

*ECCN 5E002 TSPA – Technology / Software Publicly Available*

## **MSP430x5xx and MSP430x6xx Family**

# **User's Guide**



Literature Number: SLAU208M  
June 2008–Revised February 2013

*ECCN 5E002 TSPA – Technology / Software Publicly Available*

<b>Preface .....</b>	<b>51</b>
<b>1 System Resets, Interrupts, and Operating Modes, System Control Module (SYS) .....</b>	<b>53</b>
1.1 System Control Module (SYS) Introduction .....	54
1.2 System Reset and Initialization .....	54
1.2.1 Device Initial Conditions After System Reset .....	56
1.3 Interrupts .....	56
1.3.1 (Non)Maskable Interrupts (NMIs) .....	57
1.3.2 SNMI Timing .....	58
1.3.3 Maskable Interrupts .....	59
1.3.4 Interrupt Processing .....	59
1.3.5 Interrupt Nesting .....	60
1.3.6 Interrupt Vectors .....	60
1.3.7 SYS Interrupt Vector Generators .....	61
1.4 Operating Modes .....	62
1.4.1 Entering and Exiting Low-Power Modes LPM0 Through LPM4 .....	65
1.4.2 Entering and Exiting Low-Power Modes LPMx.5 .....	65
1.4.3 Extended Time in Low-Power Modes .....	66
1.5 Principles for Low-Power Applications .....	68
1.6 Connection of Unused Pins .....	68
1.7 Reset Pin ( <b>RST/NMI</b> ) Configuration .....	69
1.8 Configuring JTAG pins .....	69
1.9 Boot Code .....	69
1.10 Bootstrap Loader (BSL) .....	69
1.11 Memory Map – Uses and Abilities .....	71
1.11.1 Vacant Memory Space .....	71
1.11.2 JTAG Lock Mechanism via the Electronic Fuse .....	71
1.12 JTAG Mailbox (JMB) System .....	72
1.12.1 JMB Configuration .....	72
1.12.2 JMBOUT0 and JMBOUT1 Outgoing Mailbox .....	72
1.12.3 JMBIN0 and JMBIN1 Incoming Mailbox .....	72
1.12.4 JMB NMI Usage .....	73
1.13 Device Descriptor Table .....	73
1.13.1 Identifying Device Type .....	74
1.13.2 TLV Descriptors .....	75
1.13.3 Peripheral Discovery Descriptor .....	76
1.13.4 CRC Computation .....	80
1.13.5 Calibration Values .....	81
1.14 SFR Registers .....	83
1.14.1 SFRIE1 Register .....	84
1.14.2 SFRIFG1 Register .....	85
1.14.3 SFRRPCR Register .....	87
1.15 SYS Registers .....	88
1.15.1 SYSCCTL Register .....	89
1.15.2 SYSSBSLC Register .....	90
1.15.3 SYSJMBC Register .....	91

1.15.4	SYSJMBI0 Register .....	92
1.15.5	SYSJMBI1 Register .....	92
1.15.6	SYSJMBO0 Register .....	93
1.15.7	SYSJMBO1 Register .....	93
1.15.8	SYSUNIV Register .....	94
1.15.9	SYSSNIV Register .....	95
1.15.10	SYSRSTIV Register .....	96
1.15.11	SYSBERRIV Register .....	97
<b>2</b>	<b>Power Management Module and Supply Voltage Supervisor .....</b>	<b>98</b>
2.1	Power Management Module (PMM) Introduction .....	99
2.2	PMM Operation .....	101
2.2.1	$V_{CORE}$ and the Regulator .....	101
2.2.2	Supply Voltage Supervisor and Monitor .....	101
2.2.3	Supply Voltage Supervisor and Monitor - Power-Up .....	107
2.2.4	Increasing $V_{CORE}$ to Support Higher MCLK Frequencies .....	107
2.2.5	Decreasing $V_{CORE}$ for Power Optimization .....	109
2.2.6	Transition From LPM3 and LPM4 Modes to AM .....	109
2.2.7	LPM3.5 and LPM4.5 .....	109
2.2.8	Brownout Reset (BOR), Software BOR, Software POR .....	109
2.2.9	SVS and SVM Performance Modes and Wakeup Times .....	110
2.2.10	PMM Interrupts .....	113
2.2.11	Port I/O Control .....	113
2.2.12	Supply Voltage Monitor Output (SVMOUT, Optional) .....	113
2.3	PMM Registers .....	114
2.3.1	PMMCTL0 Register .....	115
2.3.2	PMMCTL1 Register .....	116
2.3.3	SVSMHCTL Register .....	117
2.3.4	SVSMLCTL Register .....	118
2.3.5	SVSMIO Register .....	119
2.3.6	PMMIFG Register .....	120
2.3.7	PMMRIE Register .....	122
2.3.8	PM5CTL0 Register .....	123
<b>3</b>	<b>Battery Backup System .....</b>	<b>124</b>
3.1	Battery Backup Introduction .....	125
3.2	Battery Backup Operation .....	125
3.2.1	Battery Backup Switch Control .....	126
3.2.2	LPMx.5 and Backup Operation .....	127
3.2.3	Resistive Charger .....	127
3.3	Battery Backup Registers .....	128
3.3.1	BAKCTL Register .....	129
3.3.2	BAKCHCTL Register .....	130
<b>4</b>	<b>Auxiliary Supply System (AUX) .....</b>	<b>131</b>
4.1	Auxiliary Supply System Introduction .....	132
4.2	Auxiliary Supply Operation .....	133
4.2.1	Startup .....	134
4.2.2	Switching Control .....	134
4.2.3	Software-Controlled Switching .....	134
4.2.4	Hardware-Controlled Switching .....	135
4.2.5	Interactions Among $f_{SYS}$ , $V_{CORE}$ , $V_{DSYS}$ , $SVM_H$ , and $AUXxLVL$ .....	136
4.2.6	Auxiliary Supply Monitor .....	138
4.2.7	LPMx.5 and Auxiliary Supply Operation .....	139
4.2.8	Digital I/Os and Auxiliary Supplies .....	140
4.2.9	Measuring the Supplies .....	141

---

4.2.10	Resistive Charger .....	142
4.2.11	Auxiliary Supply Interrupts .....	142
4.2.12	Software Flow .....	143
4.2.13	Examples of AUX Operation .....	145
<b>4.3</b>	<b>AUX Registers .....</b>	<b>147</b>
4.3.1	AUXCTL0 Register .....	148
4.3.2	AUXCTL1 Register .....	149
4.3.3	AUXCTL2 Register .....	150
4.3.4	AUX2CHCTL Register .....	151
4.3.5	AUX3CHCTL Register .....	152
4.3.6	AUXADCCTL Register .....	153
4.3.7	AUXIFG Register .....	154
4.3.8	AUXIE Register .....	155
4.3.9	AUXIV Register .....	156
<b>5</b>	<b>Unified Clock System (UCS) .....</b>	<b>157</b>
5.1	Unified Clock System (UCS) Introduction .....	158
5.2	UCS Operation .....	160
5.2.1	UCS Module Features for Low-Power Applications .....	160
5.2.2	Internal Very-Low-Power Low-Frequency Oscillator (VLO) .....	160
5.2.3	Internal Trimmed Low-Frequency Reference Oscillator (REFO) .....	161
5.2.4	XT1 Oscillator .....	161
5.2.5	XT2 Oscillator .....	162
5.2.6	Digitally-Controlled Oscillator (DCO) .....	163
5.2.7	Frequency Locked Loop (FLL) .....	164
5.2.8	DCO Modulator .....	164
5.2.9	Disabling FLL Hardware and Modulator .....	165
5.2.10	FLL Operation From Low-Power Modes .....	165
5.2.11	Operation From Low-Power Modes, Requested by Peripheral Modules .....	165
5.2.12	UCS Module Fail-Safe Operation .....	167
5.2.13	Synchronization of Clock Signals .....	170
5.3	Module Oscillator (MODOSC) .....	171
5.3.1	MODOSC Operation .....	171
5.4	UCS Module Registers .....	172
5.4.1	UCSCTL0 Register .....	173
5.4.2	UCSCTL1 Register .....	174
5.4.3	UCSCTL2 Register .....	175
5.4.4	UCSCTL3 Register .....	176
5.4.5	UCSCTL4 Register .....	177
5.4.6	UCSCTL5 Register .....	178
5.4.7	UCSCTL6 Register .....	180
5.4.8	UCSCTL7 Register .....	182
5.4.9	UCSCTL8 Register .....	183
5.4.10	UCSCTL9 Register .....	184
<b>6</b>	<b>CPUX .....</b>	<b>185</b>
6.1	MSP430X CPU (CPUX) Introduction .....	186
6.2	Interrupts .....	188
6.3	CPU Registers .....	189
6.3.1	Program Counter (PC) .....	189
6.3.2	Stack Pointer (SP) .....	189
6.3.3	Status Register (SR) .....	191
6.3.4	Constant Generator Registers (CG1 and CG2) .....	192
6.3.5	General-Purpose Registers (R4 –R15) .....	193
6.4	Addressing Modes .....	195

6.4.1	Register Mode .....	196
6.4.2	Indexed Mode .....	197
6.4.3	Symbolic Mode .....	201
6.4.4	Absolute Mode .....	206
6.4.5	Indirect Register Mode .....	208
6.4.6	Indirect Autoincrement Mode .....	209
6.4.7	Immediate Mode .....	210
6.5	MSP430 and MSP430X Instructions .....	212
6.5.1	MSP430 Instructions .....	212
6.5.2	MSP430X Extended Instructions .....	217
6.6	Instruction Set Description .....	228
6.6.1	Extended Instruction Binary Descriptions .....	229
6.6.2	MSP430 Instructions .....	231
6.6.3	Extended Instructions .....	283
6.6.4	Address Instructions .....	326
<b>7</b>	<b>Flash Memory Controller .....</b>	<b>341</b>
7.1	Flash Memory Introduction .....	342
7.2	Flash Memory Segmentation .....	343
7.2.1	Segment A .....	344
7.3	Flash Memory Operation .....	345
7.3.1	Erasing Flash Memory .....	345
7.3.2	Writing Flash Memory .....	349
7.3.3	Flash Memory Access During Write or Erase .....	356
7.3.4	Stopping Write or Erase Cycle .....	357
7.3.5	Checking Flash Memory .....	357
7.3.6	Configuring and Accessing the Flash Memory Controller .....	358
7.3.7	Flash Memory Controller Interrupts .....	358
7.3.8	Programming Flash Memory Devices .....	359
7.4	FCTL Registers .....	360
7.4.1	FCTL1 Register .....	361
7.4.2	FCTL3 Register .....	362
7.4.3	FCTL4 Register .....	363
7.4.4	SFRIE1 Register .....	364
<b>8</b>	<b>Memory Integrity Detection (MID) .....</b>	<b>365</b>
8.1	MID Overview .....	366
8.2	Flash Memory With MID Support .....	367
8.3	MID Parity Check Logic .....	367
8.4	Detecting Unprogrammed Memory Accesses .....	368
8.5	MID ROM .....	368
8.6	MID Support Software Function .....	368
8.6.1	MidEnable() Function .....	369
8.6.2	MidDisable() Function .....	370
8.6.3	MidGetErrAddr() Function .....	370
8.6.4	MidCheckMem() Function .....	371
8.6.5	MidSetRaw() Function .....	371
8.6.6	MidGetParity() Function .....	372
8.6.7	MidCalcVParity() Function .....	372
8.7	User's UNMI Interrupt Handler .....	372
<b>9</b>	<b>RAM Controller (RAMCTL) .....</b>	<b>373</b>
9.1	RAM Controller (RAMCTL) Introduction .....	374
9.2	RAMCTL Operation .....	374
9.3	RAMCTL Registers .....	375

---

9.3.1	RCCTL0 Register .....	376
<b>10</b>	<b>Backup RAM .....</b>	<b>377</b>
10.1	Backup RAM Introduction and Operation .....	378
10.2	Battery Backup Registers .....	378
<b>11</b>	<b>Direct Memory Access (DMA) Controller Module .....</b>	<b>379</b>
11.1	Direct Memory Access (DMA) Introduction .....	380
11.2	DMA Operation .....	382
11.2.1	DMA Addressing Modes .....	382
11.2.2	DMA Transfer Modes .....	382
11.2.3	Initiating DMA Transfers .....	388
11.2.4	Halting Executing Instructions for DMA Transfers .....	388
11.2.5	Stopping DMA Transfers .....	389
11.2.6	DMA Channel Priorities .....	389
11.2.7	DMA Transfer Cycle Time .....	390
11.2.8	Using DMA With System Interrupts .....	390
11.2.9	DMA Controller Interrupts .....	390
11.2.10	Using the USCI_B I <sup>2</sup> C Module With the DMA Controller .....	392
11.2.11	Using ADC12 With the DMA Controller .....	392
11.2.12	Using DAC12 With the DMA Controller .....	392
11.3	DMA Registers .....	393
11.3.1	DMACTL0 Register .....	395
11.3.2	DMACTL1 Register .....	396
11.3.3	DMACTL2 Register .....	397
11.3.4	DMACTL3 Register .....	398
11.3.5	DMACTL4 Register .....	399
11.3.6	DMAxCTL Register .....	400
11.3.7	DMAxSA Register .....	402
11.3.8	DMAxDA Register .....	403
11.3.9	DMAxSZ Register .....	404
11.3.10	DMAlV Register .....	405
<b>12</b>	<b>Digital I/O Module .....</b>	<b>406</b>
12.1	Digital I/O Introduction .....	407
12.2	Digital I/O Operation .....	408
12.2.1	Input Registers (PxIN) .....	408
12.2.2	Output Registers (PxOUT) .....	408
12.2.3	Direction Registers (PxDIR) .....	408
12.2.4	Pullup or Pulldown Resistor Enable Registers (PxREN) .....	408
12.2.5	Output Drive Strength Registers (PxDS) .....	409
12.2.6	Function Select Registers (PxSEL) .....	409
12.2.7	Port Interrupts .....	409
12.2.8	Configuring Unused Port Pins .....	411
12.3	I/O Configuration and LPMx.5 Low-Power Modes .....	411
12.4	Digital I/O Registers .....	413
12.4.1	P1IV Register .....	419
12.4.2	P2IV Register .....	420
12.4.3	P1IES Register .....	421
12.4.4	P1IE Register .....	421
12.4.5	P1IFG Register .....	421
12.4.6	P2IES Register .....	422
12.4.7	P2IE Register .....	422
12.4.8	P2IFG Register .....	422
12.4.9	PxIN Register .....	423
12.4.10	PxOUT Register .....	423

12.4.11	PxDIR Register .....	423
12.4.12	PxREN Register .....	424
12.4.13	PxDS Register .....	424
12.4.14	PxSEL Register .....	424
<b>13</b>	<b>Port Mapping Controller .....</b>	<b>425</b>
13.1	Port Mapping Controller Introduction .....	426
13.2	Port Mapping Controller Operation .....	426
13.2.1	Access .....	426
13.2.2	Mapping .....	426
13.3	Port Mapping Controller Registers .....	429
13.3.1	PMAPKEYID Register .....	430
13.3.2	PMAPCTL Register .....	430
13.3.3	PxMAPy Register .....	430
<b>14</b>	<b>Cyclic Redundancy Check (CRC) Module .....</b>	<b>431</b>
14.1	Cyclic Redundancy Check (CRC) Module Introduction .....	432
14.2	CRC Standard and Bit Order .....	432
14.3	CRC Checksum Generation .....	433
14.3.1	CRC Implementation .....	433
14.3.2	Assembler Examples .....	434
14.4	CRC Registers .....	436
14.4.1	CRCDI Register .....	437
14.4.2	CRCDIRB Register .....	437
14.4.3	CRCINIRES Register .....	438
14.4.4	CRCRESR Register .....	438
<b>15</b>	<b>AES Accelerator .....</b>	<b>439</b>
15.1	AES Accelerator Introduction .....	440
15.2	AES Accelerator Operation .....	441
15.2.1	Encryption .....	442
15.2.2	Decryption .....	443
15.2.3	Decryption Key Generation .....	444
15.2.4	Using the AES Accelerator With Low-Power Modes .....	445
15.2.5	AES Accelerator Interrupts .....	445
15.2.6	Implementing Block Cipher Modes .....	445
15.3	AES_ACCEL Registers .....	446
15.3.1	AESACTL0 Register .....	447
15.3.2	AESACTL1 Register .....	448
15.3.3	AESASTAT Register .....	449
15.3.4	AESAKEY Register .....	450
15.3.5	AESADIN Register .....	451
15.3.6	AESADOUT Register .....	451
15.3.7	AESAXDIN Register .....	452
15.3.8	AESAXIN Register .....	452
<b>16</b>	<b>Watchdog Timer (WDT_A) .....</b>	<b>453</b>
16.1	WDT_A Introduction .....	454
16.2	WDT_A Operation .....	456
16.2.1	Watchdog Timer Counter (WDTCNT) .....	456
16.2.2	Watchdog Mode .....	456
16.2.3	Interval Timer Mode .....	456
16.2.4	Watchdog Timer Interrupts .....	456
16.2.5	Clock Fail-Safe Feature .....	457
16.2.6	Operation in Low-Power Modes .....	457
16.2.7	Software Examples .....	457

---

16.3	WDT_A Registers .....	458
16.3.1	WDTCTL Register .....	459
<b>17</b>	<b>Timer_A .....</b>	<b>460</b>
17.1	Timer_A Introduction .....	461
17.2	Timer_A Operation .....	463
17.2.1	16-Bit Timer Counter .....	463
17.2.2	Starting the Timer .....	463
17.2.3	Timer Mode Control .....	464
17.2.4	Capture/Compare Blocks .....	467
17.2.5	Output Unit .....	469
17.2.6	Timer_A Interrupts .....	473
17.3	Timer_A Registers .....	475
17.3.1	TAxCTL Register .....	476
17.3.2	TAxR Register .....	477
17.3.3	TAxCCTLn Register .....	478
17.3.4	TAxCCRn Register .....	480
17.3.5	TAxIV Register .....	480
17.3.6	TAxE0 Register .....	481
<b>18</b>	<b>Timer_B .....</b>	<b>482</b>
18.1	Timer_B Introduction .....	483
18.1.1	Similarities and Differences From Timer_A .....	483
18.2	Timer_B Operation .....	485
18.2.1	16-Bit Timer Counter .....	485
18.2.2	Starting the Timer .....	485
18.2.3	Timer Mode Control .....	486
18.2.4	Capture/Compare Blocks .....	489
18.2.5	Output Unit .....	492
18.2.6	Timer_B Interrupts .....	496
18.3	Timer_B Registers .....	498
18.3.1	TBxCTL Register .....	499
18.3.2	TBxR Register .....	501
18.3.3	TBxCCTLn Register .....	502
18.3.4	TBxCCRn Register .....	504
18.3.5	TBxIV Register .....	505
18.3.6	TBxE0 Register .....	506
<b>19</b>	<b>Timer_D .....</b>	<b>507</b>
19.1	Timer_D Introduction .....	508
19.1.1	Differences From Timer_B .....	508
19.2	Timer_D Operation .....	511
19.2.1	16-Bit Timer Counter .....	511
19.2.2	High-Resolution Generator .....	512
19.2.3	Starting the Timer .....	514
19.2.4	Timer Mode Control .....	514
19.2.5	PWM Generation .....	518
19.2.6	Capture/Compare Blocks .....	521
19.2.7	Compare Mode .....	524
19.2.8	Switching From Capture to Compare Mode .....	525
19.2.9	Output Unit .....	525
19.2.10	Synchronization Between Timer_D Instances .....	532
19.2.11	Timer_D Interrupts .....	532
19.3	Timer_D Registers .....	534
19.3.1	TDxCTL0 Register .....	535
19.3.2	TDxCTL1 Register .....	537

19.3.3	TDxCTL2 Register .....	538
19.3.4	TDxR Register .....	539
19.3.5	TDxCCTLn Register .....	540
19.3.6	TDxCCRn Register .....	542
19.3.7	TDxCLn Register .....	542
19.3.8	TDxHCTL0 Register .....	543
19.3.9	TDxHCTL1 Register .....	544
19.3.10	TDxHINT Register .....	545
19.3.11	TDxIV Register .....	546
<b>20</b>	<b>Timer Event Control (TEC) .....</b>	<b>547</b>
20.1	Timer Event Control Introduction .....	548
20.2	TEC Operation .....	549
20.2.1	AUXCLK Selection Sub-Block .....	549
20.2.2	External Clear Sub-Block .....	549
20.2.3	Channel Event Sub-Block .....	549
20.2.4	Module Level Connection Between TEC and Timer_D .....	550
20.2.5	Synchronization Mechanism Between Timer_D Instances .....	552
20.2.6	Timer Event Control Interrupts .....	554
20.3	TEC Registers .....	555
20.3.1	TECxCTL0 Register .....	556
20.3.2	TECxCTL1 Register .....	558
20.3.3	TECxCTL2 Register .....	560
20.3.4	TECxSTA Register .....	561
20.3.5	TECxINT Register .....	562
20.3.6	TECxIV Register .....	563
<b>21</b>	<b>Real-Time Clock (RTC) Overview .....</b>	<b>564</b>
21.1	RTC Overview .....	564
<b>22</b>	<b>Real-Time Clock (RTC_A) .....</b>	<b>565</b>
22.1	RTC_A Introduction .....	566
22.2	RTC_A Operation .....	568
22.2.1	Counter Mode .....	568
22.2.2	Calendar Mode .....	568
22.2.3	Real-Time Clock Interrupts .....	570
22.2.4	Real-Time Clock Calibration .....	572
22.3	RTC_A Registers .....	574
22.3.1	RTCCCTL0 Register .....	576
22.3.2	RTCCCTL1 Register .....	577
22.3.3	RTCCCTL2 Register .....	578
22.3.4	RTCCCTL3 Register .....	578
22.3.5	RTCCNT1 Register .....	579
22.3.6	RTCCNT2 Register .....	579
22.3.7	RTCCNT3 Register .....	579
22.3.8	RTCCNT4 Register .....	579
22.3.9	RTCCSEC Register – Calendar Mode With Hexadecimal Format .....	580
22.3.10	RTCCSEC Register – Calendar Mode With BCD Format .....	580
22.3.11	RTCCMIN Register – Calendar Mode With Hexadecimal Format .....	581
22.3.12	RTCCMIN Register – Calendar Mode With BCD Format .....	581
22.3.13	RTCHOUR Register – Calendar Mode With Hexadecimal Format .....	582
22.3.14	RTCHOUR Register – Calendar Mode With BCD Format .....	582
22.3.15	RTCDOW Register – Calendar Mode .....	583
22.3.16	RTCDAY Register – Calendar Mode With Hexadecimal Format .....	583
22.3.17	RTCDAY Register – Calendar Mode With BCD Format .....	583

---

22.3.18	RTCMON Register – Calendar Mode With Hexadecimal Format .....	584
22.3.19	RTCMON Register – Calendar Mode With BCD Format .....	584
22.3.20	RTCYEARL Register – Calendar Mode With Hexadecimal Format .....	585
22.3.21	RTCYEARL Register – Calendar Mode With BCD Format .....	585
22.3.22	RTCYEARH Register – Calendar Mode With Hexadecimal Format .....	586
22.3.23	RTCYEARH Register – Calendar Mode With BCD Format .....	586
22.3.24	RTCAMIN Register – Calendar Mode With Hexadecimal Format .....	587
22.3.25	RTCAMIN Register – Calendar Mode With BCD Format .....	587
22.3.26	RTCAHOUR Register – Calendar Mode With Hexadecimal Format .....	588
22.3.27	RTCAHOUR Register – Calendar Mode With BCD Format .....	588
22.3.28	RTCADOW Register .....	589
22.3.29	RTCADAY Register – Calendar Mode With Hexadecimal Format .....	589
22.3.30	RTCADAY Register – Calendar Mode With BCD Format .....	589
22.3.31	RTCPSP0CTL Register .....	590
22.3.32	RTCPSP1CTL Register .....	591
22.3.33	RT0PS Register .....	592
22.3.34	RT1PS Register .....	592
22.3.35	RTCIV Register .....	592
<b>23</b>	<b>Real-Time Clock B (RTC_B) .....</b>	<b>593</b>
23.1	Real-Time Clock RTC_B Introduction .....	594
23.2	RTC_B Operation .....	596
23.2.1	Real-Time Clock and Prescale Dividers .....	596
23.2.2	Real-Time Clock Alarm Function .....	596
23.2.3	Reading or Writing Real-Time Clock Registers .....	597
23.2.4	Real-Time Clock Interrupts .....	597
23.2.5	Real-Time Clock Calibration .....	599
23.2.6	Real-Time Clock Operation in LPMx.5 Low-Power Mode .....	600
23.3	RTC_B Registers .....	601
23.3.1	RTCCTL0 Register .....	603
23.3.2	RTCCTL1 Register .....	604
23.3.3	RTCCTL2 Register .....	605
23.3.4	RTCCTL3 Register .....	605
23.3.5	RTCSEC Register – Hexadecimal Format .....	606
23.3.6	RTCSEC Register – BCD Format .....	606
23.3.7	RTCMIN Register – Hexadecimal Format .....	607
23.3.8	RTCMIN Register – BCD Format .....	607
23.3.9	RTCHOUR Register – Hexadecimal Format .....	608
23.3.10	RTCHOUR Register – BCD Format .....	608
23.3.11	RTCADOW Register .....	609
23.3.12	RTCDAY Register – Hexadecimal Format .....	609
23.3.13	RTCDAY Register – BCD Format .....	609
23.3.14	RTCMON Register – Hexadecimal Format .....	610
23.3.15	RTCMON Register – BCD Format .....	610
23.3.16	RTCYEAR Register – Hexadecimal Format .....	611
23.3.17	RTCYEAR Register – BCD Format .....	611
23.3.18	RTCAMIN Register – Hexadecimal Format .....	612
23.3.19	RTCAMIN Register – BCD Format .....	612
23.3.20	RTCAHOUR Register – Hexadecimal Format .....	613
23.3.21	RTCAHOUR Register – BCD Format .....	613
23.3.22	RTCADOW Register .....	614
23.3.23	RTCADAY Register – Hexadecimal Format .....	615
23.3.24	RTCADAY Register – BCD Format .....	615
23.3.25	RTCPSP0CTL Register .....	616

23.3.26	RTCPs1CTL Register .....	617
23.3.27	RTCPs0 Register .....	618
23.3.28	RTCPs1 Register .....	618
23.3.29	RTCIV Register .....	619
23.3.30	BIN2BCD Register .....	620
23.3.31	BCD2BIN Register .....	620
<b>24</b>	<b>Real-Time Clock C (RTC_C) .....</b>	<b>621</b>
24.1	Real-Time Clock (RTC_C) Introduction .....	622
24.2	RTC_C Operation .....	624
24.2.1	Calendar Mode .....	624
24.2.2	Real-Time Clock and Prescale Dividers .....	624
24.2.3	Real-Time Clock Alarm Function .....	624
24.2.4	Real-Time Clock Protection .....	625
24.2.5	Reading or Writing Real-Time Clock Registers .....	625
24.2.6	Real-Time Clock Interrupts .....	626
24.2.7	Real-Time Clock Calibration for Crystal Offset Error .....	628
24.2.8	Real-Time Clock Compensation for Crystal Temperature Drift .....	628
24.2.9	Real-Time Clock Operation in LPM3.5 Low-Power Mode .....	631
24.3	RTC_C Operation - Device-Dependent Features .....	632
24.3.1	Counter Mode .....	632
24.3.2	Real-Time Clock Event/Tamper Detection With Time Stamp .....	633
24.4	RTC_C Registers .....	635
24.4.1	RTCCTL0_L Register .....	638
24.4.2	RTCCTL0_H Register .....	639
24.4.3	RTCCTL1 Register .....	640
24.4.4	RTCCTL3 Register .....	641
24.4.5	RTCOCAL Register .....	641
24.4.6	RTCTCMP Register .....	642
24.4.7	RTCNT1 Register .....	643
24.4.8	RTCNT2 Register .....	643
24.4.9	RTCNT3 Register .....	643
24.4.10	RTCNT4 Register .....	643
24.4.11	RTCSEC Register – Calendar Mode With Hexadecimal Format .....	644
24.4.12	RTCSEC Register – Calendar Mode With BCD Format .....	644
24.4.13	RTCMIN Register – Calendar Mode With Hexadecimal Format .....	645
24.4.14	RTCMIN Register – Calendar Mode With BCD Format .....	645
24.4.15	RTCHOUR Register – Calendar Mode With Hexadecimal Format .....	646
24.4.16	RTCHOUR Register – Calendar Mode With BCD Format .....	646
24.4.17	RTCDOW Register – Calendar Mode .....	647
24.4.18	RTCDAY Register – Calendar Mode With Hexadecimal Format .....	647
24.4.19	RTCDAY Register – Calendar Mode With BCD Format .....	647
24.4.20	RTCMON Register – Calendar Mode With Hexadecimal Format .....	648
24.4.21	RTCMON Register – Calendar Mode With BCD Format .....	648
24.4.22	RTCYEAR Register – Calendar Mode With Hexadecimal Format .....	649
24.4.23	RTCYEAR Register – Calendar Mode With BCD Format .....	649
24.4.24	RTCAMIN Register – Calendar Mode With Hexadecimal Format .....	650
24.4.25	RTCAMIN Register – Calendar Mode With BCD Format .....	650
24.4.26	RTCAHOUR Register .....	651
24.4.27	RTCAHOUR Register – Calendar Mode With BCD Format .....	651
24.4.28	RTCADOW Register – Calendar Mode .....	652
24.4.29	RTCADAY Register – Calendar Mode With Hexadecimal Format .....	652
24.4.30	RTCADAY Register – Calendar Mode With BCD Format .....	652
24.4.31	RTCPs0CTL Register .....	653

---

24.4.32	RTCPs1CTL Register .....	654
24.4.33	RTCPs0 Register .....	656
24.4.34	RTCPs1 Register .....	656
24.4.35	RTCIV Register .....	657
24.4.36	BIN2BCD Register .....	658
24.4.37	BCD2BIN Register .....	658
24.4.38	RTCSECBAKx Register – Hexadecimal Format .....	659
24.4.39	RTCSECBAKx Register – BCD Format .....	659
24.4.40	RTCMINBAKx Register – Hexadecimal Format .....	660
24.4.41	RTCMINBAKx Register – BCD Format .....	660
24.4.42	RTCHOURBAKx Register – Hexadecimal Format .....	661
24.4.43	RTCHOURBAKx Register – BCD Format .....	661
24.4.44	RTCDAYBAKx Register – Hexadecimal Format .....	662
24.4.45	RTCDAYBAKx Register – BCD Format .....	662
24.4.46	RTCMONBAKx Register – Hexadecimal Format .....	663
24.4.47	RTCMONBAKx Register – BCD Format .....	663
24.4.48	RTCYEARBAKx Register – Hexadecimal Format .....	664
24.4.49	RTCYEARBAKx Register – BCD Format .....	664
24.4.50	RTCTCCTL0 Register .....	665
24.4.51	RTCTCCTL1 Register .....	665
24.4.52	RTCCAPxCTL Register .....	666
<b>25</b>	<b>32-Bit Hardware Multiplier (MPY32) .....</b>	<b>667</b>
25.1	32-Bit Hardware Multiplier (MPY32) Introduction .....	668
25.2	MPY32 Operation .....	670
25.2.1	Operand Registers .....	671
25.2.2	Result Registers .....	672
25.2.3	Software Examples .....	673
25.2.4	Fractional Numbers .....	674
25.2.5	Putting It All Together .....	677
25.2.6	Indirect Addressing of Result Registers .....	680
25.2.7	Using Interrupts .....	680
25.2.8	Using DMA .....	681
25.3	MPY32 Registers .....	682
25.3.1	MPY32CTL0 Register .....	684
<b>26</b>	<b>REF .....</b>	<b>685</b>
26.1	REF Introduction .....	686
26.2	Principle of Operation .....	688
26.2.1	Low-Power Operation .....	688
26.2.2	REFCTL .....	689
26.2.3	Reference System Requests .....	690
26.3	REF Registers .....	692
26.3.1	REFCTL0 Register (offset = 00h) [reset = 0080h] .....	693
<b>27</b>	<b>ADC10_A .....</b>	<b>695</b>
27.1	ADC10_A Introduction .....	696
27.2	ADC10_A Operation .....	698
27.2.1	10-Bit ADC Core .....	698
27.2.2	ADC10_A Inputs and Multiplexer .....	698
27.2.3	Voltage Reference Generator .....	699
27.2.4	Auto Power Down .....	699
27.2.5	Sample and Conversion Timing .....	699
27.2.6	Conversion Result .....	701
27.2.7	ADC10_A Conversion Modes .....	701
27.2.8	Window Comparator .....	706

27.2.9	Using the Integrated Temperature Sensor .....	707
27.2.10	ADC10_A Grounding and Noise Considerations .....	708
27.2.11	ADC10_A Interrupts .....	708
27.3	ADC10_A Registers .....	710
27.3.1	ADC10CTL0 Register .....	711
27.3.2	ADC10CTL1 Register .....	712
27.3.3	ADC10CTL2 Register .....	714
27.3.4	ADC10MEM0 Register .....	715
27.3.5	ADC10MEM0 Register, 2s-Complement Format .....	715
27.3.6	ADC10MCTL0 Register .....	716
27.3.7	ADC10HI Register .....	717
27.3.8	ADC10HI Register, 2s-Complement Format .....	717
27.3.9	ADC10LO Register .....	718
27.3.10	ADC10LO Register, 2s-Complement Format .....	718
27.3.11	ADC10IE Register .....	719
27.3.12	ADC10IFG Register .....	720
27.3.13	ADC10IV Register .....	721
<b>28</b>	<b>ADC12_A .....</b>	<b>722</b>
28.1	ADC12_A Introduction .....	723
28.2	ADC12_A Operation .....	726
28.2.1	12-Bit ADC Core .....	726
28.2.2	ADC12_A Inputs and Multiplexer .....	726
28.2.3	Voltage Reference Generator .....	727
28.2.4	Auto Power Down .....	728
28.2.5	Sample and Conversion Timing .....	728
28.2.6	Conversion Memory .....	730
28.2.7	ADC12_A Conversion Modes .....	730
28.2.8	Using the Integrated Temperature Sensor .....	736
28.2.9	ADC12_A Grounding and Noise Considerations .....	737
28.2.10	ADC12_A Interrupts .....	738
28.3	ADC12_A Registers .....	740
28.3.1	ADC12CTL0 Register .....	742
28.3.2	ADC12CTL1 Register .....	744
28.3.3	ADC12CTL2 Register .....	745
28.3.4	ADC12MEMx Register .....	746
28.3.5	ADC12MCTLx Register .....	747
28.3.6	ADC12IE Register .....	748
28.3.7	ADC12IFG Register .....	750
28.3.8	ADC12IV Register .....	752
<b>29</b>	<b>SD24_B .....</b>	<b>753</b>
29.1	SD24_B Introduction .....	754
29.2	SD24_B Operation .....	758
29.2.1	Principle of Operation .....	758
29.2.2	ADC Core .....	759
29.2.3	Voltage Reference .....	759
29.2.4	Modulator Clock .....	759
29.2.5	Auto Power-Down .....	759
29.2.6	Analog Inputs .....	759
29.2.7	Digital Filter .....	760
29.2.8	Bit Stream Input and Output .....	764
29.2.9	Conversion Modes .....	764
29.2.10	Conversion Operation Using Preload .....	766
29.2.11	Grounding and Noise Considerations .....	767

---

29.2.12	Trigger Generator .....	768
29.2.13	SD24_B Interrupts .....	769
29.2.14	Using SD24_B With DMA .....	769
<b>29.3</b>	<b>SD24_B Registers .....</b>	<b>770</b>
29.3.1	SD24BCTL0 Register .....	772
29.3.2	SD24BCTL1 Register .....	774
29.3.3	SD24BTRGCTL Register .....	775
29.3.4	SD24BIFG Register .....	776
29.3.5	SD24BIE Register .....	779
29.3.6	SD24BIV Register .....	781
29.3.7	SD24BCCTLx Register .....	782
29.3.8	SD24BINCTLx Register .....	784
29.3.9	SD24BOSRx Register .....	785
29.3.10	SD24BTRGOSR Register .....	785
29.3.11	SD24BPREx Register .....	786
29.3.12	SD24BTRGPRE Register .....	786
29.3.13	SD24BMEMLx Register .....	787
29.3.14	SD24BMEMHx Register .....	787
<b>30</b>	<b>DAC12_A .....</b>	<b>788</b>
30.1	DAC12_A Introduction .....	789
30.2	DAC12_A Operation .....	792
30.2.1	DAC12_A Core .....	792
30.2.2	DAC12_A Port Selection .....	792
30.2.3	DAC12_A Reference .....	792
30.2.4	Updating the DAC12_A Voltage Output .....	792
30.2.5	DAC12_xDAT Data Formats .....	793
30.2.6	DAC12_A Output Amplifier Offset Calibration .....	793
30.2.7	Grouping Multiple DAC12_A Modules .....	794
30.2.8	DAC12_A Interrupts .....	795
30.3	DAC Outputs .....	796
30.4	DAC12_A Registers .....	797
30.4.1	DAC12_xCTL0 Register .....	798
30.4.2	DAC12_xCTL1 Register .....	800
30.4.3	DAC12_xDAT Register, Unsigned 12-Bit Binary Format, Right Justified .....	801
30.4.4	DAC12_xDAT Register, Unsigned 12-Bit Binary Format, Left Justified .....	801
30.4.5	DAC12_xDAT Register, 2s-Complement 12-Bit Binary Format, Right Justified .....	802
30.4.6	DAC12_xDAT Register, 2s-Complement 12-Bit Binary Format, Left Justified .....	802
30.4.7	DAC12_xDAT Register, Unsigned 8-Bit Binary Format, Right Justified .....	803
30.4.8	DAC12_xDAT Register, Unsigned 8-Bit Binary Format, Left Justified .....	803
30.4.9	DAC12_xDAT Register, 2s-Complement 8-Bit Binary Format, Right Justified .....	804
30.4.10	DAC12_xDAT Register, 2s-Complement 8-Bit Binary Format, Left Justified .....	804
30.4.11	DAC12_xCALCTL Register .....	805
30.4.12	DAC12_xCALDAT Register .....	805
30.4.13	DAC12IV Register .....	806
<b>31</b>	<b>Comp_B .....</b>	<b>807</b>
31.1	Comp_B Introduction .....	808
31.2	Comp_B Operation .....	809
31.2.1	Comparator .....	809
31.2.2	Analog Input Switches .....	809
31.2.3	Port Logic .....	809
31.2.4	Input Short Switch .....	809
31.2.5	Output Filter .....	810
31.2.6	Reference Voltage Generator .....	811

---

31.2.7	Comp_B, Port Disable Register CBPD .....	812
31.2.8	Comp_B Interrupts .....	812
31.2.9	Comp_B Used to Measure Resistive Elements .....	812
31.3	Comp_B Registers .....	814
31.3.1	CBCTL0 Register .....	815
31.3.2	CBCTL1 Register .....	816
31.3.3	CBCTL2 Register .....	818
31.3.4	CBCTL3 Register .....	819
31.3.5	CBINT Register .....	821
31.3.6	CBIV Register .....	822
<b>32</b>	<b>LCD_B Controller .....</b>	<b>823</b>
32.1	LCD_B Controller Introduction .....	824
32.2	LCD_B Controller Operation .....	826
32.2.1	LCD Memory .....	826
32.2.2	LCD Timing Generation .....	826
32.2.3	Blanking the LCD .....	827
32.2.4	LCD Blinking .....	827
32.2.5	LCD_B Voltage And Bias Generation .....	828
32.2.6	LCD Outputs .....	830
32.2.7	LCD_B Interrupts .....	830
32.2.8	Static Mode .....	832
32.2.9	2-Mux Mode .....	835
32.2.10	3-Mux Mode .....	838
32.2.11	4-Mux Mode .....	841
32.3	LCD_B Registers .....	844
32.3.1	LCDBCTL0 Register .....	847
32.3.2	LCDBCTL1 Register .....	848
32.3.3	LCDBBLKCTL Register .....	849
32.3.4	LCDBMEMCTL Register .....	850
32.3.5	LCDBVCTL Register .....	851
32.3.6	LCDBPCTL0 Register .....	853
32.3.7	LCDBPCTL1 Register .....	853
32.3.8	LCDBPCTL2 Register .....	854
32.3.9	LCDBPCTL3 Register .....	854
32.3.10	LCDBCPCCTL Register .....	855
32.3.11	LCDBIV Register .....	856
<b>33</b>	<b>LCD_C Controller .....</b>	<b>857</b>
33.1	LCD_C Introduction .....	858
33.2	LCD_C Operation .....	860
33.2.1	LCD Memory .....	860
33.2.2	LCD Timing Generation .....	861
33.2.3	Blanking the LCD .....	862
33.2.4	LCD Blinking .....	862
33.2.5	LCD Voltage And Bias Generation .....	863
33.2.6	LCD Outputs .....	866
33.2.7	LCD Interrupts .....	867
33.2.8	Static Mode .....	869
33.2.9	2-Mux Mode .....	870
33.2.10	3-Mux Mode .....	871
33.2.11	4-Mux Mode .....	872
33.2.12	6-Mux Mode .....	873
33.2.13	8-Mux Mode .....	874
33.3	LCD_C Registers .....	876

---

33.3.1	LCDCCCTL0 Register .....	881
33.3.2	LCDCCCTL1 Register .....	883
33.3.3	LCDCBLKCTL Register .....	884
33.3.4	LCDCMEMCTL Register .....	885
33.3.5	LCDCVCTL Register .....	886
33.3.6	LCDCPCTL0 Register .....	888
33.3.7	LCDCPCTL1 Register .....	889
33.3.8	LCDCPCTL2 Register .....	890
33.3.9	LCDCPCTL3 Register .....	891
33.3.10	LCDCCPCTL Register .....	892
33.3.11	LCDCIV Register .....	892
<b>34</b>	<b>Universal Serial Communication Interface – UART Mode .....</b>	<b>893</b>
34.1	Universal Serial Communication Interface (USCI) Overview .....	894
34.2	USCI Introduction – UART Mode .....	895
34.3	USCI Operation – UART Mode .....	897
34.3.1	USCI Initialization and Reset .....	897
34.3.2	Character Format .....	897
34.3.3	Asynchronous Communication Format .....	897
34.3.4	Automatic Baud-Rate Detection .....	900
34.3.5	IrDA Encoding and Decoding .....	901
34.3.6	Automatic Error Detection .....	902
34.3.7	USCI Receive Enable .....	903
34.3.8	USCI Transmit Enable .....	903
34.3.9	UART Baud-Rate Generation .....	904
34.3.10	Setting a Baud Rate .....	906
34.3.11	Transmit Bit Timing .....	906
34.3.12	Receive Bit Timing .....	907
34.3.13	Typical Baud Rates and Errors .....	908
34.3.14	Using the USCI Module in UART Mode With Low-Power Modes .....	911
34.3.15	USCI Interrupts .....	911
34.4	USCI_A UART Mode Registers .....	913
34.4.1	UCAxCTL0 Register .....	914
34.4.2	UCAxCTL1 Register .....	915
34.4.3	UCAxBR0 Register .....	916
34.4.4	UCAxBR1 Register .....	916
34.4.5	UCAxMCTL Register .....	916
34.4.6	UCAxSTAT Register .....	917
34.4.7	UCAxRXBUF Register .....	918
34.4.8	UCAxTXBUF Register .....	918
34.4.9	UCAxIRTCTL Register .....	919
34.4.10	UCAxIRRCTL Register .....	919
34.4.11	UCAxABCTL Register .....	920
34.4.12	UCAxIE Register .....	921
34.4.13	UCAxIFG Register .....	921
34.4.14	UCAxIV Register .....	922
<b>35</b>	<b>Universal Serial Communication Interface – SPI Mode .....</b>	<b>923</b>
35.1	Universal Serial Communication Interface (USCI) Overview .....	924
35.2	USCI Introduction – SPI Mode .....	925
35.3	USCI Operation – SPI Mode .....	927
35.3.1	USCI Initialization and Reset .....	927
35.3.2	Character Format .....	927
35.3.3	Master Mode .....	928
35.3.4	Slave Mode .....	929

35.3.5	SPI Enable .....	929
35.3.6	Serial Clock Control .....	930
35.3.7	Using the SPI Mode With Low-Power Modes .....	930
35.3.8	SPI Interrupts .....	931
35.4	USCI_A SPI Mode Registers .....	932
35.4.1	UCAxCTL0 Register .....	933
35.4.2	UCAxCTL1 Register .....	934
35.4.3	UCAxBR0 Register .....	935
35.4.4	UCAxBR1 Register .....	935
35.4.5	UCAxMCTL Register .....	935
35.4.6	UCAxSTAT Register .....	936
35.4.7	UCAxRXBUF Register .....	937
35.4.8	UCAxTXBUF Register .....	937
35.4.9	UCAxIE Register .....	938
35.4.10	UCAxIFG Register .....	938
35.4.11	UCAxIV Register .....	939
35.5	USCI_B SPI Mode Registers .....	940
35.5.1	UCBxCTL0 Register .....	941
35.5.2	UCBxCTL1 Register .....	942
35.5.3	UCBxBR0 Register .....	943
35.5.4	UCBxBR1 Register .....	943
35.5.5	UCBxMCTL Register .....	943
35.5.6	UCBxSTAT Register .....	944
35.5.7	UCBxRXBUF Register .....	945
35.5.8	UCBxTXBUF Register .....	945
35.5.9	UCBxIE Register .....	946
35.5.10	UCBxIFG Register .....	946
35.5.11	UCBxIV Register .....	947
<b>36</b>	<b>Universal Serial Communication Interface – I<sup>2</sup>C Mode .....</b>	<b>948</b>
36.1	Universal Serial Communication Interface (USCI) Overview .....	949
36.2	USCI Introduction – I <sup>2</sup> C Mode .....	950
36.3	USCI Operation – I <sup>2</sup> C Mode .....	951
36.3.1	USCI Initialization and Reset .....	952
36.3.2	I <sup>2</sup> C Serial Data .....	952
36.3.3	I <sup>2</sup> C Addressing Modes .....	954
36.3.4	I <sup>2</sup> C Module Operating Modes .....	955
36.3.5	I <sup>2</sup> C Clock Generation and Synchronization .....	966
36.3.6	Using the USCI Module in I <sup>2</sup> C Mode With Low-Power Modes .....	967
36.3.7	USCI Interrupts in I <sup>2</sup> C Mode .....	967
36.4	USCI_B I <sup>2</sup> C Mode Registers .....	970
36.4.1	UCBxCTL0 Register .....	971
36.4.2	UCBxCTL1 Register .....	972
36.4.3	UCBxBR0 Register .....	973
36.4.4	UCBxBR1 Register .....	973
36.4.5	UCBxSTAT Register .....	974
36.4.6	UCBxRXBUF Register .....	975
36.4.7	UCBxTXBUF Register .....	975
36.4.8	UCBxI2COA Register .....	976
36.4.9	UCBxI2CSA Register .....	976
36.4.10	UCBxIE Register .....	977
36.4.11	UCBxIFG Register .....	978
36.4.12	UCBxIV Register .....	979
<b>37</b>	<b>Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode .....</b>	<b>980</b>

---

37.1	Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview .....	981
37.2	eUSCI_A Introduction – UART Mode .....	981
37.3	eUSCI_A Operation – UART Mode .....	983
37.3.1	eUSCI_A Initialization and Reset .....	983
37.3.2	Character Format .....	983
37.3.3	Asynchronous Communication Format .....	983
37.3.4	Automatic Baud-Rate Detection .....	986
37.3.5	IrDA Encoding and Decoding .....	987
37.3.6	Automatic Error Detection .....	988
37.3.7	eUSCI_A Receive Enable .....	989
37.3.8	eUSCI_A Transmit Enable .....	989
37.3.9	UART Baud-Rate Generation .....	990
37.3.10	Setting a Baud Rate .....	992
37.3.11	Transmit Bit Timing - Error calculation .....	993
37.3.12	Receive Bit Timing – Error Calculation .....	993
37.3.13	Typical Baud Rates and Errors .....	994
37.3.14	Using the eUSCI_A Module in UART Mode With Low-Power Modes .....	996
37.3.15	eUSCI_A Interrupts .....	996
37.4	eUSCI_A UART Registers .....	998
37.4.1	UCAxCTLW0 Register .....	999
37.4.2	UCAxCTLW1 Register .....	1000
37.4.3	UCAxBRW Register .....	1001
37.4.4	UCAxMCTLW Register .....	1001
37.4.5	UCAxSTATW Register .....	1002
37.4.6	UCAxRXBUF Register .....	1003
37.4.7	UCAxTXBUF Register .....	1003
37.4.8	UCAxABCTL Register .....	1004
37.4.9	UCAxIRCTL Register .....	1005
37.4.10	UCAxIE Register .....	1006
37.4.11	UCAxIFG Register .....	1007
37.4.12	UCAxIV Register .....	1008
<b>38</b>	<b>Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode .....</b>	<b>1009</b>
38.1	Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview .....	1010
38.2	eUSCI Introduction – SPI Mode .....	1010
38.3	eUSCI Operation – SPI Mode .....	1012
38.3.1	eUSCI Initialization and Reset .....	1012
38.3.2	Character Format .....	1013
38.3.3	Master Mode .....	1013
38.3.4	Slave Mode .....	1014
38.3.5	SPI Enable .....	1015
38.3.6	Serial Clock Control .....	1015
38.3.7	Using the SPI Mode With Low-Power Modes .....	1016
38.3.8	SPI Interrupts .....	1016
38.4	eUSCI_A SPI Registers .....	1018
38.4.1	UCAxCTLW0 Register .....	1019
38.4.2	UCAxBRW Register .....	1021
38.4.3	UCAxSTATW Register .....	1022
38.4.4	UCAxRXBUF Register .....	1023
38.4.5	UCAxTXBUF Register .....	1024
38.4.6	UCAxIE Register .....	1025
38.4.7	UCAxIFG Register .....	1026
38.4.8	UCAxIV Register .....	1027
38.5	eUSCI_B SPI Registers .....	1028

38.5.1	UCBxCTLW0 Register .....	1029
38.5.2	UCBxBRW Register .....	1031
38.5.3	UCBxSTATW Register .....	1031
38.5.4	UCBxRXBUF Register .....	1032
38.5.5	UCBxTXBUF Register .....	1032
38.5.6	UCBxIE Register .....	1033
38.5.7	UCBxIFG Register .....	1033
38.5.8	UCBxIV Register .....	1034
<b>39</b>	<b>Enhanced Universal Serial Communication Interface (eUSCI) – I<sup>2</sup>C Mode .....</b>	<b>1035</b>
39.1	Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview .....	1036
39.2	eUSCI_B Introduction – I <sup>2</sup> C Mode .....	1036
39.3	eUSCI_B Operation – I <sup>2</sup> C Mode .....	1037
39.3.1	eUSCI_B Initialization and Reset .....	1038
39.3.2	I <sup>2</sup> C Serial Data .....	1038
39.3.3	I <sup>2</sup> C Addressing Modes .....	1039
39.3.4	I <sup>2</sup> C Quick Setup .....	1040
39.3.5	I <sup>2</sup> C Module Operating Modes .....	1041
39.3.6	Glitch Filtering .....	1051
39.3.7	I <sup>2</sup> C Clock Generation and Synchronization .....	1051
39.3.8	Byte Counter .....	1052
39.3.9	Multiple Slave Addresses .....	1053
39.3.10	Using the eUSCI_B Module in I <sup>2</sup> C Mode With Low-Power Modes .....	1054
39.3.11	eUSCI_B Interrupts in I <sup>2</sup> C Mode .....	1054
39.4	eUSCI_B I2C Registers .....	1057
39.4.1	UCBxCTLW0 Register .....	1058
39.4.2	UCBxCTLW1 Register .....	1060
39.4.3	UCBxBRW Register .....	1062
39.4.4	UCBxSTATW .....	1062
39.4.5	UCBxTBCNT Register .....	1063
39.4.6	UCBxRXBUF Register .....	1064
39.4.7	UCBxTXBUF .....	1064
39.4.8	UCBxI2COA0 Register .....	1065
39.4.9	UCBxI2COA1 Register .....	1066
39.4.10	UCBxI2COA2 Register .....	1066
39.4.11	UCBxI2COA3 Register .....	1067
39.4.12	UCBxADDRX Register .....	1067
39.4.13	UCBxADDMASK Register .....	1068
39.4.14	UCBxI2CSA Register .....	1068
39.4.15	UCBxIE Register .....	1069
39.4.16	UCBxIFG Register .....	1071
39.4.17	UCBxIV Register .....	1073
<b>40</b>	<b>USB Module .....</b>	<b>1074</b>
40.1	USB Introduction .....	1075
40.2	USB Operation .....	1077
40.2.1	USB Transceiver (PHY) .....	1077
40.2.2	USB Power System .....	1078
40.2.3	USB Phase-Locked Loop (PLL) .....	1081
40.2.4	USB Controller Engine .....	1083
40.2.5	USB Vector Interrupts .....	1087
40.2.6	Power Consumption .....	1087
40.2.7	Suspend and Resume .....	1088
40.3	USB Transfers .....	1088
40.3.1	Control Transfers .....	1088

[www.ti.com](http://www.ti.com)


---

40.3.2	Interrupt Transfers .....	1092
40.3.3	Bulk Transfers .....	1093
40.4	USB Registers .....	1095
40.4.1	USB Configuration Registers .....	1095
40.4.2	USB Control Registers .....	1103
40.4.3	USB Buffer Registers and Memory .....	1120
<b>41</b>	<b>LDO-PWR Module .....</b>	<b>1131</b>
41.1	LDO-PWR Introduction .....	1132
41.2	LDO-PWR Operation .....	1133
41.2.1	Enabling/Disabling .....	1133
41.2.2	Powering the Rest of the MSP430 from the LDO-PWR .....	1133
41.2.3	Powering Other Components in the System from LDO-PWR .....	1134
41.2.4	Applications That Do Not Require LDO-PWR .....	1134
41.2.5	Current Limitation and Overload Protection .....	1134
41.2.6	LDO-PWR Interrupts .....	1135
41.2.7	Port U Control .....	1135
41.3	LDO-PWR Registers .....	1136
41.3.1	LDOKEYPID Register .....	1137
41.3.2	PUCTL Register .....	1137
41.3.3	LDOPWRCTL Register .....	1138
<b>42</b>	<b>Embedded Emulation Module (EEM) .....</b>	<b>1139</b>
42.1	Embedded Emulation Module (EEM) Introduction .....	1140
42.2	EEM Building Blocks .....	1142
42.2.1	Triggers .....	1142
42.2.2	Trigger Sequencer .....	1142
42.2.3	State Storage (Internal Trace Buffer) .....	1142
42.2.4	Cycle Counter .....	1142
42.2.5	Clock Control .....	1143
42.3	EEM Configurations .....	1143
<b>Revision History .....</b>	<b>1144</b>	

## List of Figures

1-1.	BOR/POR/PUC Reset Circuit .....	55
1-2.	Interrupt Priority .....	57
1-3.	NMIs With Reentrance Protection .....	58
1-4.	Interrupt Processing .....	59
1-5.	Return From Interrupt .....	60
1-6.	Operation Modes .....	63
1-7.	Devices Descriptor Table .....	74
1-8.	SFRIE1 Register .....	84
1-9.	SFRIFG1 Register .....	85
1-10.	SFRRPCR Register .....	87
1-11.	SYSCCTL Register .....	89
1-12.	SYSBSDL Register .....	90
1-13.	SYSJMBC Register .....	91
1-14.	SYSJMBI0 Register .....	92
1-15.	SYSJMBI1 Register .....	92
1-16.	SYSJMBO0 Register .....	93
1-17.	SYSJMBO1 Register .....	93
1-18.	SYSUNIV Register .....	94
1-19.	SYSSNIV Register .....	95
1-20.	SYSRSTIV Register .....	96
1-21.	SYSBERRIV Register .....	97
2-1.	System Frequency, Supply Voltage, and Core Voltage – See Device-Specific Data Sheet .....	99
2-2.	PMM Block Diagram .....	100
2-3.	Available SVM <sub>H</sub> Settings Versus VCORE Settings .....	103
2-4.	High-Side and Low-Side Voltage Failure and Resulting PMM Actions .....	104
2-5.	High-Side SVS and SVM .....	105
2-6.	Low-Side SVS and SVM .....	106
2-7.	PMM Action at Device Power-Up .....	107
2-8.	Changing V <sub>CORE</sub> and SVM <sub>L</sub> and SVS <sub>L</sub> Levels .....	108
2-9.	PMMCTL0 Register .....	115
2-10.	PMMCTL1 Register .....	116
2-11.	SVSMHCTL Register .....	117
2-12.	SVSMLCTL Register .....	118
2-13.	SVSMIO Register .....	119
2-14.	PMMIFG Register .....	120
2-15.	PMMRIE Register .....	122
2-16.	PM5CTL0 Register .....	123
3-1.	Battery Backup Switch Overview .....	126
3-2.	Charger Block Diagram .....	127
3-3.	BAKCTL Register .....	129
3-4.	BAKCHCTL Register .....	130
4-1.	Auxiliary Supply Switch Overview .....	133
4-2.	System Frequency vs Supply Voltage .....	137
4-3.	Available SVM <sub>H</sub> Settings vs V <sub>CORE</sub> Settings .....	137
4-4.	Available AUXxLVL Settings vs SVM <sub>H</sub> Settings .....	138
4-5.	Auxiliary Supply Monitor Block Diagram .....	139
4-6.	I/Os Powered by Auxiliary Supplies .....	141

[www.ti.com](http://www.ti.com)


---

4-7.	AUX Connection to ADC .....	141
4-8.	Charger Block Diagram .....	142
4-9.	Software Flow Chart .....	144
4-10.	AUXCTL0 Register .....	148
4-11.	AUXCTL1 Register .....	149
4-12.	AUXCTL2 Register .....	150
4-13.	AUX2CHCTL Register.....	151
4-14.	AUX3CHCTL Register.....	152
4-15.	AUXADCCTL Register .....	153
4-16.	AUXIFG Register.....	154
4-17.	AUXIE Register.....	155
4-18.	AUXIV Register.....	156
5-1.	UCS Block Diagram .....	159
5-2.	Modulator Patterns.....	165
5-3.	Module Request Clock System .....	166
5-4.	Oscillator Fault Logic .....	169
5-5.	Switch MCLK from DCOCLK to XT1CLK .....	170
5-6.	UCSCTL0 Register .....	173
5-7.	UCSCTL1 Register .....	174
5-8.	UCSCTL2 Register .....	175
5-9.	UCSCTL3 Register .....	176
5-10.	UCSCTL4 Register .....	177
5-11.	UCSCTL5 Register .....	178
5-12.	UCSCTL6 Register .....	180
5-13.	UCSCTL7 Register .....	182
5-14.	UCSCTL8 Register .....	183
5-15.	UCSCTL9 Register .....	184
6-1.	MSP430X CPU Block Diagram .....	187
6-2.	PC Storage on the Stack for Interrupts .....	188
6-3.	Program Counter.....	189
6-4.	PC Storage on the Stack for CALLA .....	189
6-5.	Stack Pointer .....	190
6-6.	Stack Usage .....	190
6-7.	PUSHX.A Format on the Stack .....	190
6-8.	PUSH SP, POP SP Sequence .....	190
6-9.	SR Bits .....	191
6-10.	Register-Byte and Byte-Register Operation .....	193
6-11.	Register-Word Operation .....	193
6-12.	Word-Register Operation .....	194
6-13.	Register – Address-Word Operation .....	194
6-14.	Address-Word – Register Operation .....	195
6-15.	Indexed Mode in Lower 64 KB .....	197
6-16.	Indexed Mode in Upper Memory.....	198
6-17.	Overflow and Underflow for Indexed Mode.....	199
6-18.	Example for Indexed Mode .....	200
6-19.	Symbolic Mode Running in Lower 64 KB .....	202
6-20.	Symbolic Mode Running in Upper Memory .....	203
6-21.	Overflow and Underflow for Symbolic Mode .....	204
6-22.	MSP430 Double-Operand Instruction Format.....	212

6-23. MSP430 Single-Operand Instructions .....	213
6-24. Format of Conditional Jump Instructions.....	214
6-25. Extension Word for Register Modes.....	217
6-26. Extension Word for Non-Register Modes .....	217
6-27. Example for Extended Register or Register Instruction.....	218
6-28. Example for Extended Immediate or Indexed Instruction.....	219
6-29. Extended Format I Instruction Formats .....	220
6-30. 20-Bit Addresses in Memory .....	220
6-31. Extended Format II Instruction Format.....	221
6-32. PUSHM and POPM Instruction Format .....	222
6-33. RRCM, RRAM, RRUM, and RLAM Instruction Format .....	222
6-34. BRA Instruction Format .....	222
6-35. CALLA Instruction Format .....	222
6-36. Decrement Overlap .....	248
6-37. Stack After a RET Instruction .....	267
6-38. Destination Operand—Arithmetic Shift Left .....	269
6-39. Destination Operand—Carry Left Shift .....	270
6-40. Rotate Right Arithmetically RRA.B and RRA.W .....	271
6-41. Rotate Right Through Carry RRC.B and RRC.W.....	272
6-42. Swap Bytes in Memory.....	279
6-43. Swap Bytes in a Register .....	279
6-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A .....	306
6-45. Destination Operand-Arithmetic Shift Left .....	307
6-46. Destination Operand-Carry Left Shift.....	308
6-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A .....	309
6-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode .....	311
6-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode .....	311
6-50. Rotate Right Through Carry RRCM[.W] and RRCM.A.....	313
6-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode .....	315
6-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode .....	315
6-53. Rotate Right Unsigned RRUM[.W] and RRUM.A.....	316
6-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode .....	317
6-55. Swap Bytes SWPBX.A Register Mode.....	321
6-56. Swap Bytes SWPBX.A In Memory .....	321
6-57. Swap Bytes SWPBX[.W] Register Mode .....	322
6-58. Swap Bytes SWPBX[.W] In Memory .....	322
6-59. Sign Extend SXTX.A .....	323
6-60. Sign Extend SXTX[.W] .....	323
7-1. Flash Memory Module Block Diagram .....	342
7-2. 256-KB Flash Memory Segments Example .....	343
7-3. Erase Cycle Timing .....	346
7-4. Erase Cycle From Flash .....	347
7-5. Erase Cycle From RAM .....	348
7-6. Byte, Word, and Long-Word Write Timing .....	349
7-7. Initiating a Byte or Word Write From Flash.....	350
7-8. Initiating a Byte or Word Write From RAM .....	351
7-9. Initiating Long-Word Write From Flash.....	352
7-10. Initiating Long-Word Write from RAM .....	353
7-11. Block-Write Cycle Timing .....	354

[www.ti.com](http://www.ti.com)


---

7-12.	Block Write Flow .....	355
7-13.	User-Developed Programming Solution .....	359
7-14.	FCTL1 Register .....	361
7-15.	FCTL3 Register .....	362
7-16.	FCTL4 Register .....	363
7-17.	SFRIE1 Register .....	364
8-1.	Block Diagram of MID Implementation .....	366
8-2.	Overview of MSP430 Flash Memory Segmentation.....	367
8-3.	cw0 Parameter .....	369
8-4.	cw1 Parameter .....	369
9-1.	RCCTL0 Register .....	376
11-1.	DMA Controller Block Diagram.....	381
11-2.	DMA Addressing Modes .....	382
11-3.	DMA Single Transfer State Diagram .....	384
11-4.	DMA Block Transfer State Diagram .....	385
11-5.	DMA Burst-Block Transfer State Diagram .....	387
11-6.	DMACTL0 Register .....	395
11-7.	DMACTL1 Register .....	396
11-8.	DMACTL2 Register .....	397
11-9.	DMACTL3 Register .....	398
11-10.	DMACTL4 Register .....	399
11-11.	DMAxCTL Register .....	400
11-12.	DMAxSA Register.....	402
11-13.	DMAxDA Register.....	403
11-14.	DMAxSZ Register .....	404
11-15.	DMAIV Register .....	405
12-1.	P1IV Register.....	419
12-2.	P2IV Register.....	420
12-3.	P1IES Register .....	421
12-4.	P1IE Register.....	421
12-5.	P1IFG Register.....	421
12-6.	P2IES Register .....	422
12-7.	P2IE Register.....	422
12-8.	P2IFG Register.....	422
12-9.	PxIN Register.....	423
12-10.	PxOUT Register.....	423
12-11.	PxDIR Register .....	423
12-12.	PxREN Register .....	424
12-13.	PxD S Register.....	424
12-14.	PxSEL Register .....	424
13-1.	PMAPKEYID Register .....	430
13-2.	PMAPCTL Register .....	430
13-3.	PxMAPy Register .....	430
14-1.	LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result .....	432
14-2.	Implementation of CRC-CCITT Using the CRCDI and CRCINIRES Registers.....	434
14-3.	CRCDI Register .....	437
14-4.	CRCDIRB Register .....	437
14-5.	CRCINIRES Register .....	438
14-6.	CRCRESR Register .....	438

15-1.	AES Accelerator Block Diagram .....	440
15-2.	AES State Array Input and Output.....	441
15-3.	AES-128 Encryption Process.....	442
15-4.	AES-128 Decryption Process using AESOPx = 01 .....	443
15-5.	AES-128 Decryption Process using AESOPx = 10 and 11 .....	444
15-6.	AESACTL0 Register.....	447
15-7.	AESACTL1 Register.....	448
15-8.	AESASTAT Register.....	449
15-9.	AESAKEY Register .....	450
15-10.	AESADIN Register.....	451
15-11.	AESADOUT Register.....	451
15-12.	AESAXDIN Register .....	452
15-13.	AESAXIN Register .....	452
16-1.	Watchdog Timer Block Diagram .....	455
16-2.	WDTCTL Register .....	459
17-1.	Timer_A Block Diagram .....	462
17-2.	Up Mode .....	464
17-3.	Up Mode Flag Setting .....	464
17-4.	Continuous Mode .....	465
17-5.	Continuous Mode Flag Setting .....	465
17-6.	Continuous Mode Time Intervals .....	465
17-7.	Up/Down Mode .....	466
17-8.	Up/Down Mode Flag Setting.....	466
17-9.	Output Unit in Up/Down Mode .....	467
17-10.	Capture Signal (SCS = 1).....	468
17-11.	Capture Cycle .....	468
17-12.	Output Example – Timer in Up Mode .....	470
17-13.	Output Example – Timer in Continuous Mode .....	471
17-14.	Output Example – Timer in Up/Down Mode.....	472
17-15.	Capture/Compare TAxCCTL0 Interrupt Flag .....	473
17-16.	TAxCTL Register.....	476
17-17.	TAxR Register .....	477
17-18.	TAxCCTLn Register .....	478
17-19.	TAxCCRn Register .....	480
17-20.	TAxIV Register .....	480
17-21.	TAxEX0 Register.....	481
18-1.	Timer_B Block Diagram .....	484
18-2.	Up Mode .....	486
18-3.	Up Mode Flag Setting .....	486
18-4.	Continuous Mode .....	487
18-5.	Continuous Mode Flag Setting .....	487
18-6.	Continuous Mode Time Intervals .....	487
18-7.	Up/Down Mode .....	488
18-8.	Up/Down Mode Flag Setting.....	488
18-9.	Output Unit in Up/Down Mode .....	489
18-10.	Capture Signal (SCS = 1).....	490
18-11.	Capture Cycle .....	490
18-12.	Output Example – Timer in Up Mode .....	493
18-13.	Output Example – Timer in Continuous Mode .....	494

---

18-14. Output Example – Timer in Up/Down Mode.....	495
18-15. Capture/Compare TBxCCR0 Interrupt Flag .....	496
18-16. TBxCTL Register.....	499
18-17. TBxR Register .....	501
18-18. TBxCCTLn Register .....	502
18-19. TBxCCRn Register .....	504
18-20. TBxIV Register .....	505
18-21. TBxEX0 Register.....	506
19-1. Timer_D Block Diagram.....	509
19-2. High Resolution Clock Generator .....	512
19-3. Up Mode .....	514
19-4. Up Mode Flag Setting .....	515
19-5. Continuous Mode .....	515
19-6. Continuous Mode Flag Setting .....	515
19-7. Continuous Mode Time Intervals .....	516
19-8. TDxCCR0 PWM Generation Under Continuous Mode .....	516
19-9. Up/Down Mode .....	517
19-10. Up/Down Mode Flag Setting.....	517
19-11. Output Unit in Up/Down Mode .....	518
19-12. Controlling Rising and Falling Edge of PWM Output in Up Mode .....	520
19-13. Deadband Generation (TDxCMB = 1).....	521
19-14. Capture Signal (SCS = 1).....	522
19-15. Single Capture Cycle .....	522
19-16. Sequential Capture Events in Dual Capture Mode .....	523
19-17. COV in Dual Capture Mode .....	523
19-18. Output Example, Channel 1 – Timer in Up Mode .....	527
19-19. Output Example, Channel 1 - Timer in Up Mode With External Fault Signal .....	528
19-20. Output Example - Timer in Up Mode with External Timer Clear Signal .....	529
19-21. Output Example – Timer in Continuous Mode .....	530
19-22. Output Example – Timer in Up/Down Mode.....	531
19-23. Capture/Compare TDxCCR0 Interrupt Flag .....	532
19-24. TDxCTL0 Register .....	535
19-25. TDxCTL1 Register .....	537
19-26. TDxCTL2 Register .....	538
19-27. TDxR Register.....	539
19-28. TDxCCTLn Register .....	540
19-29. TDxCCRn Register .....	542
19-30. TDxCLn Register.....	542
19-31. TDxHCTL0 Register .....	543
19-32. TDxHCTL1 Register .....	544
19-33. TDxHINT Register .....	545
19-34. TDxIV Register .....	546
20-1. Timer Event Control Block Diagram .....	548
20-2. External Input Events Affect Timer_D Output .....	550
20-3. Timer_D Output With Channel Combination .....	550
20-4. Module Level Connection Between TEC and Timer_D .....	551
20-5. Synchronization Between Timer Instances .....	553
20-6. TECxCTL0 Register .....	556
20-7. TECxCTL1 Register .....	558

20-8.	TECxCTL2 Register .....	560
20-9.	TECxSTA Register.....	561
20-10.	TECxINT Register.....	562
20-11.	TECxIV Register .....	563
22-1.	RTC_A.....	567
22-2.	RTCCTL0 Register .....	576
22-3.	RTCCTL1 Register .....	577
22-4.	RTCCTL2 Register .....	578
22-5.	RTCCTL3 Register .....	578
22-6.	RTCNT1 Register .....	579
22-7.	RTCNT2 Register .....	579
22-8.	RTCNT3 Register .....	579
22-9.	RTCNT4 Register .....	579
22-10.	RTCSEC Register.....	580
22-11.	RTCSEC Register.....	580
22-12.	RTCMIN Register .....	581
22-13.	RTCMIN Register .....	581
22-14.	RTCHOUR Register .....	582
22-15.	RTCHOUR Register .....	582
22-16.	RTCDOW Register .....	583
22-17.	RTCDAY Register.....	583
22-18.	RTCDAY Register.....	583
22-19.	RTCMON Register.....	584
22-20.	RTCMON Register.....	584
22-21.	RTCYEARL Register .....	585
22-22.	RTCYEARL Register .....	585
22-23.	RTCYEARH Register.....	586
22-24.	RTCYEARH Register.....	586
22-25.	RTCAMIN Register .....	587
22-26.	RTCAMIN Register .....	587
22-27.	RTCAHOUR Register .....	588
22-28.	RTCAHOUR Register .....	588
22-29.	RTCADOW Register.....	589
22-30.	RTCADAY Register.....	589
22-31.	RTCADAY Register.....	589
22-32.	RTCPSC0CTL Register .....	590
22-33.	RTCPSC1CTL Register .....	591
22-34.	RT0PS Register .....	592
22-35.	RTPS1 Register .....	592
22-36.	RTCIV Register.....	592
23-1.	RTC_B Block Diagram .....	595
23-2.	RTCCTL0 Register .....	603
23-3.	RTCCTL1 Register .....	604
23-4.	RTCCTL2 Register .....	605
23-5.	RTCCTL3 Register .....	605
23-6.	RTCSEC Register.....	606
23-7.	RTCSEC Register.....	606
23-8.	RTCMIN Register .....	607
23-9.	RTCMIN Register .....	607

[www.ti.com](http://www.ti.com)

23-10. RTCHOUR Register .....	608
23-11. RTCHOUR Register .....	608
23-12. RTCDOW Register .....	609
23-13. RTCDAY Register.....	609
23-14. RTCDAY Register.....	609
23-15. RTCMON Register.....	610
23-16. RTCMON Register.....	610
23-17. RTCYEAR Register.....	611
23-18. RTCYEAR Register.....	611
23-19. RTCAMIN Register .....	612
23-20. RTCAMIN Register .....	612
23-21. RTCAHOUR Register .....	613
23-22. RTCAHOUR Register .....	613
23-23. RTCADOW Register.....	614
23-24. RTCADAY Register.....	615
23-25. RTCADAY Register.....	615
23-26. RTCPS0CTL Register.....	616
23-27. RTCPS1CTL Register.....	617
23-28. RTCPS0 Register .....	618
23-29. RTCPS1 Register .....	618
23-30. RTCIV Register.....	619
23-31. BIN2BCD Register .....	620
23-32. BCD2BIN Register .....	620
24-1. RTC_C Block Diagram .....	623
24-2. RTC_C Offset Error Calibration and Temperature Compensation Scheme .....	630
24-3. RTCCTL0_L Register .....	638
24-4. RTCCTL0_H Register .....	639
24-5. RTCCTL1 Register .....	640
24-6. RTCCTL3 Register .....	641
24-7. RTCOCAL Register.....	641
24-8. RTCTCMP Register .....	642
24-9. RTCNT1 Register .....	643
24-10. RTCNT2 Register .....	643
24-11. RTCNT3 Register .....	643
24-12. RTCNT4 Register .....	643
24-13. RTCSEC Register.....	644
24-14. RTCSEC Register.....	644
24-15. RTCMIN Register .....	645
24-16. RTCMIN Register .....	645
24-17. RTCHOUR Register .....	646
24-18. RTCHOUR Register .....	646
24-19. RTCDOW Register .....	647
24-20. RTCDAY Register.....	647
24-21. RTCDAY Register.....	647
24-22. RTCMON Register.....	648
24-23. RTCMON Register.....	648
24-24. RTCYEAR Register.....	649
24-25. RTCYEAR Register.....	649
24-26. RTCAMIN Register .....	650

24-27. RTCAMIN Register .....	650
24-28. RTCAHOUR Register .....	651
24-29. RTCAHOUR Register .....	651
24-30. RTCADOW Register.....	652
24-31. RTCADAY Register.....	652
24-32. RTCADAY Register.....	652
24-33. RTCPS0CTL Register.....	653
24-34. RTCPS1CTL Register.....	654
24-35. RTCPS0 Register .....	656
24-36. RTCPS1 Register .....	656
24-37. RTCIV Register.....	657
24-38. BIN2BCD Register .....	658
24-39. BCD2BIN Register .....	658
24-40. RTCSECBAKx Register.....	659
24-41. RTCSECBAKx Register.....	659
24-42. RTCMINBAKx Register .....	660
24-43. RTCMINBAKx Register .....	660
24-44. RTCHOURBAKx Register .....	661
24-45. RTCHOURBAKx Register .....	661
24-46. RTCDAYBAKx Register .....	662
24-47. RTCDAYBAKx Register .....	662
24-48. RTCMONBAKx Register .....	663
24-49. RTCMONBAKx Register .....	663
24-50. RTCYEARBAKx Register .....	664
24-51. RTCYEARBAKx Register .....	664
24-52. RTCTCCTL0 Register .....	665
24-53. RTCTCCTL1 Register .....	665
24-54. RTCCAPxCTL Register .....	666
25-1. MPY32 Block Diagram .....	669
25-2. Q15 Format Representation .....	674
25-3. Q14 Format Representation .....	674
25-4. Saturation Flow Chart .....	676
25-5. Multiplication Flow Chart .....	678
25-6. MPY32CTL0 Register .....	684
26-1. REF Block Diagram.....	687
26-2. REFCTL0 Register.....	693
27-1. ADC10_A Block Diagram .....	697
27-2. Analog Multiplexer .....	698
27-3. Extended Sample Mode.....	700
27-4. Pulse Sample Mode .....	700
27-5. Analog Input Equivalent Circuit .....	701
27-6. Single-Channel Single-Conversion Mode.....	702
27-7. Sequence-of-Channels Mode .....	703
27-8. Repeat-Single-Channel Mode .....	704
27-9. Repeat-Sequence-of-Channels Mode.....	705
27-10. Typical Temperature Sensor Transfer Function .....	707
27-11. ADC10_A Grounding and Noise Considerations .....	708
27-12. ADC10CTL0 Register .....	711
27-13. ADC10CTL1 Register .....	712

[www.ti.com](http://www.ti.com)


---

27-14. ADC10CTL2 Register .....	714
27-15. ADC10MEM0 Register .....	715
27-16. ADC10MEM0 Register .....	715
27-17. ADC10MCTL0 Register .....	716
27-18. ADC10HI Register .....	717
27-19. ADC10HI Register .....	717
27-20. ADC10LO Register .....	718
27-21. ADC10LO Register .....	718
27-22. ADC10IE Register .....	719
27-23. ADC10IFG Register .....	720
27-24. ADC10IV Register .....	721
28-1. ADC12_A Block Diagram (Devices With REF Module).....	724
28-2. ADC12_A MSP430F54xx (non-A) Block Diagram.....	725
28-3. Analog Multiplexer .....	726
28-4. Extended Sample Mode.....	728
28-5. Pulse Sample Mode .....	729
28-6. Analog Input Equivalent Circuit .....	729
28-7. Single-Channel Single-Conversion Mode.....	731
28-8. Sequence-of-Channels Mode .....	732
28-9. Repeat-Single-Channel Mode .....	733
28-10. Repeat-Sequence-of-Channels Mode.....	734
28-11. Typical Temperature Sensor Transfer Function .....	736
28-12. ADC12_A Grounding and Noise Considerations .....	737
28-13. ADC12CTL0 Register .....	742
28-14. ADC12CTL1 Register .....	744
28-15. ADC12CTL2 Register .....	745
28-16. ADC12MEMx Register .....	746
28-17. ADC12MCTLx Register .....	747
28-18. ADC12IE Register .....	748
28-19. ADC12IFG Register .....	750
28-20. ADC12IV Register .....	752
29-1. SD24_B Overview Block Diagram .....	755
29-2. SD24_B Reference- and Clock Generation Block Diagram .....	756
29-3. SD24_B Converter Block Diagram .....	757
29-4. Sigma-Delta Principle .....	758
29-5. Analog Input Equivalent Circuit .....	760
29-6. SINC <sup>3</sup> Filter Structure.....	761
29-7. Comb Filter's Frequency Response With OSR = 32 .....	761
29-8. Digital Filter Step Response and Conversion Points Digital Filter Output.....	762
29-9. SD24_B Output Encoder and Input Decoder Block Diagram .....	764
29-10. Single Conversion Examples .....	765
29-11. Grouped Operation - Internal Start-of-Conversion Trigger .....	766
29-12. Grouped Operation - External Start-of-Conversion Trigger .....	766
29-13. Conversion Delay Using Preload - Example .....	766
29-14. Start of Conversion Using Preload - Example .....	767
29-15. SD24_B Trigger Generator Block Diagram.....	768
29-16. SD24BCTL0 Register .....	772
29-17. SD24BCTL1 Register .....	774
29-18. SD24BTRGCTL Register .....	775

29-19.	SD24BIFG Register.....	776
29-20.	SD24BIE Register.....	779
29-21.	SD24BIV Register.....	781
29-22.	SD24BCCTLx Register.....	782
29-23.	SD24BINCTLx Register .....	784
29-24.	SD24BOSRx Register .....	785
29-25.	SD24BTRGOSR Register.....	785
29-26.	SD24BPREx Register .....	786
29-27.	SD24BTRGPRE Register.....	786
29-28.	SD24BMEMLx Register .....	787
29-29.	SD24BMEMHx Register .....	787
30-1.	DAC12_A Block Diagram for Two Module Devices .....	790
30-2.	DAC12_A Block Diagram For Single Module Devices .....	791
30-3.	Output Voltage vs DAC Data, 12-Bit, Straight Binary Mode .....	793
30-4.	Output Voltage vs DAC Data, 12-Bit, 2's complement Mode .....	793
30-5.	Negative Offset.....	794
30-6.	Positive Offset .....	794
30-7.	DAC12_A Group Update Example, Timer_A3 Trigger.....	795
30-8.	DAC12_xCTL0 Register .....	798
30-9.	DAC12_xCTL1 Register .....	800
30-10.	DAC12_xDAT Register.....	801
30-11.	DAC12_xDAT Register.....	801
30-12.	DAC12_xDAT Register.....	802
30-13.	DAC12_xDAT Register.....	802
30-14.	DAC12_xDAT Register.....	803
30-15.	DAC12_xDAT Register.....	803
30-16.	DAC12_xDAT Register.....	804
30-17.	DAC12_xDAT Register.....	804
30-18.	DAC12_xCALCTL Register.....	805
30-19.	DAC12_xCALDAT Register .....	805
30-20.	DAC12IV Register .....	806
31-1.	Comp_B Block Diagram.....	808
31-2.	Comp_B Sample-And-Hold.....	810
31-3.	RC-Filter Response at the Output of the Comparator.....	811
31-4.	Reference Generator Block Diagram.....	811
31-5.	Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer .....	812
31-6.	Temperature Measurement System .....	812
31-7.	Timing for Temperature Measurement Systems.....	813
31-8.	CBCTL0 Register .....	815
31-9.	CBCTL1 Register .....	816
31-10.	CBCTL2 Register .....	818
31-11.	CBCTL3 Register .....	819
31-12.	CBINT Register.....	821
31-13.	CBIV Register .....	822
32-1.	LCD_B Controller Block Diagram.....	825
32-2.	LCD Memory - Example for 160 Segments Maximum.....	826
32-3.	Bias Generation .....	829
32-4.	Example Static Waveforms .....	832
32-5.	Static LCD Example (MAB addresses need to be replaced with LCDMx) .....	833

[www.ti.com](http://www.ti.com)


---

32-6.	Example 2-Mux Waveforms .....	835
32-7.	2-Mux LCD Example (MAB addresses need to be replaced with LCDMx).....	836
32-8.	Example 3-Mux Waveforms .....	838
32-9.	3-Mux LCD Example (MAB addresses need to be replaced with LCDMx).....	839
32-10.	Example 4-Mux Waveforms .....	841
32-11.	4-Mux LCD Example (MAB addresses need to be replaced with LCDMx).....	842
32-12.	LcdbCtl0 Register.....	847
32-13.	LcdbCtl1 Register.....	848
32-14.	LcddbLkCtl Register.....	849
32-15.	LcdBmemCtl Register .....	850
32-16.	LcdBvCtl Register.....	851
32-17.	LcdBpCtl0 Register .....	853
32-18.	LcdBpCtl1 Register .....	853
32-19.	LcdBpCtl2 Register .....	854
32-20.	LcdBpCtl3 Register .....	854
32-21.	LcdBpCtl Register.....	855
32-22.	LcdBiv Register .....	856
33-1.	LCD Controller Block Diagram .....	859
33-2.	LCD Memory for Static and 2-Mux to 4-Mux Mode - Example for 160 Segments .....	860
33-3.	LCD Memory for 5-Mux to 8-Mux Mode - Example for 160 Segments .....	861
33-4.	Bias Generation .....	864
33-5.	Example Static Waveforms .....	869
33-6.	Example 2-Mux Waveforms .....	870
33-7.	Example 3-Mux Waveforms .....	871
33-8.	Example 4-Mux Waveforms .....	872
33-9.	Example 6-Mux Waveforms .....	873
33-10.	Example 8-Mux, 1/3 Bias Waveforms (LCDLP = 0) .....	874
33-11.	Example 8-Mux, 1/3 Bias Low-Power Waveforms (LCDLP = 1).....	875
33-12.	LcdCtl0 Register.....	881
33-13.	LcdCtl1 Register.....	883
33-14.	LcdCblkCtl Register .....	884
33-15.	LcdCmemCtl Register .....	885
33-16.	LcdCvCtl Register .....	886
33-17.	LcdCpCtl0 Register .....	888
33-18.	LcdCpCtl1 Register .....	889
33-19.	LcdCpCtl2 Register .....	890
33-20.	LcdCpCtl3 Register .....	891
33-21.	LcdCcpCtl Register .....	892
33-22.	LcdCiv Register .....	892
34-1.	USCI_Ax Block Diagram – UART Mode (UCSYNC = 0) .....	896
34-2.	Character Format .....	897
34-3.	Idle-Line Format.....	898
34-4.	Address-Bit Multiprocessor Format.....	899
34-5.	Auto Baud-Rate Detection – Break/Synch Sequence.....	900
34-6.	Auto Baud-Rate Detection – Synch Field.....	900
34-7.	UART vs IrDA Data Format.....	901
34-8.	Glitch Suppression, USCI Receive Not Started.....	903
34-9.	Glitch Suppression, USCI Activated.....	903
34-10.	BITCLK Baud-Rate Timing With UCOS16 = 0 .....	904

34-11. Receive Error .....	907
34-12. UCAxCTL0 Register .....	914
34-13. UCAxCTL1 Register .....	915
34-14. UCAxBR0 Register .....	916
34-15. UCAxBR1 Register .....	916
34-16. UCAxMCTL Register .....	916
34-17. UCAxSTAT Register.....	917
34-18. UCAxRXBUF Register .....	918
34-19. UCAxTXBUF Register.....	918
34-20. UCAxIRTCTL Register .....	919
34-21. UCAxIRRCTL Register.....	919
34-22. UCAxABCTL Register.....	920
34-23. UCAxIE Register .....	921
34-24. UCAxIFG Register .....	921
34-25. UCAxIV Register .....	922
35-1. USCI Block Diagram – SPI Mode .....	926
35-2. USCI Master and External Slave .....	928
35-3. USCI Slave and External Master .....	929
35-4. USCI SPI Timing With UCMSB = 1 .....	930
35-5. UCAxCTL0 Register .....	933
35-6. UCAxCTL1 Register .....	934
35-7. UCAxBR0 Register .....	935
35-8. UCAxBR1 Register .....	935
35-9. UCAxMCTL Register .....	935
35-10. UCAxSTAT Register.....	936
35-11. UCAxRXBUF Register .....	937
35-12. UCAxTXBUF Register.....	937
35-13. UCAxIE Register .....	938
35-14. UCAxIFG Register .....	938
35-15. UCAxIV Register .....	939
35-16. UCBxCTL0 Register .....	941
35-17. UCBxCTL1 Register .....	942
35-18. UCBxBR0 Register .....	943
35-19. UCBxBR1 Register .....	943
35-20. UCBxMCTL Register .....	943
35-21. UCBxSTAT Register.....	944
35-22. UCBxRXBUF Register .....	945
35-23. UCBxTXBUF Register.....	945
35-24. UCBxIE Register .....	946
35-25. UCBxIFG Register .....	946
35-26. UCBxIV Register .....	947
36-1. USCI Block Diagram – I <sup>2</sup> C Mode .....	951
36-2. I <sup>2</sup> C Bus Connection Diagram .....	952
36-3. I <sup>2</sup> C Module Data Transfer .....	953
36-4. Bit Transfer on I <sup>2</sup> C Bus .....	953
36-5. I <sup>2</sup> C Module 7-Bit Addressing Format .....	954
36-6. I <sup>2</sup> C Module 10-Bit Addressing Format.....	954
36-7. I <sup>2</sup> C Module Addressing Format With Repeated START Condition .....	954
36-8. I <sup>2</sup> C Time-Line Legend .....	955

---

36-9.	I <sup>2</sup> C Slave Transmitter Mode .....	956
36-10.	I <sup>2</sup> C Slave Receiver Mode .....	958
36-11.	I <sup>2</sup> C Slave 10-Bit Addressing Mode.....	959
36-12.	I <sup>2</sup> C Master Transmitter Mode .....	961
36-13.	I <sup>2</sup> C Master Receiver Mode .....	963
36-14.	I <sup>2</sup> C Master 10-Bit Addressing Mode .....	964
36-15.	Arbitration Procedure Between Two Master Transmitters .....	965
36-16.	Synchronization of Two I <sup>2</sup> C Clock Generators During Arbitration .....	966
36-17.	UCBxCTL0 Register .....	971
36-18.	UCBxCTL1 Register .....	972
36-19.	UCBxBR0 Register .....	973
36-20.	UCBxBR1 Register .....	973
36-21.	UCBxSTAT Register.....	974
36-22.	UCBxRXBUF Register .....	975
36-23.	UCBxTXBUF Register.....	975
36-24.	UCBxI2COA Register .....	976
36-25.	UCBxI2CSA Register.....	976
36-26.	UCBxIE Register .....	977
36-27.	UCBxIFG Register .....	978
36-28.	UCBxIV Register .....	979
37-1.	eUSCI_Ax Block Diagram – UART Mode (UCSYNC = 0).....	982
37-2.	Character Format .....	983
37-3.	Idle-Line Format.....	984
37-4.	Address-Bit Multiprocessor Format.....	985
37-5.	Auto Baud-Rate Detection – Break/Synch Sequence.....	986
37-6.	Auto Baud-Rate Detection – Synch Field.....	986
37-7.	UART vs IrDA Data Format.....	987
37-8.	Glitch Suppression, eUSCI_A Receive Not Started.....	989
37-9.	Glitch Suppression, eUSCI_A Activated .....	989
37-10.	BITCLK Baud-Rate Timing With UCOS16 = 0 .....	990
37-11.	Receive Error.....	994
37-12.	UCAxCTLW0 Register .....	999
37-13.	UCAxCTLW1 Register.....	1000
37-14.	UCAxBRW Register.....	1001
37-15.	UCAxMCTLW Register .....	1001
37-16.	UCAxSTATW Register.....	1002
37-17.	UCAxRXBUF Register .....	1003
37-18.	UCAxTXBUF Register .....	1003
37-19.	UCAxABCTL Register .....	1004
37-20.	UCAxIRCTL Register .....	1005
37-21.	UCAxIE Register.....	1006
37-22.	UCAxIFG Register.....	1007
37-23.	UCAxIV Register.....	1008
38-1.	eUSCI Block Diagram – SPI Mode.....	1011
38-2.	eUSCI Master and External Slave (UCSTEM = 0).....	1013
38-3.	eUSCI Slave and External Master .....	1014
38-4.	eUSCI SPI Timing With UCMSB = 1 .....	1016
38-5.	UCAxCTLW0 Register .....	1019
38-6.	UCAxBRW Register.....	1021

38-7.	UCAxSTATW Register.....	1022
38-8.	UCAxRXBUF Register .....	1023
38-9.	UCAxTXBUF Register .....	1024
38-10.	UCAxIE Register.....	1025
38-11.	UCAxIFG Register.....	1026
38-12.	UCAxIV Register.....	1027
38-13.	UCBxCTLW0 Register.....	1029
38-14.	UCBxBRW Register.....	1031
38-15.	UCBxSTATW Register.....	1031
38-16.	UCBxRXBUF Register .....	1032
38-17.	UCBxTXBUF Register .....	1032
38-18.	UCBxIE Register.....	1033
38-19.	UCBxIFG Register.....	1033
38-20.	UCBxIV Register.....	1034
39-1.	eUSCI_B Block Diagram – I <sup>2</sup> C Mode .....	1037
39-2.	I <sup>2</sup> C Bus Connection Diagram.....	1038
39-3.	I <sup>2</sup> C Module Data Transfer .....	1039
39-4.	Bit Transfer on I <sup>2</sup> C Bus .....	1039
39-5.	I <sup>2</sup> C Module 7-Bit Addressing Format.....	1039
39-6.	I <sup>2</sup> C Module 10-Bit Addressing Format .....	1040
39-7.	I <sup>2</sup> C Module Addressing Format With Repeated START Condition.....	1040
39-8.	I <sup>2</sup> C Time-Line Legend .....	1042
39-9.	I <sup>2</sup> C Slave Transmitter Mode .....	1043
39-10.	I <sup>2</sup> C Slave Receiver Mode .....	1044
39-11.	I <sup>2</sup> C Slave 10-Bit Addressing Mode .....	1045
39-12.	I <sup>2</sup> C Master Transmitter Mode .....	1047
39-13.	I <sup>2</sup> C Master Receiver Mode .....	1049
39-14.	I <sup>2</sup> C Master 10-Bit Addressing Mode.....	1050
39-15.	Arbitration Procedure Between Two Master Transmitters .....	1050
39-16.	Synchronization of Two I <sup>2</sup> C Clock Generators During Arbitration.....	1051
39-17.	UCBxCTLW0 Register.....	1058
39-18.	UCBxCTLW1 Register .....	1060
39-19.	UCBxBRW Register.....	1062
39-20.	UCBxSTATW Register.....	1062
39-21.	UCBxTBCNT Register .....	1063
39-22.	UCBxRXBUF Register .....	1064
39-23.	UCBxTXBUF Register .....	1064
39-24.	UCBxI2COA0 Register .....	1065
39-25.	UCBxI2COA1 Register .....	1066
39-26.	UCBxI2COA2 Register .....	1066
39-27.	UCBxI2COA3 Register .....	1067
39-28.	UCBxADDRX Register.....	1067
39-29.	UCBxADDMASK Register.....	1068
39-30.	UCBxI2CSA Register .....	1068
39-31.	UCBxIE Register.....	1069
39-32.	UCBxIFG Register.....	1071
39-33.	UCBxIV Register.....	1073
40-1.	USB Block Diagram .....	1076
40-2.	USB Power System .....	1078

[www.ti.com](http://www.ti.com)


---

40-3.	USB Power Up and Down Profile .....	1079
40-4.	Powering Entire MSP430 From VBUS .....	1080
40-5.	USB-PLL Analog Block Diagram .....	1081
40-6.	Data Buffers and Descriptors .....	1084
40-7.	USB Timer and Time Stamp Generation .....	1086
40-8.	USBKEYPID Register .....	1096
40-9.	USBCNF Register .....	1096
40-10.	USBPHYCTL Register .....	1097
40-11.	USBPWRCTL Register .....	1098
40-12.	USBPLLCTL Register .....	1100
40-13.	USBPLLDIVB Register .....	1101
40-14.	USBPLLIR Register .....	1102
40-15.	USBIEPCNF_0 Register .....	1104
40-16.	USBIEPBCNT_0 Register .....	1105
40-17.	USBOEPCNFG_0 Register .....	1106
40-18.	USBOEPBCNT_0 Register .....	1107
40-19.	USBIEPIE Register .....	1108
40-20.	USBOEPIE Register .....	1110
40-21.	USBIEPIFG Register .....	1112
40-22.	USBOEPIFG Register .....	1113
40-23.	USBVECINT Register .....	1114
40-24.	USBMAINT Register .....	1115
40-25.	USBTSREG Register .....	1116
40-26.	USBFN Register .....	1116
40-27.	USBCTL Register .....	1117
40-28.	USBIE Register .....	1118
40-29.	USBIFG Register .....	1119
40-30.	USBFUNADR Register .....	1119
40-31.	USBOEPCNF_n Register .....	1123
40-32.	USBOEPBBAX_n Register .....	1124
40-33.	USBOEPBCTX_n Register .....	1125
40-34.	USBOEPBBAY_n Register .....	1125
40-35.	USBOEPBCTY_n Register .....	1126
40-36.	USBOEPSIZXY_n Register .....	1126
40-37.	USBIEPCNF_n Register .....	1127
40-38.	USBIEPBBAX_n Register .....	1128
40-39.	USBIEPBCTX_n Register .....	1129
40-40.	USBIEPBBAY_n Register .....	1129
40-41.	USBIEPBCTY_n Register .....	1130
40-42.	USBOEPSIZXY_n Register .....	1130
41-1.	LDO Block Diagram .....	1132
41-2.	3.3-V LDO Power Up/Down Profile .....	1133
41-3.	Powering Entire MSP430 From LDOI .....	1134
41-4.	LDOKEYPID Register .....	1137
41-5.	PUCTL Register .....	1137
41-6.	LDOPWRCTL Register .....	1138
42-1.	Large Implementation of EEM .....	1141

## List of Tables

1-1.	Interrupt Sources, Flags, and Vectors .....	60
1-2.	Operation Modes .....	64
1-3.	Connection of Unused Pins .....	68
1-4.	Tag Values .....	75
1-5.	Peripheral Discovery Descriptor .....	76
1-6.	Values for Memory Entry .....	76
1-7.	Values for Peripheral Entry.....	77
1-8.	Peripheral IDs .....	77
1-9.	Sample Peripheral Discovery Descriptor .....	79
1-10.	SFR Base Address .....	83
1-11.	SFR Registers .....	83
1-12.	SFRIE1 Register Description .....	84
1-13.	SFRIFG1 Register Description .....	85
1-14.	SFRRPCR Register Description .....	87
1-15.	SYS Base Address .....	88
1-16.	SYS Registers .....	88
1-17.	SYSCCTL Register Description .....	89
1-18.	SYSBSLC Register Description .....	90
1-19.	SYSJMBC Register Description .....	91
1-20.	SYSJMBI0 Register Description .....	92
1-21.	SYSJMBI1 Register Description.....	92
1-22.	SYSJMBO0 Register Description .....	93
1-23.	SYSJMBO1 Register Description .....	93
1-24.	SYSUNIV Register Description .....	94
1-25.	SYSSNIV Register Description .....	95
1-26.	SYSRSTIV Register Description .....	96
1-27.	SYSBERRIV Register Description .....	97
2-1.	SVS and SVM Thresholds .....	102
2-2.	Recommended SVS <sub>L</sub> Settings.....	102
2-3.	Recommended SVS <sub>H</sub> Settings .....	102
2-4.	Available SVS <sub>H</sub> and SVM <sub>H</sub> Settings Versus V <sub>CORE</sub> Settings .....	103
2-5.	SVS <sub>L</sub> and SVM <sub>L</sub> Control Mode Selection .....	111
2-6.	SVS <sub>L</sub> Automatic Performance Control.....	111
2-7.	SVS <sub>L</sub> Manual Performance Modes .....	111
2-8.	SVM <sub>L</sub> Automatic Performance Control .....	111
2-9.	SVM <sub>L</sub> Manual Performance Modes .....	111
2-10.	SVS <sub>H</sub> and SVM <sub>H</sub> Control Mode Selection .....	112
2-11.	SVS <sub>H</sub> Automatic Performance Control .....	112
2-12.	SVS <sub>H</sub> Manual Performance Modes .....	112
2-13.	SVM <sub>H</sub> Automatic Performance Control .....	112
2-14.	SVM <sub>H</sub> Manual Performance Modes.....	112
2-15.	PMM Registers .....	114
2-16.	PMMCTL0 Register Description .....	115
2-17.	PMMCTL1 Register Description .....	116
2-18.	SVSMHCTL Register Description.....	117
2-19.	SVMLCTL Register Description .....	118
2-20.	SVSMIO Register Description.....	119

[www.ti.com](http://www.ti.com)


---

2-21.	PMMIFG Register Description.....	120
2-22.	PMMRIE Register Description.....	122
2-23.	PM5CTL0 Register Description .....	123
3-1.	Battery Backup Registers .....	128
3-2.	BAKCTL Register Description .....	129
3-3.	BAKCHCTL Register Description .....	130
4-1.	Next Supply Voltage Selection .....	135
4-2.	Minimum Voltage Thresholds for Selected $f_{SYS}$ .....	138
4-3.	Supply Selection During LPMx.5.....	140
4-4.	Auxiliary Supply Registers .....	147
4-5.	AUXCTL0 Register Description .....	148
4-6.	AUXCTL1 Register Description .....	149
4-7.	AUXCTL2 Register Description .....	150
4-8.	AUX2CHCTL Register Description .....	151
4-9.	AUX3CHCTL Register Description .....	152
4-10.	AUXADCCTL Register Description .....	153
4-11.	AUXIFG Register Description .....	154
4-12.	AUXIE Register Description .....	155
4-13.	AUXIV Register Description .....	156
5-1.	Clock Request System and Power Modes .....	167
5-2.	UCS Registers.....	172
5-3.	UCSCTL0 Register Description .....	173
5-4.	UCSCTL1 Register Description .....	174
5-5.	UCSCTL2 Register Description .....	175
5-6.	UCSCTL3 Register Description .....	176
5-7.	UCSCTL4 Register Description .....	177
5-8.	UCSCTL5 Register Description .....	178
5-9.	UCSCTL6 Register Description .....	180
5-10.	UCSCTL7 Register Description .....	182
5-11.	UCSCTL8 Register Description .....	183
5-12.	UCSCTL9 Register Description .....	184
6-1.	SR Bit Description .....	191
6-2.	Values of Constant Generators CG1, CG2.....	192
6-3.	Source and Destination Addressing .....	195
6-4.	MSP430 Double-Operand Instructions.....	213
6-5.	MSP430 Single-Operand Instructions .....	213
6-6.	Conditional Jump Instructions .....	214
6-7.	Emulated Instructions .....	214
6-8.	Interrupt, Return, and Reset Cycles and Length.....	215
6-9.	MSP430 Format II Instruction Cycles and Length .....	215
6-10.	MSP430 Format I Instructions Cycles and Length .....	216
6-11.	Description of the Extension Word Bits for Register Mode .....	217
6-12.	Description of Extension Word Bits for Non-Register Modes .....	218
6-13.	Extended Double-Operand Instructions.....	219
6-14.	Extended Single-Operand Instructions.....	221
6-15.	Extended Emulated Instructions .....	223
6-16.	Address Instructions, Operate on 20-Bit Register Data.....	224
6-17.	MSP430X Format II Instruction Cycles and Length .....	225
6-18.	MSP430X Format I Instruction Cycles and Length .....	226

6-19.	Address Instruction Cycles and Length .....	227
6-20.	Instruction Map of MSP430X .....	228
7-1.	Supported Simultaneous Code Execution and Flash Operations .....	345
7-2.	Erase Modes.....	345
7-3.	Write Modes .....	349
7-4.	Flash Access While Flash is Busy (BUSY = 1) .....	356
7-5.	FCTL Registers.....	360
7-6.	FCTL1 Register Description .....	361
7-7.	FCTL3 Register Description .....	362
7-8.	FCTL4 Register Description .....	363
7-9.	SFRIE1 Register Description .....	364
8-1.	Overview of MID Support Software Functions .....	368
9-1.	RAMCTL Registers .....	375
9-2.	RCCTL0 Register Description.....	376
10-1.	Backup RAM Registers .....	378
11-1.	DMA Transfer Modes.....	383
11-2.	DMA Trigger Operation .....	389
11-3.	Maximum Single-Transfer DMA Cycle Time .....	390
11-4.	DMA Registers .....	393
11-5.	DMACTL0 Register Description.....	395
11-6.	DMACTL1 Register Description.....	396
11-7.	DMACTL2 Register Description.....	397
11-8.	DMACTL3 Register Description.....	398
11-9.	DMACTL4 Register Description.....	399
11-10.	DMAxCTL Register Description .....	400
11-11.	DMAxSA Register Description .....	402
11-12.	DMAxDA Register Description .....	403
11-13.	DMAxSZ Register Description.....	404
11-14.	DMAIV Register Description .....	405
12-1.	I/O Configuration .....	408
12-2.	Digital I/O Registers .....	413
12-3.	P1IV Register Description .....	419
12-4.	P2IV Register Description .....	420
12-5.	P1IES Register Description.....	421
12-6.	P1IE Register Description .....	421
12-7.	P1IFG Register Description.....	421
12-8.	P2IES Register Description.....	422
12-9.	P2IE Register Description .....	422
12-10.	P2IFG Register Description.....	422
12-11.	PxIN Register Description .....	423
12-12.	PxOUT Register Description .....	423
12-13.	PxDIR Register Description.....	423
12-14.	PxREN Register Description .....	424
12-15.	PxDS Register Description .....	424
12-16.	PxSEL Register Description .....	424
13-1.	Examples for Port Mapping Mnemonics and Functions .....	427
13-2.	Port Mapping Control Registers.....	429
13-3.	Port Mapping Registers for Port Px – Byte Access .....	429
13-4.	Port Mapping Registers for Port Px – Word Access .....	429

---

13-5.	PMAPKEYID Register Description.....	430
13-6.	PMAPCTL Register Description.....	430
13-7.	PxMAPy Register Description .....	430
14-1.	CRC Registers .....	436
14-2.	CRCDI Register Description.....	437
14-3.	CRCDIRB Register Description .....	437
14-4.	CRCINIRES Register Description .....	438
14-5.	CRCRESR Register Description .....	438
15-1.	AES_ACCEL Registers .....	446
15-2.	AESACTL0 Register Description.....	447
15-3.	AESACTL1 Register Description.....	448
15-4.	AESASTAT Register Description .....	449
15-5.	AESAKEY Register Description.....	450
15-6.	AESADIN Register Description .....	451
15-7.	AESADOUT Register Description .....	451
15-8.	AESAXDIN Register Description.....	452
15-9.	AESAXIN Register Description.....	452
16-1.	WDT_A Registers .....	458
16-2.	WDTCTL Register Description .....	459
17-1.	Timer Modes.....	464
17-2.	Output Modes .....	469
17-3.	Timer_A Registers .....	475
17-4.	TAxCTL Register Description .....	476
17-5.	TAxR Register Description.....	477
17-6.	TAxCCTLn Register Description .....	478
17-7.	TAxCCRn Register Description .....	480
17-8.	TAxIV Register Description .....	480
17-9.	TAxEX0 Register Description .....	481
18-1.	Timer Modes.....	486
18-2.	TBxCCLn Load Events .....	491
18-3.	Compare Latch Operating Modes .....	492
18-4.	Output Modes .....	492
18-5.	Timer_B Registers .....	498
18-6.	TBxCTL Register Description .....	499
18-7.	TBxR Register Description.....	501
18-8.	TBxCCTLn Register Description .....	502
18-9.	TBxCCRn Register Description .....	504
18-10.	TBxIV Register Description .....	505
18-11.	TBxEX0 Register Description .....	506
19-1.	Factory Preprogrammed Frequency and TDHMx, TDHCLKCR Bit Settings.....	512
19-2.	Timer Modes.....	514
19-3.	High-Resolution Mode Limitation (TDHEN = 1) - Minimum Duty Cycle .....	519
19-4.	High-Resolution Mode Limitation (TDHEN = 1) - Maximum Duty Cycle.....	519
19-5.	TDCLx Load Events .....	524
19-6.	Compare Latch Operating Modes .....	524
19-7.	Output Modes .....	525
19-8.	Timer_D Registers .....	534
19-9.	TDxCTL0 Register Description .....	535
19-10.	TDxCTL1 Register Description.....	537

19-11. TDxCTL2 Register Description .....	538
19-12. TDxR Register Description .....	539
19-13. TDxCCTLn Register Description .....	540
19-14. TDxCCRn Register Description .....	542
19-15. TDxCLn Register Description .....	542
19-16. TDxHCTL0 Register Description .....	543
19-17. TDxHCTL1 Register Description .....	544
19-18. TDxHINT Register Description .....	545
19-19. TDxIV Register Description .....	546
20-1. TEC Registers .....	555
20-2. TECxCTL0 Register Description .....	556
20-3. TECxCTL1 Register Description .....	558
20-4. TECxCTL2 Register Description .....	560
20-5. TECxSTA Register Description .....	561
20-6. TECxINT Register Description .....	562
20-7. TECxIV Register Description .....	563
21-1. RTC Overview .....	564
22-1. RTC_A Registers .....	574
22-2. RTCCTL0 Register Description .....	576
22-3. RTCCTL1 Register Description .....	577
22-4. RTCCTL2 Register Description .....	578
22-5. RTCCTL3 Register Description .....	578
22-6. RTCNT1 Register Description .....	579
22-7. RTCNT2 Register Description .....	579
22-8. RTCNT3 Register Description .....	579
22-9. RTCNT4 Register Description .....	579
22-10. RTCSEC Register Description .....	580
22-11. RTCSEC Register Description .....	580
22-12. RTCMIN Register Description .....	581
22-13. RTCMIN Register Description .....	581
22-14. RTCHOUR Register Description .....	582
22-15. RTCHOUR Register Description .....	582
22-16. RTCDOW Register Description .....	583
22-17. RTCDAY Register Description .....	583
22-18. RTCDAY Register Description .....	583
22-19. RTCMON Register Description .....	584
22-20. RTCMON Register Description .....	584
22-21. RTCYEARL Register Description .....	585
22-22. RTCYEARL Register Description .....	585
22-23. RTCYEARH Register Description .....	586
22-24. RTCYEARH Register Description .....	586
22-25. RTCAMIN Register Description .....	587
22-26. RTCAMIN Register Description .....	587
22-27. RTCAHOUR Register Description .....	588
22-28. RTCAHOUR Register Description .....	588
22-29. RTCADOW Register Description .....	589
22-30. RTCADAY Register Description .....	589
22-31. RTCADAY Register Description .....	589
22-32. RTCPS0CTL Register Description .....	590

[www.ti.com](http://www.ti.com)

22-33. RTCPS1CTL Register Description.....	591
22-34. RT0PS Register Description.....	592
22-35. RT1PS Register Description.....	592
22-36. RTCIV Register Description .....	592
23-1. RTC_B Registers .....	601
23-2. RTCCTL0 Register Description .....	603
23-3. RTCCTL1 Register Description .....	604
23-4. RTCCTL2 Register Description .....	605
23-5. RTCCTL3 Register Description .....	605
23-6. RTCSEC Register Description .....	606
23-7. RTCSEC Register Description .....	606
23-8. RTCMIN Register Description .....	607
23-9. RTCMIN Register Description .....	607
23-10. RTCHOUR Register Description.....	608
23-11. RTCHOUR Register Description.....	608
23-12. RTCDOW Register Description .....	609
23-13. RTCDAY Register Description .....	609
23-14. RTCDAY Register Description .....	609
23-15. RTCMON Register Description .....	610
23-16. RTCMON Register Description .....	610
23-17. RTCYEAR Register Description .....	611
23-18. RTCYEAR Register Description .....	611
23-19. RTCAMIN Register Description .....	612
23-20. RTCAMIN Register Description .....	612
23-21. RTCAHOUR Register Description .....	613
23-22. RTCAHOUR Register Description .....	613
23-23. RTCADOW Register Description .....	614
23-24. RTCADAY Register Description .....	615
23-25. RTCADAY Register Description .....	615
23-26. RTCPS0CTL Register Description.....	616
23-27. RTCPS1CTL Register Description.....	617
23-28. RTCPS0 Register Description.....	618
23-29. RTCPS1 Register Description.....	618
23-30. RTCIV Register Description .....	619
23-31. BIN2BCD Register Description.....	620
23-32. BCD2BIN Register Description.....	620
24-1. RTCCAPx Pin Configuration .....	634
24-2. RTC_C Registers .....	635
24-3. RTC_C Event/Tamper Detection Registers .....	637
24-4. RTC_C Real-Time Clock Counter Mode Aliases .....	637
24-5. RTCCTL0_L Register Description .....	638
24-6. RTCCTL0_H Register Description .....	639
24-7. RTCCTL1 Register Description .....	640
24-8. RTCCTL3 Register Description .....	641
24-9. RTCOCAL Register Description .....	641
24-10. RTCTCMP Register Description .....	642
24-11. RTCNT1 Register Description.....	643
24-12. RTCNT2 Register Description.....	643
24-13. RTCNT3 Register Description.....	643

24-14. RTCNT4 Register Description.....	643
24-15. RTCSEC Register Description .....	644
24-16. RTCSEC Register Description .....	644
24-17. RTCMIN Register Description.....	645
24-18. RTCMIN Register Description .....	645
24-19. RTCHOUR Register Description.....	646
24-20. RTCHOUR Register Description.....	646
24-21. RTCDOW Register Description .....	647
24-22. RTCDAY Register Description .....	647
24-23. RTCDAY Register Description .....	647
24-24. RTCMON Register Description .....	648
24-25. RTCMON Register Description .....	648
24-26. RTCYEAR Register Description .....	649
24-27. RTCYEAR Register Description .....	649
24-28. RTCAMIN Register Description .....	650
24-29. RTCAMIN Register Description .....	650
24-30. RTCAHOUR Register Description .....	651
24-31. RTCAHOUR Register Description .....	651
24-32. RTCADOW Register Description .....	652
24-33. RTCADAY Register Description .....	652
24-34. RTCADAY Register Description .....	652
24-35. RTCPS0CTL Register Description.....	653
24-36. RTCPS1CTL Register Description.....	654
24-37. RTCPS0 Register Description.....	656
24-38. RTCPS1 Register Description.....	656
24-39. RTCIV Register Description .....	657
24-40. BIN2BCD Register Description.....	658
24-41. BCD2BIN Register Description.....	658
24-42. RTCSECBAXx Register Description .....	659
24-43. RTCSECBAXx Register Description .....	659
24-44. RTCMINBAKx Register Description .....	660
24-45. RTCMINBAKx Register Description .....	660
24-46. RTCHOURBAKx Register Description .....	661
24-47. RTCHOURBAKx Register Description .....	661
24-48. RTCDAYBAKx Register Description .....	662
24-49. RTCYEARBAKx Register Description .....	662
24-50. RTCMONBAKx Register Description.....	663
24-51. RTCMONBAKx Register Description.....	663
24-52. RTCYEARBAKx Register Description.....	664
24-53. RTCYEARBAKx Register Description.....	664
24-54. RTCTCCTL0 Register Description.....	665
24-55. RTCTCCTL1 Register Description.....	665
24-56. RTCCAPxCTL Register Description .....	666
25-1. Result Availability (MPYFRAC = 0, MPYSAT = 0) .....	670
25-2. OP1 Registers.....	671
25-3. OP2 Registers .....	671
25-4. SUMEXT and MPYC Contents.....	672
25-5. Result Availability in Fractional Mode (MPYFRAC = 1, MPYSAT = 0) .....	675
25-6. Result Availability in Saturation Mode (MPYSAT = 1) .....	676

[www.ti.com](http://www.ti.com)


---

25-7.	MPY32 Registers .....	682
25-8.	Alternative Registers.....	683
25-9.	MPY32CTL0 Register Description .....	684
26-1.	REF Control of Reference System (REFMSTR = 1) (Default) .....	689
26-2.	Control of Reference System (REFMSTR = 0, ADC12_A only) .....	690
26-3.	REF Registers .....	692
26-4.	REFCTL0 Register Description .....	693
27-1.	Conversion Mode Summary .....	701
27-2.	ADC10_A Registers .....	710
27-3.	ADC10CTL0 Register Description .....	711
27-4.	ADC10CTL1 Register Description .....	712
27-5.	ADC10CTL2 Register Description .....	714
27-6.	ADC10MEM0 Register Description .....	715
27-7.	ADC10MEM0 Register Description.....	715
27-8.	ADC10MCTL0 Register Description.....	716
27-9.	ADC10HI Register Description .....	717
27-10.	ADC10HI Register Description .....	717
27-11.	ADC10LO Register Description .....	718
27-12.	ADC10LO Register Description .....	718
27-13.	ADC10IE Register Description .....	719
27-14.	ADC10IFG Register Description .....	720
27-15.	ADC10IV Register Description .....	721
28-1.	ADC12_A Conversion Result Formats .....	730
28-2.	Conversion Mode Summary .....	730
28-3.	ADC12_A Registers .....	740
28-4.	ADC12CTL0 Register Description .....	742
28-5.	ADC12CTL1 Register Description .....	744
28-6.	ADC12CTL2 Register Description .....	745
28-7.	ADC12MEMx Register Description .....	746
28-8.	ADC12MCTLx Register Description .....	747
28-9.	ADC12IE Register Description .....	748
28-10.	ADC12IFG Register Description .....	750
28-11.	ADC12IV Register Description .....	752
29-1.	Offset Binary Left Aligned Mapping.....	763
29-2.	2s Complement Left Aligned Mapping .....	763
29-3.	Data Format Example for OSR = 256 .....	763
29-4.	Conversion Mode Summary .....	764
29-5.	SD24_B Registers .....	770
29-6.	SD24BCTL0 Register Description .....	772
29-7.	SD24BCTL1 Register Description .....	774
29-8.	SD24BTRGCTL Register Description .....	775
29-9.	SD24BIFG Register Description .....	776
29-10.	SD24BIE Register Description .....	779
29-11.	SD24BIV Register Description .....	781
29-12.	SD24BCCTLx Register Description .....	782
29-13.	SD24BINCTLx Register Description.....	784
29-14.	SD24BOSRx Register Description.....	785
29-15.	SD24BTRGOSR Register Description .....	785
29-16.	SD24BPREx Register Description .....	786

29-17. SD24BTRGPRE Register Description.....	786
29-18. SD24BMEMLx Register Description.....	787
29-19. SD24BMEMHx Register Description .....	787
30-1. DAC Full-Scale Range ( $V_{ref} = V_{e_{REF+}}$ or $V_{REF+}$ ) .....	792
30-2. DAC Output Selection .....	796
30-3. DAC12_A Registers .....	797
30-4. DAC12_xCTL0 Register Description .....	798
30-5. DAC12_xCTL1 Register Description .....	800
30-6. DAC12_xDAT Register Description .....	801
30-7. DAC12_xDAT Register Description .....	801
30-8. DAC12_xDAT Register Description .....	802
30-9. DAC12_xDAT Register Description .....	802
30-10. DAC12_xDAT Register Description .....	803
30-11. DAC12_xDAT Register Description .....	803
30-12. DAC12_xDAT Register Description .....	804
30-13. DAC12_xDAT Register Description .....	804
30-14. DAC12_xCALCTL Register Description .....	805
30-15. DAC12_xCALDAT Register Description .....	805
30-16. DAC12IV Register Description .....	806
31-1. Comp_B Registers.....	814
31-2. CBCTL0 Register Description .....	815
31-3. CBCTL1 Register Description .....	816
31-4. CBCTL2 Register Description .....	818
31-5. CBCTL3 Register Description .....	819
31-6. CBINT Register Description .....	821
31-7. CBIV Register Description .....	822
32-1. LCD Voltage and Biasing Characteristics .....	830
32-2. LCD_B Registers.....	844
32-3. LCD_B Memory Registers .....	845
32-4. LCD_B Blinking Memory Registers .....	846
32-5. LCDBCTL0 Register Description.....	847
32-6. LCDBCTL1 Register Description.....	848
32-7. LCDDBLKCTL Register Description .....	849
32-8. LCDMEMCTL Register Description.....	850
32-9. LCDBVCTL Register Description .....	851
32-10. LCDBPCTL0 Register Description .....	853
32-11. LCDBPCTL1 Register Description .....	853
32-12. LCDBPCTL2 Register Description .....	854
32-13. LCDBPCTL3 Register Description .....	854
32-14. LCDBCCTL Register Description .....	855
32-15. LCDBIV Register Description.....	856
33-1. Differences Between LCD_B and LCD_C .....	858
33-2. Bias Voltages and external Pins .....	865
33-3. LCD Voltage and Biasing Characteristics .....	866
33-4. LCD_C Control Registers .....	876
33-5. LCD_C Memory Registers for Static and 2-Mux to 4-Mux Modes .....	877
33-6. LCD Blinking Memory Registers for Static and 2-Mux to 4-Mux Modes.....	878
33-7. LCD Memory Registers for 5-Mux to 8-Mux.....	879
33-8. LCDCTL0 Register Description .....	881

[www.ti.com](http://www.ti.com)


---

33-9.	LCDCCTL1 Register Description .....	883
33-10.	LDCDBLKCTL Register Description .....	884
33-11.	LDCCMEMCTL Register Description.....	885
33-12.	LCDCVCTL Register Description .....	886
33-13.	LCDCPCTL0 Register Description.....	888
33-14.	LCDCPCTL1 Register Description.....	889
33-15.	LCDCPCTL2 Register Description.....	890
33-16.	LCDCPCTL3 Register Description.....	891
33-17.	LDCCCPCTL Register Description .....	892
33-18.	LCDCIV Register Description .....	892
34-1.	Receive Error Conditions .....	902
34-2.	BITCLK Modulation Pattern .....	904
34-3.	BITCLK16 Modulation Pattern .....	905
34-4.	Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 .....	908
34-5.	Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1 .....	910
34-6.	USCI_A UART Mode Registers.....	913
34-7.	UCAxCTL0 Register Description.....	914
34-8.	UCAxCTL1 Register Description.....	915
34-9.	UCAxBR0 Register Description .....	916
34-10.	UCAxBR1 Register Description .....	916
34-11.	UCAxMCTL Register Description .....	916
34-12.	UCAxSTAT Register Description .....	917
34-13.	UCAxRXBUF Register Description .....	918
34-14.	UCAxTXBUF Register Description .....	918
34-15.	UCAxIRTCTL Register Description .....	919
34-16.	UCAxIRRCTL Register Description .....	919
34-17.	UCAxABCTL Register Description.....	920
34-18.	UCAxIE Register Description.....	921
34-19.	UCAxIFG Register Description .....	921
34-20.	UCAxIV Register Description .....	922
35-1.	UCxSTE Operation .....	927
35-2.	USCI_A SPI Mode Registers .....	932
35-3.	UCAxCTL0 Register Description.....	933
35-4.	UCAxCTL1 Register Description.....	934
35-5.	UCAxBR0 Register Description .....	935
35-6.	UCAxBR1 Register Description .....	935
35-7.	UCAxMCTL Register Description .....	935
35-8.	UCAxSTAT Register Description .....	936
35-9.	UCAxRXBUF Register Description .....	937
35-10.	UCAxTXBUF Register Description .....	937
35-11.	UCAxIE Register Description.....	938
35-12.	UCAxIFG Register Description .....	938
35-13.	UCAxIV Register Description .....	939
35-14.	USCI_B SPI Mode Registers .....	940
35-15.	UCBxCTL0 Register Description.....	941
35-16.	UCBxCTL1 Register Description.....	942
35-17.	UCBxBR0 Register Description .....	943
35-18.	UCBxBR1 Register Description .....	943
35-19.	UCBxMCTL Register Description .....	943

35-20. UCBxSTAT Register Description .....	944
35-21. UCBxRXBUF Register Description .....	945
35-22. UCBxTXBUF Register Description .....	945
35-23. UCBxIE Register Description .....	946
35-24. UCBxIFG Register Description .....	946
35-25. UCBxIV Register Description .....	947
36-1. I <sup>2</sup> C State Change Interrupt Flags .....	968
36-2. USCI_B Registers .....	970
36-3. UCBxCTL0 Register Description .....	971
36-4. UCBxCTL1 Register Description .....	972
36-5. UCBxBR0 Register Description .....	973
36-6. UCBxBR1 Register Description .....	973
36-7. UCBxSTAT Register Description .....	974
36-8. UCBxRXBUF Register Description .....	975
36-9. UCBxTXBUF Register Description .....	975
36-10. UCBxI2COA Register Description .....	976
36-11. UCBxI2CSA Register Description .....	976
36-12. UCBxIE Register Description .....	977
36-13. UCBxIFG Register Description .....	978
36-14. UCBxIV Register Description .....	979
37-1. Receive Error Conditions .....	988
37-2. Modulation Pattern Examples .....	990
37-3. BITCLK16 Modulation Pattern .....	991
37-4. UCBRSx Settings for Fractional Portion of N = f <sub>BRCLK</sub> /Baudrate .....	992
37-5. Recommended Settings for Typical Crystals and Baudrates .....	995
37-6. UART State Change Interrupt Flags .....	997
37-7. eUSCI_A UART Registers .....	998
37-8. UCAxCTLW0 Register Description .....	999
37-9. UCAxCTLW1 Register Description .....	1000
37-10. UCAxBRW Register Description .....	1001
37-11. UCAxMCTLW Register Description .....	1001
37-12. UCAxSTATW Register Description .....	1002
37-13. UCAxRXBUF Register Description .....	1003
37-14. UCAxTXBUF Register Description .....	1003
37-15. UCAxABCTL Register Description .....	1004
37-16. UCAxIRCTL Register Description .....	1005
37-17. UCAxIE Register Description .....	1006
37-18. UCAxIFG Register Description .....	1007
37-19. UCAxIV Register Description .....	1008
38-1. UCxSTE Operation .....	1012
38-2. eUSCI_A SPI Registers .....	1018
38-3. UCAxCTLW0 Register Description .....	1019
38-4. UCAxBRW Register Description .....	1021
38-5. UCAxSTATW Register Description .....	1022
38-6. UCAxRXBUF Register Description .....	1023
38-7. UCAxTXBUF Register Description .....	1024
38-8. UCAxIE Register Description .....	1025
38-9. UCAxIFG Register Description .....	1026
38-10. UCAxIV Register Description .....	1027

---

38-11. eUSCI_B SPI Registers .....	1028
38-12. UCBxCTLW0 Register Description .....	1029
38-13. UCBxBRW Register Description .....	1031
38-14. UCBxSTATW Register Description .....	1031
38-15. UCBxRXBUF Register Description .....	1032
38-16. UCBxTXBUF Register Description .....	1032
38-17. UCBxIE Register Description .....	1033
38-18. UCBxIFG Register Description .....	1033
38-19. UCBxIV Register Description .....	1034
39-1. Glitch Filter Length Selection Bits .....	1051
39-2. I <sup>2</sup> C State Change Interrupt Flags .....	1055
39-3. eUSCI_B Registers .....	1057
39-4. UCBxCTLW0 Register Description .....	1058
39-5. UCBxCTLW1 Register Description .....	1060
39-6. UCBxBRW Register Description .....	1062
39-7. UCBxSTATW Register Description .....	1062
39-8. UCBxTBCNT Register Description .....	1063
39-9. UCBxRXBUF Register Description .....	1064
39-10. UCBxTXBUF Register Description .....	1064
39-11. UCBxI2COA0 Register Description .....	1065
39-12. UCBxI2COA1 Register Description .....	1066
39-13. UCBxI2COA2 Register Description .....	1066
39-14. UCBxI2COA3 Register Description .....	1067
39-15. UCBxADDRX Register Description .....	1067
39-16. UCBxADDMASK Register Description .....	1068
39-17. UCBxI2CSA Register Description .....	1068
39-18. UCBxIE Register Description .....	1069
39-19. UCBxIFG Register Description .....	1071
39-20. UCBxIV Register Description .....	1073
40-1. USB-PLL Pre-Scale Divider .....	1082
40-2. Register Settings to Generate 48 MHz Using Common Clock Input Frequencies .....	1082
40-3. USB Buffer Memory Map .....	1085
40-4. USB Interrupt Vector Generation .....	1087
40-5. USB Configuration Registers .....	1095
40-6. USBKEYPID Register Description .....	1096
40-7. USBCNF Register Description .....	1096
40-8. USBPHYCTL Register Description .....	1097
40-9. USBPWRCTL Register Description .....	1098
40-10. USBPLLCTL Register Description .....	1100
40-11. USBPLLDIVB Register Description .....	1101
40-12. USBPLLIR Register Description .....	1102
40-13. USB Control Registers .....	1103
40-14. USBIEPCNF_0 Register Description .....	1104
40-15. USBIEPBCNT_0 Register Description .....	1105
40-16. USBOEPCNFG_0 Register Description .....	1106
40-17. USBOEPBCNT_0 Register Description .....	1107
40-18. USBIEPIE Register Description .....	1108
40-19. USBOEPIE Register Description .....	1110
40-20. USBIEPIFG Register Description .....	1112

40-21.	USBOEPIFG Register Description .....	1113
40-22.	USBVECINT Register Description .....	1114
40-23.	USBMAINT Register Description .....	1115
40-24.	USBTSREG Register Description .....	1116
40-25.	USBFN Register Description.....	1116
40-26.	USBCTL Register Description .....	1117
40-27.	USBIE Register Description.....	1118
40-28.	USBIFG Register Description.....	1119
40-29.	USBFUNADR Register Description .....	1119
40-30.	USB Buffer Memory .....	1120
40-31.	USB Buffer Descriptor Registers .....	1120
40-32.	USBOEPCNF_n Register Description .....	1123
40-33.	USBOEPBBAX_n Register Description .....	1124
40-34.	USBOEPBCTX_n Register Description .....	1125
40-35.	USBOEPBBAY_n Register Description .....	1125
40-36.	USBOEPBCTY_n Register Description .....	1126
40-37.	USBOEPSIZXY_n Register Description .....	1126
40-38.	USBIEPCNF_n Register Description .....	1127
40-39.	USBIEPBBAX_n Register Description.....	1128
40-40.	USBIEPBCTX_n Register Description.....	1129
40-41.	USBIEPBBAY_n Register Description.....	1129
40-42.	USBIEPBCTY_n Register Description.....	1130
40-43.	USBIEPSIZXY_n Register Description .....	1130
41-1.	LDO-PWR Registers .....	1136
41-2.	LDOKEYPID Register Description .....	1137
41-3.	PUCTL Register Description .....	1137
41-4.	LDOPWRCTL Register Description.....	1138
42-1.	EEM Configurations .....	1143



## Read This First

### About This Manual

This manual describes the modules and peripherals of the MSP430x5xx and MSP430x6xx family of devices. Each description presents the module or peripheral in a general sense. Not all features and functions of all modules or peripherals may be present on all devices. In addition, modules or peripherals may differ in their exact implementation between device families, or may not be fully implemented on an individual device or device family.

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

### Related Documentation From Texas Instruments

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

### FCC Warning

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

### Notational Conventions

Program examples, are shown in a special typeface.

### Glossary

ACLK	Auxiliary Clock; see <a href="#">Section 5.1</a>
ADC	Analog-to-Digital Converter
BOR	Brown-Out Reset; see <a href="#">Section 1.2</a>
BSL	Bootstrap Loader; see <a href="http://www.ti.com/msp430">www.ti.com/msp430</a> for application reports
CPU	Central Processing Unit; see <a href="#">Section 6.1</a>
DAC	Digital-to-Analog Converter
DCO	Digitally Controlled Oscillator; see <a href="#">Section 5.2.6</a>
dst	Destination; see <a href="#">Section 6.5</a>
FLL	Frequency Locked Loop; see <a href="#">Section 5.2.7</a>
GIE Modes	General Interrupt Enable; see <a href="#">Section 1.3.3</a>
INT(N/2)	Integer portion of N/2
I/O	Input/Output; see <a href="#">Chapter 12</a>
ISR	Interrupt Service Routine
LSB	Least-Significant Bit
LSD	Least-Significant Digit

---

LPM	Low-Power Mode; see <a href="#">Section 1.4</a> ; also named PM for Power Mode
MAB	Memory Address Bus
MCLK	Master Clock; see <a href="#">Section 5.1</a>
Memory Data Bus	
MSB	Most-Significant Bit
MSD	Most-Significant Digit
NMI	(Non)-Maskable Interrupt; see <a href="#">Section 1.3.1</a> ; also split to UNMI and SNMI
PC	Program Counter; see <a href="#">Section 6.3.1</a>
PM	Power Mode; see <a href="#">Section 1.4</a>
POR	Power-On Reset; see <a href="#">Section 1.2</a>
PUC	Power-Up Clear; see <a href="#">Section 1.2</a>
RAM	Random Access Memory
SCG	System Clock Generator; see <a href="#">Section 6.3.3</a>
SFR	Special Function Register; <a href="#">Section 1.14</a>
SMCLK	Sub-System Master Clock; see <a href="#">Section 5.1</a>
SNMI	System NMI; see <a href="#">Section 1.3.1</a>
SP	Stack Pointer; see <a href="#">Section 6.3.2</a>
SR	Status Register; see <a href="#">Section 6.3.3</a>
src	Source; see <a href="#">Section 6.5</a>
TOS	Top of stack; see <a href="#">Section 6.3.2</a>
UNMI	User NMI; see <a href="#">Section 1.3.1</a>
WDT	Watchdog Timer; see <a href="#">Chapter 16</a>
z16	16-bit address space

---

## Register Bit Conventions

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

## Register Bit Accessibility and Initial Condition

---

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

---

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

---



---



---

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

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

Topic	Page
1.1 System Control Module (SYS) Introduction .....	54
1.2 System Reset and Initialization .....	54
1.3 Interrupts .....	56
1.4 Operating Modes .....	62
1.5 Principles for Low-Power Applications .....	68
1.6 Connection of Unused Pins .....	68
1.7 Reset Pin (RST/NMI) Configuration .....	69
1.8 Configuring JTAG pins .....	69
1.9 Boot Code .....	69
1.10 Bootstrap Loader (BSL) .....	69
1.11 Memory Map – Uses and Abilities .....	71
1.12 JTAG Mailbox (JMB) System .....	72
1.13 Device Descriptor Table .....	73
1.14 SFR Registers .....	83
1.15 SYS Registers .....	88

## 1.1 System Control Module (SYS) Introduction

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

## 1.2 System Reset and Initialization

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

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

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

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

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

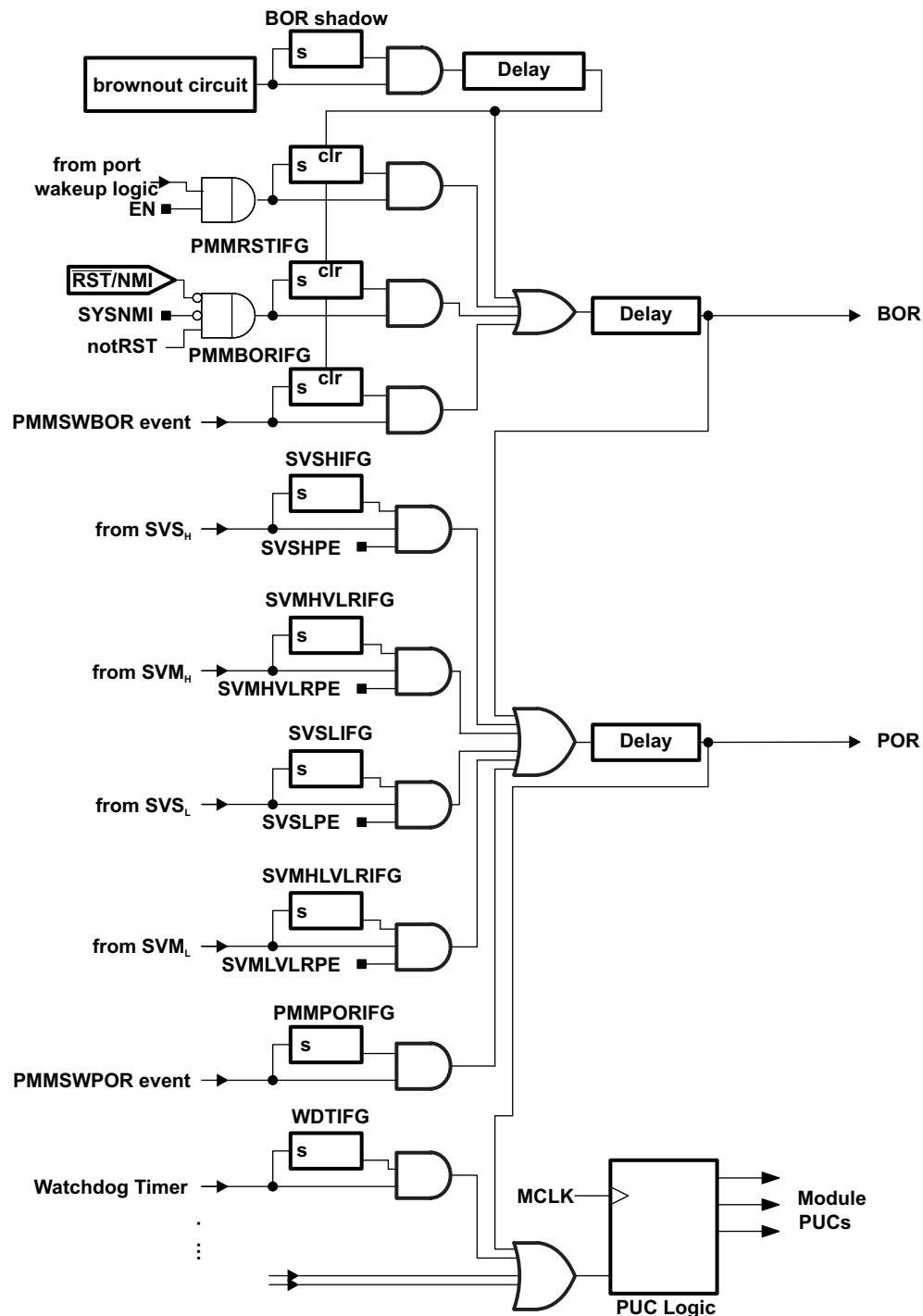
A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

- A POR signal
- Watchdog timer expiration when watchdog mode only (see the [WDT\\_A chapter](#) for details)
- Watchdog timer password violation (see the [WDT\\_A chapter](#) for details)
- A Flash memory password violation (see the [Flash Controller chapter](#) for details)
- Power Management Module password violation (see the [PMM chapter](#) for details)
- Fetch from peripheral area

---

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

---


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

### 1.2.1 Device Initial Conditions After System Reset

After a BOR, the initial device conditions are:

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

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

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

---

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

---



---

**NOTE:** Some SRAM locations can be modified by the boot code (refer to [Section 1.9](#)) after a BOR event. These SRAM locations, when available, are at SRAM locations 01CFAh through 01CFFh and 023FAh through 023FFh.

---

## 1.3 Interrupts

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

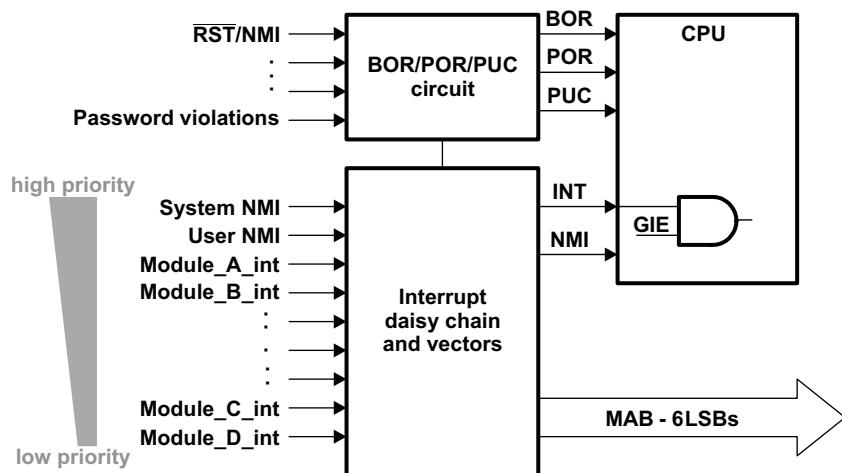
There are three types of interrupts:

- System reset
- (Non)maskable
- Maskable

---

**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. The block diagram for NMI sources is shown in [Figure 1-3](#).

A UNMI interrupt can be generated by following sources:

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

A SNMI interrupt can be generated by following sources:

- Power Management Module (PMM) SVM<sub>L</sub>/SVM<sub>H</sub> supply voltage fault
- PMM high/low side delay expiration
- Vacant memory access
- JTAG mailbox (JMB) event

---

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

---

### 1.3.2 SNMI Timing

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

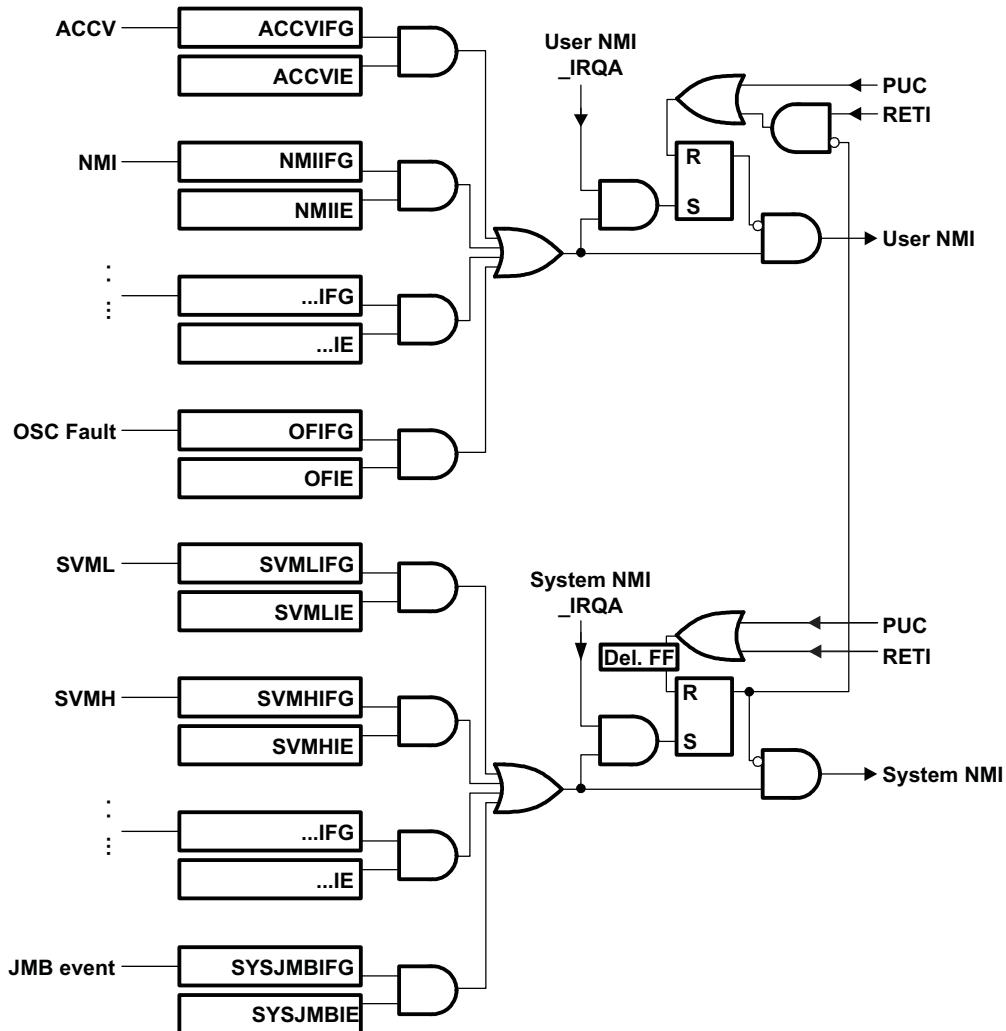


Figure 1-3. NMIs With Reentrance Protection

### 1.3.3 Maskable Interrupts

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

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

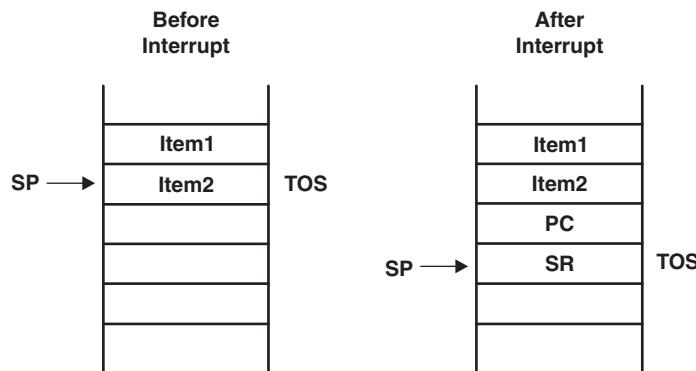
### 1.3.4 Interrupt Processing

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

#### 1.3.4.1 Interrupt Acceptance

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

1. Any currently executing instruction is completed.
2. The PC, which points to the next instruction, is pushed onto the stack.
3. The SR is pushed onto the stack.
4. The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
5. The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
6. 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 1-4. Interrupt Processing**

---

#### NOTE: Enable and Disable Interrupt

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

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

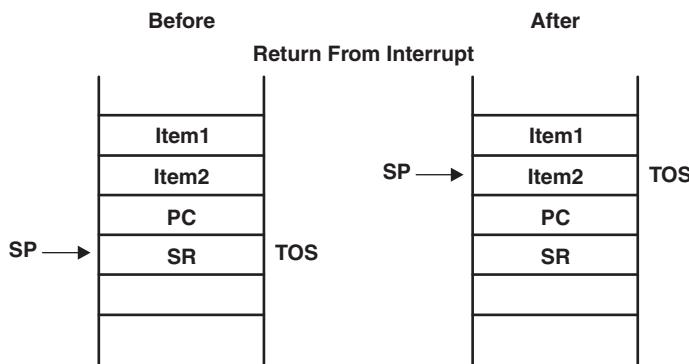
### 1.3.4.2 Return From Interrupt

The interrupt handling routine terminates with the instruction:

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

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

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



**Figure 1-5. Return From Interrupt**

### 1.3.5 Interrupt Nesting

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

### 1.3.6 Interrupt Vectors

The interrupt vectors are located in the address range 0FFFFh to OFF80h, for a maximum of 64 interrupt sources. A vector is programmed by the user and points to the start location of the corresponding interrupt service routine. [Table 1-1](#) is an example of the interrupt vectors available. See the device-specific data sheet for the complete interrupt vector list.

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

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

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

### 1.3.6.1 Alternate Interrupt Vectors

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

### 1.3.7 SYS Interrupt Vector Generators

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

### 1.3.7.1 SYSSNIV Software Example

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

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

### 1.3.7.2 SYSBERRIV Bus Error Interrupt Vector Generator

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

## 1.4 Operating Modes

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

The operating modes take into account three different needs:

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

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

When LPMx.5 (LPM3.5 or LPM4.5) is entered, the voltage regulator of the Power Management Module (PMM) is disabled. All RAM and register contents are lost. Although the I/O register contents are lost, the I/O pin states are locked upon LPMx.5 entry. See the [Digital I/O chapter](#) for further details. Wakeup from LPM4.5 is possible via a power sequence, a  $\overline{\text{RST}}$  event, or from specific I/O. Wakeup from LPM3.5 is possible via a power sequence, a  $\overline{\text{RST}}$  event, RTC event, or from specific I/O.

---

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

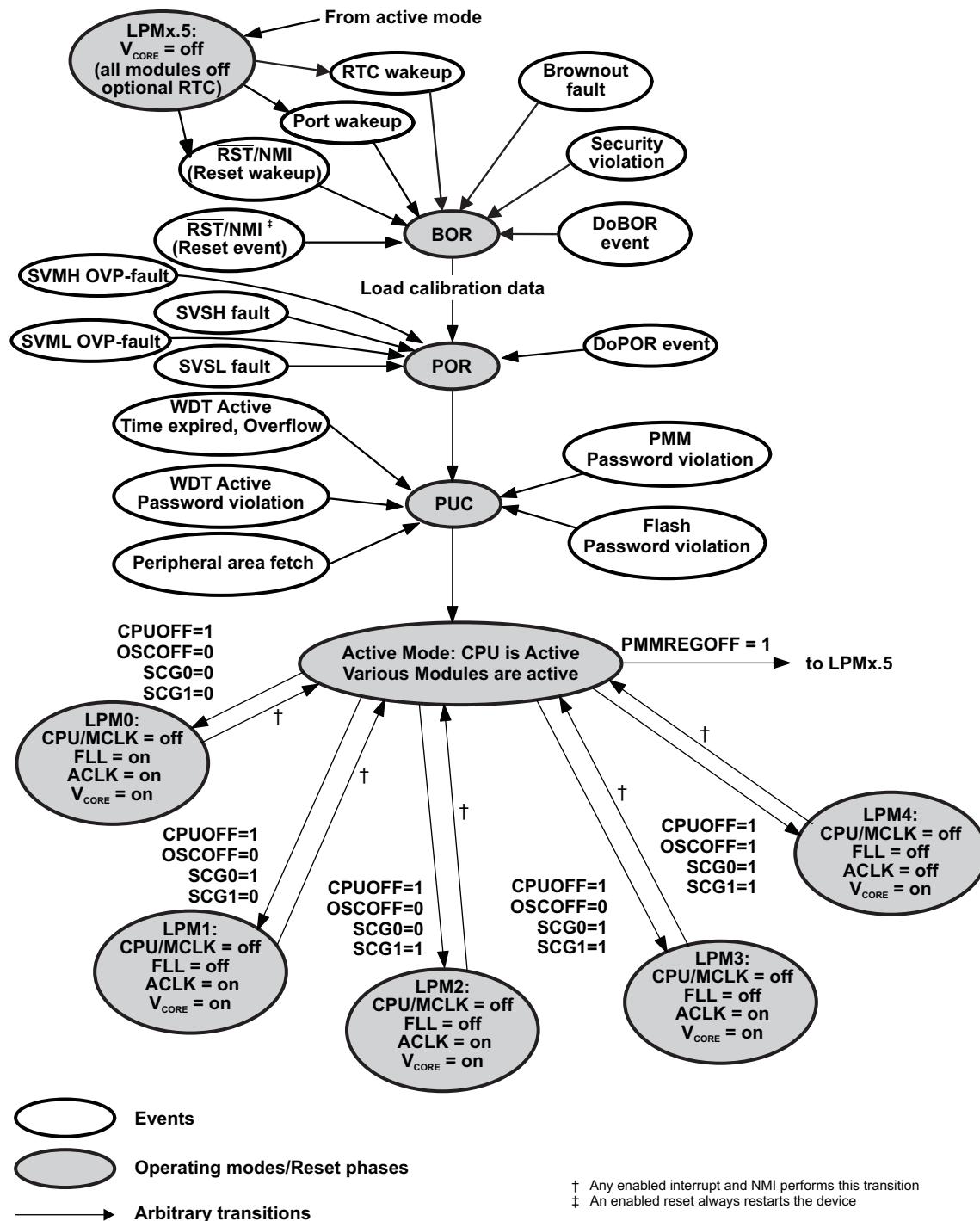
---



---

**NOTE:** The TEST/SBWTCK pin is used for interfacing to the development tools via Spy-Bi-Wire and JTAG. When the TEST/SBWTCK pin is high, wakeup times from LPM2, LPM3, and LPM4 may be different compared to when TEST/SBWTCK is low. Pay careful attention to the real-time behavior when exiting from LPM2, LPM3, and LPM4 with the device connected to a development tool (for example, MSP-FET430UIF). See the [PMM chapter](#) for details.

---


**Figure 1-6. Operation Modes**

## Operating Modes

www.ti.com

**Table 1-2. Operation Modes**

<b>SCG1 <sup>(1)</sup></b>	<b>SCG0</b>	<b>OSCOFF <sup>(1)</sup></b>	<b>CPUOFF <sup>(1)</sup></b>	<b>Mode</b>	<b>CPU and Clocks Status <sup>(2)</sup></b>
0	0	0	0	Active	CPU, MCLK are active. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK, MCLK, or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	0	0	1	LPM0	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	1	0	1	LPM1	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is disabled.
1	0	0	1	LPM2	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	0	1	LPM3	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	1	1	LPM4	CPU and all clocks are disabled.
1	1	1	1	LPM3.5 <sup>(3)</sup>	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, RTC operation is possible when configured properly. See the <i>RTC module</i> for further details.
1	1	1	1	LPM4.5 <sup>(3)</sup>	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, all clock sources are disabled; that is, no RTC operation is possible.

<sup>(1)</sup> This bit is automatically reset when exiting low power modes. Please refer to [Section 1.4.1](#) for details.<sup>(2)</sup> The low-power modes and, hence, the system clocks can be affected by the clock request system. See the [UCS chapter](#) for details.<sup>(3)</sup> LPM3.5 and LPM4.5 modes are not available on all devices. See the device-specific data sheet for availability.

### **1.4.1 Entering and Exiting Low-Power Modes LPM0 Through LPM4**

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

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

```
; Enter LPM0 Example
    BIS    #GIE+CPUOFF,SR           ; Enter LPM0
; ...
; Program stops here

; Exit LPM0 Interrupt Service Routine
    BIC    #CPUOFF,0(SP)          ; Exit LPM0 on RETI
    RETI

; Enter LPM3 Example
    BIS    #GIE+CPUOFF+SCG1+SCG0,SR      ; Enter LPM3
; ...
; Program stops here

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

; Enter LPM4 Example
    BIS    #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR      ; Enter LPM4
; ...
; Program stops here

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

### **1.4.2 Entering and Exiting Low-Power Modes LPMx.5**

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

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

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

The program flow for entering LPMx.5 is:

1. Configure I/O appropriately. See the [Digital I/O chapter](#) for complete details on configuring I/O for LPMx.5.
  - Set all ports to general purpose I/O. Configure each port to ensure no floating inputs based on the application requirements.
  - If wakeup from I/O is desired, configure input ports with interrupt capability appropriately.
2. If LPM3.5 is available, and desired, enable RTC operation. In addition, configure any RTC interrupts, if desired for LPM3.5 wakeup event. See the [RTC Overview chapter](#) for complete details.
3. Ensure clock system settings allow LPMx.5 entry according to [Table 5-1](#) in [UCS chapter](#).
4. Enter LPMx.5 by setting PMMREGOFF=1 and LPM4 status register bits. The following code example shows how to enter LPMx.5 mode. See the [PMM chapter](#) for further details.

```
; Enter LPMx.5 Example
MOV.B #PMMMPW_H, &PMMCTL0_H           ; Open PMM registers for write
BIS.B #PMMREGOFF, &PMMCTL0_L          ;
BIS    #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR ; Enter LPMx.5 when PMMREGOFF is set.
```

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

Although it is recommended to set GIE = 1 prior to entering LPMx.5, it is not required. Device wakeup from LPMx.5 with an enabled wakeup function will still cause the device to wake up from LPMx.5 even with GIE = 0. If GIE = 0 prior to LPMx.5, additional care may be required. Should the respective interrupt event should occur during LPMx.5 entry, the device may not recognize this or any future interrupt wakeup event on this function.

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

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

#### 1.4.3 Extended Time in Low-Power Modes

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

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

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

RETI

## 1.5 Principles for Low-Power Applications

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

- Use interrupts to wake the processor and control program flow.
- Peripherals should be switched on only when needed.
- Use low-power integrated peripheral modules in place of software driven functions. For example, Timer\_A and Timer\_B can automatically generate PWM and capture external timing with no CPU resources.
- Calculated branching and fast table look-ups should be used in place of flag polling and long software calculations.
- Avoid frequent subroutine and function calls due to overhead.
- For longer software routines, single-cycle CPU registers should be used.
- Overwrite RAM control register RCCTL0 with all not available and unused segments set to powered down (= 1). For information about used RAM segments see the device-specific data sheet.

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

## 1.6 Connection of Unused Pins

The correct termination of all unused pins is listed in [Table 1-3](#).

**Table 1-3. Connection of Unused Pins<sup>(1)</sup>**

Pin	Potential	Comment
AVCC	DV <sub>cc</sub>	
AVSS	DV <sub>ss</sub>	
Px.y	Open	Switched to port function, output direction (PxDIR.n = 1). Px.y represents port x and bit y of port x (for example, P1.0, P1.1, P2.2, PJ.0, PJ.1)
XIN	DV <sub>ss</sub>	For dedicated XIN pins only. XIN pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.
XOUT	Open	For dedicated XOUT pins only. XOUT pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.
XT2IN	DV <sub>ss</sub>	For dedicated XT2IN pins only. XT2IN pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.
XT2OUT	Open	For dedicated XT2OUT pins only. XT2OUT pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.
LCDCAP	DV <sub>ss</sub>	
RST/NMI	DV <sub>cc</sub> or V <sub>cc</sub>	47-kΩ pullup or internal pullup selected with 10-nF (2.2 nF) pulldown <sup>(2)</sup>
PJ.0/TDO	Open	The JTAG pins are shared with general purpose I/O function (PJ.x). If not being used, these should be switched to port function, output direction (PJDIR.n = 1). When used as JTAG pins, these pins should remain open.
PJ.1/TDI		
PJ.2/TMS		
PJ.3/TCK		
TEST	Open	This pin always has an internal pulldown enabled.
V18	Open	For USB devices only when USB module is not being used in the application
PUR <sup>(3)</sup>	DV <sub>ss</sub>	For USB devices only when USB module is not being used in the application
VUSB	Open	For USB devices only when USB module is not being used in the application
PU.0/DP	Open	For USB devices only when USB module is not being used in the application
PU.1/DM		
VBUS, VSSU	DV <sub>ss</sub>	For USB devices only when USB module is not being used in the application

<sup>(1)</sup> Any unused pin with a secondary function that is shared with general purpose I/O should follow the Px.y unused pin connection guidelines.

<sup>(2)</sup> The pulldown capacitor should not exceed 2.2 nF when using devices with Spy-Bi-Wire interface in Spy-Bi-Wire mode or in 4-wire JTAG mode with TI tools such as FET interfaces or GANG programmers.

<sup>(3)</sup> The default USB BSL evaluates the state of the PUR pin after a BOR reset. If it is pulled high externally, then the BSL is invoked. Therefore, unless invoking the BSL, it is important to keep PUR pulled low after a BOR reset, even if BSL or USB is never used. A 1-MΩ resistor to ground is recommended.

**Table 1-3. Connection of Unused Pins<sup>(1)</sup> (continued)**

Pin	Potential	Comment
LDOI	DV <sub>SS</sub>	For devices with LDO-PWR module when not being used in the application.
LDOO	Open	For devices with LDO-PWR module when not being used in the application.

## 1.7 Reset Pin (RST/NMI) Configuration

The reset pin can be configured as a reset function (default) or as an NMI function via the Special Function Register (SFR), SFRRPCR. Setting SYSNMI causes the 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 NMIIIE enables the interrupt of the external NMI. Upon an external NMI event, the NMIIIFG is set.

The RST/NMI pin can have either a pullup or pulldown present or not. SYSRSTUP selects either pullup or pulldown and SYRSRSTRE causes the pullup or pulldown to be enabled or not. If the RST/NMI pin is unused, it is required to have either the internal pullup selected and enabled or an external resistor connected to the RST/NMI pin as shown in [Table 1-3](#).

---

**NOTE:** All devices except the MSP430F543x (non-A devices) have the internal pullup enabled. In this case, no external pullup resistor is required.

---

## 1.8 Configuring JTAG pins

The JTAG pins are shared with general purpose I/O pins. There are several ways that the JTAG pins can be selected for four wire JTAG mode via software. Normally, upon a BOR, SYSJTAGPIN is cleared. With SYSJTAGPIN cleared, the JTAG are configured as general-purpose I/O. See the [Digital I/O chapter](#) for details on controlling the JTAG pins as general purpose I/O. If SYSJTAG = 1, the JTAG pins are configured to four wire JTAG mode and remain in this mode until another BOR condition occurs. Therefore, SYSJTAGPIN is a write only once function. Clearing it by software is not possible, and the device does not change from four wire JTAG mode to general purpose I/O.

## 1.9 Boot Code

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

## 1.10 Bootstrap Loader (BSL)

The BSL is software that is executed after start-up when a certain BSL entry condition is applied. The BSL enables the user to communicate with the embedded memory in the microcontroller during the prototyping phase, final production, and in service. All memory mapped resources, the programmable memory (flash memory), the data memory (RAM), and the peripherals, can be modified by the BSL as required. The user can define custom BSL code for flash-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 SYSBSSLIND 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 via 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 *MSP430 Programming Via the Bootstrap Loader User's Guide (SLAU319)* at [www.ti.com/msp430](http://www.ti.com/msp430).

The amount of BSL memory that is available is device specific. The BSL memory size is organized into segments and can be set using the SYSBSLSIZE bits. 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. Once set, SYSPMMPE can only be cleared by a BOR event.

## 1.11 Memory Map – Uses and Abilities

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

Can generate NMI on read/write/fetch							
Generates PUC on fetch access							
Protectable for read/write accesses							
Always able to access PMM registers from <sup>(1)</sup> ; Mass erase by user possible							
Mass erase by user possible							
Bank erase by user possible							
Segment erase by user possible							
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 <sup>(2)</sup>					x	
00FF4h-00FF7h	Start address of descriptor structure					x	
01000h-011FFh	BSL 0	x			x		
01200h-013FFh	BSL 1	x			x		
01400h-015FFh	BSL 2	x			x		
01600h-017FFh	BSL 3	x		x	x		
017FCh-017FFh	BSL Signature Location						
01800h-0187Fh	Info D	x					
01880h-018FFh	Info C	x					
01900h-0197Fh	Info B	x					
01980h-019FFh	Info A	x					
01A00h-01A7Fh	Device Descriptor Table					x	
01C00h-05BFFh	RAM 16 KB						
05B80-05BFFh	Alternate Interrupt Vectors						
05C00h-0FFFFh	Program	x	x <sup>(1)</sup>	x			
0FF80h-0FFFFh	Interrupt Vectors						
10000h-45BFFh	Program	x	x	x			
45C00h-FFFFFh	Vacant						x <sup>(3)</sup>

<sup>(1)</sup> Access rights are separately programmable for SYS and PMM.

<sup>(2)</sup> Fixed ID for all MSP430 devices. See [Section 1.13.1](#) for further details.

<sup>(3)</sup> On vacant memory space, the value 03FFFh is driven on the data bus.

### 1.11.1 Vacant Memory Space

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

### 1.11.2 JTAG Lock Mechanism via the Electronic Fuse

A device can be protected from unauthorized access by disabling the JTAG and SBW interface. This is achieved by programming the electronic fuse. Programming the electronic fuse, completely disables the debug and access capabilities associated with the JTAG and Spy-Bi-Wire interface. The JTAG is locked by programming a certain signature into the device's flash memory at dedicated addresses. The JTAG security lock key resides at the end of the bootstrap loader (BSL) memory at addresses 17FCh through 17FFh. Anything other than 0h or FFFFFFFFh programmed to these addresses locks the JTAG interface.

All of the 5xx MSP430 devices come with a preprogrammed BSL (TI-BSL) code that, by default, protects itself from unintended erase and write access. This is done by setting SYSBSLPE in the SYSBSLC register. Since the JTAG security lock key resides in the BSL memory address range, appropriate action must be taken to unprotect the BSL memory area before programming the protection key. For more details on the electronic fuse see the *MSP430 Programming Via the JTAG Interface User's Guide (SLAU320)* at [www.ti.com/msp430](http://www.ti.com/msp430).

Some JTAG commands are still possible after the device is secured, including the BYPASS command (see IEEE1149-2001 Standard) and the JMB\_EXCHANGE command which allows access to the JTAG Mailbox System (see [Section 1.12](#) for details).

---

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

---

## 1.12 JTAG Mailbox (JMB) System

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

Applications of the JMB are:

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

### 1.12.1 JMB Configuration

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

### 1.12.2 JMBOUT0 and JMBOUT1 Outgoing Mailbox

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

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

### 1.12.3 JMBIN0 and JMBIN1 Incoming Mailbox

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

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

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

#### 1.12.4 JMB NMI Usage

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

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

### 1.13 Device Descriptor Table

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

## Device Descriptor Table

www.ti.com

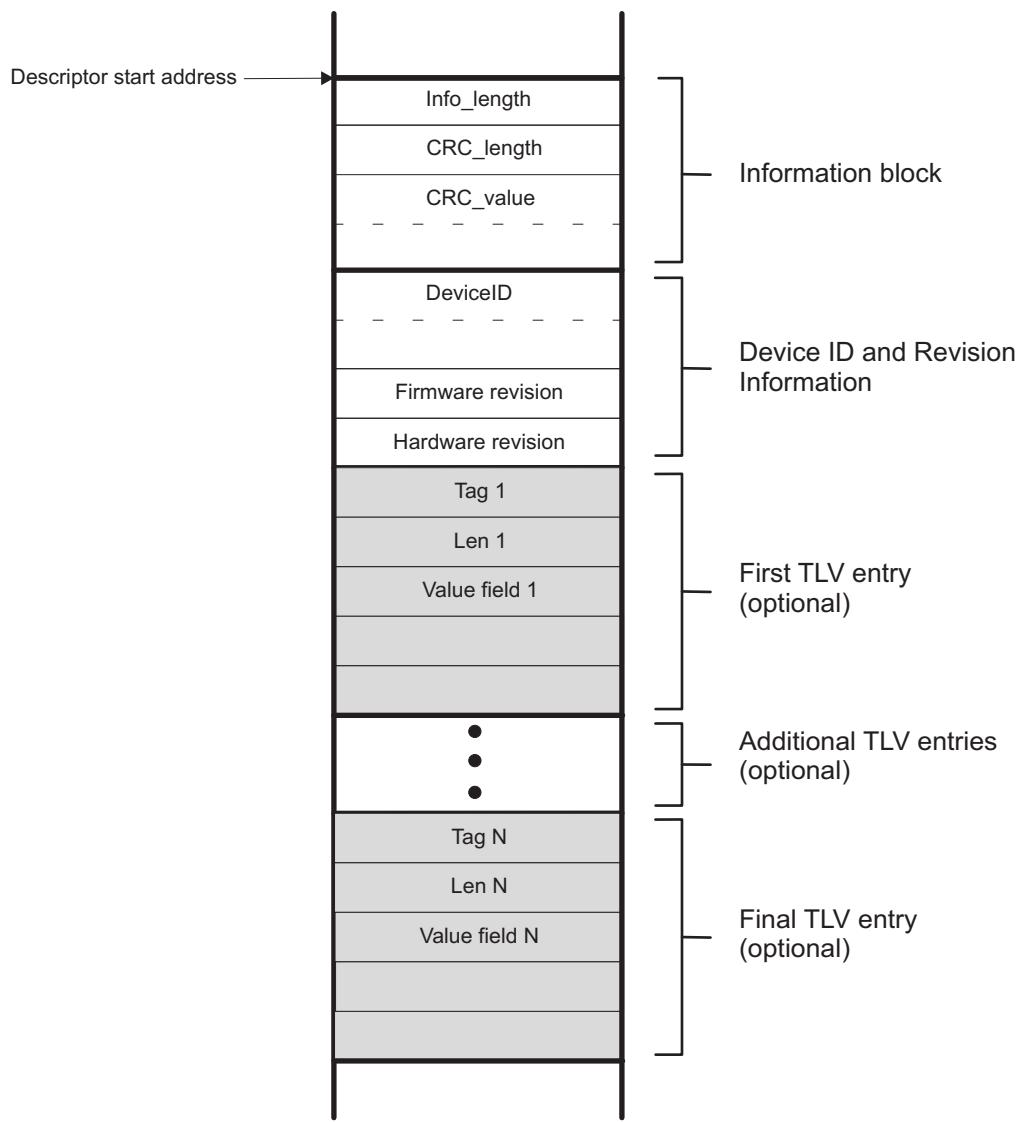


Figure 1-7. Devices Descriptor Table

**1.13.1 Identifying Device Type**

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

The length of the descriptors represented by Info\_length is computed as follows:

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

For example, if Info\_length = 5, then the length of the descriptors equals 128 bytes.

### 1.13.2 TLV Descriptors

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

The TLV descriptors are unique to their respective TLV block and are always followed by the descriptor block length.

Each TLV descriptor contains a tag field which identifies the descriptor type. [Table 1-4](#) shows the currently supported tags.

**Table 1-4. Tag Values**

Short Name	Value	Description
LDTAG	01h	Legacy descriptor (1xx, 2xx, 4xx families)
PDTAG	02h	Peripheral discovery descriptor
Reserved	03h	Future usage
Reserved	04h	Future usage
BLANK	05h	Blank descriptor
Reserved	06h	Future usage
ADCCAL	11h	ADC calibration
REFCAL	12h	REF calibration
Reserved	13h - FDh	Future usage
TAGEEXT	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 (TAGEEXT), the next byte extends the tag values, and the following two bytes represent the length of the descriptor in bytes. In this way, a user can search through the TLV descriptor table for a particular tag value, using a routine similar to below written in pseudo code:

```

// 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 Peripheral Discovery Descriptor

This descriptor type can describe concatenated or distributed memory or peripheral mappings, as well as, the number of interrupt vectors and their order. The peripheral discovery descriptor has tag value 02h (PDTAG). [Table 1-5](#) shows the structure of the peripheral discovery descriptor.

**NOTE:** Peripheral Discovery Descriptor is not available in every device. See the Device Descriptors section in the device-specific data sheet for the availability and details on Peripheral Discovery Descriptor.

**Table 1-5. Peripheral Discovery Descriptor**

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

The structures for a memory entry and peripheral entry are shown below. A memory entry consists of two bytes (one word). [Table 1-6](#) shows the individual bit fields of a memory entry word and their respective meanings. Similarly, a peripheral entry consists of two bytes (one word). [Table 1-7](#) shows the individual bit fields of a peripheral entry word and their respective meanings.

**Table 1-6. Values for Memory Entry**

Bit fields				
[15:13]	[12:9]	[8]	[7]	[6:0]
Memory type	Size	More	Unit Size	Address value
000: None	0000: 0 B	0: End Entry	0: 0200h	0000000
001: RAM	0001: 128 B	1: More Entries	1: 010000h	0000001

**Table 1-6. Values for Memory Entry (continued)**

Bit fields				
[15:13]	[12:9]	[8]	[7]	[6:0]
Memory type	Size	More	Unit Size	Address value
010: EEPROM	0010: 256 B			0000010
011: Reserved	0011: 512 B			0000011
100: FLASH	0100: 1 KB			0000100
101: ROM	0101: 2KB			0000101
110: MemType appended	0110: 4 KB			0000110
111: Undefined	0111: 8 KB			0000111
	1000: 16 KB			0001000
	1001: 32 KB			0001001
	1010: 64 KB			0001010
	1011: 128 KB			0001011
	1100: 256 KB			0001100
	1101: 512 KB			...
	1110: Size appended			...
	1111: Undefined			1111111

**Table 1-7. Values for Peripheral Entry**

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

<sup>(1)</sup> The Peripheral IDs are listed in [Table 1-8](#). This is not a complete list, but shown as an example.

**Table 1-8. Peripheral IDs<sup>(1)</sup>**

Peripheral or Module	PID
No Module	00h
WDT	01h
SFR	02h
UCS	03h
SYS	04h
PMM	05h
Flash Controller	08h
CRC16	09h
Port 1, 2	51h

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

**Table 1-8. Peripheral IDs<sup>(1)</sup> (continued)**

Peripheral or Module	PID
Port 3, 4	52h
Port 5, 6	53h
Port 7, 8	54h
Port 9, 10	55h
Port J	5Fh
Timer A0	81h
Timer A1	82h
Special info appended	FEh
Undefined module	FFh

**Table 1-9** shows a simple example for a peripheral discovery descriptor of a hypothetical device:

**Table 1-9. Sample Peripheral Discovery Descriptor**

Hex	Binary	Entry type	Description
030h, 0Eh	001_1000_0_0001110	memory	RAM 16 KB; Start address = 01C00h (0Eh * 0200h) <sup>(1)</sup>
09Bh, 02Eh	100_1011_0_0_0101110	memory	FLASH 128 KB Start address = 05C00h (2Eh * 0200h)
00h	0000_0000_0000_0000	delimiter	No more memory entries
0Fh	0000_1111	peripheral count	Peripheral count = 15
02h, 10h	00000010_0_0010000	peripheral	SFR at address = 0100h (10h * 10h)
01h, 01h	00000001_0_0000001	peripheral	WDT at address = 0110h (0100h + 10h)
05h, 01h	00000101_0_0000001	peripheral	PMM at address = 0120h (0110h + 10h)
03h, 01h	00000011_0_0000001	peripheral	UCS at address = 0130h (0120h + 10h)
08h, 01h	00001000_0_0000001	peripheral	FLCTL at address = 0140h (0130h + 10h)
09h, 01h	00001001_0_0000001	peripheral	CRC16 at address = 0150h (0140h + 10h)
04h, 01h	00000100_0_0000001	peripheral	SYS at address = 0160h (0150h + 10h)
51h, 0Ah	01010001_0_0001010	peripheral	Port 1, 2 at address = 0200h (0160h + 10h * 10h)
52h, 02h	01010010_0_0000010	peripheral	Port 3, 4 at address = 0220h (0200h + 02h * 10h)
53h, 02h	01010011_0_0000010	peripheral	Port 5, 6 at address = 0240h (0220h + 02h * 10h)
54h, 02h	01010100_0_0000010	peripheral	Port 7, 8 at address = 0260h (0240h + 02h * 10h)
55h, 02h	01010101_0_0000010	peripheral	Port 9, 10 at address = 0280h (0260h + 02h * 10h)
5Fh, 0Ah	01011111_0_0001010	peripheral	Port J at address = 0320h (0280h + 0Ah * 10h)
81h, 02h	10000001_0_0000010	peripheral	Timer A0 at address = 0340h (0320h + 02h * 10h)
82h, 04h	10000010_0_0000100	peripheral	Timer A1 at address = 0380h (0340h + 04h * 10h)
–			No appended entries
			SYSRSTIV @ 0FFEh (implied)
			SYSSNIV @ 0FFFC (implied)
			SYSUNIV @ 0FFFA (implied)
81h	1000_0001	interrupt	TA0 CCR0 @ 0FFF8
81h	1000_0001	interrupt	TA0 CCR1, CCR1, TA0IFG @ 0FFF6
51h	0101_0001	interrupt	Port 1 @ 0FFF4
82h	1000_0010	interrupt	TA1CCR0 @ 0FFF2
51h	0101_0001	interrupt	Port 2 @ 0FFF0
81h	1000_0010	interrupt	TA1CCR1, CCR1, TA1IFG @ 0FFEE
00h	0000_0000	delimiter	No more interrupt entries

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

**NOTE:** The interrupt ordering has some implied rules:

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

### 1.13.4 CRC Computation

The CRC checksum for the TLV structure is stored at memory locations 0x1A02 and 0x1A03. The least significant byte (LSB) and most significant byte (MSB) reside at memory locations 0x1A02 and 0x1A03, respectively. The checksum is computed using data stored at memory locations 0x1A04 through 0x1AFF. The CRC checksum can be easily computed using the CRC16 module. The following simplified C code utilizes the CRC16 module to compute the checksum. Please see the CRC16 chapter for further details on the CRC algorithm implementation.

---

**NOTE:** The CRC module on the MSP430F543x and MSP430F541x non-A versions does not support the bit-wise reverse feature used in this code example. Registers CRCDIRB and CRCRESR, along with their respective functionality, are not available.

---

```

unsigned int i;
unsigned char CRCRESULT_LSB, CRCRESULT_MSB;

WDTCTL = WDTPW + WDTHOLD;
CRCINIRES = 0xFFFF;
for (i = 0x01A04; i <= 0x01AFF; i++){
    CRCDIRB_L = *(unsigned char*)(i);
}

CRCRESULT_LSB = CRCINIRES_L; // value stored @ 0x1A02
CRCRESULT_MSB = CRCINIRES_H; // value stored @ 0x1A03

```

### 1.13.5 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.

#### 1.13.5.1 REF Calibration

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

$$\begin{aligned} \text{CAL\_ADC\_15VREF\_FACTOR} &= \frac{V_{\text{REF+}}}{1.5V} \times 2^{15} \\ \text{CAL\_ADC\_20VREF\_FACTOR} &= \frac{V_{\text{REF+}}}{2.0V} \times 2^{15} \\ \text{CAL\_ADC\_25VREF\_FACTOR} &= \frac{V_{\text{REF+}}}{2.5V} \times 2^{15} \end{aligned} \quad (2)$$

In this way, a conversion result is corrected by multiplying it with the CAL\_15VREF\_FACTOR (or CAL\_20VREF\_FACTOR, CAL\_25VREF\_FACTOR) and dividing the result by  $2^{15}$  as shown for each of the respective reference voltages:

$$\begin{aligned} \text{ADC}(\text{corrected}) &= \text{ADC}(\text{raw}) \times \text{CAL\_ADC15VREF\_FACTOR} \times \frac{1}{2^{15}} \\ \text{ADC}(\text{corrected}) &= \text{ADC}(\text{raw}) \times \text{CAL\_ADC20VREF\_FACTOR} \times \frac{1}{2^{15}} \\ \text{ADC}(\text{corrected}) &= \text{ADC}(\text{raw}) \times \text{CAL\_ADC25VREF\_FACTOR} \times \frac{1}{2^{15}} \end{aligned} \quad (3)$$

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 (CAL\_15VREF\_FACTOR) : 0x7BBB

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

- Multiply the conversion result by 2 (this step simplifies the final division):  $0x0100 \times 0x0002 = 0x0200$
- Multiply the result by CAL\_15VREF\_FACTOR:  $0x200 \times 0x7FEE = 0x00F7_7600$
- Divide the result by  $2^{16}$ :  $0x00F7_7600 / 0x0001_0000 = 0x0000_00F7 = 247$  decimal

#### 1.13.5.2 ADC Offset and Gain Calibration

The offset of the ADC is determined and stored as a two's-complement number in the TLV structure. The offset error correction is done by adding the CAL\_ADC\_OFFSET to the conversion result.

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

The gain of the ADC12 is calculated by [Equation 5](#):

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

The conversion result is gain corrected by multiplying it with the CAL\_ADC\_GAIN\_FACTOR and dividing the result by  $2^{15}$ :

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

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

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

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

### 1.13.5.3 Temperature Sensor Calibration

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

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

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

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

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

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

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

## 1.14 SFR Registers

The SFRs are listed in [Table 1-11](#). The base address for the SFRs is listed in [Table 1-10](#). Many of the bits inside the SFRs are described in other chapters throughout this user's guide. These bits are marked with a note and a reference. See the specific chapter of the respective module for details.

---

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

---

**Table 1-10. SFR Base Address**

Module	Base Address
SFR	00100h

**Table 1-11. SFR Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SFRIE1	Interrupt Enable	Read/write	Word	0000h	<a href="#">Section 7.4.4</a>
00h	SFRIE1_L (IE1)		Read/write	Byte	00h	
01h	SFRIE1_H (IE2)		Read/write	Byte	00h	
02h	SFRIFG1	Interrupt Flag	Read/write	Word	0082h	<a href="#">Section 1.14.2</a>
02h	SFRIFG1_L (IFG1)		Read/write	Byte	82h	
03h	SFRIFG1_H (IFG2)		Read/write	Byte	00h	
04h	SFRRPCR	Reset Pin Control	Read/write	Word	0000h	<a href="#">Section 1.14.3</a>
04h	SFRRPCR_L		Read/write	Byte	00h	
05h	SFRRPCR_H		Read/write	Byte	00h	

**1.14.1 SFRIE1 Register**

Interrupt Enable Register

**Figure 1-8. 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	ACCVIE <sup>(1)</sup>	NMIIE	VMAIE	Reserved	OFIE <sup>(2)</sup>	WDTIE <sup>(3)</sup>
rw-0	rw-0	rw-0	rw-0	rw-0	r0	rw-0	rw-0

<sup>(1)</sup> See the [Flash Controller chapter](#) for details.<sup>(2)</sup> See the [UCS chapter](#) for details.<sup>(3)</sup> See the [WDT\\_A chapter](#) for details.**Table 1-12. 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	ACCVIE	RW	0h	Flash controller access violation interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
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 ~IE1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instruction 0b = Interrupts disabled 1b = Interrupts enabled

### 1.14.2 SFRIFG1 Register

Interrupt Flag Register

**Figure 1-9. 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 <sup>(1)</sup>	WDTIFG <sup>(2)</sup>
rw-(1)	rw-(0)	r0	rw-0	rw-0	r0	rw-(1)	rw-0

<sup>(1)</sup> See the [UCS chapter](#) for details.

<sup>(2)</sup> See the [WDT\\_A chapter](#) for details.

**Table 1-13. SFRIFG1 Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	JMBOUTIFG	RW	1h	<p>JTAG mailbox output interrupt flag</p> <p>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.</p> <p>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.</p>
6	JMBINIFG	RW	0h	<p>JTAG mailbox input interrupt flag</p> <p>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</p> <p>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.</p>
5	Reserved	R	0h	Reserved. Always reads as 0.
4	NMIIFG	RW	0h	<p>NMI pin interrupt flag</p> <p>0b = No interrupt pending</p> <p>1b = Interrupt pending</p>
3	VMAIFG	RW	0h	<p>Vacant memory access interrupt flag</p> <p>0b = No interrupt pending</p> <p>1b = Interrupt pending</p>
2	Reserved	R	0h	Reserved. Always reads as 0.
1	OFIFG	RW	1h	<p>Oscillator fault interrupt flag</p> <p>0b = No interrupt pending</p> <p>1b = Interrupt pending</p>

**Table 1-13. SFRIFG1 Register Description (continued)**

Bit	Field	Type	Reset	Description
0	WDTIFG	RW	0h	<p>Watchdog timer interrupt flag. In watchdog mode, WDTIFG will self clear 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 ~IFG1 may be used for other modules, it is recommended to set or clear WDTIFG by using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions.</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>

### 1.14.3 SFRRPCR Register

Reset Pin Control Register

**Figure 1-10. 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				SYSRSTRE <sup>(1)</sup>	SYSRSTUP <sup>(1)</sup>	SYSNMIIES	SYSNMI
r0	r0	r0	r0	rw-1	rw-1	rw-0	rw-0

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

**Table 1-14. SFRRPCR Register Description**

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved. Always reads as 0.
3	SYSRSTRE	RW	1h	Reset pin resistor enable 0b = Pullup/pulldown resistor at the RST/NMI pin is disabled 1b = Pullup/pulldown resistor at the 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 configuration registers are listed in [Table 1-16](#) and the base address is listed in [Table 1-15](#). A detailed description of each register and its bits is also provided. Each register starts at a word boundary. Either word or byte data can be written to the SYS configuration registers.

---

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

---

**Table 1-15. SYS Base Address**

Module	Base Address
SYS	00180h

**Table 1-16. SYS Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SYSCTL	System Control	Read/write	Word	0000h	<a href="#">Section 1.15.1</a>
00h	SYSCTL_L		Read/write	Byte	00h	
01h	SYSCTL_H		Read/write	Byte	00h	
02h	SYSBSLC	Bootstrap Loader Configuration	Read/write	Word	0003h	<a href="#">Section 1.15.2</a>
02h	SYSBSLC_L		Read/write	Byte	03h	
03h	SYSBSLC_H		Read/write	Byte	00h	
06h	SYSJMBC	JTAG Mailbox Control	Read/write	Word	0000h	<a href="#">Section 1.15.3</a>
06h	SYSJMBC_L		Read/write	Byte	00h	
07h	SYSJMBC_H		Read/write	Byte	00h	
08h	SYSJMBI0	JTAG Mailbox Input 0	Read/write	Word	0000h	<a href="#">Section 1.15.4</a>
08h	SYSJMBI0_L		Read/write	Byte	00h	
09h	SYSJMBI0_H		Read/write	Byte	00h	
0Ah	SYSJMBI1	JTAG Mailbox Input 1	Read/write	Word	0000h	<a href="#">Section 1.15.5</a>
0Ah	SYSJMBI1_L		Read/write	Byte	00h	
0Bh	SYSJMBI1_H		Read/write	Byte	00h	
0Ch	SYSJMBO0	JTAG Mailbox Output 0	Read/write	Word	0000h	<a href="#">Section 1.15.6</a>
0Ch	SYSJMBO0_L		Read/write	Byte	00h	
0Dh	SYSJMBO0_H		Read/write	Byte	00h	
0Eh	SYSJMBO1	JTAG Mailbox Output 1	Read/write	Word	0000h	<a href="#">Section 1.15.7</a>
0Eh	SYSJMBO1_L		Read/write	Byte	00h	
0Fh	SYSJMBO1_H		Read/write	Byte	00h	
18h	SYSBERRIV	Bus Error Vector Generator	Read	Word	0000h	<a href="#">Section 1.15.11</a>
1Ah	SYSUNIV	User NMI Vector Generator	Read	Word	0000h	<a href="#">Section 1.15.8</a>
1Ch	SYSSNIV	System NMI Vector Generator	Read	Word	0000h	<a href="#">Section 1.15.9</a>
1Eh	SYSRSTIV	Reset Vector Generator	Read	Word	0002h	<a href="#">Section 1.15.10</a>

### 1.15.1 SYSCTL Register

SYS Control Register

**Figure 1-11. 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	Reserved	SYSRIVECT
r0	r0	rw-[0]	r-0	r0	rw-[0]	r0	rw-[0]

**Table 1-17. SYSCTL Register Description**

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5	SYSJTAGPIN	RW	0h	Dedicated JTAG pins enable. Setting this bit disables the shared functionality of the JTAG pins and permanently enables the JTAG function. This bit can only be set once. Once it is set it remains set until a BOR occurs. 0b = Shared JTAG pins (JTAG mode selectable via 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. Once 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 64k flash FFFFh 1b = Interrupt vectors generated with end address TOP of RAM

**1.15.2 SYSBSLC Register**

Bootstrap Loader Configuration Register

**Figure 1-12. 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	SYSBSLSIZE
r0	r0	r0	r0	r0	rw-[0]	rw-[1]	rw-[1]

**Table 1-18. SYSBSLC Register Description**

Bit	Field	Type	Reset	Description
15	SYSBSLPE	RW	0h	Bootstrap loader memory protection enable for the size covered in SYSBSLSIZE. By default, this bit is cleared by hardware with a BOR event (as indicated above), however the boot code that checks for an available BSL may set this bit via software in order to protect the BSL. Since devices normally come with a TI BSL preprogrammed and protected, the boot code sets this bit. 0b = Area not protected. Read, program, and erase of BSL memory is possible. 1b = Area protected
14	SYSBSLOFF	RW	0h	Bootstrap loader memory disable for the size covered in SYSBSLSIZE 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	SYSBSLSIZE	RW	03h	Bootstrap loader size. Defines the space and size of flash memory that is reserved for the BSL. 00b = Size: BSL segment 3 01b = Size: BSL segments 2 and 3 10b = Size: BSL segments 1, 2, and 3 11b = Size: BSL segments 1, 2, 3, and 4

### 1.15.3 SYSJMBC Register

JTAG Mailbox Control Register

**Figure 1-13. SYSJMBC Register**

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
JMBCLR1OFF	JMBCLR0OFF	Reserved	JMBM0DE	JMBOUT1FG	JMBOUT0FG	JMBIN1FG	JMBIN0FG
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

**Table 1-19. SYSJMBC Register Description**

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

**1.15.4 SYSJMBI0 Register**

JTAG Mailbox Input 0 Register

**Figure 1-14. 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 1-20. 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**

JTAG Mailbox Input 0 Register

**Figure 1-15. 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 1-21. 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

JTAG Mailbox Output 0 Register

**Figure 1-16. 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 1-22. 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

JTAG Mailbox Output 1 Register

**Figure 1-17. 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 1-23. 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**

User NMI Vector Register

**NOTE:** Additional events for more complex devices are 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 1-18. SYSUNIV Register**

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

**Table 1-24. 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. 00h = No interrupt pending 02h = NMIIFG interrupt pending (highest priority) 04h = OFIFG interrupt pending 06h = ACCVIFG interrupt pending 08h = BUSIFG interrupt pending (Not present on all devices. See device-specific datasheet)

### **1.15.9 SYSSNIV Register**

System NMI Vector Register

---

**NOTE:** Additional events for more complex devices are 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 used device.

---

**Figure 1-19. SYSSNIV Register**

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

**Table 1-25. SYSSNIV Register Description**

Bit	Field	Type	Reset	Description
15-0	SYSSNIV	R	0h	<p>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.</p> <p>00h = No interrupt pending            02h = SVMLIFG interrupt pending (highest priority)            04h = SVMHIFG interrupt pending            06h = SVSMLDLYIFG interrupt pending            08h = SVSMHDLYIFG interrupt pending            0Ah = VMAIFG interrupt pending            0Ch = JMBINIFG interrupt pending            0Eh = JMBOUTIFG interrupt pending            10h = SVMLVLRIFG interrupt pending            12h = SVMHVLRIFG interrupt pending            14h = Reserved</p>

**1.15.10 SYSRSTIV Register**

Reset Interrupt Vector Register

NOTE: Additional events for more complex devices are 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 used device.

**Figure 1-20. SYSRSTIV Register**

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

**Table 1-26. SYSRSTIV Register Description**

Bit	Field	Type	Reset	Description
15-0	SYSRSTIV	R	0h	<p>Reset interrupt vector. Generates a value that can be used as address offset for fast interrupt service routine handling to identify the last cause of a reset (BOR, POR, PUC). Writing to this register clears all pending reset source flags.</p> <p>00h = No interrupt pending            02h = Brownout (BOR) (highest priority)            04h = <del>RST</del>/NMI (BOR)            06h = PMMSWBOR (BOR)            08h = Wakeup from LPMx.5 (BOR)            0Ah = Security violation (BOR)            0Ch = SVSL (POR)            0Eh = SVSH (POR)            10h = SVM_L_OVP (POR)            12h = SVM_H_OVP (POR)            14h = PMMSWPOR (POR)            16h = WDT time out (PUC)            18h = WDT password violation (PUC)            1Ah = Flash password violation (PUC)            1Ch = Reserved            1Eh = PERF peripheral/configuration area fetch (PUC)            20h = PMM password violation (PUC)            22h to 3Eh = Reserved</p>

### 1.15.11 SYSBERRIV Register

System Bus Error Interrupt Vector Register

NOTE: Additional events for more complex devices are 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 used device.

**Figure 1-21. SYSBERRIV Register**

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

**Table 1-27. SYSBERRIV Register Description**

Bit	Field	Type	Reset	Description
15-0	SYSBERRIV	R	0h	<p>System bus error interrupt vector. Generates a value that can be used as an address offset for fast interrupt service routine handling. Writing to this register clears all pending flags.</p> <p>00h = No interrupt pending</p> <p>02h = USB module timed out. Wait state time out of 8 clock cycles. 16 clock cycles only on the F552x and F551x devices.</p> <p>04h = Reserved for future extensions</p> <p>06h = Reserved for future extensions</p> <p>08h = Reserved for future extensions</p>

## **Power Management Module and Supply Voltage Supervisor**

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

Topic	Page
2.1 Power Management Module (PMM) Introduction .....	99
2.2 PMM Operation .....	101
2.3 PMM Registers .....	114

## 2.1 Power Management Module (PMM) Introduction

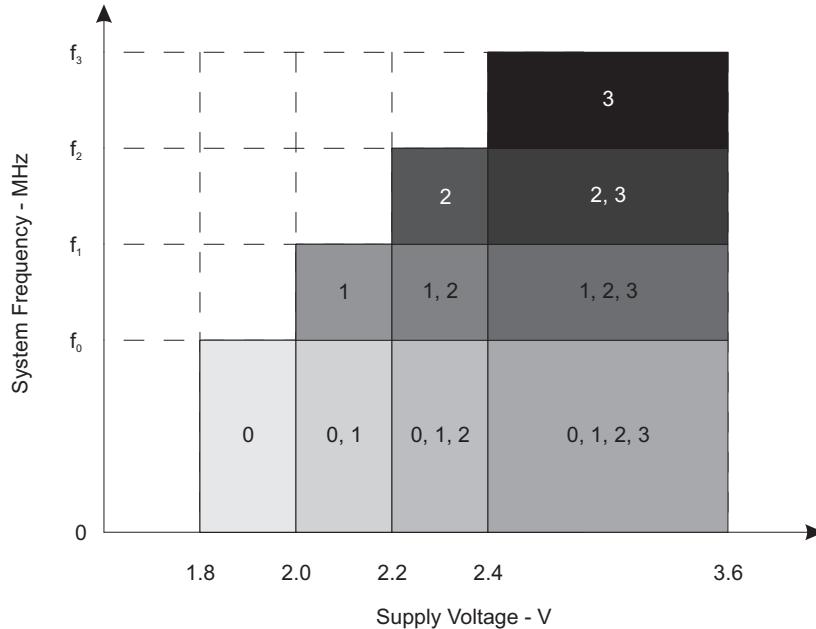
PMM features include:

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

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

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

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



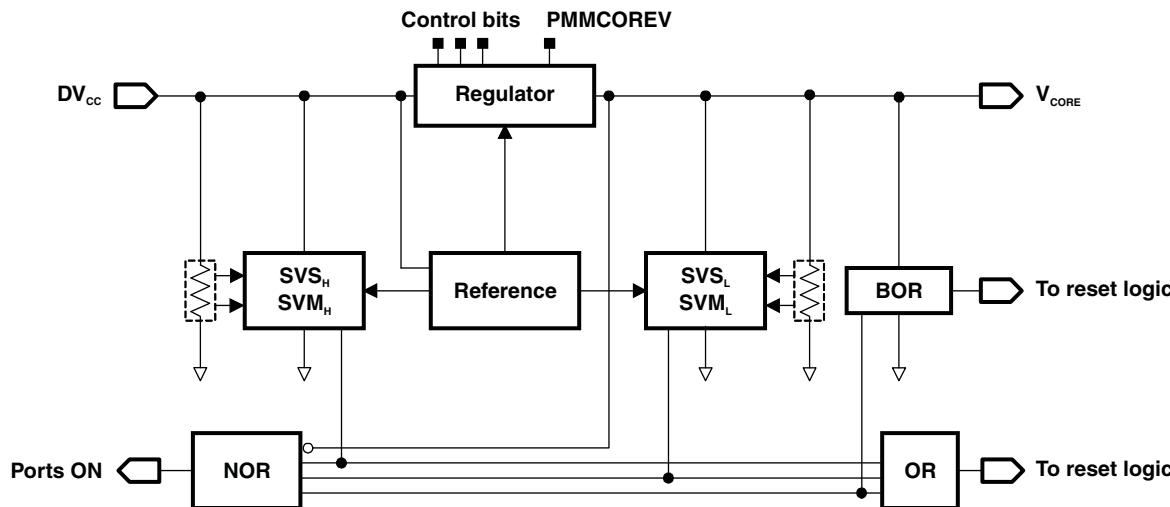
The numbers within the fields denote the supported PMMCOREVx settings.

**Figure 2-1. System Frequency, Supply Voltage, and Core Voltage – See Device-Specific Data Sheet**

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

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

The block diagram of the PMM is shown in [Figure 2-2](#).



**Figure 2-2. PMM Block Diagram**

## 2.2 PMM Operation

### 2.2.1 $V_{CORE}$ and the Regulator

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

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

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

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

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

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

### 2.2.2 Supply Voltage Supervisor and Monitor

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

Typical application scenarios for supply voltage supervisors and monitors are:

- High-Side Supervisor, SVSH
  - Supervision of external power supply ( $DV_{CC}$ )
  - Device reset because of low battery or supply voltage
- High-Side Monitor, SVMH
  - Monitoring of external power supply ( $DV_{CC}$ )
  - Detection of low battery voltage (Pre-warning)
- Low-Side Supervisor, SVSL
  - Supervision of internal core voltage used to supply digital core
  - Device reset because of disruptive conditions at external VCORE pin (for example a short). The internal core voltage never drops below a critical level if parasitic events at the external VCORE pin are avoided.
- Low-Side Monitor, SVML
  - Monitoring of internal core voltage used to supply digital core
  - Detection of correct internal voltage levels when changing (especially increasing) the core voltage level before changing, for example, to higher system frequencies (also see [Section 2.2.4](#)).

#### 2.2.2.1 SVS and SVM Thresholds

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

**Table 2-1. SVS and SVM Thresholds**

Register	Description	Threshold	Available Steps
SVSHRVL	SVS <sub>H</sub> reset voltage level	SVS <sub>H_IT-</sub>	4
SVSMHRRRL	SVS <sub>H</sub> , SVM <sub>H</sub> reset release voltage level	SVS <sub>H_IT+</sub> , SVM <sub>H</sub>	8
SVSLRVL	SVS <sub>L</sub> reset voltage level	SVS <sub>L_IT-</sub>	4
SVSMLRRL	SVS <sub>L</sub> , SVM <sub>L</sub> reset release voltage level	SVS <sub>L_IT+</sub> , SVM <sub>L</sub>	4 <sup>(1)</sup>

<sup>(1)</sup> The register settings support up to eight levels (0 through 7); however, levels 3 through 7 are identical.

### Recommended SVS<sub>L</sub> Settings

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

**Table 2-2. Recommended SVS<sub>L</sub> Settings**

PMMCOREV[1:0]	DVCC (V)	SVSLRVL[1:0] Sets SVS <sub>L_IT-</sub> .Level	SVSMLRRL[2:0] Sets SVS <sub>L_IT+</sub> and SVM <sub>L</sub> levels
00	≥ 1.8	00	000
01	≥ 2.0	01	001
10	≥ 2.2	10	010
11	≥ 2.4	11	011

### Recommended SVS<sub>H</sub> Settings

For the high-side supply, there are two supply voltage supervisor levels available. The SVSMHRRRL bits define the voltage level of DVCC at which the reset is released. The SVSHRVL register defines the voltage level of DVCC below which the reset is turned on. These settings should be selected according to the minimum voltages required for device operation in a given application, as well as system power supply characteristics. See the device-specific data sheet for threshold values corresponding to the settings shown here. Although various settings are available, the most common are based on the maximum frequency required which, in turn, determines the minimum DVCC level supervised. By default, an SVS<sub>H</sub> event always generates a POR (SVSHPE = 1), and it is recommended to always configure SVSHPE = 1 for reliable device startup. The most commonly used and recommended settings are shown in [Table 4-2](#).

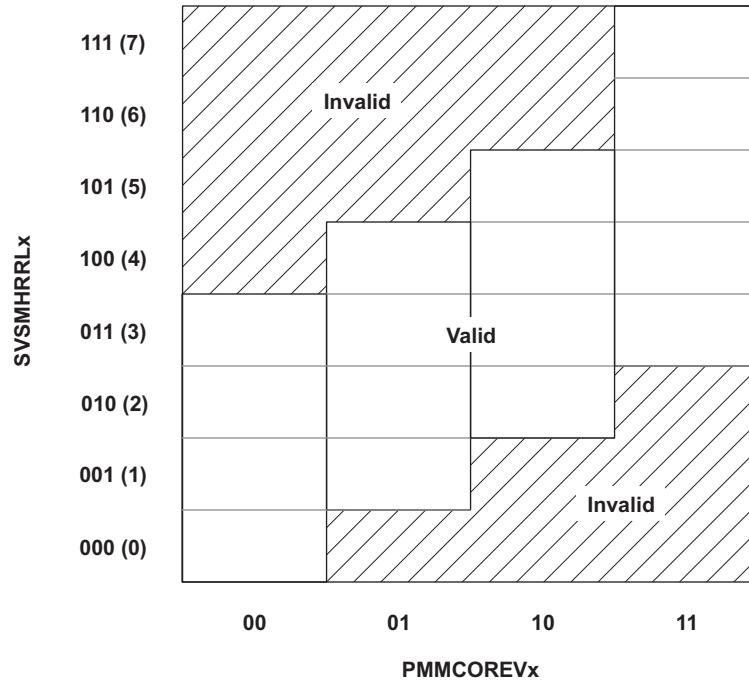
**Table 2-3. Recommended SVS<sub>H</sub> Settings**

f <sub>sys</sub> Max (MHz)	DVCC (V)	SVSHRVL[1:0] Sets SVS <sub>H_IT-</sub> .Level	SVSMHRRRL[2:0] Sets SVS <sub>H_IT+</sub> and SVM <sub>H</sub> Levels	PMMCOREV[1:0]
8	>1.8	00	000	00
12	>2.0	01	001	01
20	>2.2	10	010	10
25	>2.4	11	011	11

The available voltage threshold settings of SVS<sub>H</sub> and SVM<sub>H</sub> are dependent on the voltage level setting of VCORE. [Table 2-4](#) summarizes all the possible settings available. All other settings not listed are invalid and should not be used. [Figure 2-3](#) shows the available settings for the SVM<sub>H</sub>.

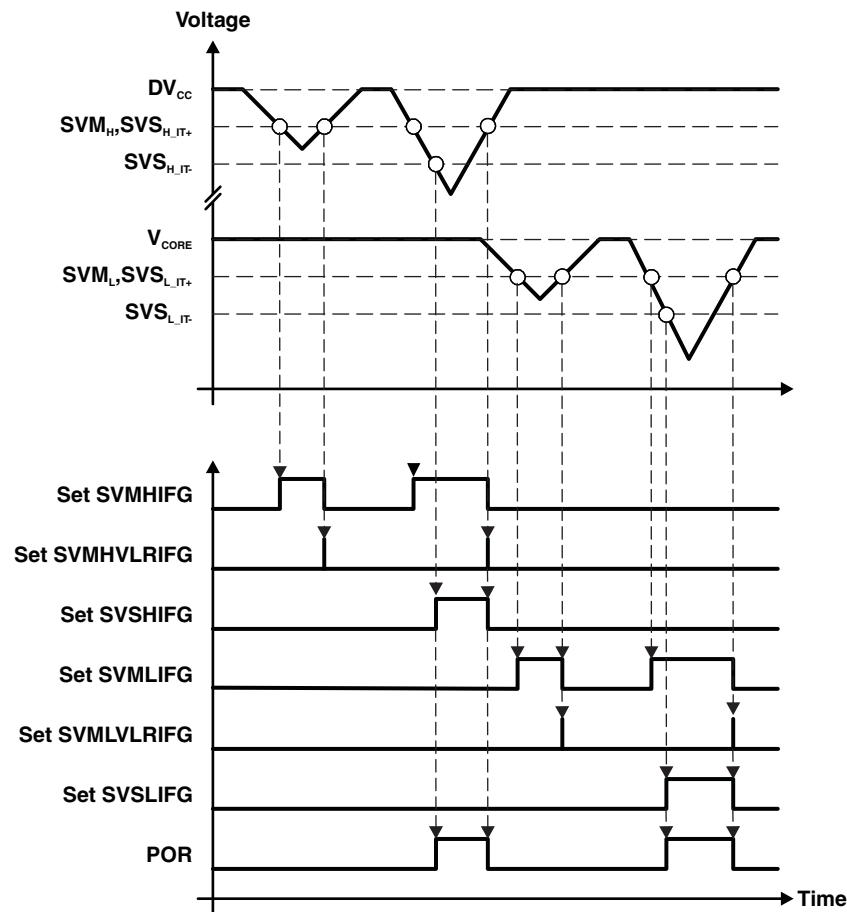
**Table 2-4. Available SVSHRL[1:0] and SVMHRRRL[2:0] Settings Versus V<sub>CORE</sub> Settings**

PMMCOREV[1:0]	SVSHRL[1:0] Sets SVSHRL <sub>H_IT</sub> Level	SVMHRRRL[2:0] Sets SVMHRRRL <sub>H_IT+</sub> and SVM <sub>H</sub> Levels
00	00 through 11	000 through 011
01	00 through 11	001 through 100
10	00 through 11	010 through 101
11	00 through 11	011 through 111


**Figure 2-3. Available SVM<sub>H</sub> Settings Versus VCORE Settings**

The behavior of the SVS and SVM according to these thresholds is best portrayed graphically. [Figure 2-4](#) shows how the supervisors and monitors respond to various supply failure conditions.

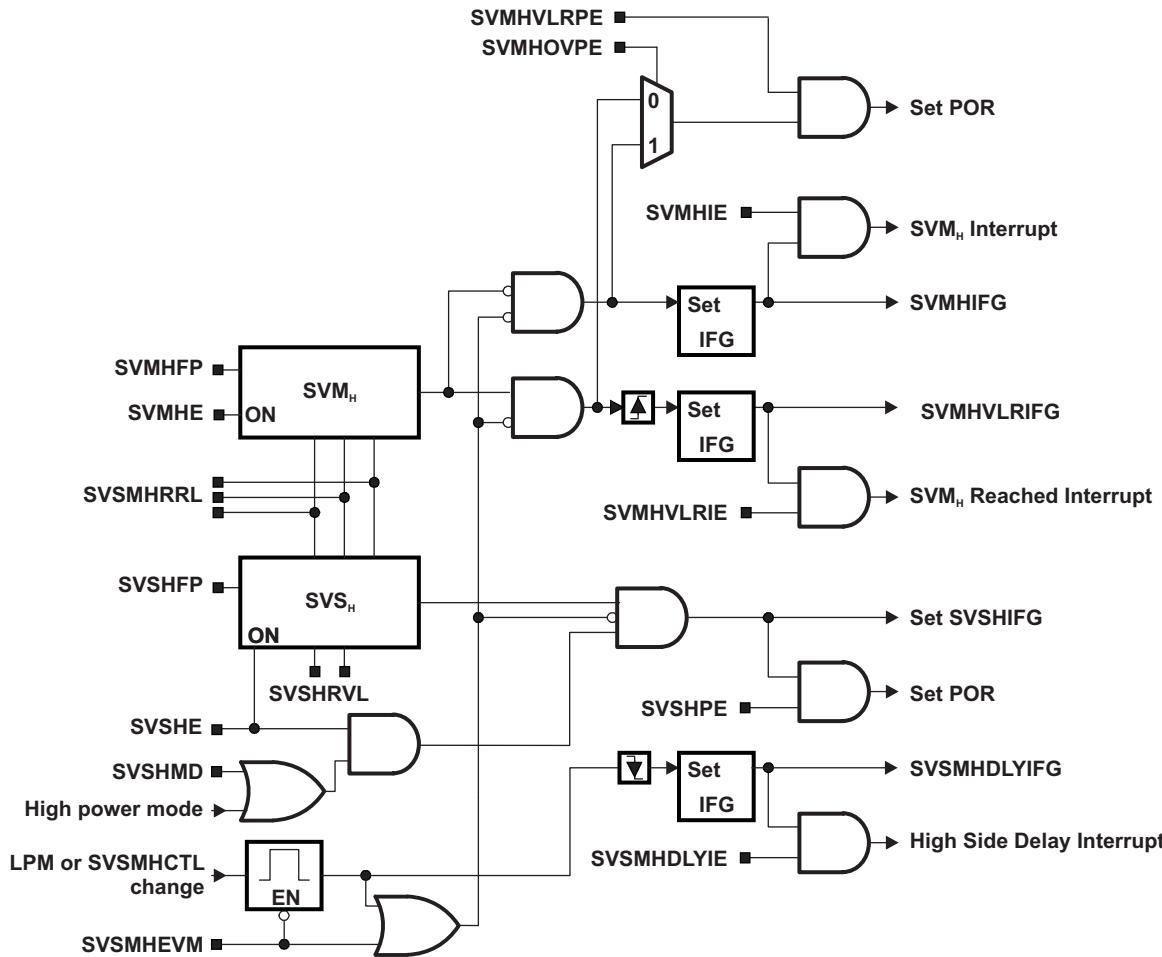
As [Figure 2-4](#) shows, there is hysteresis built into the supervision thresholds, such that the thresholds in force depend on whether the voltage rail is going up or down. There is no hysteresis in the monitoring thresholds.



**Figure 2-4. High-Side and Low-Side Voltage Failure and Resulting PMM Actions**

### 2.2.2.2 High-Side Supervisor ( $SVS_H$ ) and High-Side Monitor ( $SVM_H$ )

The  $SVS_H$  and  $SVM_H$  modules are enabled by default. They can be disabled by clearing the  $SVSHE$  and  $SVMHE$  bits, respectively. Their block diagrams are shown in [Figure 2-5](#).



**Figure 2-5. High-Side SVS and SVM**

If DV<sub>CC</sub> falls below the SVS<sub>H</sub> level, SVSHIFG (SVS<sub>H</sub> interrupt flag) is set. If DV<sub>CC</sub> remains below the SVS<sub>H</sub> level and software attempts to clear SVSHIFG, it is immediately set again by hardware. If the SVSHPE (SVS<sub>H</sub> POR enable) bit is set when SVSHIFG gets set, a POR is generated.

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

If DV<sub>CC</sub> rises above the SVM<sub>H</sub> level, the SVMHVLRIFG (SVM<sub>H</sub> voltage level reached) interrupt flag is set. If SVMHVLRIE (SVM<sub>H</sub> voltage level reached interrupt enable) is set when this occurs, an interrupt is also generated.

Alternatively the SVM<sub>H</sub> module can be used for overvoltage detection, but only with the highest core voltage setting (PMMCOREV = 11b). This is accomplished by setting the SVMHOVPE (SVM<sub>H</sub> overvoltage POR enable) bit in addition to setting SVMHVLRPE. Under these conditions, if a rising DV<sub>CC</sub> exceeds safe device operation, a POR is generated.

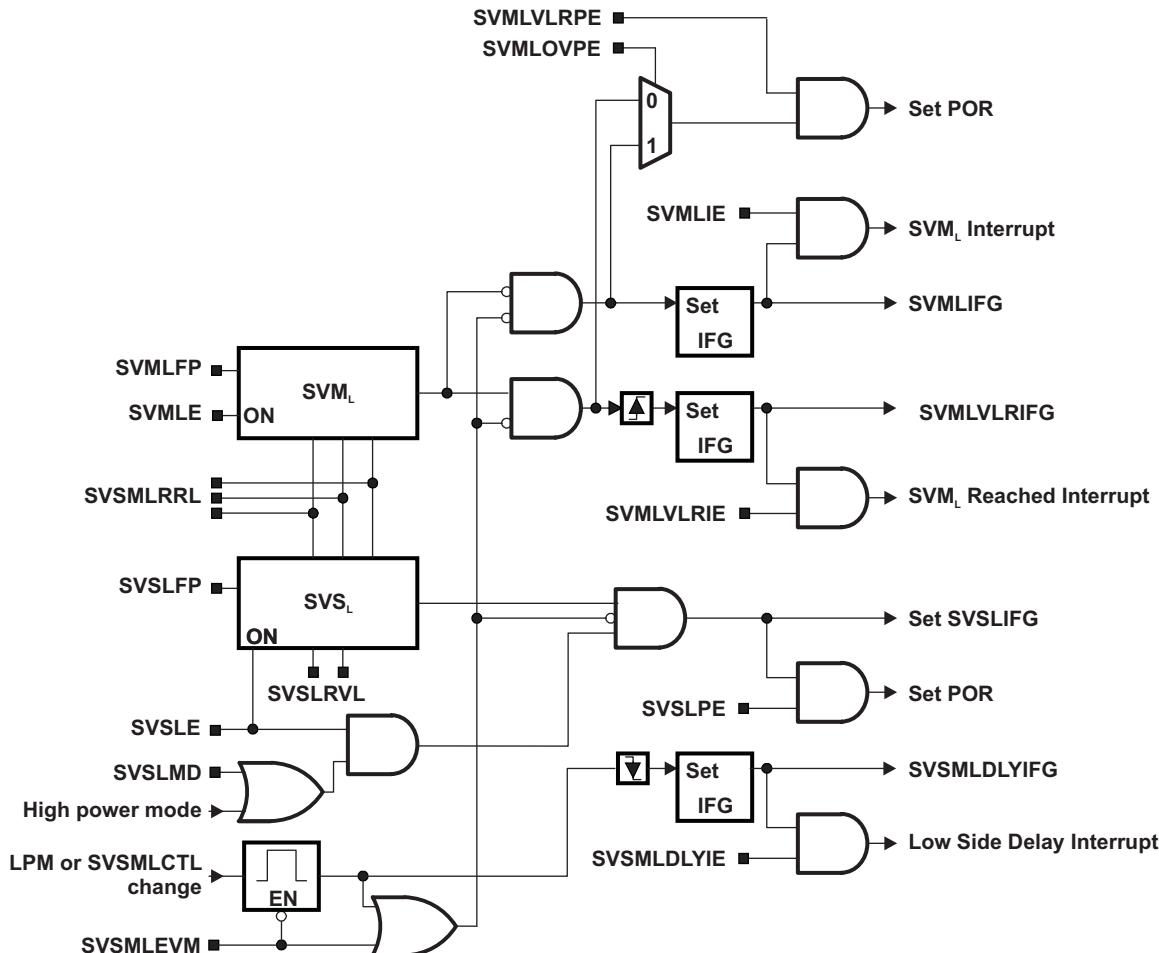
The SVS<sub>H</sub> and SVM<sub>H</sub> modules have configurable performance modes for power-saving operation. (See [Section 2.2.9](#) for more information.) If these SVS<sub>H</sub> and SVM<sub>H</sub> power modes are modified, or if a voltage level is modified, a delay element masks the interrupts and POR sources until the SVS<sub>H</sub> and SVM<sub>H</sub> circuits have settled. When SVSMHDLYST (delay status) reads zero, the delay has expired. In addition, the SVSMHDLYIFG (SVS<sub>H</sub> and SVM<sub>H</sub> delay expired) interrupt flag is set. If the SVSMHDLYIE (SVS<sub>H</sub> and SVM<sub>H</sub> delay expired interrupt enable) is set when this occurs, an interrupt is also generated.

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

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

### 2.2.2.3 Low-Side Supervisor ( $SVS_L$ ) and Low-Side Monitor ( $SVM_L$ )

The  $SVS_L$  and  $SVM_L$  modules are enabled by default. They can be disabled by clearing SVSLE and SVMLE bits, respectively. Their block diagrams are shown in [Figure 2-6](#).



**Figure 2-6. Low-Side SVS and SVM**

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

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

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

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

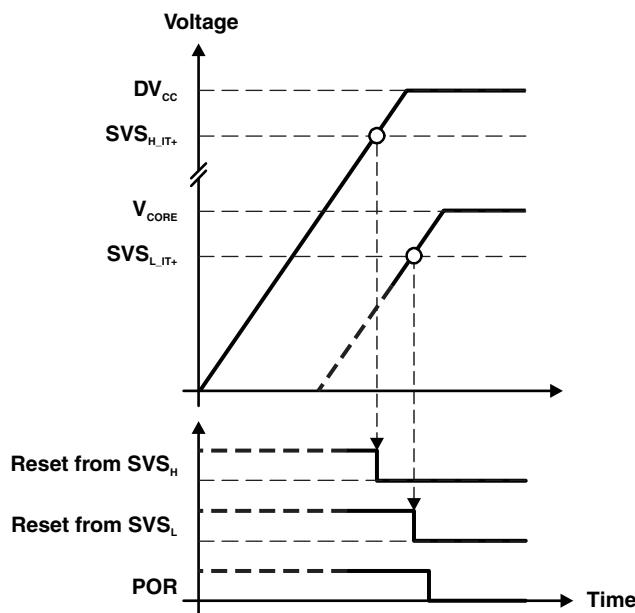
The SVS<sub>L</sub> and SVM<sub>L</sub> modules have configurable performance modes for power-saving operation. (See [Section 2.2.9](#) for more information.) If these SVS<sub>L</sub> and SVM<sub>L</sub> power modes are modified, or if a voltage level is modified, a delay element masks the interrupts and POR sources until the SVS<sub>L</sub> and SVM<sub>L</sub> circuits have settled. When SVSMLDLYST (delay status) reads zero, the delay has expired. In addition, the SVSMLDLYIFG (SVS<sub>L</sub>/SVM<sub>L</sub> delay expired) interrupt flag is set. If the SVSMLDLYIE (SVS<sub>L</sub>/SVM<sub>L</sub> delay expired interrupt enable) is set when this occurs, an interrupt is also generated.

In case of power-fail conditions, setting SVSLMD causes the SVS<sub>L</sub> interrupt flag to be set in LPM2, LPM3, and LPM4. If SVSLMD is not set, the SVS<sub>L</sub> interrupt flag is not set in LPM2, LPM3, and LPM4. In addition, all SVS<sub>L</sub> and SVM<sub>L</sub> events can be masked by setting SVSMLEVM. For most applications, SVSMLEVM should be cleared.

All the interrupt flags of SVS<sub>L</sub> and SVM<sub>L</sub> remain set until cleared by a BOR or by software.

### 2.2.3 Supply Voltage Supervisor and Monitor - Power-Up

When the device is powering up, the SVS<sub>H</sub> and SVS<sub>L</sub> functions are enabled by default. Initially, DV<sub>CC</sub> is low, and therefore the PMM holds the device in POR reset. When both the SVS<sub>H</sub> and SVS<sub>L</sub> levels are met, the reset is released. [Figure 2-7](#) shows this process.



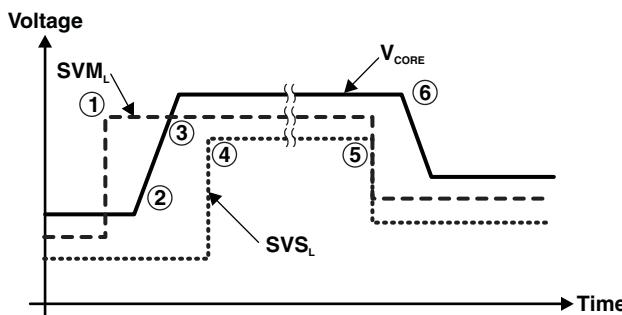
**Figure 2-7. PMM Action at Device Power-Up**

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

### 2.2.4 Increasing $V_{CORE}$ to Support Higher MCLK Frequencies

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

After setting PMMCOREV to increase  $V_{CORE}$ , there is a time delay until the new voltage has been established. Software must not raise MCLK until the necessary core voltage has settled. SVM<sub>L</sub> can be used to verify that  $V_{CORE}$  has met the required minimum value, prior to increasing MCLK. [Figure 2-8](#) shows this procedure.



**Figure 2-8. Changing  $V_{CORE}$  and  $SVM_L$  and  $SVS_L$  Levels**

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

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

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

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

```
void SetVCoreUp (unsigned int level)  
{  
    // Open PMM registers for write access  
    PMMCTL0_H = 0xA5;  
    // Make sure no flags are set for iterative sequences  
    while ((PMMIFG & SVSMHDLYIFG) == 0);  
    while ((PMMIFG & SVSMLDLYIFG) == 0);  
    // Set SVS/SVM high side new level  
    SVSMHCTL = SVSHE + SVSHRVL0 * level + SVMHE + SVSMHRRL0 * level;  
    // Set SVM low side to new level  
    SVSMLCTL = SVSLE + SVMLE + SVSMLRRL0 * level;  
    // Wait till SVM is settled  
    while ((PMMIFG & SVSMLDLYIFG) == 0);  
    // Clear already set flags  
    PMMIFG &= ~(SVMLVLRIFG + SVMLIFG);  
    // Set VCore to new level  
    PMMCTL0_L = PMMCOREV0 * level;  
    // Wait till new level reached  
    if ((PMMIFG & SVMLIFG))  
        while ((PMMIFG & SVMLVLRIFG) == 0);  
    // Set SVS/SVM low side to new level  
    SVSMLCTL = SVSLE + SVSLRVL0 * level + SVMLE + SVSMLRRL0 * level;  
    // Lock PMM registers for write access  
    PMMCTL0_H = 0x00;  
}
```

---

**NOTE:** See the *MSP430x5xx and MSP430x6xx Core Libraries* ([SLAA448](#)). These libraries contain useful and ready-to-use functions for easily configuring and using the PMM module.

---

### 2.2.5 Decreasing $V_{CORE}$ for Power Optimization

The risk posed by increasing MCLK frequency does not exist when decreasing MCLK from the current  $V_{CORE}$  or higher settings, because higher  $V_{CORE}$  levels can still support MCLK frequencies below the ones for which they were intended. However, significant power efficiency gains can be made by operating  $V_{CORE}$  at the lowest value required for a given MCLK frequency. It is critical that the  $V_{CORE}$  level be decreased by only one level at a time. The following steps show the procedure to decrease  $V_{CORE}$  by one level. This sequence is repeated to change the  $V_{CORE}$  level until the targeted level is obtained:

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

- Step 5: Program the SVM<sub>L</sub> and SVS<sub>L</sub> to the new level and wait for (SVSMLDLYIFG) to be set.
- Step 6: Program PMMCOREV to the new  $V_{CORE}$  level.

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

### 2.2.6 Transition From LPM3 and LPM4 Modes to AM

The LDO requires time to settle when the application transitions from low-power modes to active modes. If a transition from LPM3 or LPM4 occurs and the device does not stay in active mode long enough, the LDO does not have time to settle sufficiently. Circuitry inside the LDO ensures that the LDO has its minimum required time to settle to its proper operating voltage. The circuitry ensures that every eighth transition from LPM3 or LPM4 causes the LDO to remain on long enough to properly settle. This is handled automatically and requires no setting by the application.

### 2.2.7 LPM3.5 and LPM4.5

LPM3.5 and LPM4.5 are additional low-power modes in which the regulator of the PMM is completely disabled, providing additional power savings. Not all devices support all LPMx.5 modes, so see the device-specific data sheet. Because there is no power supplied to VCORE during LPMx.5, the CPU and all digital modules including RAM are unpowered. This disables the entire device and, as a result, the contents of the registers and RAM are lost. Any essential values should be stored to flash prior to entering LPMx.5. PMMREGOFF bit is used to disable the regulator. See the SYS module for complete descriptions and proper uses of LPMx.5.

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

### 2.2.8 Brownout Reset (BOR), Software BOR, Software POR

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

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

### **2.2.9 SVS and SVM Performance Modes and Wakeup Times**

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

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

In automatic mode, hardware changes the normal or full-performance selection depending on the operational mode in effect.

The wakeup time of the device from low-power modes is affected by the settings of the SVS<sub>L</sub> and SVM<sub>L</sub> performance modes as listed in [Table 2-6](#), [Table 2-7](#), [Table 2-8](#), and [Table 2-9](#). The wakeup time from low-power modes is not affected by the settings of the SVS<sub>H</sub> and SVM<sub>H</sub>. All wake-ups from LPMx.5 (LPM3.5 or LPM4.5), are defined by the data sheet parametric, t<sub>WAKE-UP-LPM5</sub>, regardless of the performance modes for SVS<sub>L</sub> or SVM<sub>L</sub>, because these are disabled in LPMx.5.

The tables in [Section 2.2.9.1](#) and [Section 2.2.9.2](#) show the required settings to select the control and performance modes for SVS<sub>L</sub>, SVM<sub>L</sub>, SVS<sub>H</sub>, and SVM<sub>H</sub>.

---

**NOTE: Low-Power Modes**

Even if the CPU requests a specific low-power mode, the device might not go into that state because of modules requesting clocks that should be switched off or have higher frequencies or because of modules requesting a higher drive capability of the LDO. The low-power modes mentioned in the tables assume that the device is actually in the requested state; that is, no module is requesting a deviating clock setting or drive capability.

---

## 2.2.9.1 Low-Side SVS and SVM Control and Performance Mode Selection

**Table 2-5. SVS<sub>L</sub> and SVM<sub>L</sub> Control Mode Selection**

SVSMLACE	SVSLMD	SVS <sub>L</sub> Control Mode	SVM <sub>L</sub> Control Mode
0	0	Automatic (see Table 2-6)	Manual (see Table 2-8)
0	1	Manual (see Table 2-7)	Manual (see Table 2-8)
1	0	Automatic (see Table 2-6)	Automatic (see Table 2-9)
1	1	Automatic (see Table 2-6)	Automatic (see Table 2-9)

**Table 2-6. SVS<sub>L</sub> Automatic Performance Control**

SVSLE	SVSLMD	SVSLFP	AM, LPM0, LPM1 SVS <sub>L</sub> State	LPM2, LPM3, LPM4 SVS <sub>L</sub> State	Wakeup Time LPM2, LPM3, LPM4
0	x	x	Off	Off	t <sub>WAKE-UP-FAST</sub>
1	0	0	Normal	Off	t <sub>WAKE-UP-SLOW</sub>
1	0	1	Full performance	Off	t <sub>WAKE-UP-FAST</sub>
1	1	0	Normal	Off	t <sub>WAKE-UP-SLOW</sub>
1	1	1	Full performance	Normal	t <sub>WAKE-UP-FAST</sub>

**Table 2-7. SVS<sub>L</sub> Manual Performance Modes**

SVSLE		SVSLFP	AM, LPM0, LPM1 SVS <sub>L</sub> State	LPM2, LPM3, LPM4 SVS <sub>L</sub> State	Wakeup Time LPM2, LPM3, LPM4
0		x	Off	Off	t <sub>WAKE-UP-FAST</sub>
1		0	Normal	Normal	t <sub>WAKE-UP-SLOW</sub>
1		1	Full performance	Full performance	t <sub>WAKE-UP-FAST</sub>

**Table 2-8. SVM<sub>L</sub> Automatic Performance Control**

SVMLE		SVMLFP	AM, LPM0, LPM1 SVM <sub>L</sub> State	LPM2, LPM3, LPM4 SVM <sub>L</sub> State	Wakeup Time LPM2, LPM3, LPM4
0		x	Off	Off	t <sub>WAKE-UP-FAST</sub>
1		0	Normal	Off	t <sub>WAKE-UP-SLOW</sub>
1		1	Full performance	Normal	t <sub>WAKE-UP-FAST</sub>

**Table 2-9. SVM<sub>L</sub> Manual Performance Modes**

SVMLE		SVMLFP	AM, LPM0, LPM1 SVM <sub>L</sub> State	LPM2, LPM3, LPM4 SVM <sub>L</sub> State	Wakeup Time LPM2, LPM3, LPM4
0		x	Off	Off	t <sub>WAKE-UP-FAST</sub>
1		0	Normal	Normal	t <sub>WAKE-UP-SLOW</sub>
1		1	Full performance	Full performance	t <sub>WAKE-UP-FAST</sub>

## 2.2.9.2 High-Side SVS and SVM Control and Performance Mode Selection

**Table 2-10. SVS<sub>H</sub> and SVM<sub>H</sub> Control Mode Selection**

SVSMHACE	SVSHMD	SVS <sub>H</sub> Control Mode	SVM <sub>H</sub> Control Mode
0	0	Automatic (see <a href="#">Table 2-11</a> )	Manual (see <a href="#">Table 2-13</a> )
0	1	Manual (see <a href="#">Table 2-12</a> )	Manual (see <a href="#">Table 2-13</a> )
1	0	Automatic (see <a href="#">Table 2-11</a> )	Automatic (see <a href="#">Table 2-14</a> )
1	1	Automatic (see <a href="#">Table 2-11</a> )	Automatic (see <a href="#">Table 2-14</a> )

**Table 2-11. SVS<sub>H</sub> Automatic Performance Control**

SVSHE	SVSHMD	SVSHFP	AM, LPM0, LPM1 SVS <sub>H</sub> State	LPM2, LPM3, LPM4 SVS <sub>H</sub> State
0	x	x	Off	Off
1	0	0	Normal	Off
1	0	1	Full performance	Off
1	1	0	Normal	Off
1	1	1	Full performance	Normal

**Table 2-12. SVS<sub>H</sub> Manual Performance Modes**

SVSHE		SVSHFP	AM, LPM0, LPM1 SVS <sub>H</sub> State	LPM2, LPM3, LPM4 SVS <sub>H</sub> State
0		x	Off	Off
1		0	Normal	Normal
1		1	Full performance	Full performance

**Table 2-13. SVM<sub>H</sub> Automatic Performance Control**

SVMHE		SVMHFP	AM, LPM0, LPM1 SVM <sub>H</sub> State	LPM2, LPM3, LPM4 SVM <sub>H</sub> State
0		x	Off	Off
1		0	Normal	Off
1		1	Full performance	Normal

**Table 2-14. SVM<sub>H</sub> Manual Performance Modes**

SVMHE		SVMHFP	AM, LPM0, LPM1 SVM <sub>H</sub> State	LPM2, LPM3, LPM4 SVM <sub>H</sub> State
0		x	Off	Off
1		0	Normal	Normal
1		1	Full performance	Full performance

## 2.2.9.3 Wakeup Times in Debug Mode

The TEST/SBWTCK pin is used for interfacing to the development tools via Spy-Bi-Wire and JTAG. When the TEST/SBWTCK pin is high, wakeup times from LPM2, LPM3, and LPM4 may be different compared to when TEST/SBWTCK is low. When the TEST/SBWTCK pin is high, all delays associated with the SVS<sub>L</sub> and SVM<sub>L</sub> settings have no effect and the device wakes within  $t_{WAKE-UP-FAST}$ . Pay careful attention to the real-time behavior when exiting from LPM2, LPM3, and LPM4 with the device connected to a development tool (for example, MSP-FET430UIF).

### 2.2.10 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.

### 2.2.11 Port I/O Control

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

### 2.2.12 Supply Voltage Monitor Output (SVMOUT, Optional)

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

## 2.3 PMM Registers

The PMM registers are listed in [Table 2-15](#). The base address of the PMM module can be found in the device-specific data sheet. The address offset of each PMM register is given in [Table 2-15](#). The password, PMMPW, defined in the PMMCTL0 register controls access to all PMM, SVS, and SVM registers. Once the correct password is written, the write access is enabled. The write access is disabled by writing a wrong password in byte mode to the PMMCTL0 upper byte. Word accesses to PMMCTL0 with a wrong password triggers a PUC. A write access to a register other than PMMCTL0 while write access is not enabled causes a PUC.

---

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

---

**Table 2-15. PMM Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PMMCTL0	PMM control register 0	Read/write	Word	9600h	<a href="#">Section 2.3.1</a>
00h	PMMCTL0_L		Read/write	Byte	00h	
01h	PMMCTL0_H		Read/write	Byte	96h	
02h	PMMCTL1	PMM control register 1	Read/write	Word	0000h	<a href="#">Section 2.3.2</a>
02h	PMMCTL1_L		Read/write	Byte	00h	
03h	PMMCTL1_H		Read/write	Byte	00h	
04h	SVSMHCTL	SVS and SVM high side control register	Read/write	Word	4400h	<a href="#">Section 2.3.3</a>
04h	SVSMHCTL_L		Read/write	Byte	00h	
05h	SVSMHCTL_H		Read/write	Byte	44h	
06h	SVSMLCTL	SVS and SVM low side control register	Read/write	Word	4400h	<a href="#">Section 2.3.4</a>
06h	SVSMLCTL_L		Read/write	Byte	00h	
07h	SVSMLCTL_H		Read/write	Byte	44h	
08h	SVSMIO	SVSIN and SVMOUT control register (optional)	Read/write	Word	0020h	<a href="#">Section 2.3.5</a>
08h	SVSMIO_L		Read/write	Byte	20h	
09h	SVSMIO_H		Read/write	Byte	00h	
0Ch	PMMIFG	PMM interrupt flag register	Read/write	Word	0000h	<a href="#">Section 2.3.6</a>
0Ch	PMMIFG_L		Read/write	Byte	00h	
0Dh	PMMIFG_H		Read/write	Byte	00h	
0Eh	PMMRIE	PMM interrupt enable register	Read/write	Word	1100h	<a href="#">Section 2.3.7</a>
0Eh	PMMRIE_L		Read/write	Byte	00h	
0Fh	PMMRIE_H		Read/write	Byte	11h	
10h	PM5CTL0	Power mode 5 control register 0	Read/write	Word	0000h	<a href="#">Section 2.3.8</a>
10h	PM5CTL0_L		Read/write	Byte	00h	
11h	PM5CTL0_H		Read/write	Byte	00h	

### 2.3.1 PMMCTL0 Register

Power Management Module Control Register 0

**Figure 2-9. 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	Reserved	PMMREGOFF	PMMSWPOR	PMMSWBOR	PMMCOREV		
rw-0	r-0	r-0	rw-0	rw-0	rw-0	rw-[0]	rw-[0]

**Table 2-16. PMMCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-8	PMMPW	RW	96h	PMM password. Always read as 096h. When using word operations, must be written with 0A5h or a PUC is generated. When using byte operation, writing 0A5h unlocks all PMM registers. When using byte operation, writing anything different than 0A5h locks all PMM registers.
7	Reserved	RW	0h	Reserved. Must always be written as 0.
6-5	Reserved	R	0h	Reserved. Always reads as 0.
4	PMMREGOFF	RW	0h	Regulator off (see the SYS chapter for details)
3	PMMSWPOR	RW	0h	Software power-on reset. Setting this bit to 1 triggers a POR. This bit is self clearing.
2	PMMSWBOR	RW	0h	Software brownout reset. Setting this bit to 1 triggers a BOR. This bit is self clearing.
1-0	PMMCOREV	RW	0h	Core voltage (see the device-specific data sheet for supported levels and corresponding voltages) 00b = V(CORE) level 0 01b = V(CORE) level 1 10b = V(CORE) level 2 11b = V(CORE) level 3

### 2.3.2 PMMCTL1 Register

Power Management Module Control Register 1

**Figure 2-10. PMMCTL1 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		Reserved		Reserved		Reserved	
r-0	r-0	rw-[0]	rw-[0]	r-0	r-0	rw-0	rw-0

**Table 2-17. PMMCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	Reserved	RW	0h	Reserved. Must always be written with 0.
3-2	Reserved	R	0h	Reserved. Always reads as 0.
1-0	Reserved	RW	0h	Reserved. Must always be written with 0.

### 2.3.3 SVSMHCTL Register

Supply Voltage Supervisor and Monitor High-Side Control Register

**Figure 2-11. SVSMHCTL Register**

15	14	13	12	11	10	9	8
SVMHFP	SVMHE	Reserved	SVMHOVPE	SVSHFP	SVSHE	SVSHRVL	
rw-[0]	rw-1	r-0	rw-[0]	rw-[0]	rw-1	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
SVSMHACE	SVSMHEVM	Reserved	SVSHMD	SVSMHDLYST	SVSMHRRL		
rw-[0]	rw-0	r-0	rw-0	r-0	rw-[0]	rw-[0]	rw-[0]

**Table 2-18. SVSMHCTL Register Description**

Bit	Field	Type	Reset	Description
15	SVMHFP	RW	0h	SVM high-side full-performance mode. If this bit is set, the SVMH operates in full-performance mode. 0b = Normal mode. See the device-specific data sheet for response times. 1b = Full-performance mode. See the device-specific data sheet for response times.
14	SVMHE	RW	1h	SVM high-side enable. If this bit is set, the SVMH is enabled.
13	Reserved	R	0h	Reserved. Always reads as 0.
12	SVMHOVPE	RW	0h	SVM high-side overvoltage enable. If this bit is set, the SVMH overvoltage detection is enabled. If SVMHVLRPE is also set, a POR occurs on an overvoltage condition.
11	SVSHFP	RW	0h	SVS high-side full-performance mode. If this bit is set, the SVSH operates in full-performance mode. 0b = Normal mode. See the device-specific data sheet for response times. 1b = Full-performance mode. See the device-specific data sheet for response times.
10	SVSHE	RW	1h	SVS high-side enable. If this bit is set, the SVSH is enabled.
9-8	SVSHRVL	RW	0h	SVS high-side reset voltage level. If DVCC falls short of the SVSH voltage level selected by SVSHRVL, a reset is triggered (if SVSHPE = 1). The voltage levels are defined in the device-specific data sheet.
7	SVSMHACE	RW	0h	SVS and SVM high-side automatic control enable. If this bit is set, the low-power mode of the SVSH and SVMH circuits is under hardware control.
6	SVSMHEVM	RW	0h	SVS and SVM high-side event mask. If this bit is set, the SVSH and SVMH events are masked. 0b = No events are masked. 1b = All events are masked.
5	Reserved	R	0h	Reserved. Always reads as 0.
4	SVSHMD	RW	0h	SVS high-side mode. If this bit is set, the SVSH interrupt flag is set in LPM2, LPM3, and LPM4 in case of power-fail conditions. If this bit is not set, the SVSH interrupt is not set in LPM2, LPM3, and LPM4.
3	SVSMHDLYST	RW	0h	SVS and SVM high-side delay status. If this bit is set, the SVSH and SVMH events are masked for some delay time. The delay time depends on the power mode of the SVSH and SVMH. If SVMHFP = 1 and SVSHFP = 1 (that is, full-performance mode), the delay is shorter. See the device-specific data sheet for details. The bit is cleared by hardware if the delay has expired.
2-0	SVSMHRRL	RW	0h	SVS and SVM high-side reset release voltage level. These bits define the reset release voltage level of the SVSH. It is also used for the SVMH to define the voltage reached level. The voltage levels are defined in the device-specific data sheet.

### 2.3.4 SVSMLCTL Register

Supply Voltage Supervisor and Monitor Low-Side Control Register

**Figure 2-12. SVSMLCTL Register**

15	14	13	12	11	10	9	8
SVMLFP	SVMLE	Reserved	SVMLOVPE	SVSLFP	SVSLE	SVSLRVL	
rw-[0]	rw-1	r-0	rw-[0]	rw-[0]	rw-1	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
SVSMLACE	SVSMLEV	Reserved	SVSLMD	SVSMLDLYST	SVSMLRRL		
rw-[0]	rw-0	r-0	rw-0	r-0	rw-[0]	rw-[0]	rw-[0]

**Table 2-19. SVSMLCTL Register Description**

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

### 2.3.5 SVSMIO Register

SVSIN and SVMOUT Control Register

**Figure 2-13. SVSMIO Register**

15	14	13	12	11	10	9	8
Reserved		SVMHVLROE	SVMHOE		Reserved		
r-0	r-0	r-0	rw-[0]	rw-[0]	r-0	r-0	r-0
7	6	5	4	3	2	1	0
Reserved		SVMOUTPOL	SVMLVLROE	SVMLOE		Reserved	
r-0	r-0	rw-[1]	rw-[0]	rw-[0]	r-0	r-0	r-0

**Table 2-20. SVSMIO Register Description**

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

## 2.3.6 PMMIFG Register

Power Management Module Interrupt Flag Register

**Figure 2-14. PMMIFG Register**

15	14	13	12	11	10	9	8
PMMILPM5IFG	Reserved	SVSLIFG <sup>(1)</sup>	SVSHIFG <sup>(1)</sup>	Reserved	PMMPORIFG	PMMRSTIFG	PMMBORIFG
rw-[0]	r-0	rw-[0]	rw-[0]	r-0	rw-[0]	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
Reserved	SVMHVLRIFG <sup>(1)</sup>	SVMHIFG	SVSMHDLYIFG	Reserved	SVMLVLRIFG <sup>(1)</sup>	SVMLIFG	SVSMLDLYIFG
r-0	rw-[0]	rw-[0]	rw-0	r-0	rw-[0]	rw-[0]	rw-0

<sup>(1)</sup> After power up, the reset value depends on the power sequence.

<sup>(1)</sup> After power up, the reset value depends on the power sequence.

**Table 2-21. PMMIFG Register Description**

Bit	Field	Type	Reset	Description
15	PMMILPM5IFG	RW	0h	LPMx.5 flag. This bit is set if the system was in LPMx.5 before. The bit is cleared by software or by reading the reset vector word. A power failure on the DVCC domain clears the bit. 0b = No interrupt pending 1b = Interrupt pending
14	Reserved	R	0h	Reserved. Always reads as 0.
13	SVSLIFG	RW	0h	SVS low-side interrupt flag. The bit is cleared by software or by reading the reset vector word. 0b = No interrupt pending 1b = Interrupt pending
12	SVSHIFG	RW	0h	SVS high-side interrupt flag. The bit is cleared by software or by reading the reset vector word. 0b = No interrupt pending 1b = Interrupt pending
11	Reserved	R	0h	Reserved. Always reads as 0.
10	PMMPORIFG	RW	0h	PMM software power-on reset interrupt flag. This interrupt flag is set if a software POR is triggered. The bit is cleared by software or by reading the reset vector word, SYSRSTIV. 0b = No interrupt pending 1b = Interrupt pending
9	PMMRSTIFG	RW	0h	PMM reset pin interrupt flag. This interrupt flag is set if the RST/NMI pin is the reset source. The bit is cleared by software or by reading the reset vector word. 0b = No interrupt pending 1b = Interrupt pending
8	PMMBORIFG	RW	0h	PMM software brownout reset interrupt flag. This interrupt flag is set if a software BOR (PMMSWBOR) is triggered. The bit is cleared by software or by reading the reset vector word, SYSRSTIV. 0b = No interrupt pending 1b = Interrupt pending
7	Reserved	R	0h	Reserved. Always reads as 0.
6	SVMHVLRIFG	RW	0h	SVM high-side voltage level reached interrupt flag. The bit is cleared by software or by reading the reset vector (SVSHPE = 1) word or by reading the interrupt vector (SVSHPE = 0) word. 0b = No interrupt pending 1b = Interrupt pending
5	SVMHIFG	RW	0h	SVM high-side interrupt flag. The bit is cleared by software. 0b = No interrupt pending 1b = Interrupt pending

**Table 2-21. PMMIFG Register Description (continued)**

Bit	Field	Type	Reset	Description
4	SVSMHDLYIFG	RW	0h	SVS and SVM high-side delay expired interrupt flag. This interrupt flag is set if the delay element expired. The bit is cleared by software or by reading the interrupt vector word. 0b = No interrupt pending 1b = Interrupt pending
3	Reserved	R	0h	Reserved. Always reads as 0.
2	SVMLVLIFG	RW	0h	SVM low-side voltage level reached interrupt flag. The bit is cleared by software or by reading the reset vector (SVSLPE = 1) word or by reading the interrupt vector (SVSLPE = 0) word. 0b = No interrupt pending 1b = Interrupt pending
1	SVMLIFG	RW	0h	SVM low-side interrupt flag. The bit is cleared by software. 0b = No interrupt pending 1b = Interrupt pending
0	SVSMLDLYIFG	RW	0h	SVS and SVM low-side delay expired interrupt flag. This interrupt flag is set if the delay element expired. The bit is cleared by software or by reading the interrupt vector word. 0b = No interrupt pending 1b = Interrupt pending

### 2.3.7 PMMRIE Register

Power Management Module Reset and Interrupt Enable Register

**Figure 2-15. PMMRIE Register**

15	14	13	12	11	10	9	8
Reserved	SVMHVRPE	SVSHPE		Reserved	SVMLVRPE	SVSLPE	
r-0	r-0	rw-[0]	rw-[1]	r-0	r-0	rw-[0]	rw-[1]
7	6	5	4	3	2	1	0
Reserved	SVMHVLRIE	SVMHIE	SVSMHDLYIE	Reserved	SVMLVLRIE	SVMLIE	SVSMLDLYIE
r-0	rw-0	rw-0	rw-0	r-0	rw-0	rw-0	rw-0

**Table 2-22. PMMRIE Register Description**

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0.
13	SVMHVRPE	RW	0h	SVM high-side voltage level reached power-on reset enable. If this bit is set, exceeding the SVMH voltage level triggers a POR.
12	SVSHPE	RW	1h	SVS high-side power-on reset enable. If this bit is set, falling below the SVSH voltage level triggers a POR.
11-10	Reserved	R	0h	Reserved. Always reads as 0.
9	SVMLVRPE	RW	0h	SVM low-side voltage level reached power-on reset enable. If this bit is set, exceeding the SVML voltage level triggers a POR.
8	SVSLPE	RW	1h	SVS low-side power-on reset enable. If this bit is set, falling below the SVSL voltage level triggers a POR.
7	Reserved	R	0h	Reserved. Always reads as 0.
6	SVMHVLRIE	RW	0h	SVM high-side reset voltage level interrupt enable
5	SVMHIE	RW	0h	SVM high-side interrupt enable. This bit is cleared by software or if the interrupt vector word is read.
4	SVSMHDLYIE	RW	0h	SVS and SVM high-side delay expired interrupt enable
3	Reserved	R	0h	Reserved. Always reads as 0.
2	SVMLVLRIE	RW	0h	SVM low-side reset voltage level interrupt enable
1	SVMLIE	RW	0h	SVM low-side interrupt enable. This bit is cleared by software or if the interrupt vector word is read.
0	SVSMLDLYIE	RW	0h	SVS and SVM low-side delay expired interrupt enable

### **2.3.8 PM5CTL0 Register**

Power Mode 5 Control Register 0

**Figure 2-16. 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							
r0	r0	r0	r0	r0	r0	r0	rw-[0]

**Table 2-23. PM5CTL0 Register Description**

Bit	Field	Type	Reset	Description
15-1	Reserved	R	0h	Reserved. Always reads as 0.
0	LOCKLPM5	RW	0h	<p>Lock I/O pin configuration upon entry to or exit from LPMx.5. When power is applied to the device, this bit, once set, can only be cleared by the user or via another power cycle.</p> <p>Note: This bit was formerly named LOCKIO, and some application reports and code examples may continue to use this terminology.</p> <p>0b = I/O pin configuration is not locked and defaults to its reset condition.</p> <p>1b = I/O pin configuration remains locked. Pin state is held during LPMx.5 entry and exit.</p>

## **Battery Backup System**

The battery backup system provides the possibility to operate a real-time clock (RTC\_B module) and retain some bytes in a backup RAM from a backup source when the primary supply fails. The battery backup system also includes a simple charging circuitry to charge capacitors connected to the backup supply. This chapter describes the battery backup system.

Topic	Page
3.1 <b>Battery Backup Introduction</b> .....	<b>125</b>
3.2 <b>Battery Backup Operation</b> .....	<b>125</b>
3.3 <b>Battery Backup Registers</b> .....	<b>128</b>

### 3.1 Battery Backup Introduction

Battery backup system features include:

- Automatic and manual switching to the backup supply
- Backup-supplied backup subsystem that can contain:
  - Backup-supplied real-time clock with 32-kHz crystal oscillator (see the [RTC\\_B chapter](#))
  - Backup-supplied backup RAM (see the [Backup RAM chapter](#))
- Resistive charger for backup capacitors

---

**NOTE: Operation without separate battery backup supply**

If there is no separate battery backup supply in the system, the VBAT pin **must** be connected to DVCC. In addition, it is recommended to disable the switching by setting the BAKDIS bit.

---

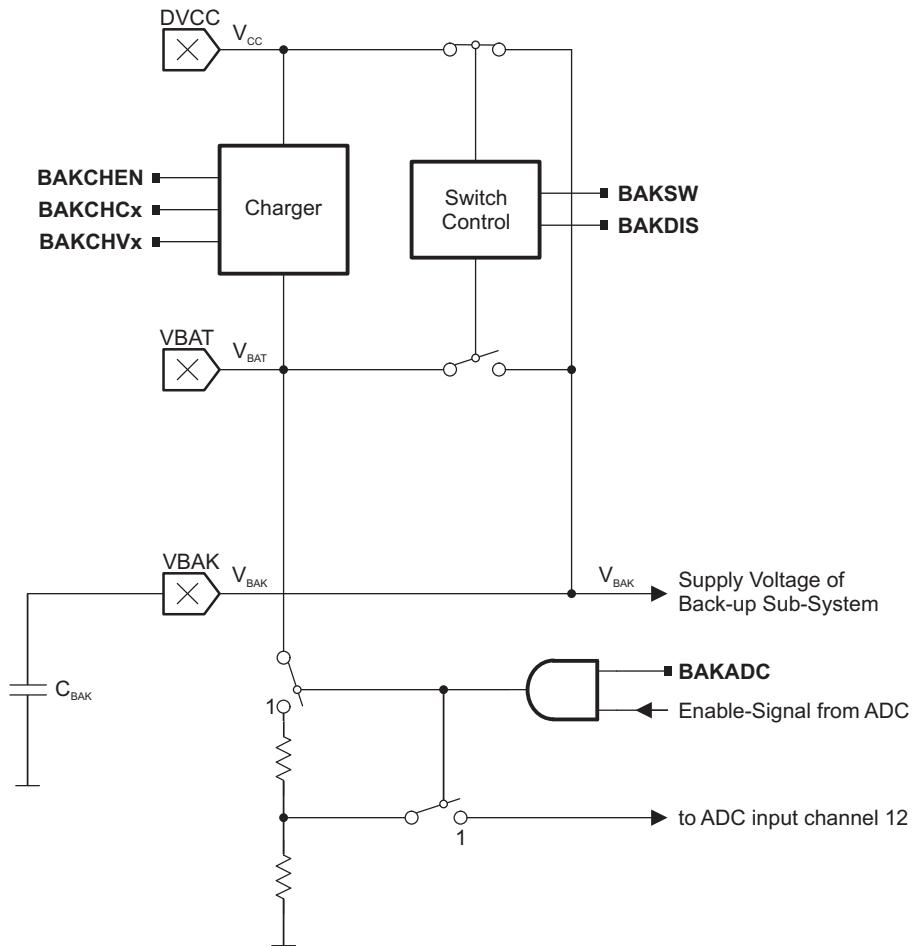
### 3.2 Battery Backup Operation

Upon switching to backup supply, the LOCKBAK bit is automatically set by the hardware. Usually (when not switching to backup supply manually) the part of the device that is not powered by the backup supply is unpowered and loses all configuration and data. The device starts with the BOR entry sequence when the primary supply is high enough again. All peripheral registers are set to their default conditions. The battery backup system remains locked, and the device can be configured. Also, the RTC registers are not retained during backup operation, and they must be restored. Then the LOCKBAK bit can be cleared, which releases potential pending backup input pin interrupt conditions and sets the corresponding port interrupt flags and the RTC\_B interrupt flags. Any pending backup input pin interrupt can be serviced.

If the RTC is enabled (RTCHOLD = 0), the 32-kHz oscillator remains active during backup operation. Also, the fault detection remains functional. If a fault occurs during backup operation, the RTCOFIFG flag is set after LOCKBAK is cleared, and the flag can be serviced.

### 3.2.1 Battery Backup Switch Control

Figure 3-1 shows an overview of the battery backup switch.



**Figure 3-1. Battery Backup Switch Overview**

The switch is controlled by the high-side SVS that supervises the primary supply, DVCC.

By default (for example, at power on) the backup subsystem is supplied from the backup battery (VBAT). As soon as primary supply is available and the high-side SVS releases the device, the backup supply is switched to the primary supply. When the primary supply falls below the configured high-side SVS level, the supply is switched back to the battery voltage VBAT. This means that the automatic switch control requires that the high-side SVS is always enabled (also in low-power modes). If the high-side SVS is disabled, the switch also selects the battery voltage VBAT to supply the backup subsystem.

---

**NOTE:**  $C_{BAK}$  shown in Figure 3-1 is used to ensure proper decoupling during a switchover event. Please see device specific datasheet for recommended values. This capacitance should not be confused with an external capacitor that may be placed on VBAT to maintain charge during backup operation in the application.

---

The LOCKBAK bit can be reset only when the supplies in the backup domain have settled. If for example a discharged capacitor is connected to VBAT, this might take a couple of milliseconds after DVCC is supplied. The recommended flow is to check if LOCKBAK reads 0 after having reset it by software.

**NOTE: Restrictions**

When the lowest high-side SVS level (00b) is used to monitor the primary supply, the temperature range is restricted to 0°C to 85°C.

In addition to the automatic switching based on the high-side SVS, it is possible to "manually" switch to battery backup supply by setting the BAKSW to 1. A POR resets BAKSW, and the system returns to automatic switch control.

The battery backup voltage can be measured if the device provides an ADC. In this case, the BAKADC bit must be set, and the ADC channel 12 must be selected. (This is the same ADC channel that can also be used to measure the supply voltage  $V_{CC}$ ; see the respective ADC chapter for details.) The resistive divider is connected to the battery only during the sampling phase of the ADC.

### 3.2.2 LPMx.5 and Backup Operation

During LPMx.5 (LPM3.5 or LPM4.5), the backup subsystem is always supplied from the backup battery, except when switching is completely disabled by setting the BAKDIS bit.

If using a capacitor to source the backup supply, the device can wake up regularly from LPMx.5, recharge the capacitor, and return to LPMx.5. The time interval must be designed such that the remaining charge on the capacitor is always sufficient to bridge the worst-case backup time (that is, the time without any primary supply).

### 3.2.3 Resistive Charger

Together with the battery backup switch, a resistive charging circuit is implemented to charge capacitors connected to the backup supply. A simplified block diagram of the charger is shown in [Figure 3-2](#). The charger is enabled by writing the correct password (069h) into the upper byte of BAKCHCTL, together with BAKCHEN = 1, selecting a charging resistor with  $BAKCHCx \neq 00b$  and a charge end voltage with  $BAKCHVx \neq 00b$ . Writing to the charger control register with an incorrect password disables the charger and all control register bits are reset to 0.

If  $V_{CC}$  is selected as charge end voltage with  $BACKCHVx = 01b$  (or if  $V_{CC} < 2.7$  V with  $BACKCHVx = 10b$ ), an attached capacitor is charged to  $V_{CC}$  with  $V_{BAT}(t) \approx V_{CC} \times (1 - \exp(-t/RC))$ , with R being the selected charging resistor and C being the capacitor attached to pin VBAT (this is **not**  $C_{BAK}$ ).

If a charge end voltage of 2.7 V is selected ( $BACKCHVx = 10b$ ) and  $V_{CC} > 2.7$  V, then an attached capacitor is charged with  $V_{BAT}(t) \approx V_{CC} \times (1 - \exp(-t/RC))$  (same as above) but as soon as  $V_{BAT}$  reaches approximately 2.7 V, the charging process is halted. If  $V_{BAT}$  drops by approximately 70 mV (the comparator hysteresis), the charging process continues again until the capacitor connected to pin VBAT is again charged to approximately 2.7 V. Note: For low power reasons, the  $V_{BAT}$  voltage is compared against the 2.7-V limit only once during each VLO clock cycle and only then is charging disabled or re-enabled.

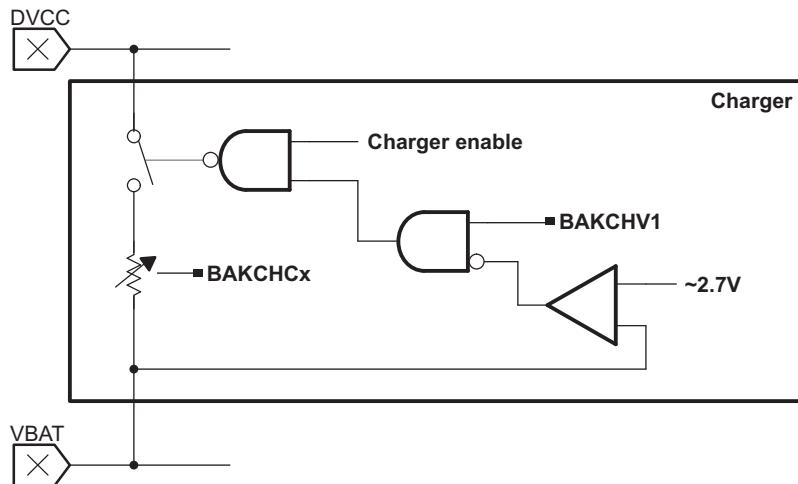


Figure 3-2. Charger Block Diagram

### 3.3 Battery Backup Registers

The battery backup registers are listed in [Table 3-1](#). The base address for the backup RAM registers can be found in the device-specific data sheet. The address offsets are given in [Table 3-1](#).

**Table 3-1. Battery Backup Registers**

Offset	Acronym	Register Name	Type	LPMx.5, Backup Retention	Section
00h	BAKCTL	Battery Backup Control	Read/write	not retained	<a href="#">Section 3.3.1</a>
02h	BAKCHCTL	Battery Charger Control	Read/write	not retained	<a href="#">Section 3.3.2</a>

### 3.3.1 BAKCTL Register

Battery Backup Control Register

**Figure 3-3. BAKCTL Register**

15	14	13	12	11	10	9	8
Reserved							
r	r	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved				BAKDIS	BAKADC	BAKSW	LOCKBAK
r0	r0	r0	r0	rw	rw-(0)	rw-(0)	r/w0-[1]

**Table 3-2. BAKCTL Register Description**

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved. Always reads as 0.
3	BAKDIS	RW	0h	Disable backup supply switching. Reset to 0 after a complete power cycle. 0b = Backup supply switching enabled 1b = Backup supply switching disabled. Backup subsystem always powered from VCC (also during LPMx.5).
2	BAKADC	RW	0h	Battery backup supply to ADC 0b = Vbat measurement disabled 1b = Vbat measurement enabled
1	BAKSW	RW	0h	Manual switch to battery backup supply 0b = Switching is automatic 1b = Switch to battery backup supply
0	LOCKBAK	RW	0h	Lock backup subsystem. Can only be written as 0. The LOCKBAK bit should only be written as 0 after configuring the RTC control registers. This ensures that RTC will not be stopped after leaving backup or LPMx.5 mode. SVSH has to be active when LOCKBAK bit is cleared. LOCKBAK is always set to 1 by hardware after the core was powered down either due to a complete power cycle of the main supply DVCC or due to LPMx.5 operation. 0b = Backup subsystem not locked 1b = Backup subsystem locked

### 3.3.2 BAKCHCTL Register

Battery Charger Control Register

**Figure 3-4. BAKCHCTL Register**

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

**Table 3-3. BAKCHCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	BAKCHKEYx	RW	5Ah	Charger access key. Always read as 05Ah. Must be written as 069h together with low byte; any other write disables the charger and all control register bits are reset to 0.
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	BAKCHVx	RW	0h	Charger end voltage 00b = Charger disabled 01b = VCC 10b = Approximately 2.7 V, or VCC if VCC is lower than 2.7 V 11b = Reserved
3	Reserved	R	0h	Reserved. Always reads as 0.
2-1	BAKCHCx	RW	0h	Charger charge current 00b = Charger disabled 01b = Charge current defined by a maximum 5-kΩ resistor 10b = Charge current defined by a maximum 10-kΩ resistor 11b = Charge current defined by a maximum 20-kΩ resistor
0	BAKCHEN	RW	0h	Charger enable 0b = Charger disabled 1b = Charger enabled

## **Auxiliary Supply System (AUX)**

The auxiliary supply system (AUX) allows the device to operate from alternate supplies (also called auxiliary supplies) if the primary supply (DVCC and AVCC) fails. The AUX includes simple charging circuitry to charge capacitors connected to the auxiliary supplies. This chapter describes the AUX.

<b>Topic</b>	<b>Page</b>
<b>4.1 Auxiliary Supply System Introduction .....</b>	<b>132</b>
<b>4.2 Auxiliary Supply Operation .....</b>	<b>133</b>
<b>4.3 AUX Registers .....</b>	<b>147</b>

## 4.1 Auxiliary Supply System Introduction

The auxiliary supply system features include:

- Automatic or manual switching from the primary supply to an auxiliary supply while maintaining full functionality.
- One or two auxiliary supplies (AUXVCC1 and AUXVCC2), depending on the specific device.
- Automatic threshold-based monitoring of primary and auxiliary supplies.
- At startup, automatically chooses between the primary supply (DVCC/AVCC) and AUXVCC1, based on which one is higher voltage.
- A separate auxiliary supply (AUXVCC3) can power a backup subsystem <sup>(1)</sup>.
- Simple charger for capacitors on AUXVCC2 and AUXVCC3.

---

**NOTE: Unused auxiliary supplies**

Any unused auxiliary supply inputs (AUXVCC1, AUXVCC2, or AUXVCC3) must be connected to DVSS.

If AUXVCC1 or AUXVCC2 are unused, their usage should be disabled by setting AUXxMD = 1 and AUXxOK = 0 in software, too.

---

<sup>(1)</sup> The backup subsystem usually contains a real-time clock (RTC) module with a 32-kHz crystal oscillator, backup RAM, and optionally (device-specific) up to two digital I/O pins.

## 4.2 Auxiliary Supply Operation

The AUX module allows the device to switch between the primary supply (DVCC and AVCC) and up to two auxiliary supplies (AUXVCC1 and AUXVCC2) while maintaining full device functionality. When using an auxiliary supply, both AVCC and DVCC are switched to the same auxiliary supply.

Switching can be controlled automatically through hardware or manually through software. In the hardware-controlled mode, switching is triggered by the high-side supply voltage monitor (SVM), which must be enabled and configured as described in [Section 4.2.4](#). In the software-controlled mode, switching is triggered by changing values in AUX module registers, as described in [Section 4.2.3](#).

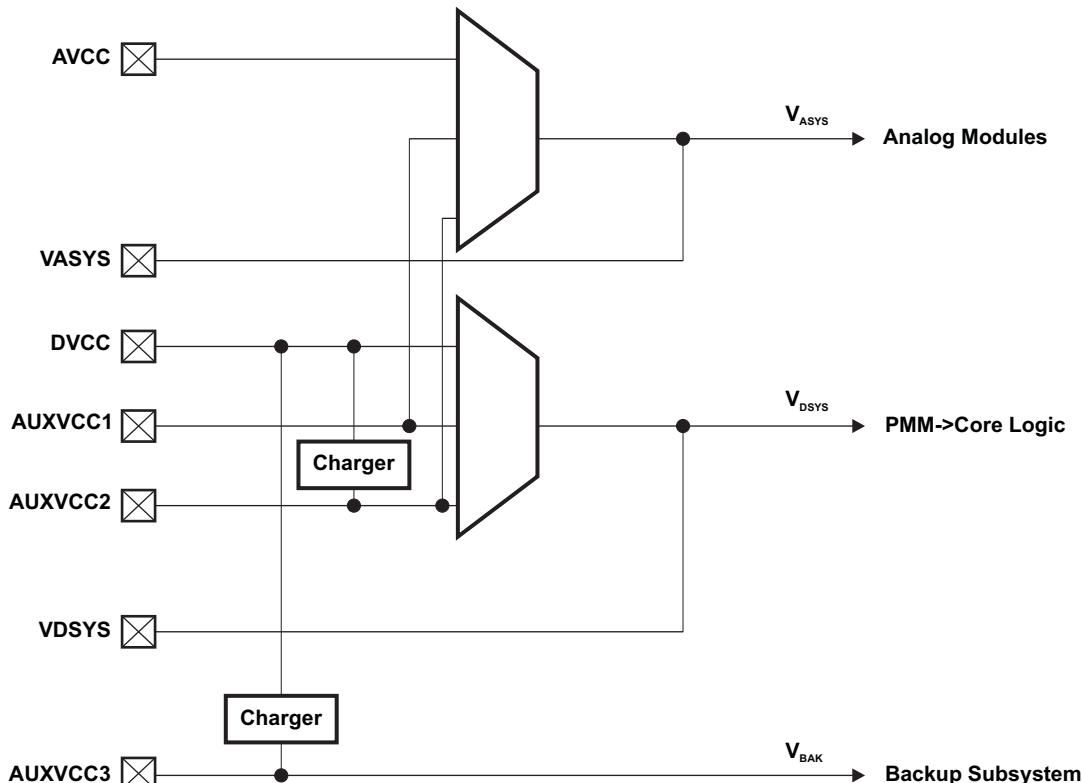
[Figure 4-1](#) shows an overview of the auxiliary supply switches.

The digital core of the device is supplied through the Power Management Module (PMM) by the internal digital system voltage  $V_{DSYS}$ . In the PMM chapter of this user's guide, that supply is assumed to be DVCC; however, on devices with the AUX module, AUX switches  $V_{DSYS}$  among the DVCC, AUXVCC1, and AUXVCC2 inputs.  $V_{DSYS}$  is also output on the VDSYS pin, which must be connected to an external capacitor as specified in the device-specific data sheet.

The analog modules of the device are supplied by the internal analog system voltage  $V_{ASYS}$ . The AUX module switches  $V_{ASYS}$  among the AVCC, AUXVCC1, and AUXVCC2 inputs.  $V_{ASYS}$  is also output on the VASYS pin, which must be connected to an external capacitor as specified in the device-specific data sheet.

The switches for the digital and the analog system voltages are controlled by the same signals and always connect to the same voltage (primary voltage DVCC/AVCC, first auxiliary voltage AUXVCC1, or second auxiliary voltage AUXVCC2).

Auxiliary voltage AUXVCC3 supplies  $V_{BAK}$  to the backup subsystem, which usually contains a real-time clock (RTC) module with a 32-kHz crystal oscillator, some backup RAM, and optionally (device-specific) up to two digital I/O pins.



**Figure 4-1. Auxiliary Supply Switch Overview**

#### 4.2.1 Startup

The device starts whenever a supply is connected to DVCC or AUXVCC1. If both supplies are connected, AUX uses whichever voltage is higher to supply the digital and analog system voltages,  $V_{DSYS}$  and  $V_{ASYS}$ . If the supplies (DVCC/AVCC and AUXVCC1) differ by less than 100mV the selected supply can be either DVCC/AVCC or AUXVCC1 - only if the difference is greater than 100mV always the supply with the higher voltage is selected.

After startup from any condition that causes a BOR event (including connection of power or wake from LPMx.5), the AUX module is automatically locked (that is, the LOCKAUX bit is set). The behavior of the auxiliary supply system can then be configured in software (see [Section 4.2.2](#)).

After configuration is completed, unlock the AUX module by clearing the LOCKAUX bit. The AUX module is then operational and switches between the input supplies as defined during configuration.

If the LOCKAUX bit remains set, the programmed configuration is ignored, and the device continues to be supplied by DVCC or AUXVCC1 (whichever was selected at startup).

[Section 4.2.7](#) describes additional considerations for setting the power supply before entering LPMx.5.

---

**NOTE: Highest Supply Voltage**

To ensure reliable operation, the selected supply voltage (by default, the highest voltage in the system) must always supply at least 0.5  $\mu$ A to critical circuitry.

---

**NOTE: DVCC vs AUXVCC1 at Startup**

Under normal operating conditions it is recommended to have DVCC supplied by a voltage at least 100mV higher than AUXVCC1. This ensures a reliable startup from DVCC and avoids an unwanted startup from AUXVCC1.

---

#### 4.2.2 Switching Control

During normal operation (that is, when the LOCKAUX bit is cleared) switching to another supply is triggered either by software or by hardware (specifically, by the high-side SVM). [Section 4.2.3](#) describes the configuration and control of the AUX for software-controlled switching. [Section 4.2.4](#) describes configuration and control of the AUX for hardware-controlled switching. [Section 4.2.13](#) includes examples of how to configure hardware-controlled mode based on different usage scenarios.

#### 4.2.3 Software-Controlled Switching

To enable or disable a supply using software-controlled switching:

1. To enable software control of a supply, set  $AUXxMD = 1$  ( $AUX0MD$  for DVCC,  $AUX1MD$  for  $AUXVCC1$ , and  $AUX2MD$  for  $AUXVCC2$ ).
2. To select the supply to use, set  $AUXxOK = 1$  ( $AUX0OK$  for DVCC,  $AUX1OK$  for  $AUXVCC1$ , and  $AUX2OK$  for  $AUXVCC2$ ).

When  $AUXxOK$  is set, the AUX module immediately switches to the specified supply.

If  $AUXxOK = 1$  for more than one supply, AUX uses the one with the highest priority. The default priority is DVCC, then  $AUXVCC1$ , then  $AUXVCC2$ . To make  $AUXVCC2$  priority higher than  $AUXVCC1$ , set  $AUX2PRIO = 1$ .

3. To disable a supply, clear  $AUXxOK = 0$ . If the current supply is software-controlled ( $AUXxMD = 1$ ) and  $AUXxOK$  is changed from 1 to 0, the next available supply (considering the priority defined by  $AUX2PRIO$ ) is used to source the system voltages.

When a switch from one supply to another occurs, interrupts are generated as described in [Section 4.2.11](#).

The software control can be used to permanently disable a supply or, for example, to qualify the quality of the supplies by measuring the actual supply voltage with an ADC (see [Section 4.2.9](#)) instead of using the auxiliary supply monitor (see [Section 4.2.6](#)).

When using software-controlled mode for all supplies, the SVS and SVM in the PMM are not required. They may be enabled, and they will operate as described in the PMM chapter, but they do not interact with the AUX during software-controlled mode.

#### 4.2.4 Hardware-Controlled Switching

To enable hardware-controlled switching, clear  $AUXxMD = 0$  ( $AUX0MD$  for DVCC,  $AUX1MD$  for  $AUXVCC1$ , and  $AUX2MD$  for  $AUXVCC2$ ). A supply can be disabled and, therefore, excluded from the automatic switching system by setting the corresponding  $AUXxMD = 1$  and  $AUXxOK = 0$ . For example, to disable  $AUXVCC2$ , set  $AUX2MD = 1$  and  $AUX2OK = 0$ .

During hardware-controlled switching, the SVM in the PMM must be enabled and configured. The SVS can also be enabled and operates as described in the PMM chapter, but it does not interact with the AUX module.

The high-side SVM of the PMM monitors  $V_{DSYS}$ , which is output from the AUX module. If  $V_{DSYS}$  falls below the voltage set by the  $SVSMHRRVL$  bits, the SVM notifies AUX to switch to the next valid supply. AUX switches to the next valid supply immediately after receiving the trigger from the SVM. When a switch occurs, interrupts are generated as described in [Section 4.2.11](#).

---

**NOTE: Voltage Dip on  $V_{DSYS}$**

Because the SVM does not signal AUX to change the supply until the voltage on  $V_{DSYS}$  falls to the SVM monitoring level (set in the  $SVSMHRRVL$  bits), there is a dip in  $V_{DSYS}$  from the nominal operating level. This change in voltage must be considered when selecting system frequency and core voltage (see [Section 4.2.5](#)).

---

AUX determines which supplies are valid based on the threshold voltage set in the  $AUXxLVL$  bits. The validity of a supply is reported by the corresponding  $AUXxOK$  bit. When  $AUXxMD = 0$ , the  $AUXxOK$  bit is controlled by hardware and cannot be written by software.  $AUXxOK = 1$  indicates a valid/"good" supply voltage, and  $AUXxOK = 0$  indicates an invalid/"bad" supply voltage. See [Section 4.2.6](#) for details on the monitoring of the auxiliary supplies.

---

**NOTE: Interactions Among  $SVM_H$ ,  $V_{CORE}$ , and AUX**

Because of the relationship between supply voltage, core voltage, and maximum system frequency, see [Section 4.2.5](#) for considerations when setting the  $AUXxLVL$  and  $SVM_H$  levels.

In particular, note that  $AUX0LVL$  must be higher than the  $SVM_H$  level.

---

The selection of the next valid supply depends on the state of the  $AUXxOK$  and  $AUX2PRIO$  bits when the trigger occurs as shown in [Table 4-1](#). If there is no valid supply available to switch to (that is, all other  $AUXxOK$  bits are 0) no switching takes place, and the device eventually goes into reset if the current supply continues to fall. To avoid rapid switching back and forth between supplies, any further switching is prevented during a "recovery time" of several 100  $\mu$ s as specified in the device-specific data sheet after each switch-over.

**Table 4-1. Next Supply Voltage Selection**

Current Supply	Next Supply <sup>(1)</sup>	DVCC OK? ( $AUX0OK$ )	AUXVCC1 OK? ( $AUX1OK$ )	AUXVCC2 OK? ( $AUX2OK$ )	AUXVCC1/ AUXVCC2 Priority ( $AUX2PRIO$ )
DVCC/AVCC	AUXVCC1	don't care	1	don't care	0
		don't care	1	0	1
	AUXVCC2	don't care	0	1	don't care
		don't care	1	1	1
AUXVCC1	DVCC	1	don't care	don't care	don't care
	AUXVCC2	0	don't care	1	don't care

<sup>(1)</sup> If there is no valid supply available to switch to, no switching takes place.

**Table 4-1. Next Supply Voltage Selection (continued)**

Current Supply	Next Supply <sup>(1)</sup>	DVCC OK? (AUX0OK)	AUXVCC1 OK? (AUX1OK)	AUXVCC2 OK? (AUX2OK)	AUXVCC1/ AUXVCC2 Priority (AUX2PRIO)
AUXVCC2	DVCC	1	don't care	don't care	don't care
	AUXVCC1	0	1	don't care	don't care

**NOTE: Special Case for Switching to DVCC/AVCC**

If the device is supplied by AUXVCC1 or AUXVCC2, and the AUX0OK bit transitions from 0 to 1 (either by hardware or by software), AUX switches to the DVCC/AVCC supply without any signal from the SVM.

AUX does not automatically switch from AUX2VCC to AUX1VCC when AUX1OK transitions from 0 to 1, unless the SVM signals that a switch is necessary.

**4.2.5 Interactions Among  $f_{SYS}$ ,  $V_{CORE}$ ,  $V_{DSYS}$ ,  $SVM_H$ , and  $AUXxLVL$** 

The interactions that must be considered when setting the threshold levels in the SVM and AUX are:

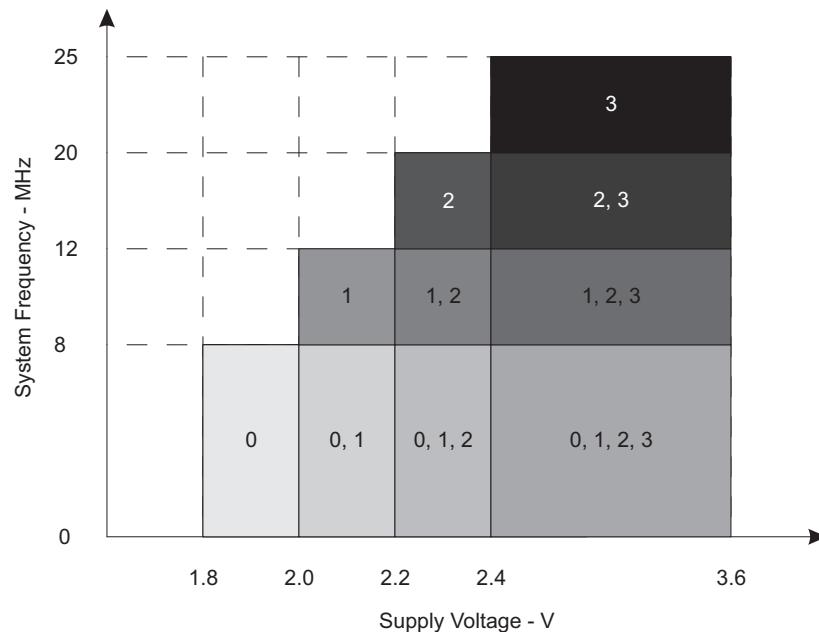
- Minimum  $V_{CORE}$  required to support the selected system frequency ( $f_{SYS}$ )
- Valid  $SVM_H$  to support the selected  $V_{CORE}$
- Minimum  $V_{DSYS}$  and minimum  $AUXxLVL$  required to support the selected  $V_{CORE}$

The interactions among  $f_{SYS}$ , DVCC ( $V_{DSYS}$  for devices with AUX),  $V_{CORE}$ , and  $SVM_H$  are described in detail in the PMM chapter. This section adds considerations for the valid  $AUXxLVL$  values.

**NOTE: Maximum System Frequency**

The following discussion describes all system frequencies supported in this family. However, the maximum system frequency varies by device; therefore, see the device-specific data sheet to determine this value.

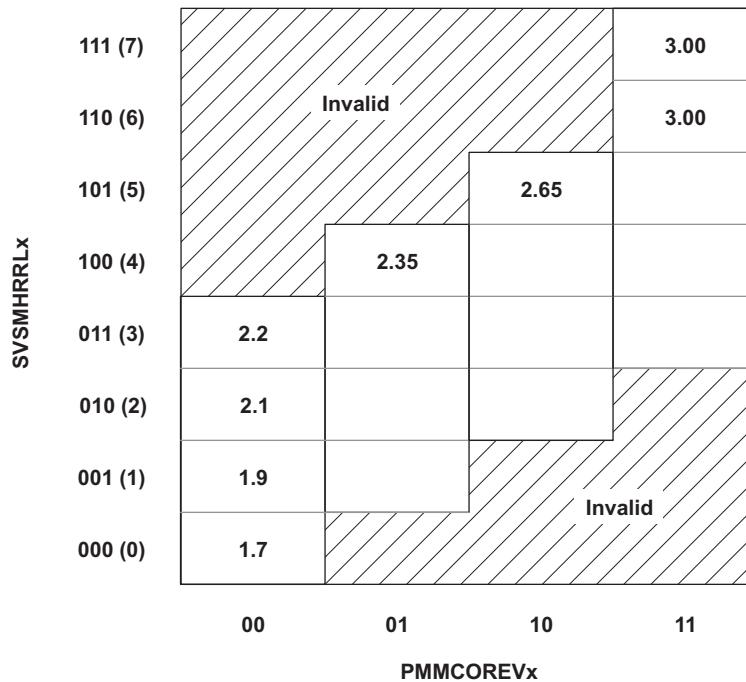
Figure 4-2 shows typical requirements for supply voltage and  $V_{CORE}$  compared to the system frequency (see the device-specific data sheet for the values required for each device). As shown here, there is a minimum  $V_{CORE}$  (set by PMMCOREV[1:0]) and a minimum supply voltage ( $V_{DSYS}$ ) for each system frequency. For details on the recommended settings for PMMCOREV[1:0] and supply voltage, see Section 2.2.2.1.



The numbers within the fields denote the supported PMMCOREVx settings.

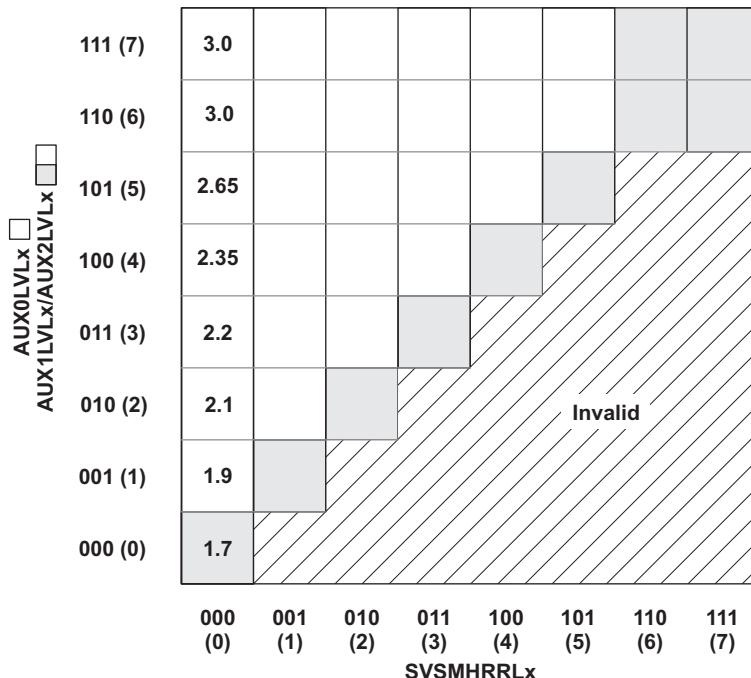
**Figure 4-2. System Frequency vs Supply Voltage**

After selecting the system frequency and PMMCOREV[1:0] values, the SVM threshold must be selected. [Figure 4-3](#) shows the valid values for  $SVM_H$  ( $SVSMHRRRL[2:0]$ ) based on the selected  $V_{CORE}$  (PMMCOREV[1:0]). This information is also included in [Section 2.2.2.1](#).



**Figure 4-3. Available  $SVM_H$  Settings vs  $V_{CORE}$  Settings**

Based on the preceding selections, the auxiliary supply threshold levels must be set. These thresholds are the minimum supply voltage that are available from each auxiliary supply. [Figure 4-4](#) shows the valid AUXxLVL settings compared to the selected SVM setting. The level setting for DVCC/AVCC AUX0LVL must be chosen at least one step above the SVSMHRRRLx level to avoid any unwanted switching between DVCC/AVCC and AUXVCC1 or AUXVCC2.



**Figure 4-4. Available AUXxLVL Settings vs SVM<sub>H</sub> Settings**

[Table 4-2](#) combines the information from the preceding figures and discussions.

**Table 4-2. Minimum Voltage Thresholds for Selected f<sub>SYS</sub>**

f <sub>SYS</sub> (max) (MHz)	Minimum PMMCOREV[1:0]	Minimum SVSMHRRRL[2:0] (Sets SVM <sub>H</sub> Level)	Minimum V <sub>DSYS</sub>	Minimum AUX0LVL	Minimum AUX1LVL, AUX2LVL
8	00	000	1.8 V	001	000
12	01	001	2.0 V	010	001
20	10	010	2.2 V	011	010
25	11	011	2.4 V	100	011

#### 4.2.6 Auxiliary Supply Monitor

Supplies that are not currently in use and that are not software controlled (AUXxMD = 0) are monitored using a low-power comparator. For example, if DVCC supplies the device and the AUXxMD bits are 0 for both AUXVCC1 and AUXVCC2, then AUXVCC1 and AUXVCC2 are monitored. As another example, if AUXVCC1 supplies the device and AUX2MD = 1, only DVCC is monitored. If all unselected supplies are controlled by software (AUXxMD = 1), then the automatic monitoring is disabled. The level the supply voltages are compared against is programmable with the AUXxLVL bits. If a supply drops below the selected threshold level the corresponding AUXxDRPIFG is set. [Figure 4-5](#) shows a principle block diagram of the monitoring circuitry.

If two supplies are monitored this happens in a time-division multiplexing scheme clocked by the VLO. During one VLO clock period, one supply is compared against its AUXxLVL threshold; during the next VLO clock period, the other supply is compared against its threshold. This means the AUXxOK bits of these supplies are updated approximately every 300  $\mu$ s (worst case) with VLO clock frequency of approximately 6 kHz. If only one supply is monitored, the output of the comparator is sampled using the VLO clock, so that an update of the corresponding AUXxOK bit occurs approximately every 150  $\mu$ s (worst case).

Each monitoring cycle uses some charge from the monitored supply. The discharge of the monitored supplies can be reduced by decreasing the monitoring rate by changing the AUXMRx bits. By default (with AUXMRx = 00b) the supplies are monitored continuously as described above. By setting AUXMRx = 01b, the supplies are monitored every 32 VLO clock cycles; by setting AUXMRx = 10b, the supplies are monitored every 1024 VLO clock cycles. The AUXMONIFG signals the completion of each monitoring cycle.

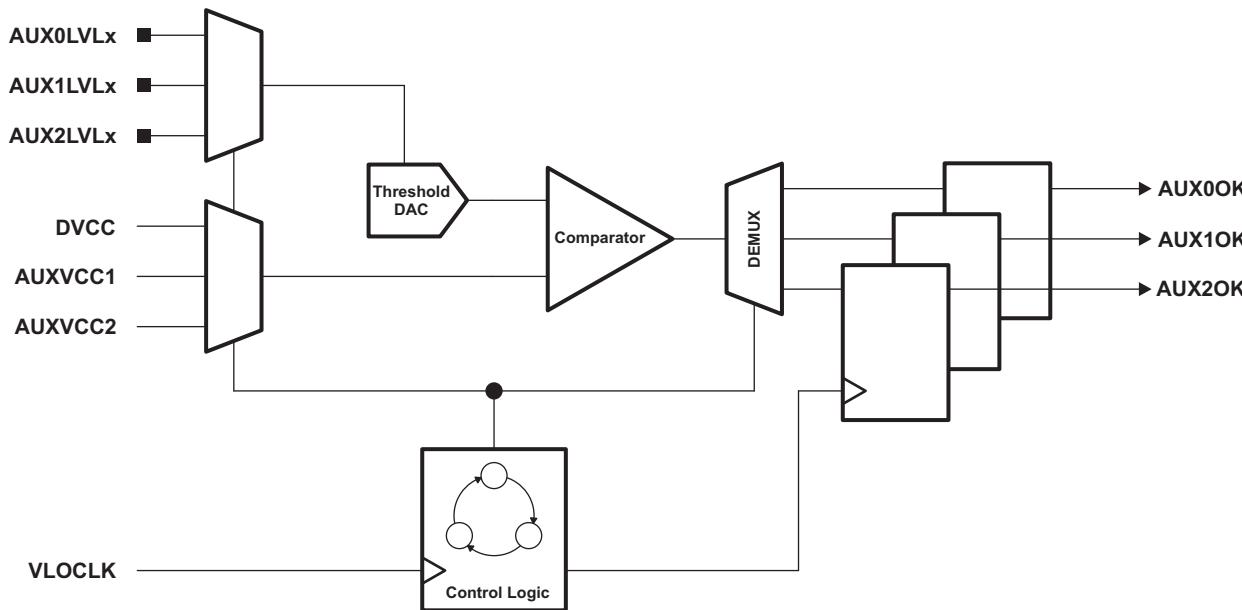
If an unused supply is changed from software to hardware control (AUXxMD is changed from 1 to 0) a new monitoring cycle is started with the next VLO clock cycle. Switching of the supplies does not affect the timing of the monitoring cycles, it only changes which supplies are monitored.

The auxiliary supply monitor is enabled after clearing the LOCKAUX bit unless all unused supplies are controlled by software. The monitored supplies are considered "not okay" until the first monitoring cycle completes.

---

**NOTE: Switching and Monitoring**

Switching is independent from the update interval of the AUXxOK bits. Switching is triggered either by the high-side SVM or by software and takes place immediate after the trigger occurs, taking the current AUXxOK states into account to select the next supply. Only a change of AUX0OK from 0 to 1 indicating the DVCC changed from a "not okay" to an "okay" state triggers a switch back to DVCC.



**Figure 4-5. Auxiliary Supply Monitor Block Diagram**

#### 4.2.7 LPMx.5 and Auxiliary Supply Operation

During LPMx.5, the device is powered by whichever input supply was active when entering the low-power mode. To force the device to use a specific supply during LPMx.5, activate that supply before entering LPMx.5. [Table 4-3](#) lists the supply selection during LPMx.5 based on the disabled supplies. The automatic threshold-based supply monitoring scheme is always disabled during LPMx.5.

**Table 4-3. Supply Selection During LPMx.5**

Supply During LPMx.5	DVCC/AVCC Status <sup>(1)</sup>	AUXVCC1 Status <sup>(2)</sup>
DVCC/AVCC or AUXVCC1 (whichever is higher <sup>(3)</sup> )	not disabled	not disabled
DVCC/AVCC	not disabled	disabled
AUXVCC1	disabled	not disabled
AUXVCC2	disabled	disabled

<sup>(1)</sup> DVCC/AVCC is disabled if AUX0MD = 1 and AUX0OK = 0; otherwise it is enabled.

<sup>(2)</sup> AUXVCC1 is disabled if AUX1MD = 1 and AUX1OK = 0; otherwise it is enabled.

<sup>(3)</sup> If the supplies (DVCC/AVCC and AUXVCC1) differ by less than 100mV the selected supply can be either DVCC/AVCC or AUXVCC1 - only if the difference is greater than 100mV always the supply with the higher voltage is selected.

After wake up from LPMx.5, the LOCKAUX bit is set. The LPMx.5 supply selection remains active until the LOCKAUX bit is cleared. When the LOCKAUX bit is cleared, the auxiliary supply system is controlled as defined by the control register settings that were configured before clearing the LOCKAUX bit. None of the AUX registers are retained during LPMx.5; thus, all registers must be reconfigured after wake-up from LPMx.5 and prior to releasing LOCKAUX.

The supplies monitored by hardware (AUXxMD = 0) are considered "not okay" until the status is updated for the first time by the auxiliary supply monitor. Only the supply that was used during LPMx.5 is considered "okay" unless it drops below the programmed SVM level. If this behavior is unwanted, the state can be set to "okay" by temporarily switching to software mode (AUXxMD = 1), setting the AUXxOK bit to 1, and then return to hardware mode (AUXxMD = 0). The last state defined in software mode is retained when switching back to hardware mode until the first update occurs by the auxiliary supply monitor.

The supplies controlled by software (AUXxMD = 1) are considered "not okay" or "okay" according to their AUXxOK setting. If the supply that was used during LPMx.5 is set to "not okay" (AUXxMD = 1 and AUXxOK = 0) the auxiliary supply system tries to switch to another ("okay") supply according to [Table 4-1](#) as soon as LOCKAUX is cleared.

---

**NOTE: DVCC vs AUXVCC1 during LPMx.5**

Under normal operating conditions it is recommended to have DVCC supplied by a voltage at least 100mV higher than AUXVCC1 if automatic switching between DVCC and AUXVCC1 is active i.e. both supplies are not disabled when entering LPMx.5 (see [Table 4-3](#)). This ensures a reliable operation from DVCC and avoids a potentially unwanted operation from AUXVCC1.

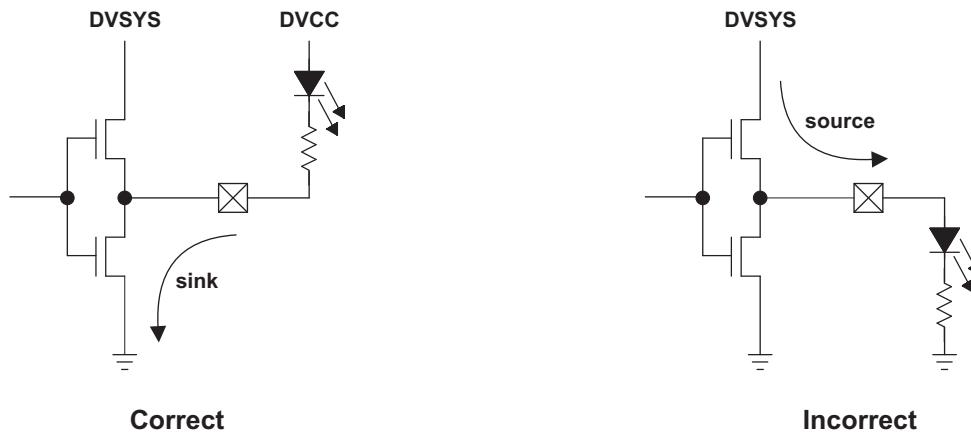
---

#### 4.2.8 Digital I/Os and Auxiliary Supplies

In most devices that implement the auxiliary supply system, the digital I/Os can be powered by the switched supplies. In this case, care must be taken that large currents are sunk into DVSS and not sourced from the switched supplies, because of the voltage drop across the supply switches as shown in [Figure 4-6](#).

**CAUTION**

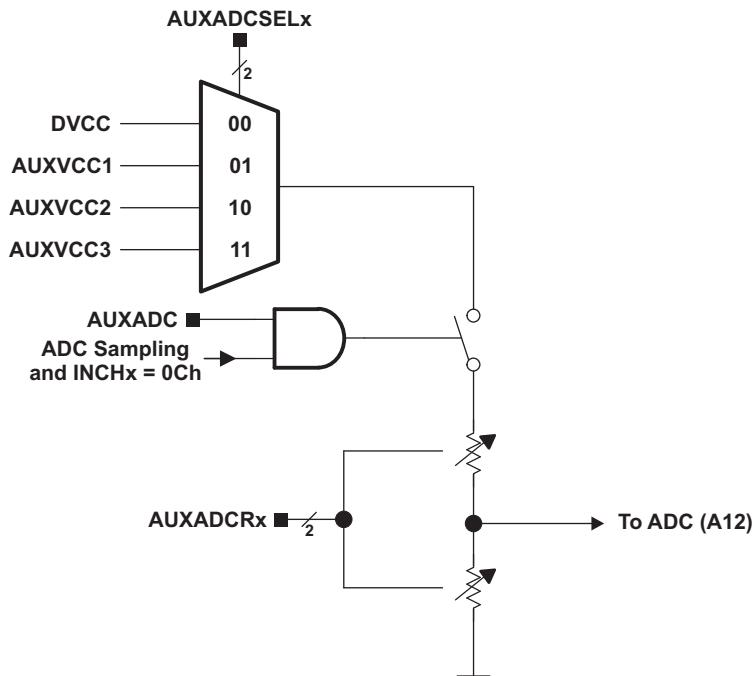
Check connection of external loads at digital I/Os to avoid high voltage drops across switches. This might cause unwanted resets.



**Figure 4-6. I/Os Powered by Auxiliary Supplies**

#### 4.2.9 Measuring the Supplies

The primary and all auxiliary supply voltages can be measured if the device provides an ADC. The supply to be measured is selected with the AUXADCSELx bit in the AUXADCCTL register. In addition the resistive load applied to the selected supply during the sampling phase of the ADC can be selected with the AUXADCRx bits. This allows to perform a "health" check of the supplies even when not loaded by the application. The ADC supply voltage channel (usually channel 12 (0Ch)) must be selected and the auxiliary supply voltage measurement must be enabled with AUXADC = 1. The resistive divider is connected to the supplies only during the sampling phase of the ADC.

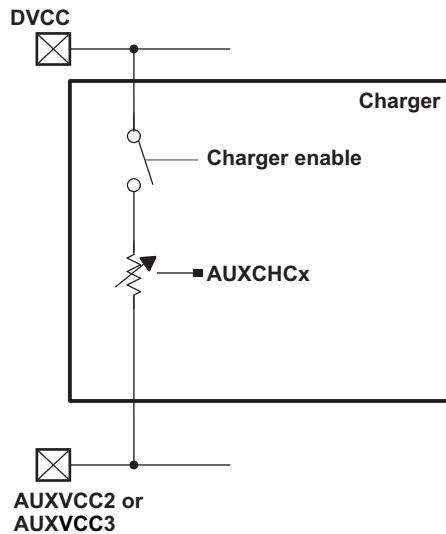


**Figure 4-7. AUX Connection to ADC**

#### 4.2.10 Resistive Charger

Two simple resistive charging circuits are implemented to be able to charge capacitors connected to AUXVCC2 and AUXVCC3. A simplified block diagram of the charger is shown in [Figure 4-8](#). The charger for AUXVCC2 or AUXVCC3 is enabled by writing the correct key (069h) into the upper byte of AUX2CHCTL or AUX3CHCTL, respectively, together with AUXCHVx=01b, AUXCHEN=1 and selecting a charging resistor with AUXCHCx#00b. Writing to the charger control register with an incorrect key disables the charger and all control register bits are reset to 0.

Both chargers will be disabled when DVCC is not selected as supply source and AUXCHEN is reset by hardware.



**Figure 4-8. Charger Block Diagram**

#### 4.2.11 Auxiliary Supply Interrupts

The auxiliary supply system provides 7 interrupt sources:

- AUXSWNMIFG: (non-)maskable supplies switched interrupt
- AUX0SWIFG: switched to DVCC interrupt
- AUX1SWIFG: switched to AUXVCC1 interrupt
- AUX2SWIFG: switched to AUXVCC2 interrupt
- AUX1DRPIFG: AUXVCC1 dropped below threshold interrupt
- AUX2DRPIFG: AUXVCC2 dropped below threshold interrupt
- AUXMONIFG: supply monitor interrupt

The AUXSWNMIFG is set after the system switched from one supply to another supply. A non-maskable interrupt request is generated if the AUXSWNMIE bit is set - otherwise if (only) the AUXSWGIE bit and additionally the GIE bit are set a maskable interrupt request is generated.

The AUXxSWIFG bits are set if the auxiliary supply system switched to the corresponding supply (DVCC, AUXVCC1 or AUXVCC2). This information can be used in the interrupt service routine together with the interrupt vector generator AUXIV to re-configure the device to the current/new supply situation. A (maskable) interrupt request is generated if the corresponding AUXxSWIE bit and the GIE bit are set.

The AUXxDRPIFG bits are set if the corresponding supplies AUXxOK state changes from 1 to 0 due to the supply voltage dropping below the selected threshold value AUXxLVL with the hardware monitor being enabled (AUXxMD = 0). A (maskable) interrupt request is generated if the corresponding AUXxDRPIE bit and the GIE bit are set.

The AUXMONIFG bit is set if a hardware monitoring cycle is completed and the supply AUXxOK states are updated accordingly. A (maskable) interrupt request is generated if the corresponding AUXMONIE bit and the GIE bit are set.

#### 4.2.11.1 AUXIV, Interrupt Vector Generator

All (maskable) auxiliary supply system interrupt sources are prioritized and combined to source a single interrupt vector. AUXIV is used to determine which enabled interrupt source requested an interrupt. The highest priority interrupt request that is enabled generates a number in the AUXIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled interrupts do not affect the AUXIV value.

Any read access of the AUXIV register automatically resets the highest pending interrupt flag. A write access to the AUXIV register automatically resets all pending interrupt flags. All interrupt flags can also be cleared via software.

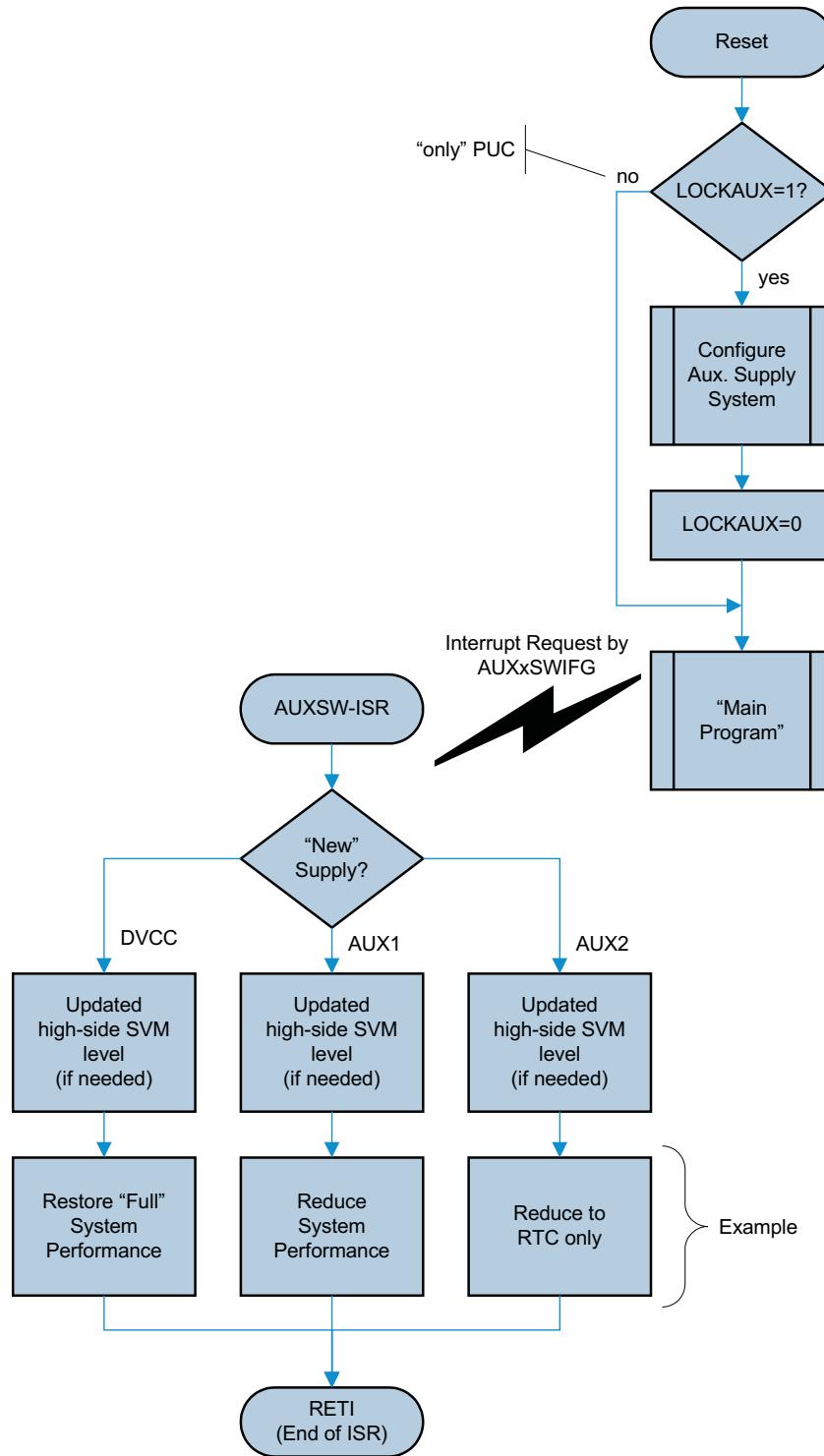
#### 4.2.11.2 Auxiliary Supply Non-Maskable Interrupt

If AUXSWNMIFG is configured as non-maskable interrupt source (with AUXSWNMIE=1) it will source (together with interrupt flags from other modules) the user-NMI interrupt vector. In the user-NMI interrupt service routine the SYSUNIV interrupt vector word register can be evaluated or added to the program counter to automatically enter the appropriate part of the user-NMI interrupt service routine. Refer to device-specific data sheet concerning the user-NMI interrupt vector sources and priorities.

If both AUXSWNMIE and AUXSWGIE are set always the non-maskable interrupt service routine will be called when AUXSWNMIFG is set because the user-NMI has a higher priority. In this case both interrupt vector generators, the SYSUNIV and the AUXIV, will indicate a pending AUXSWNMIFG.

#### 4.2.12 Software Flow

Figure 4-9 shows a sample software flow chart for the use and control of the auxiliary supply system.



NOTE: Configuration of the auxiliary supply system is required after wake-up from LPMx.5. None of the AUX registers are retained during LPMx.5.

**Figure 4-9. Software Flow Chart**

#### 4.2.13 Examples of AUX Operation

The following sections show sample configurations of the AUX module. Some settings in these examples do not apply to all devices; see the device-specific data sheet.

##### 4.2.13.1 Example 1

This example shows configuration for running at 25-MHz system frequency at a supply voltage of  $3\text{ V} \pm 0.3\text{ V}$ , maintaining that frequency when switching the AUX1 or AUX2.

The maximum  $\text{SVM}_H$  level must be less than or equal to 2.7 V, so that switching occurs only when  $V_{\text{DSYS}}$  goes outside  $3\text{ V} \pm 0.3\text{ V}$ . To switch back to DVCC/AVCC as soon as possible when it is within  $3\text{ V} \pm 0.3\text{ V}$ , the maximum AUX0LVL must also be less than or equal to 2.7 V. Because AUX0LVL must be 1 higher than SVSMHRRL, set = 4 and SVSMHRRL = 3.

To support 25-MHz operation, PMMCOREV must equal 3. With SVSMHRRL = 3, all core voltage settings are supported, so this is valid. Based on this core voltage setting, configure SVS levels (see for details).

To continue operation at 25-MHz without glitches even when using AUX1 and AUX2, the thresholds for these must be equal to or higher than SVSMHRRL. To minimize use of the auxiliary supplies, they should be set equal to SVSMHRRL. Therefore, set AUX1LVL and AUX2LVL = 3.

##### 4.2.13.2 Example 2

This example shows configuration for running at a supply voltage of  $3.3\text{ V} \pm 0.3\text{ V}$  and allowing for switching supplies without functional glitches.

The maximum  $\text{SVM}_H$  level must be less than or equal to 3 V, so that switching occurs only when  $V_{\text{DSYS}}$  goes outside  $3.3\text{ V} \pm 0.3\text{ V}$ . To switch back to DVCC/AVCC as soon as possible when it is within  $3.3\text{ V} \pm 0.3\text{ V}$ , the maximum AUX0LVL must also be less than or equal to 3 V. Because AUX0LVL must be 1 higher than SVSMHRRL, set = 6 and SVSMHRRL = 5.

With SVSMHRRL = 5, the core voltage setting (PMMCOREV) must be 2 or 3. Assuming the application requires a system frequency up to 20 MHz, set PMMCOREV = 2. Based on this core voltage setting, configure SVS levels (see for details).

To continue operation without glitches when switching to AUX1 and AUX2, the thresholds for these must be equal to or higher than SVSMHRRL. To minimize use of the auxiliary supplies, they should be set equal to SVSMHRRL. Therefore, set AUX1LVL and AUX2LVL = 5.

##### 4.2.13.3 Example 3

This example shows configuration for running at a system frequency of 8 MHz an a nominal supply voltage of 3.3 V with settings designed to minimize power consumption.

To minimize power consumption at 8 MHz, set PMMCOREV = 0. With this core voltage, the recommended  $\text{SVM}_H$  setting is SVSMHRRL = 0. Because AUX0LVL must be at least 1 higher than SVSMHRRL, set AUX0LVL = 1.

To continue operation without glitches when switching to AUX1 and AUX2, the thresholds for these must be equal to or higher than SVSMHRRL. To minimize use of the auxiliary supplies, they should be set equal to SVSMHRRL. Therefore, set AUX1LVL and AUX2LVL = 0.

##### 4.2.13.4 Example 4

This example shows configuration for running at a system frequency of 25 MHz when operating on a supply voltage of  $3\text{ V} \pm 0.3\text{ V}$  from DVCC, and then changing the system frequency to 12 MHz when supplied from AUX1 and to 8 MHz when supplied from AUX2.

The maximum  $\text{SVM}_H$  level must be less than or equal to 2.7 V, so that switching occurs only when  $V_{\text{DSYS}}$  goes outside  $3\text{ V} \pm 0.3\text{ V}$ . To switch back to DVCC/AVCC as soon as possible when it is within  $3\text{ V} \pm 0.3\text{ V}$ , the maximum AUX0LVL must also be less than or equal to 2.7 V. Because AUX0LVL must be 1 higher than SVSMHRRL, set = 4 and SVSMHRRL = 3.

*Auxiliary Supply Operation*

www.ti.com

To support 25-MHz operation, PMMCOREV must equal 3. With SVSMHRRRL = 3, all core voltage settings are supported, so this is valid. Based on this core voltage setting, configure SVS levels (see for details).

When switching back to AUX0 (DVCC/AVCC) after having failed over to AUX1 or AUX2, these same settings must be restored.

Given that the system frequency should be changed to 12 MHz when running from AUX1, settings must be changed when switching to AUX1.

When switching from AUX0 (DVCC/AVCC) to AUX1:

1. Decrease system frequency to 12 MHz
2. Decrease core voltage level by setting PMMCOREV = 1 (also change SVS settings)
3. Decrease the SVM level by setting SVSMHRRRL = 1
4. Set AUX1LVL and AUX2LVL to 1

When switching from AUX2 to AUX1:

1. Increase AUX1LVL and AUX2LVL to 1
2. Increase the SVM level by setting SVSMHRRRL = 1
3. Increase core voltage level by setting PMMCOREV = 1 (also change SVS settings)
4. Increase system frequency to 12 MHz

Given that the system frequency should be changed to 8 MHz when running from AUX2, settings must be changed when switching to AUX2.

1. Decrease system frequency to 8 MHz
2. Decrease core voltage level by setting PMMCOREV = 0 (also change SVS settings)
3. Decrease the SVM level by setting SVSMHRRRL = 0
4. Set AUX1LVL and AUX2LVL to 0

## 4.3 AUX Registers

The registers to control the auxiliary supplies are listed in [Table 4-4](#). The base address for the registers can be found in the device-specific data sheet. The address offsets are given in [Table 4-4](#).

The access key, AUXKEY, defined in the AUXCTL0 register controls access to the AUXCTLx registers. Once the correct key is written, the write access is enabled. The write access is disabled by writing a wrong key in byte mode to the AUXCTL0 upper byte. Word accesses to AUXCTL0 with a wrong key also disables the write access. A write access to a AUXCTLx register other than AUXCTL0 while write access is not enabled is ignored.

---

**NOTE: Bit Naming Convention**

The bits and bit fields in the AUXCTL0 to AUXCTL2 registers are named according to what supplies they refer to: AUX0... refers to DVCC, AUX1... refers to AUXVCC1, and AUX2... refers to AUXVCC2. In any description, the occurrence of a bit name like AUXx... refers to all the bit names AUX0..., AUX1..., and AUX2....

---

**Table 4-4. Auxiliary Supply Registers**

Offset	Acronym	Register Name	Type	Reset	Section
00h	AUXCTL0	Auxiliary Supply Control 0 register <sup>(1)</sup>	Read/write	9601h	<a href="#">Section 4.3.1</a>
02h	AUXCTL1	Auxiliary Supply Control 1 register <sup>(1)</sup>	Read/write	0000h	<a href="#">Section 4.3.2</a>
04h	AUXCTL2	Auxiliary Supply Control 2 register <sup>(1)</sup>	Read/write	0000h	<a href="#">Section 4.3.3</a>
06h		Reserved			
08h		Reserved			
0Ah		Reserved			
0Ch		Reserved			
0Eh		Reserved			
10h		Reserved			
12h	AUX2CHCTL	AUX2 Charger Control	Read/write	5A00h	<a href="#">Section 4.3.4</a>
14h	AUX3CHCTL	AUX3 Charger Control	Read/write	5A00h	<a href="#">Section 4.3.5</a>
16h	AUXADCCTL	AUX ADC Control	Read/write	0000h	<a href="#">Section 4.3.6</a>
18h		Reserved			
1Ah	AUXIFG	AUX Interrupt Flag	Read/write	0000h	<a href="#">Section 4.3.7</a>
1Ch	AUXIE	AUX Interrupt Enable	Read/write	0000h	<a href="#">Section 4.3.8</a>
1Eh	AUXIV	AUX Interrupt Vector Word	Read/write	0000h	<a href="#">Section 4.3.9</a>

<sup>(1)</sup> Access protected by key AUXKEY in AUXCTL0.

**4.3.1 AUXCTL0 Register**

Auxiliary Supply Control 0 Register

**Figure 4-10. AUXCTL0 Register**

15	14	13	12	11	10	9	8
AUXKEY							
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				AUX2SW	AUX1SW	AUX0SW	LOCKAUX
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r/w0-[1]

**Table 4-5. AUXCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-8	AUXKEY	RW	96h	AUX access key. Always read as 096h. Must be written with 0A5h or all writes to AUXCTLx registers are ignored.
7-4	Reserved	R	0h	Reserved. Always reads as 0.
3	AUX2SW	RW	0h	AUXVCC2 switch state 0b = AUXVCC2 switch open 1b = AUXVCC2 switch closed
2	AUX1SW	RW	0h	AUXVCC1 switch state 0b = AUXVCC1 switch open 1b = AUXVCC1 switch closed
1	AUX0SW	RW	0h	DVCC switch state 0b = DVCC switch open 1b = DVCC switch closed
0	LOCKAUX	RW	0h	Lock auxiliary supply system. Can only be written as 0. LOCKAUX is always set to 1 by hardware after the core was powered down either due to a complete power cycle of the main supplies (DVCC, AUXVCC1, or AUXVCC2) or due to LPMx.5 operation. 0b = Auxiliary supply system not locked 1b = Auxiliary supply system locked - operating from either DVCC or AUXVCC1

#### 4.3.2 AUXCTL1 Register

Auxiliary Supply Control 1 Register

**Figure 4-11. AUXCTL1 Register**

15	14	13	12	11	10	9	8
		Reserved			AUX2MD	AUX1MD	AUX0MD
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
		Reserved		AUX2PRIO	AUX2OK	AUX1OK	AUX0OK
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 4-6. AUXCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10	AUX2MD	RW	0h	AUXVCC2 auxiliary supply mode 0b = Hardware controlled 1b = Software controlled
9	AUX1MD	RW	0h	AUXVCC1 auxiliary supply mode 0b = Hardware controlled 1b = Software controlled
8	AUX0MD	RW	0h	DVCC auxiliary supply mode 0b = Hardware controlled 1b = Software controlled
7-4	Reserved	R	0h	Reserved. Always reads as 0.
3	AUX2PRIO	RW	0h	Auxiliary supply AUXVCC2 priority. Defines order of switching between AUXVCC1 and AUXVCC2. 0b = AUXVCC2 has lower priority than AUXVCC1 1b = AUXVCC2 has higher priority than AUXVCC1
2	AUX2OK	RW	0h	AUXVCC2 okay flag. Read-only if AUX2MD = 0 and indicates the status monitored by the hardware based on the selected level AUX2LVLx. If AUX2MD = 1 the bit must be controlled by software to indicate the status of the supply. It is not modified by hardware in this case. 0b = Supply not okay - below AUX2LVLx if AUX2MD = 0 1b = Supply okay - above AUX2LVLx if AUX2MD = 0
1	AUX1OK	RW	0h	AUXVCC1 okay flag. Read-only if AUX1MD = 0 and indicates the status monitored by the hardware based on the selected level AUX1LVLx. If AUX1MD = 1 the bit must be controlled by software to indicate the status of the supply. It is not modified by hardware in this case. 0b = Supply not okay - below AUX1LVLx if AUX1MD = 0 1b = Supply okay - above AUX1LVLx if AUX1MD = 0
0	AUX0OK	RW	0h	DVCC okay flag. Read-only if AUX0MD = 0 and indicates the status monitored by the hardware based on the selected level AUX0LVLx. If AUX0MD = 1 the bit must be controlled by software to indicate the status of the supply. It is not modified by hardware in this case. 0b = Supply not okay - below AUX0LVLx if AUX0MD = 0 1b = Supply okay - above AUX0LVLx if AUX0MD = 0

**4.3.3 AUXCTL2 Register**

Auxiliary Supply Control 2 Register

**Figure 4-12. AUXCTL2 Register**

15	14	13	12	11	10	9	8
Reserved		AUXMRx		Reserved	AUX2LVLx		
r0	r0	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved		AUX1LVLx		Reserved	AUX0LVLx		
r0	rw-(0)	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)

**Table 4-7. AUXCTL2 Register Description**

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0.
13-12	AUXMRx	RW	0h	Auxiliary supply monitoring rate 00b = Continous monitoring 01b = Monitoring every 32 VLO clock cycles ( $\approx$ 5ms) 10b = Monitoring every 1024 VLO clock cycles ( $\approx$ 150ms) 11b = Reserved
11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	AUX2LVLx	RW	0h	AUXVCC2 auxiliary supply threshold level. The levels are specified in the device-specific data sheet. 000b = Approximately 1.74 V 001b = Approximately 1.94 V 010b = Approximately 2.14 V 011b = Approximately 2.26 V 100b = Approximately 2.40 V 101b = Approximately 2.70 V 110b = Approximately 3.00 V 111b = Approximately 3.00 V
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	AUX1LVLx	RW	0h	AUXVCC1 auxiliary supply threshold level. The levels are specified in the device-specific data sheet. 000b = Approximately 1.74 V 001b = Approximately 1.94 V 010b = Approximately 2.14 V 011b = Approximately 2.26 V 100b = Approximately 2.40 V 101b = Approximately 2.70 V 110b = Approximately 3.00 V 111b = Approximately 3.00 V
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	AUX0LVLx	RW	0h	DVCC auxiliary supply threshold level. The levels are specified in the device-specific data sheet. 000b = Approximately 1.74 V 001b = Approximately 1.94 V 010b = Approximately 2.14 V 011b = Approximately 2.26 V 100b = Approximately 2.40 V 101b = Approximately 2.70 V 110b = Approximately 3.00 V 111b = Approximately 3.00 V

#### 4.3.4 AUX2CHCTL Register

AUX Charger Control Register for AUX2

**Figure 4-13. AUX2CHCTL Register**

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

**Table 4-8. AUX2CHCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	AUXCHKEYx	RW	5Ah	Charger access key. Always read as 05Ah. Must be written as 069h together with low byte; writing any other value disables the charger and all control register bits are reset to 0.
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	AUXCHVx	RW	0h	Charger end voltage 00b = Charger disabled 01b = VCC 10b = Reserved 11b = Reserved
3	Reserved	R	0h	Reserved. Always reads as 0.
2-1	AUXCHCx	RW	0h	Charger charge current 00b = Charger disabled 01b = Charge current defined by a maximum 5-kΩ resistor 10b = Charge current defined by a maximum 10-kΩ resistor 11b = Charge current defined by a maximum 20-kΩ resistor
0	AUXCHEN	RW	0h	Charger enable 0b = Charger disabled 1b = Charger enabled

**4.3.5 AUX3CHCTL Register**

AUX Charger Control Register for AUX3

**Figure 4-14. AUX3CHCTL Register**

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

**Table 4-9. AUX3CHCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	AUXCHKEYx	RW	5Ah	Charger access key. Always read as 05Ah. Must be written as 069h together with low byte; writing any other value disables the charger and all control register bits are reset to 0.
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	AUXCHVx	RW	0h	Charger end voltage 00b = Charger disabled 01b = VCC 10b = Reserved 11b = Reserved
3	Reserved	R	0h	Reserved. Always reads as 0.
2-1	AUXCHCx	RW	0h	Charger charge current 00b = Charger disabled 01b = Charge current defined by a maximum 5-kΩ resistor 10b = Charge current defined by a maximum 10-kΩ resistor 11b = Charge current defined by a maximum 20-kΩ resistor
0	AUXCHEN	RW	0h	Charger enable 0b = Charger disabled 1b = Charger enabled

#### 4.3.6 AUXADCCTL Register

Auxiliary Supply ADC Control Register

**Figure 4-15. AUXADCCTL 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		AUXADCRx		Reserved	AUXADCSELx		AUXADC
r0	r0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

**Table 4-10. AUXADCCTL Register Description**

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	AUXADCRx	RW	0h	Load resistance ( $R_{(tot)} = 3R$ ) during sampling of selected supply. Refer to the device-specific data sheet, too. 00b = $R_{(tot)} \approx 15 \text{ k}\Omega - I = U/R = 3.6 \text{ V} / 15 \text{ k}\Omega = 240 \mu\text{A}$ ; $I = 1.8 \text{ V} / 16 \text{ k}\Omega = 120 \mu\text{A}$ 01b = $R_{(tot)} \approx 1.5 \text{ k}\Omega - I = 2.4 \text{ mA}$ at 3.6 V; $I = 1.2 \text{ mA}$ at 1.8V 10b = $R_{(tot)} \approx 0.5 \text{ k}\Omega - I = 7.2 \text{ mA}$ at 3.6 V; $I = 3.6 \text{ mA}$ at 1.8 V 11b = Reserved
3	Reserved	R	0h	Reserved. Always reads as 0.
2-1	AUXADCSELx	RW	0h	Select supply to be measured with ADC. 00b = DVCC 01b = AUXVCC1 10b = AUXVCC2 11b = AUXVCC3
0	AUXADC	RW	0h	Auxiliary supplies to ADC 0b = Auxiliary supply measurement disabled 1b = Auxiliary supply measurement enabled

**4.3.7 AUXIFG Register**

Auxiliary Supply Interrupt Flag Register

**Figure 4-16. AUXIFG Register**

15	14	13	12	11	10	9	8
Reserved							AUXSWMIFG
r0	r0	r0	r0	r0	r0	r0	rw-(0)
7	6	5	4	3	2	1	0
AUXMONIFG	AUX2DRPIFG	AUX1DRPIFG	Reserved	Reserved	AUX2SWIFG	AUX1SWIFG	AUX0SWIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)

**Table 4-11. AUXIFG Register Description**

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	AUXSWMIFG	RW	0h	Supplies switched (non-)maskable interrupt flag. Set if a switch from any supply to any other supply happened. Sources an NMI if AUXSWMIE is set, otherwise sources a maskable interrupt if AUXSWGIE and GIE is set. 0b = No interrupt pending 1b = Interrupt pending
7	AUXMONIFG	RW	0h	Supply monitor interrupt flag. Set after completion of a hardware monitoring cycle. 0b = No interrupt pending 1b = Interrupt pending
6	AUX2DRPIFG	RW	0h	AUXVCC2 dropped below its threshold interrupt flag. 0b = No interrupt pending 1b = Interrupt pending
5	AUX1DRPIFG	RW	0h	AUXVCC1 dropped below its threshold interrupt flag. 0b = No interrupt pending 1b = Interrupt pending
4	Reserved	RW	0h	Reserved. Always write as 0.
3	Reserved	R	0h	Reserved. Always reads as 0.
2	AUX2SWIFG	RW	0h	Switched to AUXVCC2 interrupt flag. 0b = No interrupt pending 1b = Interrupt pending
1	AUX1SWIFG	RW	0h	Switched to AUXVCC1 interrupt flag. 0b = No interrupt pending 1b = Interrupt pending
0	AUX0SWIFG	RW	0h	Switched to DVCC interrupt flag. 0b = No interrupt pending 1b = Interrupt pending

#### 4.3.8 AUXIE Register

Auxiliary Supply Interrupt Enable Register

**Figure 4-17. AUXIE Register**

15	14	13	12	11	10	9	8
Reserved							AUXSWNMIE
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
AUXMONIE	AUX2DRPIE	AUX1DRPIE	Reserved	AUXSWGIE	AUX2SWIE	AUX1SWIE	AUX0SWIE
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 4-12. AUXIE Register Description**

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	AUXSWNMIE	R	0h	Supplies switched non-maskable interrupt enable. 0b = Non-maskable interrupt disabled 1b = Non-maskable interrupt enabled
7	AUXMONIE	RW	0h	Supply monitor interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
6	AUX2DRPIE	RW	0h	AUXVCC2 dropped below its threshold interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
5	AUX1DRPIE	RW	0h	AUXVCC1 dropped below its threshold interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
4	Reserved	RW	0h	Reserved. Always write as 0.
3	AUXSWGIE	RW	0h	Global supply switched interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
2	AUX2SWIE	RW	0h	Switched to AUXVCC2 interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
1	AUX1SWIE	RW	0h	Switched to AUXVCC1 interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
0	AUX0SWIE	RW	0h	Switched to DVCC interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled

**4.3.9 AUXIV Register**

Auxiliary Supply Interrupt Vector Register

**Figure 4-18. AUXIV Register**

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

**Table 4-13. AUXIV Register Description**

Bit	Field	Type	Reset	Description
15-0	AUXIVx	R	0h	<p>Auxiliary Supply Interrupt vector value. It generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending interrupt flags.</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: Global (non-)maskable supply switched interrupt flag; Interrupt Flag: AUXSWNMIFG; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: Switched to DVCC interrupt flag; Interrupt Flag: AUX0SWIFG</p> <p>06h = Interrupt Source: Switched to AUXVCC1 interrupt flag; Interrupt Flag: AUX1SWIFG</p> <p>08h = Interrupt Source: Switched to AUXVCC2 interrupt flag; Interrupt Flag: AUX2SWIFG</p> <p>0Ah = Interrupt Source: Reserved; Interrupt Flag: -</p> <p>0Ch = Interrupt Source: AUXVCC1 below threshold interrupt flag; Interrupt Flag: AUX1DRPIFG</p> <p>0Eh = Interrupt Source: AUXVCC2 below threshold interrupt flag; Interrupt Flag: AUX2DRPIFG</p> <p>10h = Interrupt Source: Supply monitor interrupt flag; Interrupt Flag: AUXMONIFG; Interrupt Priority: Lowest</p>

## **Unified Clock System (UCS)**

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

<b>Topic</b>	<b>Page</b>
5.1 <b>Unified Clock System (UCS) Introduction</b> .....	<b>158</b>
5.2 <b>UCS Operation</b> .....	<b>160</b>
5.3 <b>Module Oscillator (MODOSC)</b> .....	<b>171</b>
5.4 <b>UCS Module Registers</b> .....	<b>172</b>

## 5.1 Unified Clock System (UCS) Introduction

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

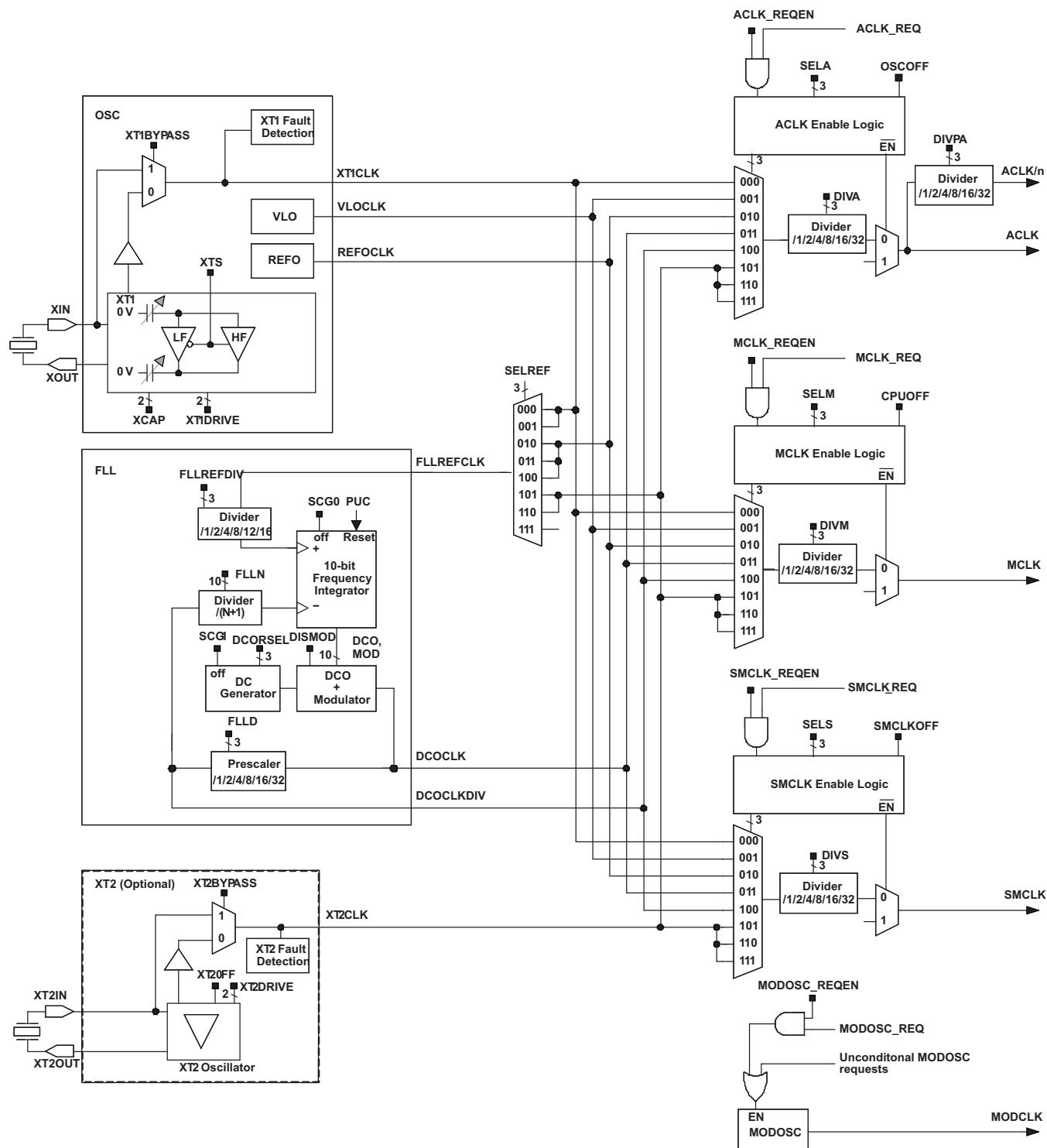
The UCS module includes up to five clock sources:

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

Three clock signals are available from the UCS module:

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

The block diagram of the UCS module is shown in [Figure 5-1](#).


**Figure 5-1. UCS Block Diagram**

## 5.2 UCS Operation

After a PUC, the UCS module default configuration is:

- XT1 in LF mode is selected as the oscillator source for XT1CLK. XT1CLK is selected for ACLK.
- DCOCLKDIV is selected for MCLK.
- DCOCLKDIV is selected for SMCLK.
- FLL operation is enabled and XT1CLK is selected as the FLL reference clock, FLLREFCLK.
- On devices that have XIN and XOUT shared with general-purpose I/O, XIN and XOUT pins are set to general-purpose I/Os and XT1 remains disabled until the I/O ports are configured for XT1 operation. If XIN and XOUT are not shared with general-purpose I/O, XT1 is enabled.
- When available, XT2IN and XT2OUT pins are set to general-purpose I/Os and XT2 is disabled.

As previously stated, FLL operation with XT1 is selected by default. If the crystal pins (XIN, XOUT) are shared with general-purpose I/Os, XT1 will remain disabled until the PSEL bits associated with the crystal pins are set. If XIN and XOUT are not shared with general-purpose I/O, XT1 is enabled. When a 32,768 Hz crystal is used for XT1CLK, the fault control logic immediately causes ACLK to be sourced by the REFOCLK, because XT1 is not stable immediately (see [Section 5.2.12](#)). Once crystal startup is obtained and settled, the FLL stabilizes MCLK and SMCLK to 1.048576 MHz and  $f_{DCO} = 2.097152$  MHz.

Status register control bits (SCG0, SCG1, OSCOFF, and CPUOFF) configure the MSP430 operating modes and enable or disable portions of the UCS module (see the [SYS chapter](#)). Registers UCSCTL0 through UCSCTL8, configure the UCS module.

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

---

**NOTE:** For devices using RTC\_B, RTC\_C, or RTC\_D (RTC modules supporting LPM3.5) setting bit RTCHOLD = 0 in register RTCCTL1 also enables XT1, independent from UCS configuration.

---

### 5.2.1 UCS Module Features for Low-Power Applications

Conflicting requirements typically exist in battery-powered applications:

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

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

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

### 5.2.2 Internal Very-Low-Power Low-Frequency Oscillator (VLO)

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

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

### 5.2.3 Internal Trimmed Low-Frequency Reference Oscillator (REFO)

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

REFO is enabled under any of the following conditions:

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

---

#### NOTE: REFO Enable for MSP430F543x, MSP430F541x devices

REFO is enabled under any of the following conditions:

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

---

### 5.2.4 XT1 Oscillator

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

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

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

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

Some devices support XT1 bypass operation with external clock inputs that reside on a different external supply domain, called DV<sub>IO</sub>. Please refer to the device specific datasheet. On these devices, DV<sub>IO</sub> has a voltage range of  $1.8V \pm 10\%$ . When using the XT1 bypass operation with external clock inputs that reside on DV<sub>IO</sub>, it is required that XT1BYPASSLV = 1. For example, when XT1BYPASSLV = 1, it is assumed the external clock signal swings from 0V to DV<sub>IO</sub>. With XT1BYPASS = 0, it is assumed the external clock signal swings from 0V to DV<sub>CC</sub>. The usage of XT1BYPASSLV allows for interfacing to external clock sources that reside on either the DV<sub>CC</sub> or DV<sub>IO</sub> supply domains. When used with an external signal, the external frequency must meet the data sheet parameters for the chosen mode. XT1 is powered down when used in bypass mode.

On many devices, the XT1 pins are shared with general-purpose I/O ports (refer to the device specific datasheet for availability). At power up, the default operation is XT1, LF mode of operation. However, for devices that have XT1 shared with general-purpose I/O ports, XT1 will remain disabled until the ports shared with XT1 are configured for XT1 operation. The configuration of the shared I/O is determined by the PSEL bit associated with XIN and the XT1BYPASS bit. Setting the PSEL bit causes the XIN and XOUT ports to be configured for XT1 operation. If XT1BYPASS is also set, XT1 is configured for bypass mode of operation, and the oscillator associated with XT1 is powered down. In bypass mode of operation, XIN can accept an external clock input signal and XOUT is configured as a general-purpose I/O. The PSEL bit associated with XOUT is a don't care. If the PSEL bit associated with XIN is cleared, both XIN and XOUT ports are configured as general-purpose I/Os, and XT1 is disabled.

On devices where XT1 is not shared with general-purpose I/O ports, XT1 is enabled at power up. In bypass mode of operation (XT1BYPASS = 1), XIN can accept an external clock input signal, and XT1 is powered down.

XT1 is enabled under any of the following conditions:

- XT1 is a source for ACLK (SELA = {0}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT1 is a source for MCLK (SELM = {0}) and in active mode (AM) (CPUOFF = 0)
- XT1 is a source for SMCLK (SELS = {0}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT1 is a source for FLLREFCLK (SELREF = {0}) and the DCO is a source for ACLK (SELA = {3,4}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT1 is a source for FLLREFCLK (SELREF = {0}) and the DCO is a source for MCLK (SELM = {3,4}) and in active mode (AM) (CPUOFF = 0)
- XT1 is a source for FLLREFCLK (SELREF = {0}) and the DCO is a source for SMCLK (SELS = {3,4}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT1OFF = 0. XT1 enabled in active mode (AM) through LPM4. For devices that support LPMx.5, XT1 also remains enabled.

---

**NOTE: XT1 Enable for MSP430F543x, MSP430F541x devices**

XT1 is enabled under any of the following conditions:

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

### 5.2.5 XT2 Oscillator

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

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

Some devices support XT2 bypass operation with external clock inputs that reside on a different external supply domain, called DV<sub>IO</sub>. Please refer to the device specific datasheet. On these devices, DV<sub>IO</sub> has a voltage range of  $1.8V \pm 10\%$ . When using the XT2 bypass operation with external clock inputs that reside on DV<sub>IO</sub>, it is required that XT2BYPASSLV = 1. For example, when XT2BYPASSLV = 1, it is assumed the external clock signal swings from 0V to DV<sub>IO</sub>. With XT2BYPASS = 0, it is assumed the external clock signal swings from 0V to DV<sub>CC</sub>. The usage of XT2BYPASSLV allows for interfacing to external clock sources that reside on either the DV<sub>CC</sub> or DV<sub>IO</sub> supply domains. When used with an external signal, the external frequency must meet the data sheet parameters for the chosen mode. XT2 is powered down when used in bypass mode.

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

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

XT2 is enabled under any of the following conditions:

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

---

**NOTE: XT2 Enable for MSP430F543x, MSP430F541x devices**

XT2 is enabled under any of the following conditions:

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

---

### 5.2.6 Digitally-Controlled Oscillator (DCO)

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

---

**NOTE: For the F543x and F541x non-A versions only.**

Setting SELREF = {7} sets XT2CLK as the FLL reference clock.

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

The divider ( $N + 1$ ) and the divider value D define the DCOCLK and DCOCLKDIV frequencies. The divider ( $N + 1$ ) can be set using the FLLN bits, where  $N > 0$ . The smallest divider ( $N + 1$ ) that can be used is a divider of two. The logic will cause FLLN = 1h, if FLLN = 0h is unintentionally written. Therefore setting FLLN = 0h is also equivalent to setting FLLN = 1h and will result in a divider of 2. All other FLLN settings behave as described e.g. FLLN = 2h results in a divider of 3, FLLN = 3h results in a divider of 4, etc.

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

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

### Adjusting DCO Frequency

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

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

The frequency of DCOCLK is set by the following functions:

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

### 5.2.7 Frequency Locked Loop (FLL)

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

---

#### NOTE: Reading MOD and DCO bits

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

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

---

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

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

### 5.2.8 DCO Modulator

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

The modulator mixing formula is:

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

Figure 5-2 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.

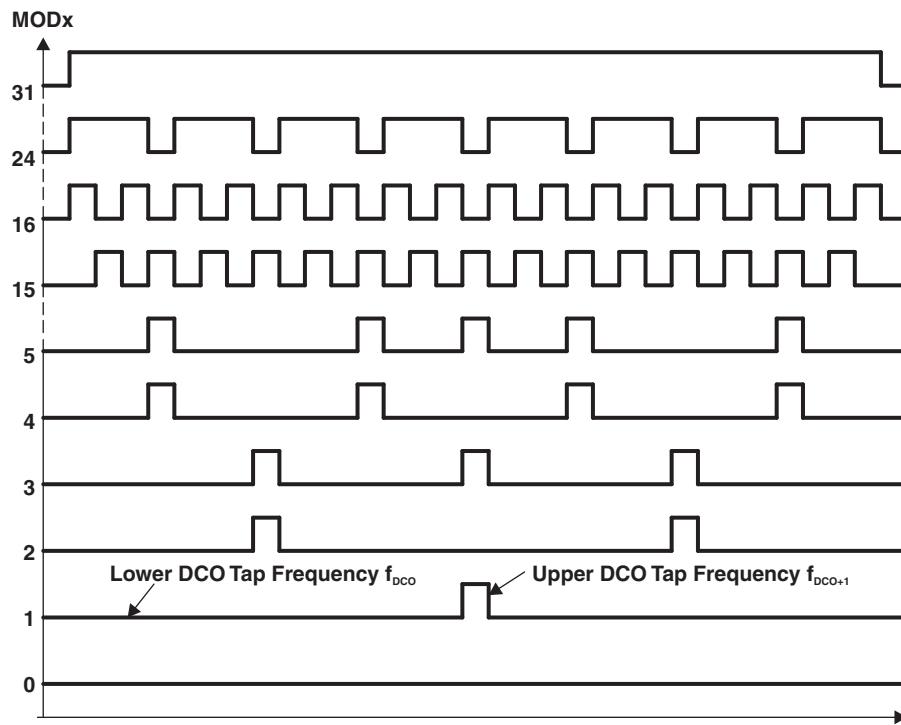


Figure 5-2. Modulator Patterns

### 5.2.9 Disabling FLL Hardware and Modulator

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

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

---

**NOTE: DCO operation without FLL**

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

---

### 5.2.10 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, 3, or 4, the FLL remains disabled and the DCO operates at the previous setting as defined in UCSCTL0 and UCSCTL1. SCG0 can be cleared by user software if FLL operation is required.

### 5.2.11 Operation From Low-Power Modes, Requested by Peripheral Modules

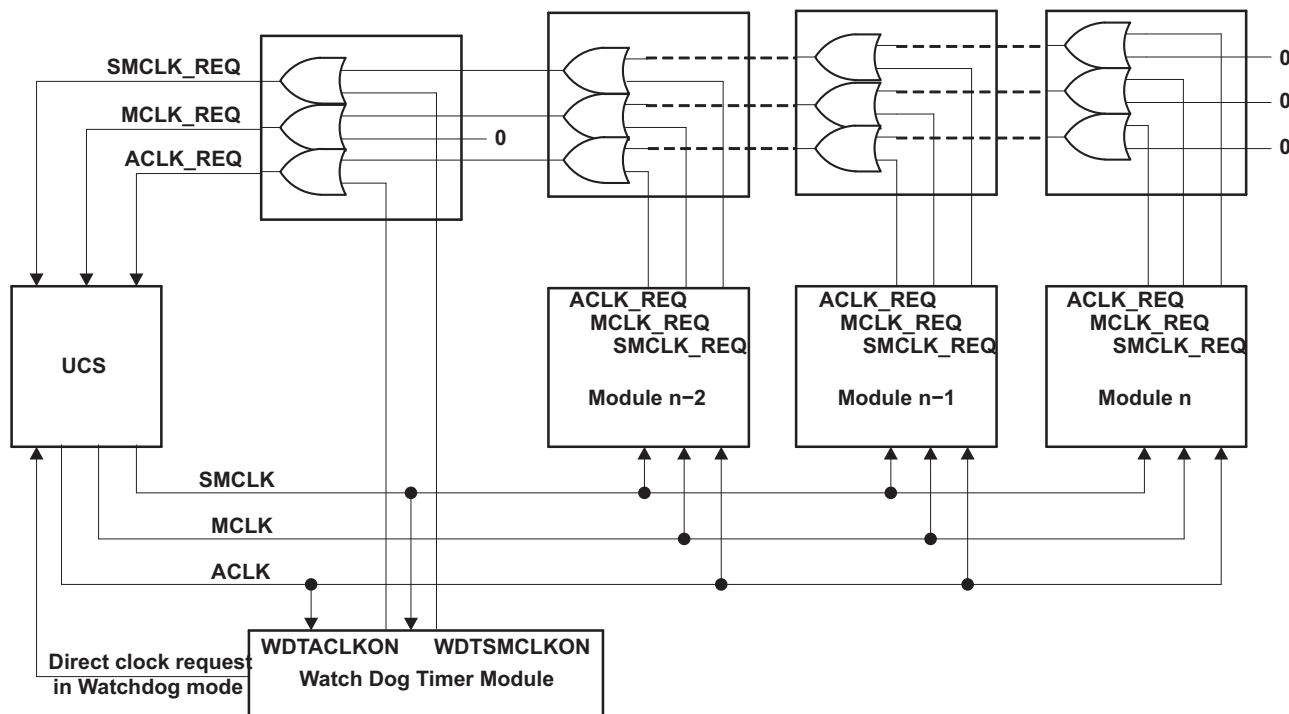
A peripheral module requests its clock sources automatically from the UCS module if required for its proper operation, regardless of the current mode of operation, as shown in [Figure 5-3](#).

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

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

If the requested source is not active, the software NMI handler must take care of the required actions. For the previous example, if ACLK was sourced by XT1 and XT1 was not enabled, an oscillator fault condition 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 specified values in the data sheet.



**Figure 5-3. Module Request Clock System**

By default, the clock request logic is enabled. The clock request logic can be disabled by clearing ACLKREQEN, MCLKREQEN, or SMCLKREQEN, for each respective system clock. When ACLKREQEN or MCLKREQEN bits are set, or active, the clock is available to the system and 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 is also influenced by the SMCLKOFF bit in the UCSCTL6 register. [Table 5-1](#) shows the relationship between the system clocks and the low-power modes in conjunction with the clock request logic.

**Table 5-1. Clock Request System and Power Modes**

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

<sup>(1)</sup> Any clock request prior to entry into LPM3.5 or LPM4.5 is ignored and LPM3.5 or LPM4.5 entry occurs. For the special case when XT1OFF = 0 or XT2OFF = 0, the LPMx.5 request is ignored and the device does not enter LPMx.5.

### 5.2.12 UCS Module Fail-Safe Operation

The UCS module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for XT1, DCO, and XT2 as shown in [Figure 5-4](#). The available fault conditions are:

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

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

When using XT1 operation in LF mode as the reference source into the FLL ( $SELREF = \{0\}$ ), a crystal fault automatically causes the FLL reference source, FLLREFCLK, to be sourced by the REFO.

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

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

The OFIFG oscillator-fault interrupt flag is set and latched at POR or when any oscillator fault (XT1LFOFFG, XT1HFOFFG, XT2OFFG, or DCOFG) 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 and exit circuitry removes this requirement. The OFIFG flag must be cleared by software. The source of the fault can be identified by checking the individual fault bits.

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

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

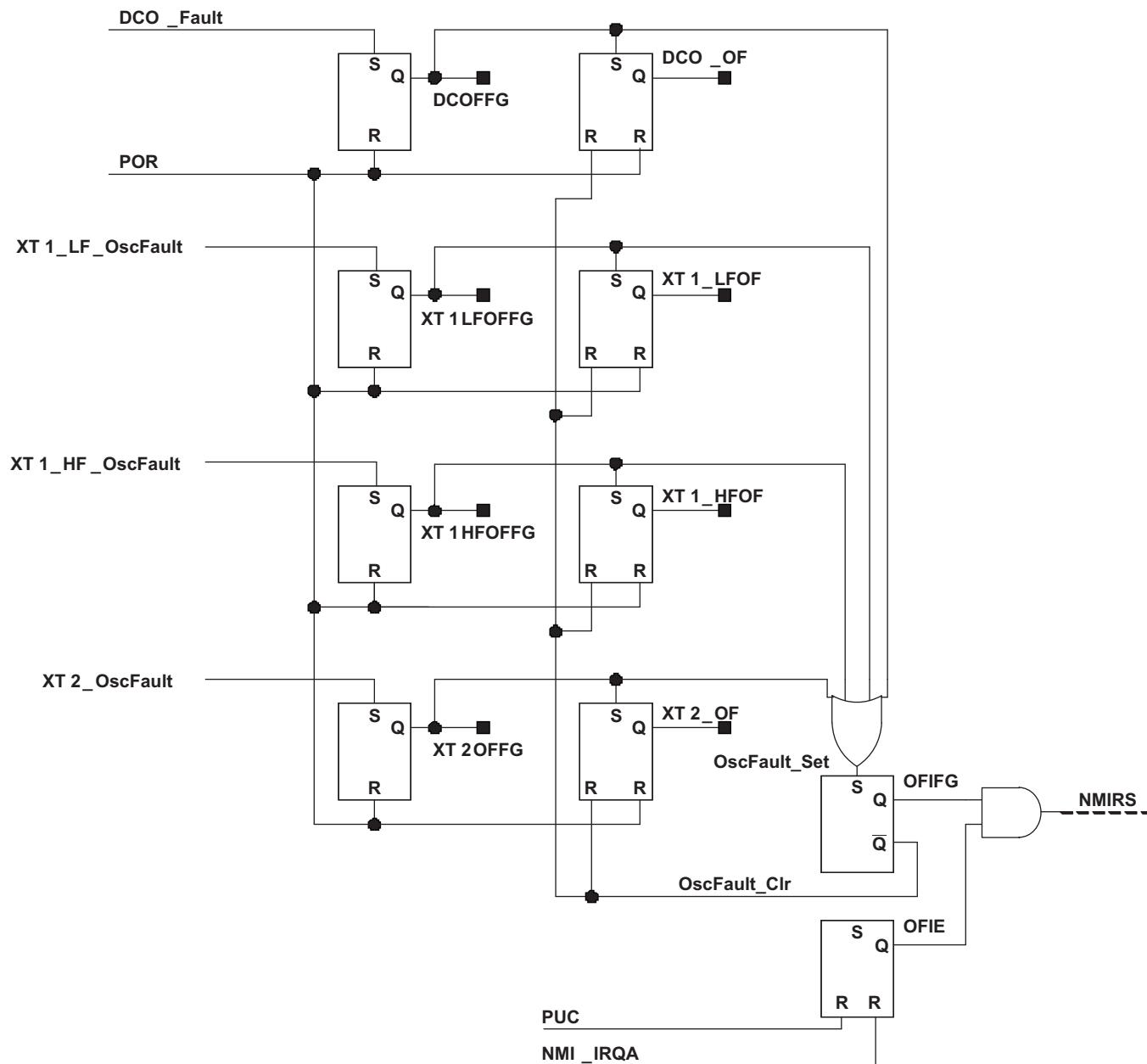
If a fault is detected for the oscillator sourcing ACLK, ACLK is automatically switched to the DCO for its clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If ACLK is sourced from XT1 in LF mode, an oscillator fault causes ACLK to be automatically switched to the REFO for its clock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

---

**NOTE: DCO active during oscillator fault**

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

---


**Figure 5-4. Oscillator Fault Logic**

**NOTE: Fault conditions**

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

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

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

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

**NOTE: Fault logic**

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

**NOTE: Fault logic counters**

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

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

### 5.2.13 Synchronization of Clock Signals

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

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

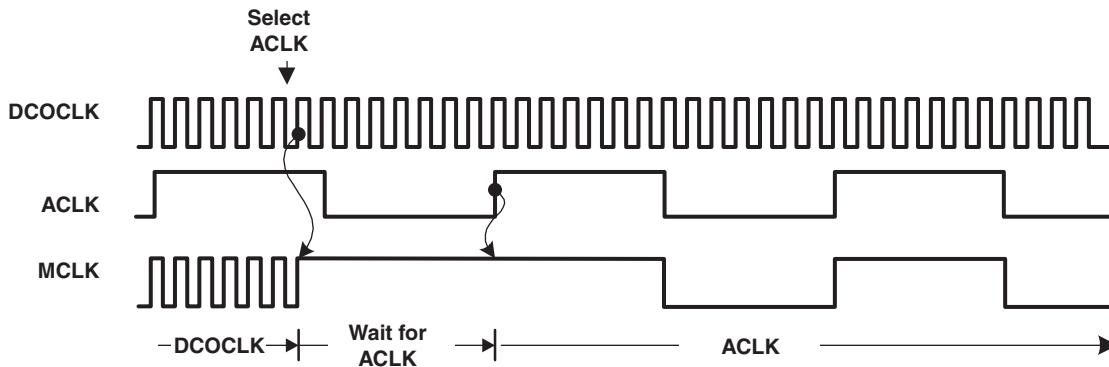


Figure 5-5. Switch MCLK from DCOCLK to XT1CLK

## 5.3 Module Oscillator (MODOSC)

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

### 5.3.1 MODOSC Operation

To conserve power, MODOSC is powered down when not needed and enabled only when required. When the MODOSC source is required, the respective module requests it. MODOSC is enabled based on unconditional and conditional requests. Setting MODOSCREQEN enables conditional requests. Unconditional requests are always enabled. It is not necessary to set MODOSCREQEN for modules that use unconditional requests; for example, flash controller or ADC12\_A.

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

The ADC12\_A may optionally use MODOSC as a clock source for its conversion clock. The user chooses the ADC12OSC as the conversion clock source. During a conversion, the ADC12\_A module issues an unconditional request for the ADC12OSC clock source. Upon doing so, the MODOSC source is enabled, if not already enabled from other modules' previous requests.

## 5.4 UCS Module Registers

The UCS module registers are listed in [Table 5-2](#). The base address can be found in the device-specific data sheet. The address offset is listed in [Table 5-2](#).

---

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

---

**Table 5-2. UCS Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCSCTL0	Unified Clock System Control 0	Read/write	Word	0000h	<a href="#">Section 5.4.1</a>
00h	UCSCTL0_L		Read/write	Byte	00h	
01h	UCSCTL0_H		Read/write	Byte	00h	
02h	UCSCTL1	Unified Clock System Control 1	Read/write	Word	0020h	<a href="#">Section 5.4.2</a>
02h	UCSCTL1_L		Read/write	Byte	20h	
03h	UCSCTL1_H		Read/write	Byte	00h	
04h	UCSCTL2	Unified Clock System Control 2	Read/write	Word	101Fh	<a href="#">Section 5.4.3</a>
04h	UCSCTL2_L		Read/write	Byte	1Fh	
05h	UCSCTL2_H		Read/write	Byte	10h	
06h	UCSCTL3	Unified Clock System Control 3	Read/write	Word	0000h	<a href="#">Section 5.4.4</a>
06h	UCSCTL3_L		Read/write	Byte	00h	
07h	UCSCTL3_H		Read/write	Byte	00h	
08h	UCSCTL4	Unified Clock System Control 4	Read/write	Word	0044h	<a href="#">Section 5.4.5</a>
08h	UCSCTL4_L		Read/write	Byte	44h	
09h	UCSCTL4_H		Read/write	Byte	00h	
0Ah	UCSCTL5	Unified Clock System Control 5	Read/write	Word	0000h	<a href="#">Section 5.4.6</a>
0Ah	UCSCTL5_L		Read/write	Byte	00h	
0Bh	UCSCTL5_H		Read/write	Byte	00h	
0Ch	UCSCTL6	Unified Clock System Control 6	Read/write	Word	C1CDh	<a href="#">Section 5.4.7</a>
0Ch	UCSCTL6_L		Read/write	Byte	CDh	
0Dh	UCSCTL6_H		Read/write	Byte	C1h	
0Eh	UCSCTL7	Unified Clock System Control 7	Read/write	Word	0703h	<a href="#">Section 5.4.8</a>
0Eh	UCSCTL7_L		Read/write	Byte	03h	
0Fh	UCSCTL7_H		Read/write	Byte	07h	
10h	UCSCTL8	Unified Clock System Control 8	Read/write	Word	0707h	<a href="#">Section 5.4.9</a>
10h	UCSCTL8_L		Read/write	Byte	07h	
11h	UCSCTL8_H		Read/write	Byte	07h	
12h	UCSCTL9	Unified Clock System Control 9 <sup>(1)</sup>	Read/write	Word	0000h	<a href="#">Section 5.4.10</a>
12h	UCSCTL9_L		Read/write	Byte	00h	
13h	UCSCTL9_H		Read/write	Byte	00h	

<sup>(1)</sup> This register is not available on all devices. See the device-specific data sheet.

### 5.4.1 UCSCTL0 Register

Unified Clock System Control 0 Register

**Figure 5-6. UCSCTL0 Register**

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

**Table 5-3. UCSCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DCO	RW	0h	DCO tap selection. These bits select the DCO tap and are modified automatically during FLL operation.
7-3	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 decremented.
2-0	Reserved	R	0h	Reserved. Always reads as 0.

### 5.4.2 UCSCTL1 Register

Unified Clock System Control 1 Register

**Figure 5-7. UCSCTL1 Register**

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

**Table 5-4. UCSCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	DCORSEL	RW	2h	DCO frequency range select. These bits select the DCO frequency range of operation defined in the device-specific datasheet.
3-2	Reserved	R	0h	Reserved. Always reads as 0.
1	Reserved	RW	0h	Reserved. Always reads as 0.
0	DISMOD	RW	0h	Modulation. This bit enables or disables the modulation. 0b = Modulation enabled 1b = Modulation disabled

### 5.4.3 UCSCTL2 Register

Unified Clock System Control 2 Register

**Figure 5-8. UCSCTL2 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 5-5. UCSCTL2 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 010b = f(DCOCLK)/4 011b = f(DCOCLK)/8 100b = f(DCOCLK)/16 101b = f(DCOCLK)/32 110b = Reserved for future use. Defaults to f(DCOCLK)/32. 111b = Reserved for future use. Defaults to f(DCOCLK)/32.
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.

**5.4.4 UCSCTL3 Register**

Unified Clock System Control 3 Register

**Figure 5-9. UCSCTL3 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		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

**Table 5-6. UCSCTL3 Register Description**

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	SELREF	RW	0h	FLL reference select. These bits select the FLL reference clock source. 000b = XT1CLK 001b = Reserved for future use. Defaults to XT1CLK. 010b = REFOCLK 011b = Reserved for future use. Defaults to REFOCLK. 100b = Reserved for future use. Defaults to REFOCLK. 101b = XT2CLK when available, otherwise REFOCLK. 110b = Reserved for future use. XT2CLK when available, otherwise REFOCLK. 111b = Reserved for future use. XT2CLK when available, otherwise REFOCLK.
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). The divided frequency is used as the FLL reference frequency. 000b = f(FLLREFCLK)/1 001b = f(FLLREFCLK)/2 010b = f(FLLREFCLK)/4 011b = f(FLLREFCLK)/8 100b = f(FLLREFCLK)/12 101b = f(FLLREFCLK)/16 110b = Reserved for future use. Defaults to f(FLLREFCLK)/16. 111b = Reserved for future use. Defaults to f(FLLREFCLK)/16.

### 5.4.5 UCSCTL4 Register

Unified Clock System Control 4 Register

**Figure 5-10. UCSCTL4 Register**

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

**Table 5-7. UCSCTL4 Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	SELA	RW	0h	Selects the ACLK source 000b = XT1CLK 001b = VLOCLK 010b = REFOCLK 011b = DCOCLK 100b = DCOCLKDIV 101b = XT2CLK when available, otherwise DCOCLKDIV 110b = Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111b = Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	SELS	RW	4h	Selects the SMCLK source 000b = XT1CLK 001b = VLOCLK 010b = REFOCLK 011b = DCOCLK 100b = DCOCLKDIV 101b = XT2CLK when available, otherwise DCOCLKDIV 110b = Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111b = Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	SELM	RW	4h	Selects the MCLK source 000b = XT1CLK 001b = VLOCLK 010b = REFOCLK 011b = DCOCLK 100b = DCOCLKDIV 101b = XT2CLK when available, otherwise DCOCLKDIV 110b = Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111b = Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.

**5.4.6 UCSCTL5 Register**

Unified Clock System Control 5 Register

**Figure 5-11. UCSCTL5 Register**

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

**Table 5-8. UCSCTL5 Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-12	DIVPA	RW	0h	ACLK source divider available at external pin. Divides the frequency of ACLK and presents it to an external pin. 000b = f(ACLK)/1 001b = f(ACLK)/2 010b = f(ACLK)/4 011b = f(ACLK)/8 100b = f(ACLK)/16 101b = f(ACLK)/32 110b = Reserved for future use. Defaults to f(ACLK)/32. 111b = Reserved for future use. Defaults to f(ACLK)/32.
11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	DIVA	RW	0h	ACLK source divider. Divides the frequency of the ACLK clock source. 000b = f(ACLK)/1 001b = f(ACLK)/2 010b = f(ACLK)/4 011b = f(ACLK)/8 100b = f(ACLK)/16 101b = f(ACLK)/32 110b = Reserved for future use. Defaults to f(ACLK)/32. 111b = Reserved for future use. Defaults to f(ACLK)/32.
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	DIVS	RW	0h	SMCLK source divider 000b = f(SMCLK)/1 001b = f(SMCLK)/2 010b = f(SMCLK)/4 011b = f(SMCLK)/8 100b = f(SMCLK)/16 101b = f(SMCLK)/32 110b = Reserved for future use. Defaults to f(SMCLK)/32. 111b = Reserved for future use. Defaults to f(SMCLK)/32.
3	Reserved	R	0h	Reserved. Always reads as 0.

**Table 5-8. UCSCTL5 Register Description (continued)**

Bit	Field	Type	Reset	Description
2-0	DIVM	RW	0h	MCLK source divider 000b = f(MCLK)/1 001b = f(MCLK)/2 010b = f(MCLK)/4 011b = f(MCLK)/8 100b = f(MCLK)/16 101b = f(MCLK)/32 110b = Reserved for future use. Defaults to f(MCLK)/32. 111b = Reserved for future use. Defaults to f(MCLK)/32.

### 5.4.7 UCSCTL6 Register

Unified Clock System Control 6 Register

**Figure 5-12. UCSCTL6 Register**

15	14	13	12	11	10	9	8
XT2DRIVE		Reserved	XT2BYPASS		Reserved		XT2OFF
rw-1	rw-1	r0	rw-0	r0	r0	r0	rw-1
7	6	5	4	3	2	1	0
XT1DRIVE		XTS	XT1BYPASS	XCAP	SMCLKOFF		XT1OFF
rw-1	rw-1	rw-0	rw-0	rw-1	rw-1	rw-0	rw-1

**Table 5-9. UCSCTL6 Register Description**

Bit	Field	Type	Reset	Description
15-14	XT2DRIVE	RW	3h	The XT2 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength. 00b = Lowest current consumption. XT2 oscillator operating range is 4 MHz to 8 MHz. 01b = Increased drive strength XT2 oscillator. XT2 oscillator operating range is 8 MHz to 16 MHz. 10b = Increased drive capability XT2 oscillator. XT2 oscillator operating range is 16 MHz to 24 MHz. 11b = Maximum drive capability and maximum current consumption for both XT2 oscillator. XT2 oscillator operating range is 24 MHz to 32 MHz.
13	Reserved	R	0h	Reserved. Always reads as 0.
12	XT2BYPASS	RW	0h	XT2 bypass select 0b = XT2 sourced internally 1b = XT2 sourced externally from pin
11-9	Reserved	R	0h	Reserved. Always reads as 0.
8	XT2OFF	RW	1h	Turns off the XT2 oscillator 0b = XT2 is on if XT2 is selected via the port selection and XT2 is not in bypass mode of operation 1b = XT2 is off if it is not used as a source for ACLK, MCLK, or SMCLK or is not used as a reference source required for FLL operation
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. 00b = Lowest current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 4 MHz to 8 MHz. 01b = Increased drive strength for XT1 LF mode. XT1 oscillator operating range in HF mode is 8 MHz to 16 MHz. 10b = Increased drive capability for XT1 LF mode. XT1 oscillator operating range in HF mode is 16 MHz to 24 MHz. 11b = Maximum drive capability and maximum current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 24 MHz to 32 MHz.
5	XTS	RW	0h	XT1 mode select 0b = Low-frequency mode. XCAP bits define the capacitance at the XIN and XOUT pins. 1b = High-frequency mode. XCAP bits are not used.
4	XT1BYPASS	RW	0h	XT1 bypass select 0b = XT1 sourced internally 1b = XT1 sourced externally from pin

**Table 5-9. UCSCTL6 Register Description (continued)**

Bit	Field	Type	Reset	Description
3-2	XCAP	RW	3h	Oscillator capacitor selection. These bits select the capacitors applied to the LF crystal or resonator in the LF mode (XTS = 0). The effective capacitance (seen by the crystal) is $C(\text{eff}) \approx (C(XIN) + 2 \text{ pF}) / 2$ . It is assumed that $C(XIN) = C(XOUT)$ and that a parasitic capacitance of 2 pF is added by the package and the printed circuit board. For details about the typical internal and the effective capacitors, see the device-specific data sheet.
1	SMCLKOFF	RW	0h	SMCLK off. This bit turns off the SMCLK. 0b = SMCLK on 1b = SMCLK off
0	XT1OFF	RW	1h	XT1 off. This bit turns off the XT1. 0b = XT1 is on if XT1 is selected via 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.

### 5.4.8 UCSCTL7 Register

Unified Clock System Control 7 Register

**Figure 5-13. UCSCTL7 Register**

15	14	13	12	11	10	9	8
Reserved		Reserved		Reserved		Reserved	
r0	r0	rw-0	rw-(0)	rw-(1)	rw-(1)	r-1	r-1
7	6	5	4	3	2	1	0
Reserved		Reserved		XT2OFFG <sup>(1)</sup>	XT1HFOFFG <sup>(1)</sup>	XT1LFOFFG	DCOFFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(1)	rw-(1)

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

**Table 5-10. UCSCTL7 Register Description**

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0.
13-12	Reserved	RW	0h	Reserved. Must always be written with 0.
11-10	Reserved	RW	3h	Reserved. The states of these bits should be ignored.
9-8	Reserved	R	3h	Reserved. The states of these bits should be ignored.
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4	Reserved	RW	0h	Reserved. The state of this bit should be ignored.
3	XT2OFFG <sup>(1)</sup>	RW	0h	XT2 oscillator fault flag. If this bit is set, the OFIFG flag is also set. XT2OFFG is set if a XT2 fault condition exists. XT2OFFG can be cleared via software. If the XT2 fault condition still remains, XT2OFFG is set. 0b = No fault condition occurred after the last reset. 1b = XT2 fault. An XT2 fault occurred after the last reset.
2	XT1HFOFFG <sup>(1)</sup>	RW	0h	XT1 oscillator fault flag (HF mode). If this bit is set, the OFIFG flag is also set. XT1HFOFFG is set if a XT1 fault condition exists. XT1HFOFFG can be cleared via software. If the XT1 fault condition still remains, XT1HFOFFG is set. 0b = No fault condition occurred after the last reset. 1b = XT1 fault. An XT1 fault occurred after the last reset.
1	XT1LFOFFG	RW	1h	XT1 oscillator fault flag (LF mode). If this bit is set, the OFIFG flag is also set. XT1LFOFFG is set if a XT1 fault condition exists. XT1LFOFFG can be cleared via software. If the XT1 fault condition still remains, XT1LFOFFG is set. 0b = No fault condition occurred after the last reset. 1b = XT1 fault (LF mode). A XT1 fault occurred after the last reset.
0	DCOFFG	RW	1h	DCO fault flag. If this bit is set, the OFIFG flag is also set. The DCOFFG bit is set if DCO = {0} or DCO = {31}. DCOFFG can be cleared via software. If the DCO fault condition still remains, DCOFFG is set. 0b = No fault condition occurred after the last reset. 1b = DCO fault. A DCO fault occurred after the last reset.

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

### **5.4.9 UCSCTL8 Register**

Unified Clock System Control 8 Register

**Figure 5-14. UCSCTL8 Register**

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

**Table 5-11. UCSCTL8 Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	Reserved	R	0h	Reserved. Must always be written as 1.
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4	Reserved	R	0h	Reserved. Must always be written 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.

### 5.4.10 UCSCTL9 Register

Unified Clock System Control 9 Register

This register is not available on all devices. See the device-specific data sheet.

**Figure 5-15. UCSCTL9 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							
r0	r0	r0	r0	r0	r0	rw-0	rw-0

**Table 5-12. UCSCTL9 Register Description**

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved. Always reads as 0.
1	XT2BYPASSLV	RW	0h	Selects XT2 bypass input swing level. Must be set for reduced swing operation. 0b = Input range from 0 to DVCC 1b = Input range from 0 to DVIO
0	XT1BYPASSLV	RW	0h	Selects XT1 bypass input swing level. Must be set for reduced swing operation. 0b = Input range from 0 to DVCC 1b = Input range from 0 to DVIO

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

---

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

---

Topic	Page
<b>6.1 MSP430X CPU (CPUX) Introduction .....</b>	<b>186</b>
<b>6.2 Interrupts .....</b>	<b>188</b>
<b>6.3 CPU Registers .....</b>	<b>189</b>
<b>6.4 Addressing Modes .....</b>	<b>195</b>
<b>6.5 MSP430 and MSP430X Instructions .....</b>	<b>212</b>
<b>6.6 Instruction Set Description .....</b>	<b>228</b>

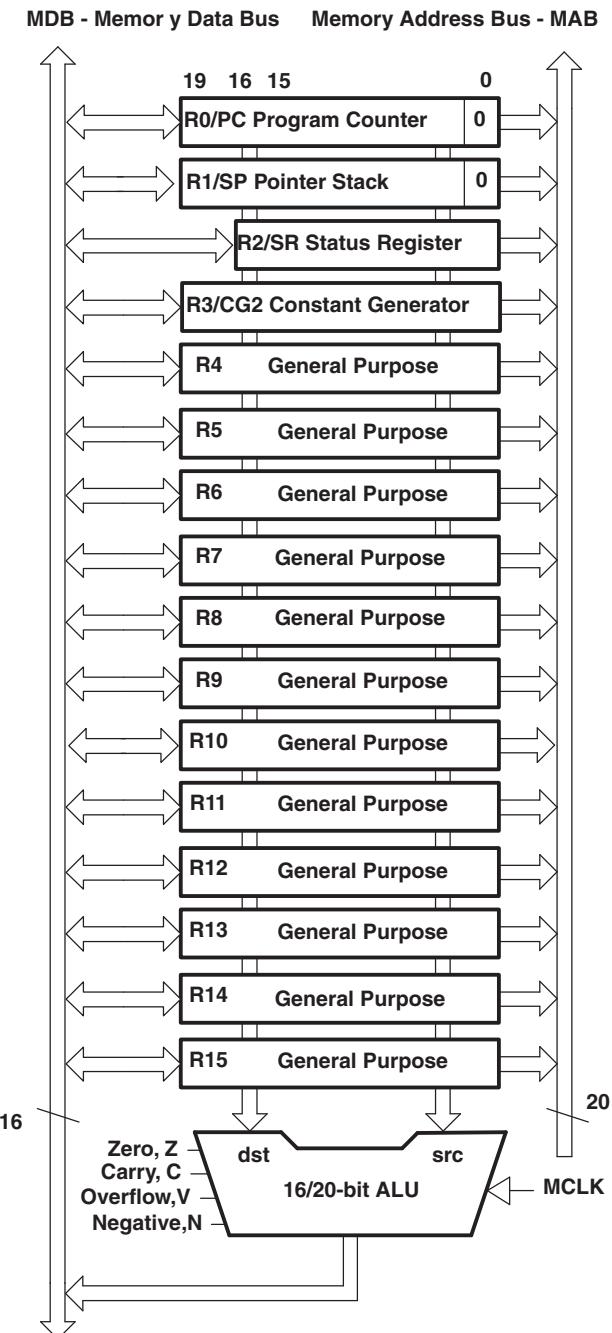
## 6.1 MSP430X CPU (CPUX) Introduction

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 6-1](#).



**Figure 6-1. MSP430X CPU Block Diagram**

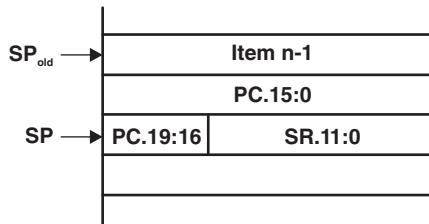
## 6.2 Interrupts

The MSP430X has the following interrupt structure:

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

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

During an interrupt, the program counter (PC) and the status register (SR) are pushed onto the stack as shown in [Figure 6-2](#). 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.



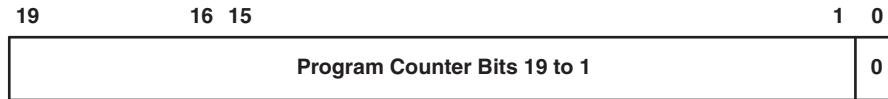
**Figure 6-2. PC Storage on the Stack for Interrupts**

## 6.3 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.

### 6.3.1 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 6-3](#) shows the PC.



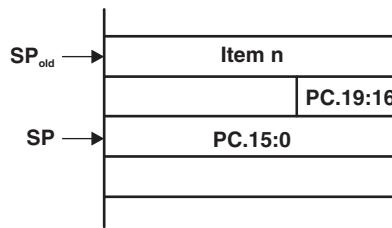
**Figure 6-3. Program Counter**

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

```
MOV.W #LABEL,PC ; Branch to address LABEL (lower 64 KB)  
  
MOVA #LABEL,PC ; Branch to address LABEL (1MB memory)  
  
MOV.W LABEL,PC ; Branch to address in word LABEL  
; (lower 64 KB)  
  
MOV.W @R14,PC ; Branch indirect to address in  
; R14 (lower 64 KB)  
  
ADDA #4,PC ; Skip two words (1 MB memory)
```

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

The PC is automatically stored on the stack with CALL (or CALLA) instructions and during an interrupt service routine. [Figure 6-4](#) 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.



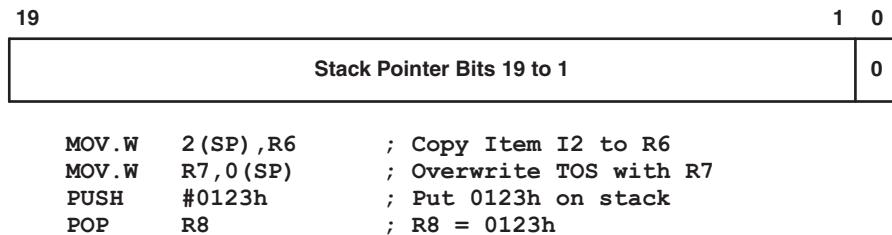
**Figure 6-4. PC Storage on the Stack for CALLA**

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.

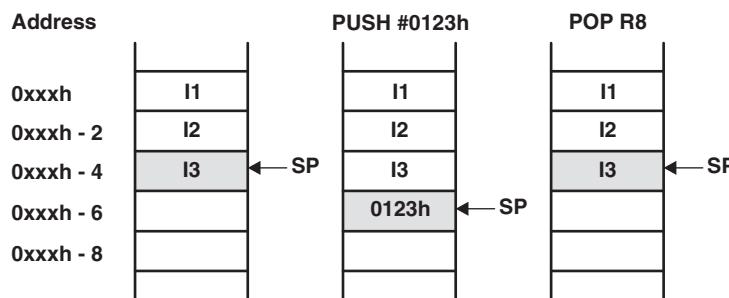
### 6.3.2 Stack Pointer (SP)

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 6-5](#) shows the SP. The SP is initialized into RAM by the user, and is always aligned to even addresses.

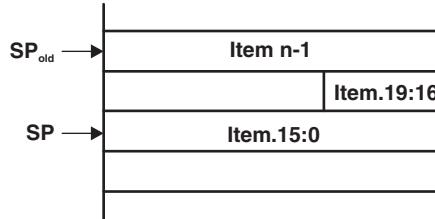
Figure 6-6 shows the stack usage. Figure 6-7 shows the stack usage when 20-bit address words are pushed.



**Figure 6-5. Stack Pointer**



**Figure 6-6. Stack Usage**



**Figure 6-7. 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 6-8.



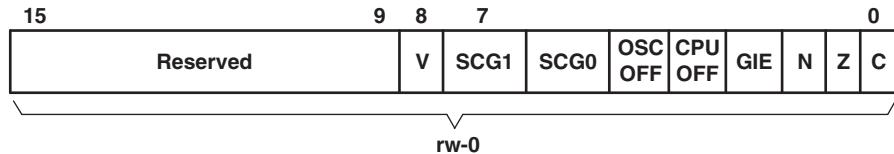
The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction. The POP SP instruction places SP1 into the stack pointer SP ( $SP_2 = SP_1$ )

**Figure 6-8. PUSH SP, POP SP Sequence**

### 6.3.3 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 6-9 shows the SR bits. Do not write 20-bit values to the SR. Unpredictable operation can result.



**Figure 6-9. SR Bits**

Table 6-1 describes the SR bits.

**Table 6-1. 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  SUB(.B), SUBX(.B,.A), SUBC(.B), SUBCX(.B,.A), SUBA, CMP(.B), CMPX(.B,.A), CMPA
SCG1	System clock generator 1. This bit may be used to enable or disable functions in the clock system depending on the device family; for example, DCO bias enable or disable.
SCG0	System clock generator 0. This bit may be used to enable or disable functions in the clock system depending on the device family; for example, FLL enable or disable.
OSCOFF	Oscillator off. This bit, when set, turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK.
CPUOFF	CPU off. This bit, when set, 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. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.
N	Negative. This bit is set when the result of an operation is negative and cleared when the result is positive.
Z	Zero. This bit is set when the result of an operation is 0 and cleared when the result is not 0.
C	Carry. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.

---

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

---

### 6.3.4 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 6-2](#).

**Table 6-2. 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.

#### 6.3.4.1 Constant Generator – Expanded Instruction Set

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

CLR dst

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

MOV R3,dst

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

INC dst

is replaced by:

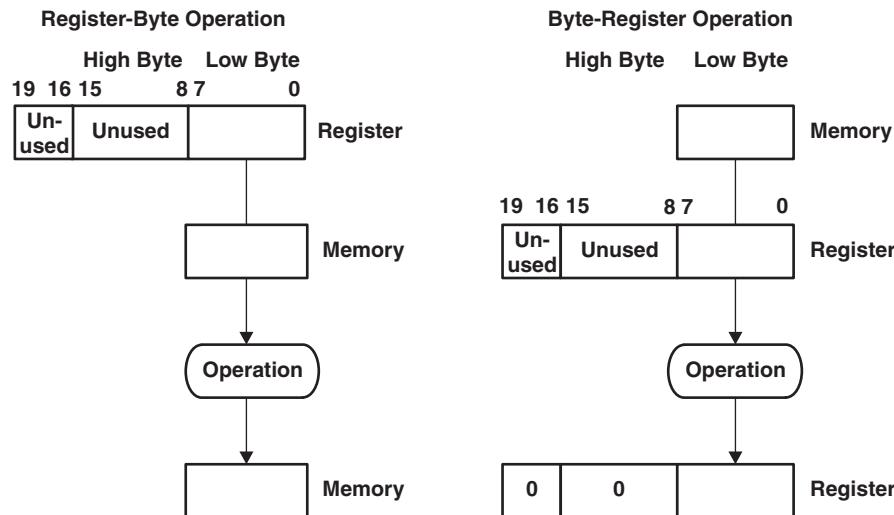
ADD 0(R3),dst

### 6.3.5 General-Purpose Registers (R4 –R15)

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

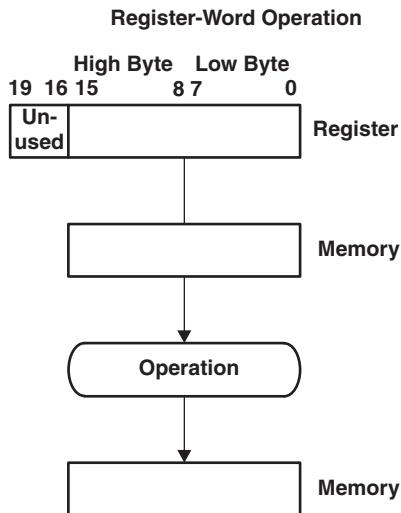
[Figure 6-10](#) through [Figure 6-14](#) show the handling of byte, word, and address-word data. Note the reset of the leading most significant bits (MSBs) if a register is the destination of a byte or word instruction.

[Figure 6-10](#) 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.

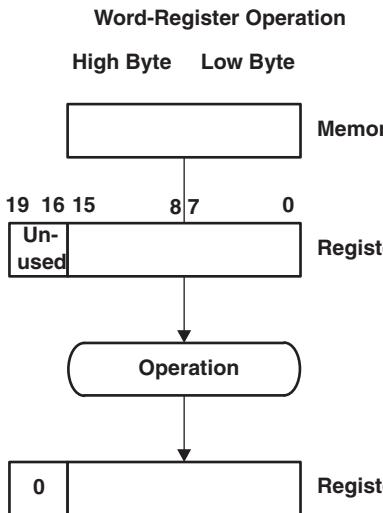


**Figure 6-10. Register-Byte and Byte-Register Operation**

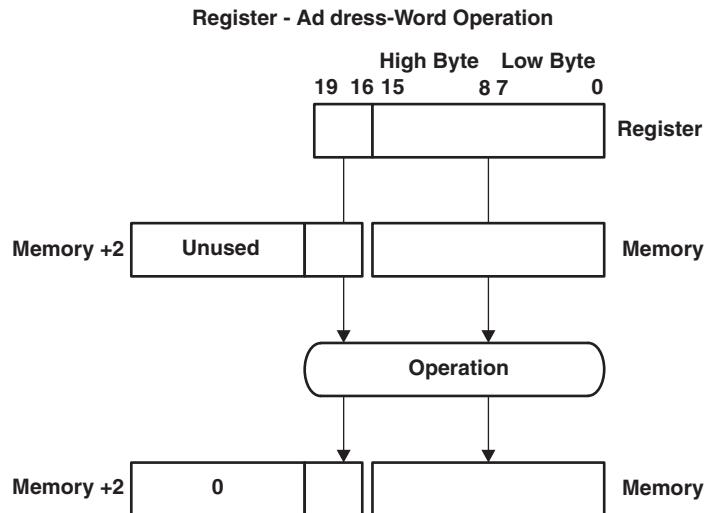
[Figure 6-11](#) and [Figure 6-12](#) show 16-bit word handling (.W suffix). The handling is shown for a source register and a destination memory word and for a source memory word and a destination register.

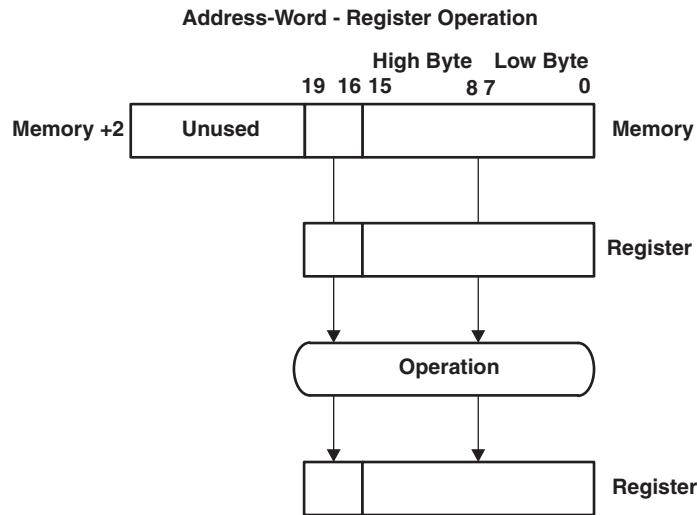


**Figure 6-11. Register-Word Operation**

**Figure 6-12. Word-Register Operation**

[Figure 6-13](#) and [Figure 6-14](#) 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 6-13. Register – Address-Word Operation**



**Figure 6-14. Address-Word – Register Operation**

## 6.4 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 6-3](#)). The MSP430 and MSP430X instructions are usable throughout the entire 1MB memory range.

**Table 6-3. 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.

### 6.4.1 Register Mode

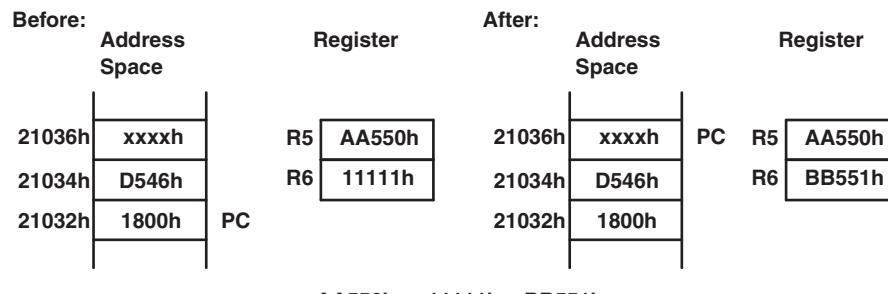
Operation:	The operand is the 8-, 16-, or 20-bit content of the used CPU register.
Length:	One, two, or three words
Comment:	Valid for source and destination
Byte operation:	Byte operation reads only the eight least significant bits (LSBs) of the source register Rsrc and writes the result to the eight 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:	<p>BIS.W R5 ,R6 ;</p> <p>This instruction logically ORs the 16-bit data contained in R5 with the 16-bit contents of R6. R6.19:16 is cleared.</p>



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:



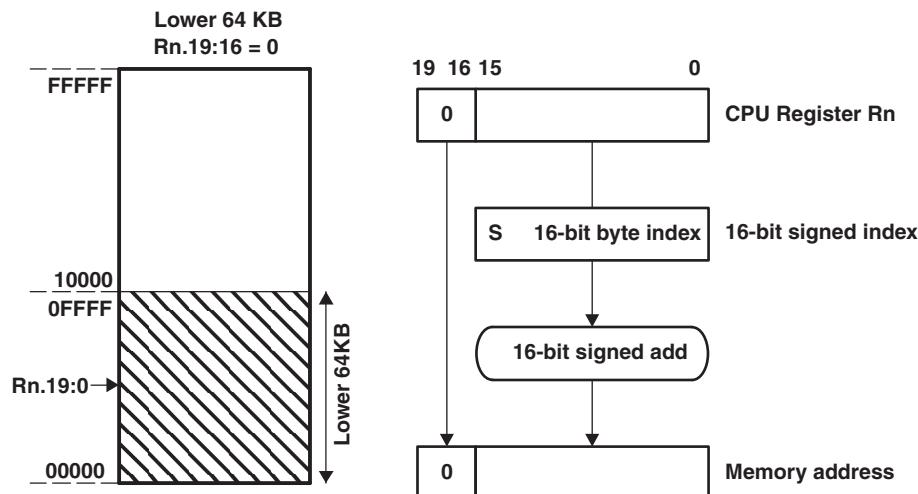
## 6.4.2 Indexed Mode

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

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

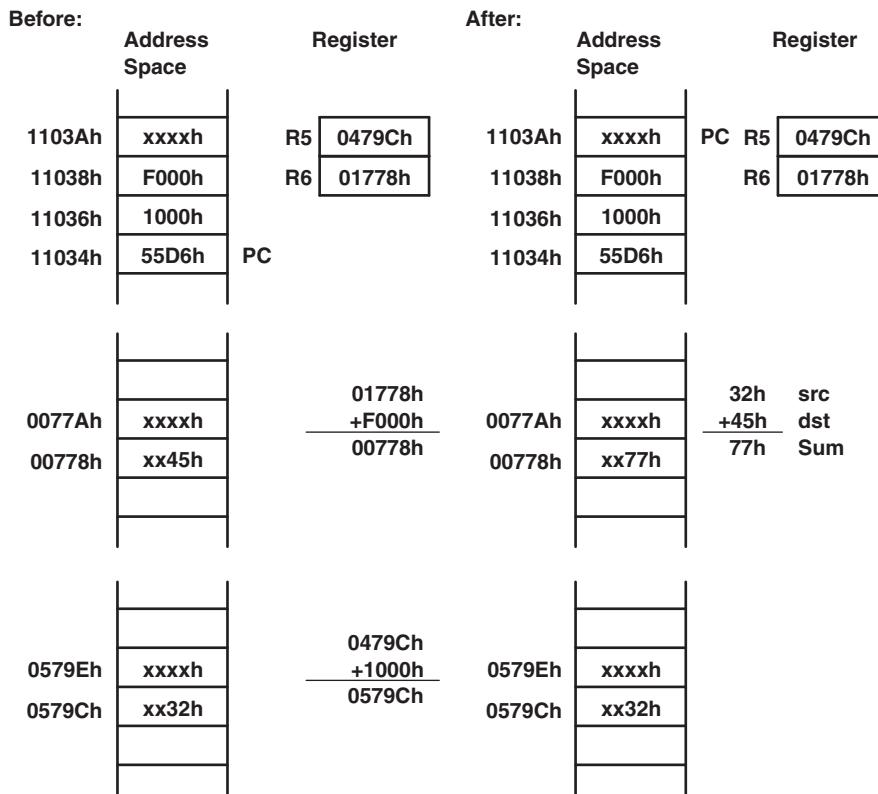
### 6.4.2.1 Indexed Mode in Lower 64-KB Memory

If the CPU register Rn points to an address in the lower 64 KB of the memory range, the calculated memory address bits 19:16 are cleared after the addition of the CPU register Rn and the signed 16-bit index. This means the calculated memory address is always located in the lower 64 KB and does not overflow or underflow out of the lower 64-KB memory space. The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications as shown in [Figure 6-15](#).



**Figure 6-15. Indexed Mode in Lower 64 KB**

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



#### 6.4.2.2 MSP430 Instruction With Indexed Mode in Upper Memory

If the CPU register Rn points to an address above the lower 64-KB memory, the Rn bits 19:16 are used for the address calculation of the operand. The operand may be located in memory in the range  $Rn \pm 32$  KB, because the index, X, is a signed 16-bit value. In this case, the address of the operand can overflow or underflow into the lower 64-KB memory space (see [Figure 6-16](#) and [Figure 6-17](#)).

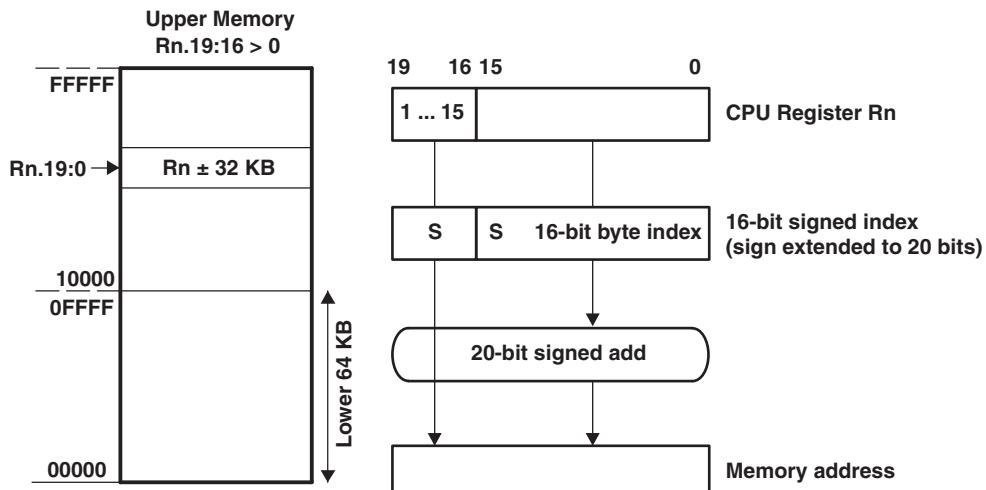
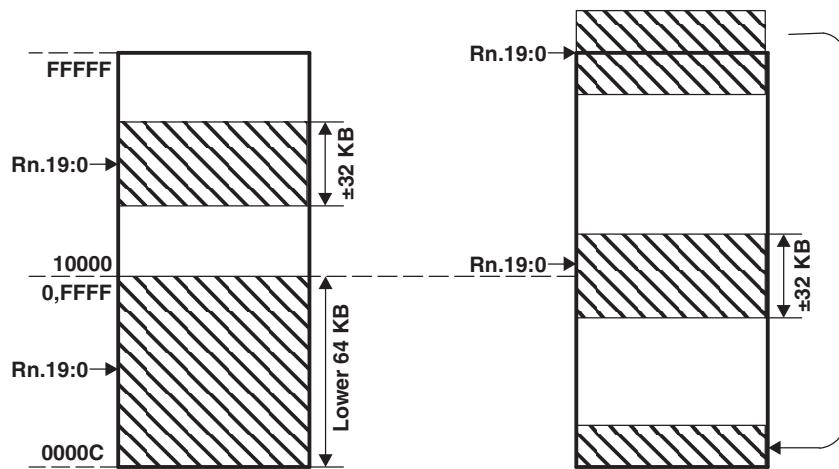


Figure 6-16. Indexed Mode in Upper Memory



**Figure 6-17. 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.

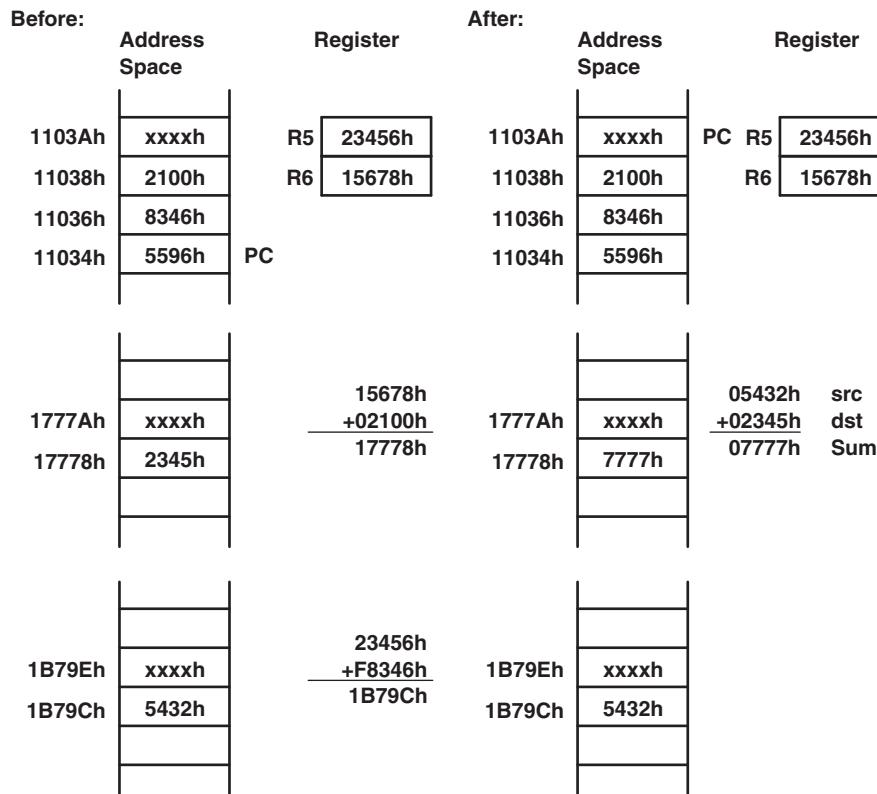


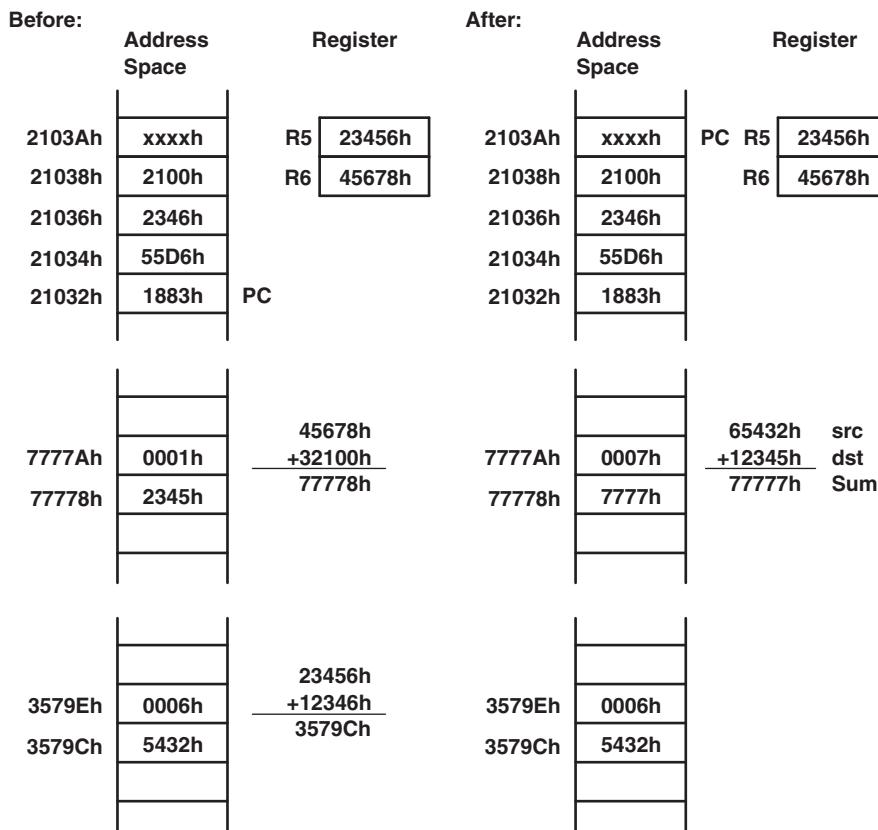
Figure 6-18. Example for Indexed Mode

#### 6.4.2.3 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.



#### **6.4.3 Symbolic Mode**

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

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

##### **6.4.3.1 Symbolic Mode in Lower 64 KB**

If the PC points to an address in the lower 64 KB of the memory range, the calculated memory address bits 19:16 are cleared after the addition of the PC and the signed 16-bit index. This means the calculated memory address is always located in the lower 64 KB and does not overflow or underflow out of the lower 64-KB memory space. The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications as shown in [Figure 6-19](#).

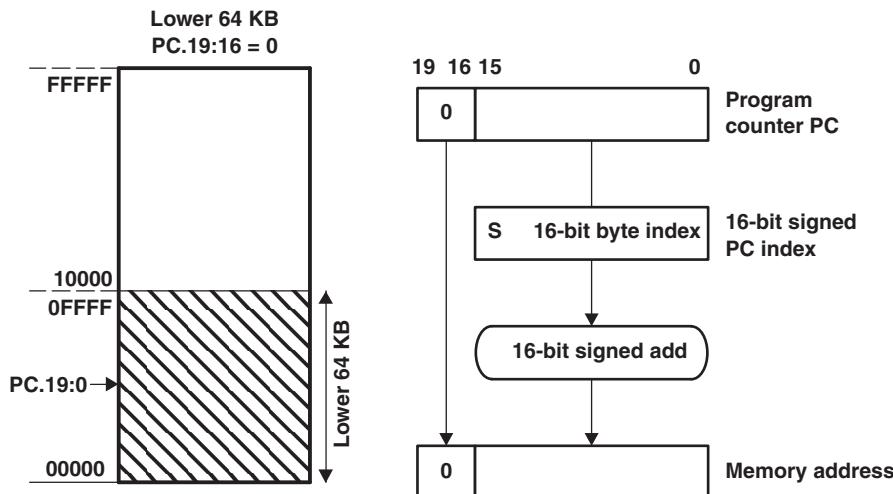
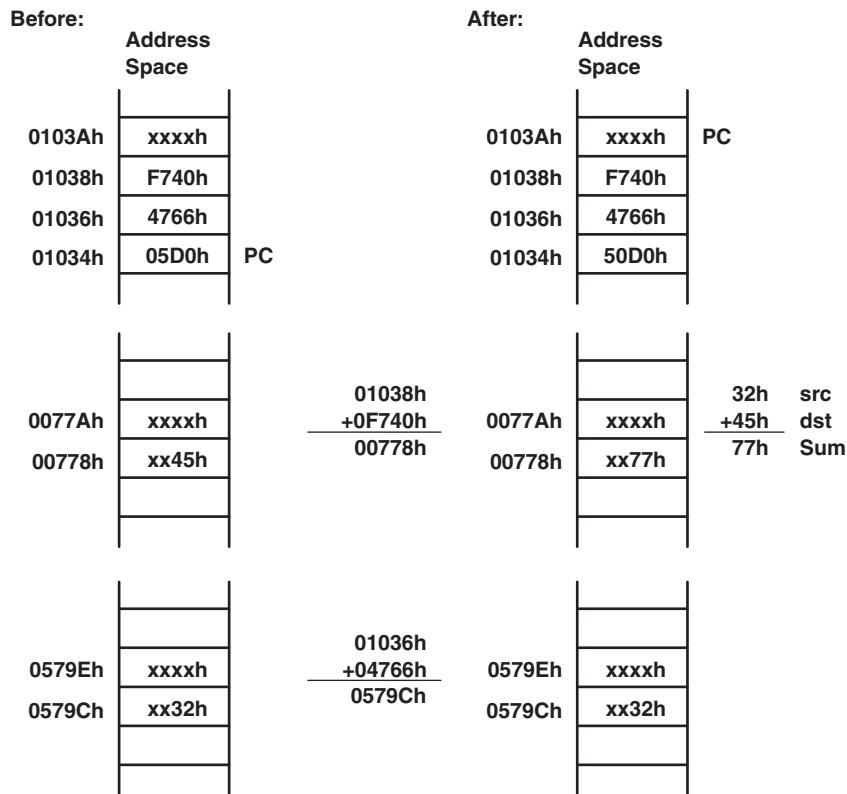


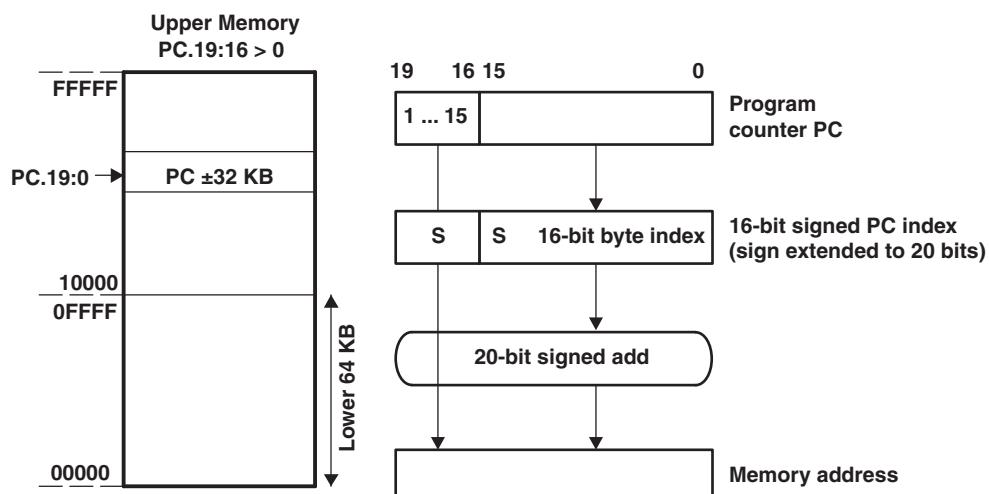
Figure 6-19. Symbolic Mode Running in Lower 64 KB

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



#### 6.4.3.2 MSP430 Instruction With Symbolic Mode in Upper Memory

If the PC points to an address above the lower 64-KB memory, the PC bits 19:16 are used for the address calculation of the operand. The operand may be located in memory in the range  $PC \pm 32\text{ KB}$ , because the index, X, is a signed 16-bit value. In this case, the address of the operand can overflow or underflow into the lower 64-KB memory space as shown in [Figure 6-20](#) and [Figure 6-21](#).



**Figure 6-20. Symbolic Mode Running in Upper Memory**

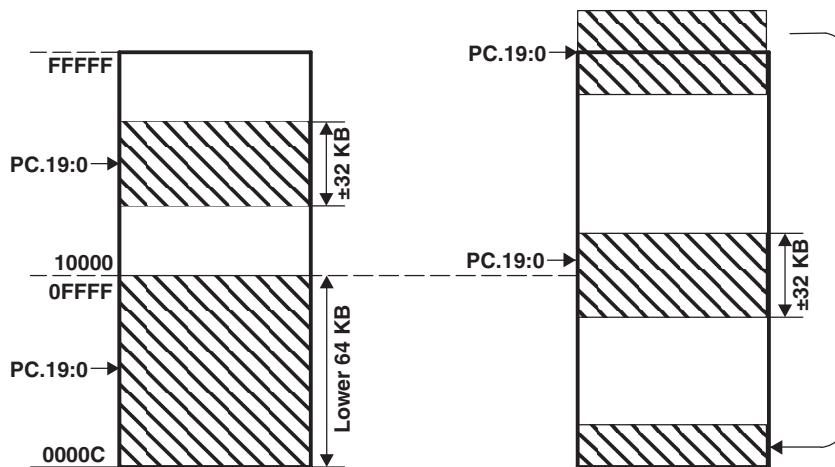
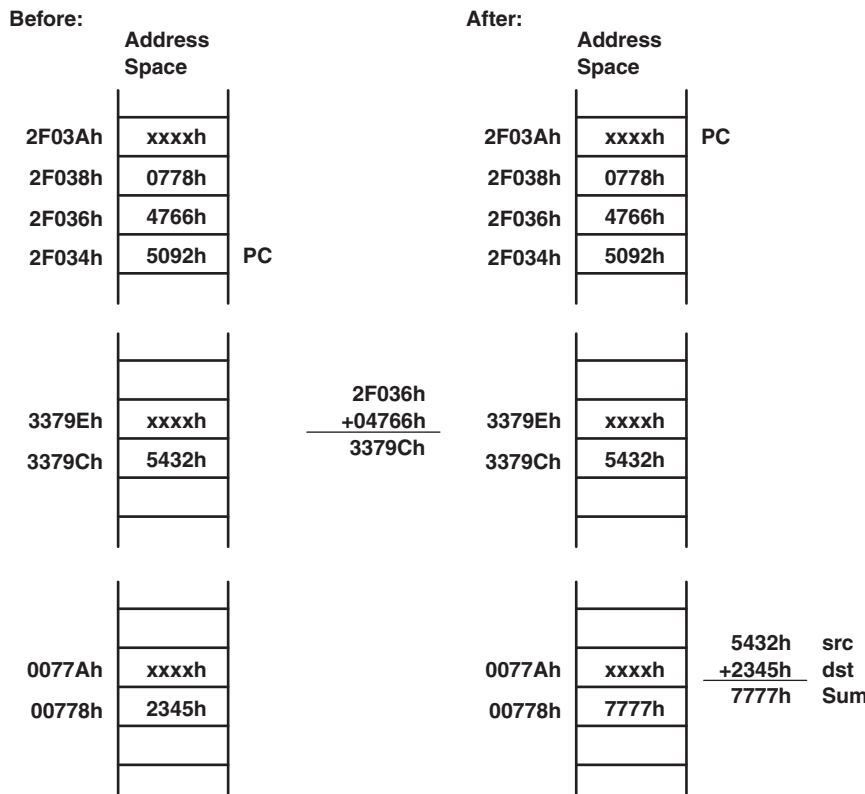


Figure 6-21. Overflow and Underflow for Symbolic Mode

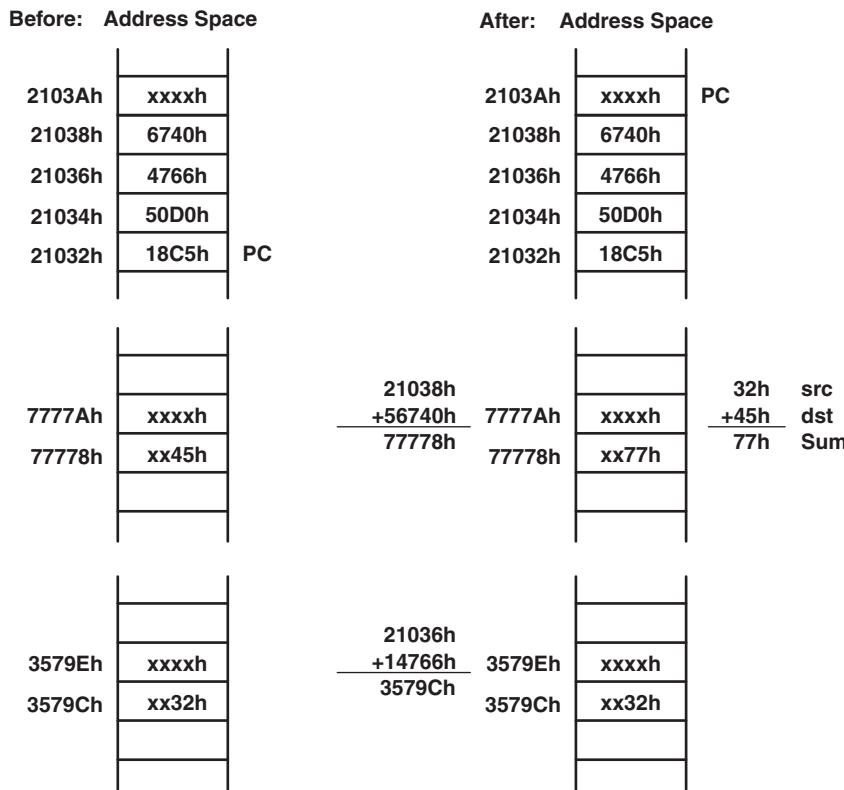
Length:	Two or three words
Operation:	The sign-extended 16-bit index in the next word after the instruction is added to the 20 bits of the PC. This delivers a 20-bit address, which points to an address in the range 0 to FFFFh. The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the PC index and inserts it
Example:	<pre>ADD.W EDE ,&amp;TONI ;</pre> <p>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.</p>
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



#### 6.4.3.3 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.



#### 6.4.4 Absolute Mode

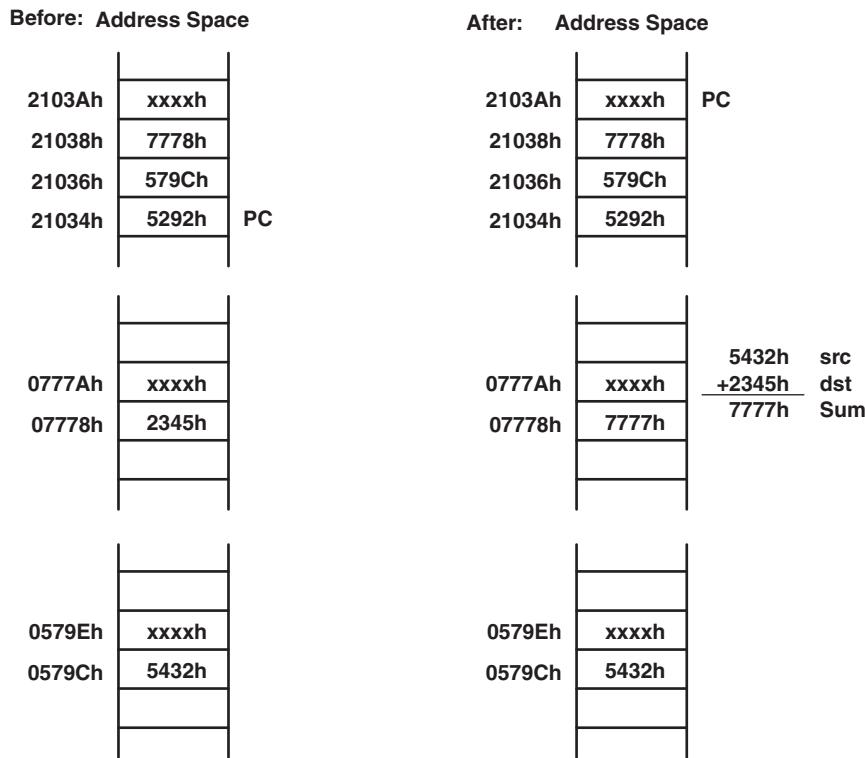
The Absolute mode uses the contents of the word following the instruction as the address of the operand. The Absolute mode has two addressing possibilities:

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

##### 6.4.4.1 Absolute Mode in Lower 64 KB

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

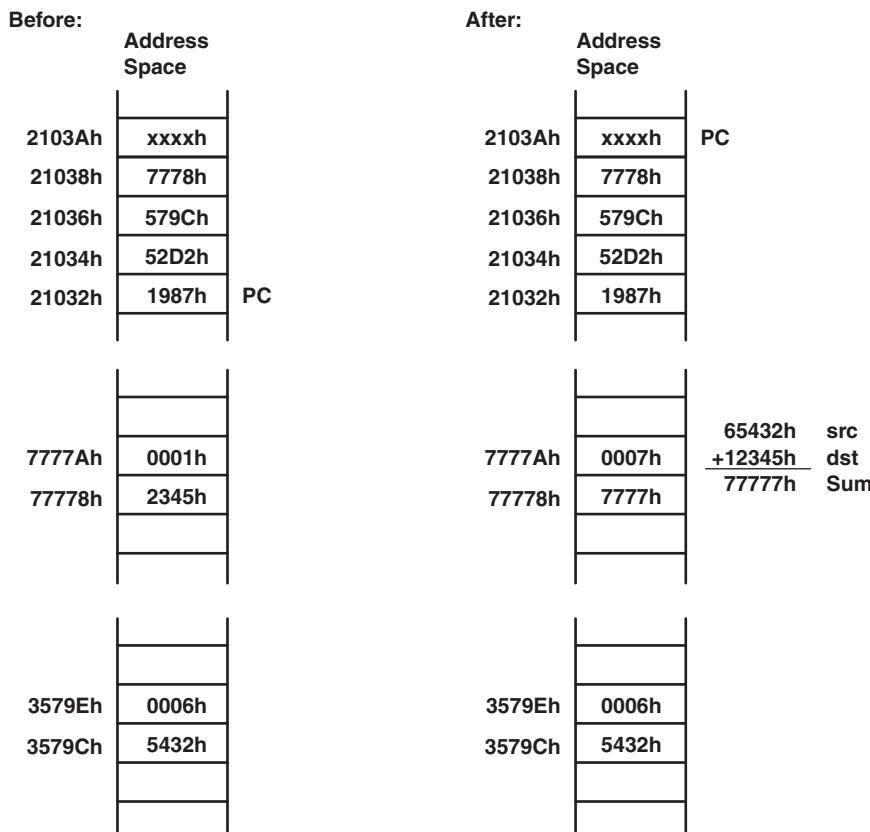
Length:	Two or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	ADD.W &EDE ,&TONI ; This instruction adds the 16-bit data contained in the absolute source and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



#### 6.4.4.2 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



#### 6.4.5 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)

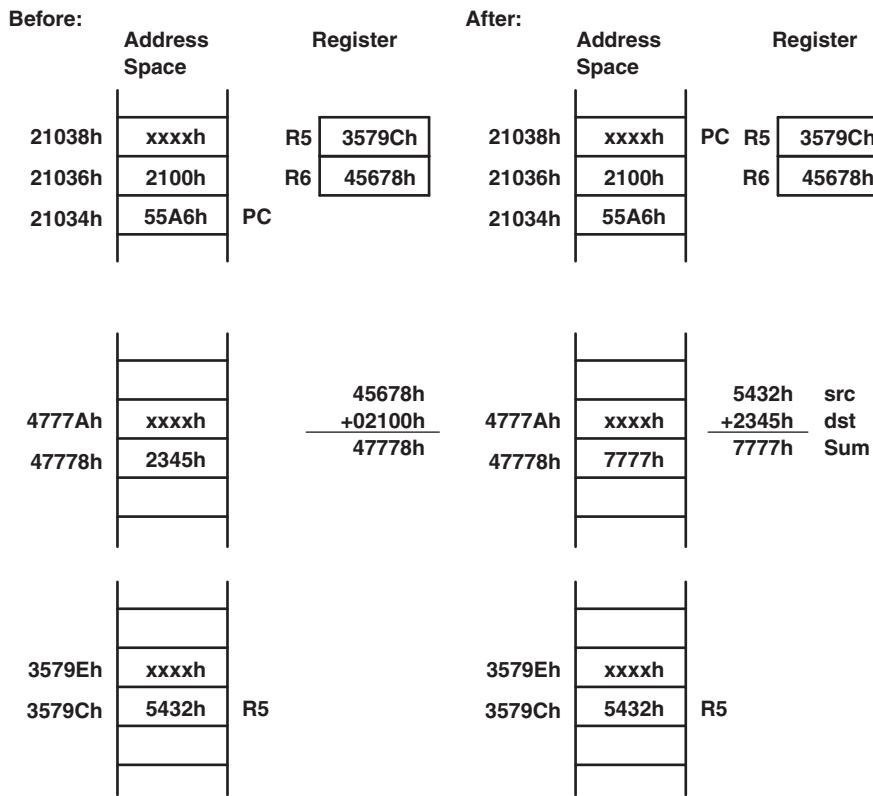
This instruction adds the two 16-bit operands contained in the source and the destination addresses and places the result into the destination.

Source:

Word pointed to by R5. R5 contains address 3579Ch for this example.

Destination:

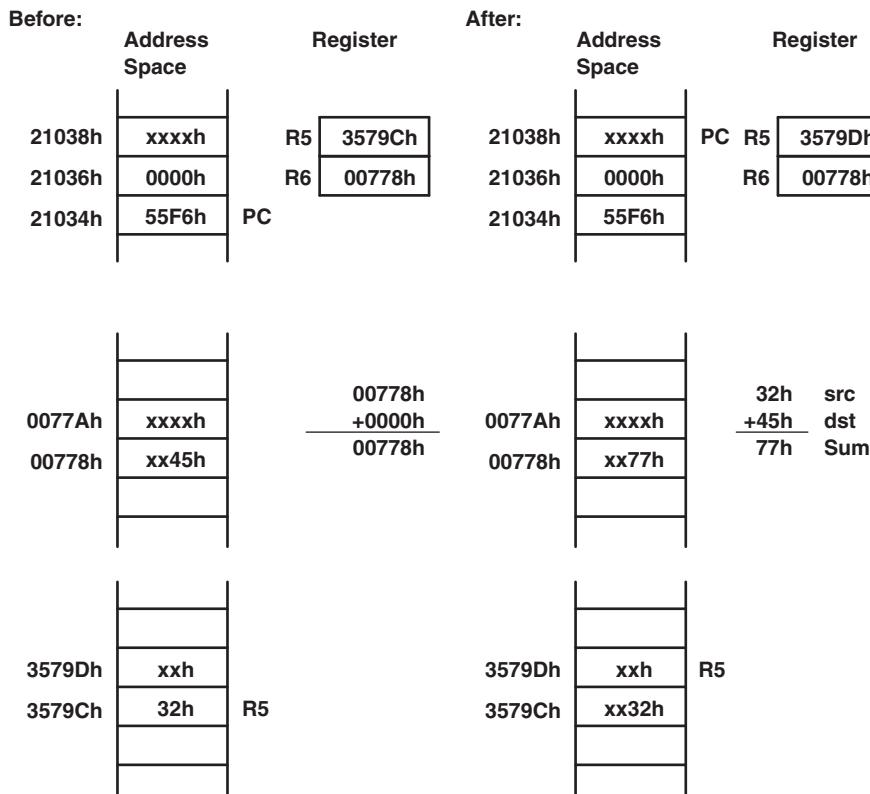
Word pointed to by R6 + 2100h, which results in address 45678h + 2100h = 7778h



#### 6.4.6 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



#### 6.4.7 Immediate Mode

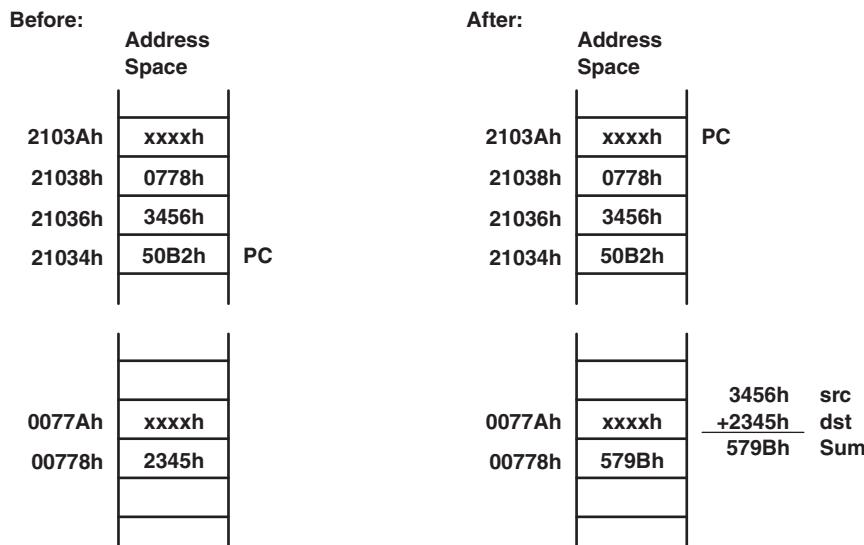
The Immediate mode allows accessing constants as operands by including the constant in the memory location following the instruction. The PC is used with the Indirect Autoincrement mode. The PC points to the immediate value contained in the next word. After the fetching of the immediate operand, the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

##### 6.4.7.1 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.

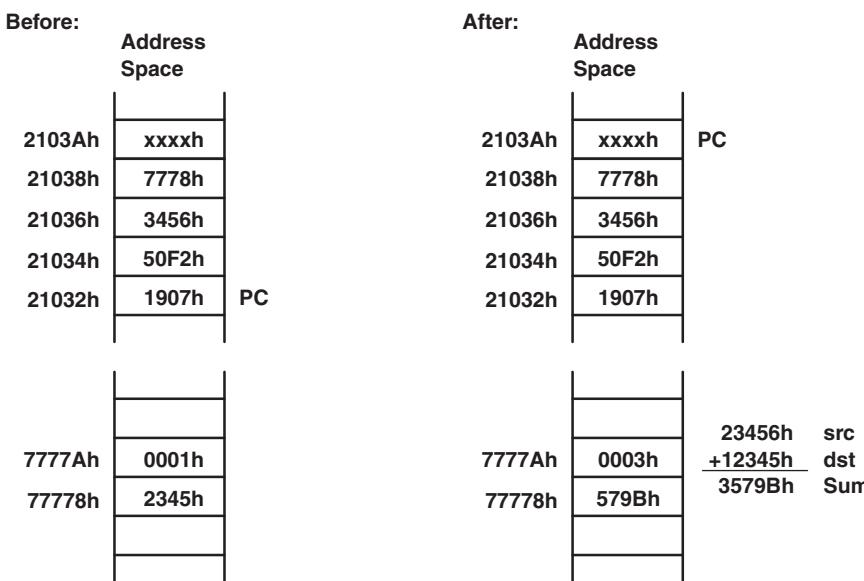
Length:	Two or three words. One word less if a constant of the constant generator can be used for the immediate operand.
Operation:	The 16-bit immediate source operand is used together with the 16-bit destination operand.
Comment:	Valid only for the source operand
Example:	ADD #3456h,&TONI This instruction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



#### 6.4.7.2 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



## 6.5 MSP430 and MSP430X Instructions

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

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

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

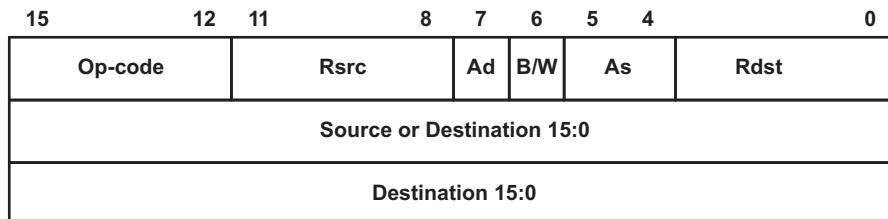
[Section 6.5.1](#) lists and describes the MSP430 instructions, and [Section 6.5.2](#) lists and describes the MSP430X instructions.

### 6.5.1 MSP430 Instructions

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

#### 6.5.1.1 MSP430 Double-Operand (Format I) Instructions

[Figure 6-22](#) shows the format of the MSP430 double-operand instructions. Source and destination words are appended for the Indexed, Symbolic, Absolute, and Immediate modes. [Table 6-4](#) lists the 12 MSP430 double-operand instructions.



**Figure 6-22. MSP430 Double-Operand Instruction Format**

**Table 6-4. MSP430 Double-Operand Instructions**

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

<sup>(1)</sup> \* = Status bit is affected.

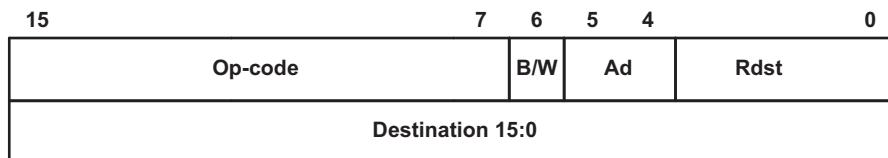
— = Status bit is not affected.

0 = Status bit is cleared.

1 = Status bit is set.

### 6.5.1.2 MSP430 Single-Operand (Format II) Instructions

Figure 6-23 shows the format for MSP430 single-operand instructions, except RETI. The destination word is appended for the Indexed, Symbolic, Absolute, and Immediate modes. Table 6-5 lists the seven single-operand instructions.


**Figure 6-23. MSP430 Single-Operand Instructions**
**Table 6-5. MSP430 Single-Operand Instructions**

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

<sup>(1)</sup> \* = Status bit is affected.

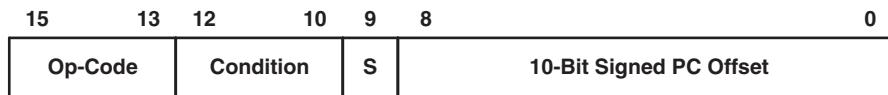
— = Status bit is not affected.

0 = Status bit is cleared.

1 = Status bit is set.

### 6.5.1.3 Jump Instructions

Figure 6-24 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 6-6 lists and describes the eight jump instructions.



**Figure 6-24. Format of Conditional Jump Instructions**

**Table 6-6. 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

### 6.5.1.4 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 6-7.

**Table 6-7. Emulated Instructions**

Instruction	Explanation	Emulation	Status Bits <sup>(1)</sup>			
			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 decimaly	DADD(.B) #0,dst	*	*	*	*
DEC(.B) dst	Decrement dst by 1	SUB(.B) #1,dst	*	*	*	*
DECD(.B) dst	Decrement dst by 2	SUB(.B) #2,dst	*	*	*	*
DINT	Disable interrupt	BIC #8,SR	-	-	-	-
EINT	Enable interrupt	BIS #8,SR	-	-	-	-
INC(.B) dst	Increment dst by 1	ADD(.B) #1,dst	*	*	*	*
INCD(.B) dst	Increment dst by 2	ADD(.B) #2,dst	*	*	*	*
INV(.B) dst	Invert dst	XOR(.B) #-1,dst	*	*	*	*

<sup>(1)</sup> \* = Status bit is affected.

– = Status bit is not affected.

0 = Status bit is cleared.

1 = Status bit is set.

**Table 6-7. Emulated Instructions (continued)**

<b>Instruction</b>	<b>Explanation</b>	<b>Emulation</b>	<b>Status Bits <sup>(1)</sup></b>			
			<b>V</b>	<b>N</b>	<b>Z</b>	<b>C</b>
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

### 6.5.1.5 MSP430 Instruction Execution

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

#### Instruction Cycles and Length for Interrupt, Reset, and Subroutines

Table 6-8 lists the length and the CPU cycles for reset, interrupts, and subroutines.

**Table 6-8. Interrupt, Return, and Reset Cycles and Length**

<b>Action</b>	<b>Execution Time (MCLK Cycles)</b>	<b>Length of Instruction (Words)</b>
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 (RST/NMI)	4	–

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

Table 6-9 lists the length and the CPU cycles for all addressing modes of the MSP430 single-operand instructions.

**Table 6-9. MSP430 Format II Instruction Cycles and Length**

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

**Jump Instructions Cycles and Lengths**

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

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

[Table 6-10](#) lists the length and CPU cycles for all addressing modes of the MSP430 Format I instructions.

**Table 6-10. 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 <sup>(1)</sup>	2	ADD R5 , 4(R6)
	EDE	4 <sup>(1)</sup>	2	XOR R8 , EDE
	&EDE	4 <sup>(1)</sup>	2	MOV R5 , &EDE
@Rn	Rm	2	1	AND @R4 ,R5
	PC	4	1	BR @R8
	x(Rm)	5 <sup>(1)</sup>	2	XOR @R5 , 8(R6)
	EDE	5 <sup>(1)</sup>	2	MOV @R5 , EDE
	&EDE	5 <sup>(1)</sup>	2	XOR @R5 ,&EDE
@Rn+	Rm	2	1	ADD @R5+ ,R6
	PC	4	1	BR @R9+
	x(Rm)	5 <sup>(1)</sup>	2	XOR @R5 , 8(R6)
	EDE	5 <sup>(1)</sup>	2	MOV @R9+ ,EDE
	&EDE	5 <sup>(1)</sup>	2	MOV @R9+ ,&EDE
#N	Rm	2	2	MOV #20 ,R9
	PC	3	2	BR #2AEh
	x(Rm)	5 <sup>(1)</sup>	3	MOV #0300h ,0(SP)
	EDE	5 <sup>(1)</sup>	3	ADD #33 ,EDE
	&EDE	5 <sup>(1)</sup>	3	ADD #33 ,&EDE
x(Rn)	Rm	3	2	MOV 2(R5) ,R7
	PC	5	2	BR 2(R6)
	TONI	6 <sup>(1)</sup>	3	MOV 4(R7) ,TONI
	x(Rm)	6 <sup>(1)</sup>	3	ADD 4(R4) ,6(R9)
	&TONI	6 <sup>(1)</sup>	3	MOV 2(R4) ,&TONI
EDE	Rm	3	2	AND EDE ,R6
	PC	5	2	BR EDE
	TONI	6 <sup>(1)</sup>	3	CMP EDE ,TONI
	x(Rm)	6 <sup>(1)</sup>	3	MOV EDE ,0(SP)
	&TONI	6 <sup>(1)</sup>	3	MOV EDE ,&TONI
&EDE	Rm	3	2	MOV &EDE ,R8
	PC	5	2	BR &EDE
	TONI	6 <sup>(1)</sup>	3	MOV &EDE ,TONI
	x(Rm)	6 <sup>(1)</sup>	3	MOV &EDE ,0(SP)
	&TONI	6 <sup>(1)</sup>	3	MOV &EDE ,&TONI

<sup>(1)</sup> MOV, BIT, and CMP instructions execute in one fewer cycle.

## 6.5.2 MSP430X Extended Instructions

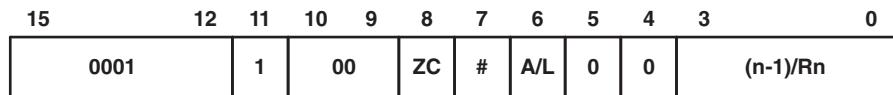
The extended MSP430X instructions give the MSP430X CPU full access to its 20-bit address space. Most MSP430X instructions require an additional word of op-code called the extension word. Some extended instructions do not require an additional word and are noted in the instruction description. All addresses,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

### 6.5.2.1 Register Mode Extension Word

The register mode extension word is shown in [Figure 6-25](#) and described in [Table 6-11](#). An example is shown in [Figure 6-27](#).



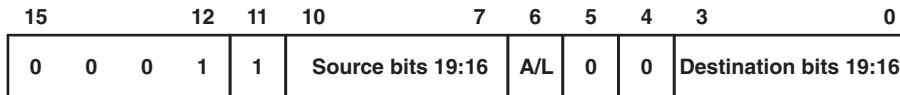
**Figure 6-25. Extension Word for Register Modes**

**Table 6-11. 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	
	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.	

### 6.5.2.2 Non-Register Mode Extension Word

The extension word for non-register modes is shown in [Figure 6-26](#) and described in [Table 6-12](#). An example is shown in [Figure 6-28](#).



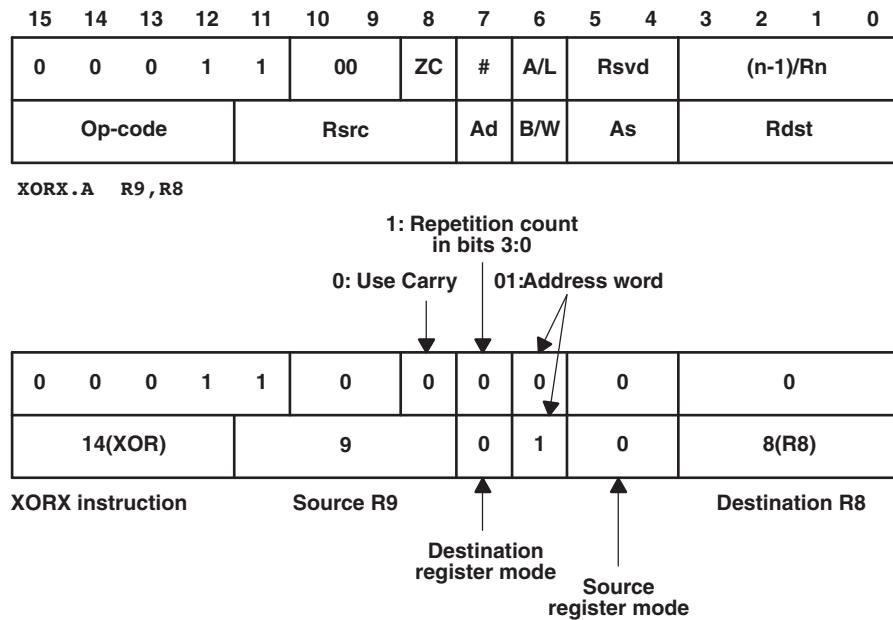
**Figure 6-26. Extension Word for Non-Register Modes**

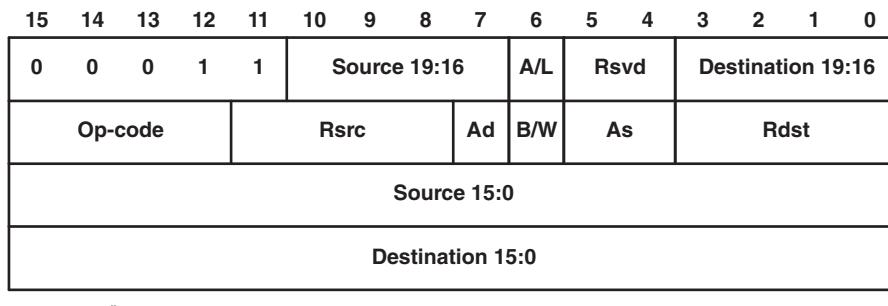
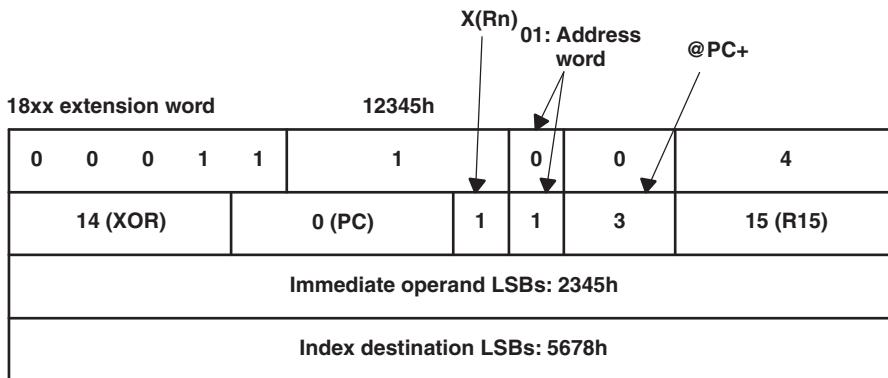
**Table 6-12. Description of Extension Word Bits for Non-Register Modes**

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.
	<b>A/L    B/W    Comment</b>
0    0	Reserved
0    1	20-bit address word
1    0	16-bit word
1    1	8-bit byte
5:4	Reserved
Destination Bits 19:16	The four MSBs of the 20-bit destination. Depending on the destination addressing mode, these four MSBs may belong to an index or to an absolute address.

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

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

**Figure 6-27. Example for Extended Register or Register Instruction**


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

**Figure 6-28. Example for Extended Immediate or Indexed Instruction**

### 6.5.2.3 Extended Double-Operand (Format I) Instructions

All 12 double-operand instructions have extended versions as listed in [Table 6-13](#).

**Table 6-13. Extended Double-Operand Instructions**

<b>Mnemonic</b>	<b>Operands</b>	<b>Operation</b>	<b>Status Bits <sup>(1)</sup></b>			
			<b>V</b>	<b>N</b>	<b>Z</b>	<b>C</b>
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	*	*	$\bar{Z}$
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	*	*	*	$\bar{Z}$
ANDX( .B , .A )	src,dst	src .and. dst → dst	0	*	*	$\bar{Z}$

<sup>(1)</sup> \* = 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 6-29.

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										
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 6-29. 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 6-30.

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 6-30. 20-Bit Addresses in Memory**

#### 6.5.2.4 Extended Single-Operand (Format II) Instructions

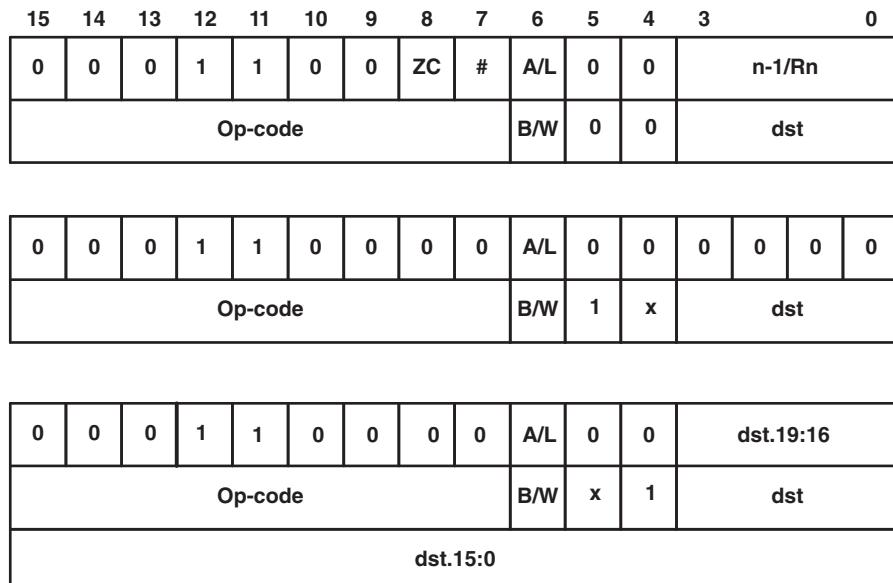
Extended MSP430X Format II instructions are listed in [Table 6-14](#).

**Table 6-14. Extended Single-Operand Instructions**

<b>Mnemonic</b>	<b>Operands</b>	<b>Operation</b>	<b>Status Bits <sup>(1)</sup></b>				
			<b>n</b>	<b>V</b>	<b>N</b>	<b>Z</b>	<b>C</b>
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	*	*	Z
SXTX(.A)	dst	Bit7 → bit8 ... MSB	1	0	*	*	Z

- <sup>(1)</sup> \* = Status bit is affected.  
 – = Status bit is not affected.  
 0 = Status bit is cleared.  
 1 = Status bit is set.

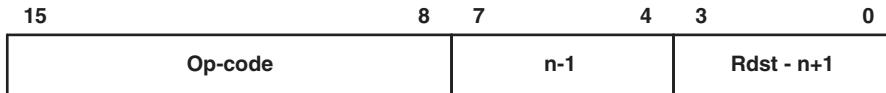
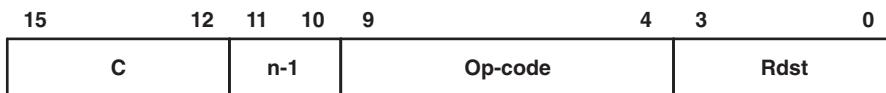
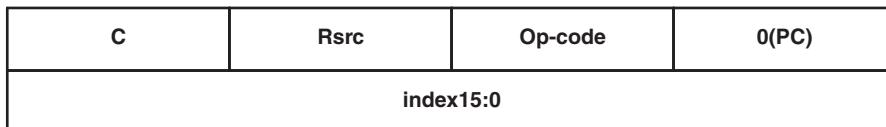
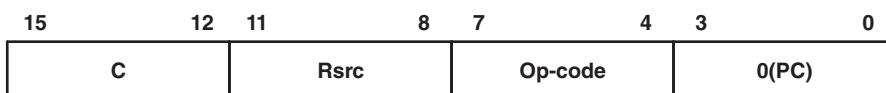
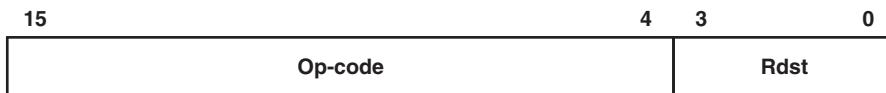
The three possible addressing mode combinations for Format II instructions are shown in [Figure 6-31](#).



**Figure 6-31. Extended Format II Instruction Format**

**Extended Format II Instruction Format Exceptions**

Exceptions for the Format II instruction formats are shown in [Figure 6-32](#) through [Figure 6-35](#).

**Figure 6-32. PUSHM and POPM Instruction Format****Figure 6-33. RRCM, RRAM, RRUM, and RLAM Instruction Format****Figure 6-34. BRA Instruction Format****Figure 6-35. CALLA Instruction Format**

### **6.5.2.5 Extended Emulated Instructions**

The extended instructions together with the constant generator form the extended emulated instructions. [Table 6-15](#) lists the emulated instructions.

**Table 6-15. Extended Emulated Instructions**

<b>Instruction</b>	<b>Explanation</b>	<b>Emulation</b>
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 decimaly	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

### 6.5.2.6 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 6-16](#). 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 6-16. Address Instructions, Operate on 20-Bit Register Data**

Mnemonic	Operands	Operation	Status Bits <sup>(1)</sup>			
			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	*	*	*	*

<sup>(1)</sup> \* = Status bit is affected.  
 – = Status bit is not affected.  
 0 = Status bit is cleared.  
 1 = Status bit is set.

### 6.5.2.7 MSP430X Instruction Execution

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

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

Table 6-17 lists the length and the CPU cycles for all addressing modes of the MSP430X extended single-operand instructions.

**Table 6-17. 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 <sup>(1)</sup> , 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 <sup>(1)</sup> , 3	5, 3	5, 3
PUSHX.A	5, 2	6, 2	6, 2	5, 3	7 <sup>(1)</sup> , 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

<sup>(1)</sup> Add one cycle when Rn = SP

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

**Table 6-18** lists the length and CPU cycles for all addressing modes of the MSP430X extended Format I instructions.

**Table 6-18. 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 <sup>(1)</sup>	2	2	2	BITX.B R5,R8
	PC	4	4	2	ADDX R9,PC
	x(Rm)	5 <sup>(2)</sup>	7 <sup>(3)</sup>	3	ANDX.A R5,4(R6)
	EDE	5 <sup>(2)</sup>	7 <sup>(3)</sup>	3	XORX R8,EDE
	&EDE	5 <sup>(2)</sup>	7 <sup>(3)</sup>	3	BITX.W R5,&EDE
@Rn	Rm	3	4	2	BITX @R5,R8
	PC	5	6	2	ADDX @R9,PC
	x(Rm)	6 <sup>(2)</sup>	9 <sup>(3)</sup>	3	ANDX.A @R5,4(R6)
	EDE	6 <sup>(2)</sup>	9 <sup>(3)</sup>	3	XORX @R8,EDE
	&EDE	6 <sup>(2)</sup>	9 <sup>(3)</sup>	3	BITX.B @R5,&EDE
@Rn+	Rm	3	4	2	BITX @R5+,R8
	PC	5	6	2	ADDX.A @R9+,PC
	x(Rm)	6 <sup>(2)</sup>	9 <sup>(3)</sup>	3	ANDX @R5+,4(R6)
	EDE	6 <sup>(2)</sup>	9 <sup>(3)</sup>	3	XORX.B @R8+,EDE
	&EDE	6 <sup>(2)</sup>	9 <sup>(3)</sup>	3	BITX @R5+,&EDE
#N	Rm	3	3	3	BITX #20,R8
	PC <sup>(4)</sup>	4	4	3	ADDX.A #FE000h,PC
	x(Rm)	6 <sup>(2)</sup>	8 <sup>(3)</sup>	4	ANDX #1234,4(R6)
	EDE	6 <sup>(2)</sup>	8 <sup>(3)</sup>	4	XORX #A5A5h,EDE
	&EDE	6 <sup>(2)</sup>	8 <sup>(3)</sup>	4	BITX.B #12,&EDE
x(Rn)	Rm	4	5	3	BITX 2(R5),R8
	PC <sup>(4)</sup>	6	7	3	SUBX.A 2(R6),PC
	TONI	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	ANDX 4(R7),4(R6)
	x(Rm)	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	XORX.B 2(R6),EDE
	&TONI	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	BITX 8(SP),&EDE
EDE	Rm	4	5	3	BITX.B EDE,R8
	PC <sup>(4)</sup>	6	7	3	ADDX.A EDE,PC
	TONI	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	ANDX EDE,4(R6)
	x(Rm)	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	ANDX EDE,TONI
	&TONI	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	BITX EDE,&TONI
&EDE	Rm	4	5	3	BITX &EDE,R8
	PC <sup>(4)</sup>	6	7	3	ADDX.A &EDE,PC
	TONI	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	ANDX.B &EDE,4(R6)
	x(Rm)	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	XORX &EDE,TONI
	&TONI	7 <sup>(2)</sup>	10 <sup>(3)</sup>	4	BITX &EDE,&TONI

<sup>(1)</sup> Repeat instructions require n + 1 cycles, where n is the number of times the instruction is executed.

<sup>(2)</sup> Reduce the cycle count by one for MOV, BIT, and CMP instructions.

<sup>(3)</sup> Reduce the cycle count by two for MOV, BIT, and CMP instructions.

<sup>(4)</sup> Reduce the cycle count by one for MOV, ADD, and SUB instructions.

### **MSP430X Address Instruction Cycles and Lengths**

[Table 6-19](#) lists the length and the CPU cycles for all addressing modes of the MSP430X address instructions.

**Table 6-19. Address Instruction Cycles and Length**

Addressing Mode		Execution Time (MCLK Cycles)			Length of Instruction (Words)		Example
Source	Destination	MOVA BRA	CMPA ADD A SUBA	MOVA	CMPA ADD A 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	

## 6.6 Instruction Set Description

Table 6-20 shows all available instructions:

**Table 6-20. Instruction Map of MSP430X**

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
10xx	RRC	RRC. B	SWP B		RRA	RRA. B	SXT		PUS H	PUS H.B	CALL		RETI	CALL A		
14xx																
18xx																
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							

### **6.6.1 Extended Instruction Binary Descriptions**

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

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

<b>Instruction</b>	<b>Instruction Group</b>		<b>Bit Loc.</b>	<b>Inst. ID</b>	<b>Instruction Identifier</b>			<b>dst</b>		
	15	12	11	10	9	8	7	4	3	0
RRCM.A	0	0	0	0	n - 1	0	0	0	1	dst
RRAM.A	0	0	0	0	n - 1	0	1	0	1	0
RLAM.A	0	0	0	0	n - 1	1	0	0	1	0
RRUM.A	0	0	0	0	n - 1	1	1	0	1	0
RRCM.W	0	0	0	0	n - 1	0	0	0	1	0
RRAM.W	0	0	0	0	n - 1	0	1	0	1	0
RLAM.W	0	0	0	0	n - 1	1	0	0	1	0
RRUM.W	0	0	0	0	n - 1	1	1	0	1	0

## Instruction Set Description

www.ti.com

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
CALLA	0	0	0	1	0	0	1	1	0	1	0	0		dst
	0	0	0	1	0	0	1	1	0	1	0	1		dst
	x.15:0													
	0	0	0	1	0	0	1	1	0	1	1	0		dst
	0	0	0	1	0	0	1	1	0	1	1	1		dst
	0	0	0	1	0	0	1	1	1	0	0	0		&abs.19:16
	&abs.15:0													
	0	0	0	1	0	0	1	1	1	0	0	1		x.19:16
	x.15:0													
	0	0	0	1	0	0	1	1	1	0	1	1		imm.19:16
	imm.15:0													
Reserved	0	0	0	1	0	0	1	1	1	0	1	0	x	x
Reserved	0	0	0	1	0	0	1	1	1	1	x	x	x	x
PUSHM.A	0	0	0	1	0	1	0	0		n - 1				dst
PUSHM.W	0	0	0	1	0	1	0	1		n - 1				dst
POPM.A	0	0	0	1	0	1	1	0		n - 1				dst - n + 1
POPM.W	0	0	0	1	0	1	1	1		n - 1				dst - n + 1

### **6.6.2 MSP430 Instructions**

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

**6.6.2.1 ADC**

<b>* ADC[.W]</b>	Add carry to destination
<b>* ADC.B</b>	Add carry to destination
<b>Syntax</b>	ADC dst or            ADC.W dst ADC.B dst
<b>Operation</b>	dst + C → dst
<b>Emulation</b>	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
```

### 6.6.2.2 ADD

<b>ADD[.W]</b>	Add source word to destination word
<b>ADD.B</b>	Add source byte to destination byte
<b>Syntax</b>	ADD src,dst or ADD.W src,dst ADD.B src,dst
<b>Operation</b>	src + dst → dst
<b>Description</b>	The source operand is added to the destination operand. The previous content of the destination is lost.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Ten is added to the 16-bit counter CNTR located in lower 64 K.

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

<b>Example</b>	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
```

<b>Example</b>	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
```

**6.6.2.3 ADDC**

<b>ADDC.W</b>	Add source word and carry to destination word
<b>ADDC.B</b>	Add source byte and carry to destination byte
<b>Syntax</b>	ADDC src,dst or ADDC.W src,dst ADDC.B src,dst
<b>Operation</b>	src + dst + C → dst
<b>Description</b>	The source operand and the carry bit C are added to the destination operand. The previous content of the destination is lost.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

### 6.6.2.4 AND

<b>AND[.W]</b>	Logical AND of source word with destination word
<b>AND.B</b>	Logical AND of source byte with destination byte
<b>Syntax</b>	AND src,dst or AND.W src,dst AND.B src,dst
<b>Operation</b>	src .and. dst → dst
<b>Description</b>	The source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
...
```

or shorter:

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

<b>Example</b>	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
```

**6.6.2.5 BIC**

<b>BIC[W]</b>	Clear bits set in source word in destination word
<b>BIC.B</b>	Clear bits set in source byte in destination byte
<b>Syntax</b>	BIC src,dst or BIC.W src,dst BIC.B src,dst
<b>Operation</b>	(.not. src) .and. dst → dst
<b>Description</b>	The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The bits 15:14 of R5 (16-bit data) are cleared. R5.19:16 = 0

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

<b>Example</b>	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
```

<b>Example</b>	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
```

### 6.6.2.6 BIS

<b>BIS[.W]</b>	Set bits set in source word in destination word
<b>BIS.B</b>	Set bits set in source byte in destination byte
<b>Syntax</b>	<code>BIS src,dst</code> or <code>BIS.W src,dst</code> <code>BIS.B src,dst</code>
<b>Operation</b>	<code>src .or. dst → dst</code>
<b>Description</b>	The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Bits 15 and 13 of R5 (16-bit data) are set to one. R5.19:16 = 0

`BIS #A000h,R5 ; Set R5 bits`

<b>Example</b>	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`

<b>Example</b>	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`

**6.6.2.7 BIT**

<b>BIT[.W]</b>	Test bits set in source word in destination word
<b>BIT.B</b>	Test bits set in source byte in destination byte
<b>Syntax</b>	BIT src,dst or BIT.W src,dst BIT.B src,dst
<b>Operation</b>	src .and. dst
<b>Description</b>	The source operand and the destination operand are logically ANDed. The result affects only the status bits in SR.
<b>Status Bits</b>	Register mode: the register bits Rdst.19:16 (.W) resp. Rdst. 19:8 (.B) are not cleared! 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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

## 6.6.2.8 BR, BRANCH

<b>* BR, BRANCH</b>	Branch to destination in lower 64K address space
<b>Syntax</b>	BR dst
<b>Operation</b>	dst → PC
<b>Emulation</b>	MOV dst,PC
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	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

```

**6.6.2.9 CALL**

<b>CALL</b>	Call a subroutine in lower 64 K
<b>Syntax</b>	CALL dst
<b>Operation</b>	<p>dst → tmp 16-bit dst is evaluated and stored</p> <p>SP – 2 → SP</p> <p>PC → @SP updated PC with return address to TOS</p> <p>tmp → PC saved 16-bit dst to PC</p>
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected. PC.19:16 cleared (address in lower 64 K)
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Examples</b>	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
```

### 6.6.2.10 CLR

* CLR[.W]	Clear destination
* CLR.B	Clear destination
<b>Syntax</b>	CLR dst or            CLR.W dst CLR.B dst
<b>Operation</b>	$0 \rightarrow \text{dst}$
<b>Emulation</b>	MOV #0,dst MOV.B #0,dst
<b>Description</b>	The destination operand is cleared.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	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

**6.6.2.11 CLRC**

**\* CLRC** Clear carry bit

**Syntax** CLRC

**Operation**  $0 \rightarrow C$

**Emulation** BIC #1,SR

**Description** The carry bit (C) is cleared. The clear carry instruction is a word instruction.

**Status Bits** N: Not affected

Z: Not affected

C: Cleared

V: Not affected

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

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

```

CLRC          ; C=0: defines start
DADD  @R13,0(R12)    ; add 16-bit counter to low word of 32-bit counter
DADC  2(R12)        ; add carry to high word of 32-bit counter

```

### 6.6.2.12 CLRN

**\* CLRN** Clear negative bit

**Syntax** CLRN

**Operation**  $0 \rightarrow N$

or

$(.\text{NOT}.\text{src} .\text{AND}. \text{dst} \rightarrow \text{dst})$

**Emulation** BIC #4,SR

**Description** The constant 04h is inverted (0FFF8h) 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

```

**6.6.2.13 CLRZ**

<b>* CLRZ</b>	Clear zero bit
<b>Syntax</b>	CLRZ
<b>Operation</b>	$0 \rightarrow Z$ or $(\text{.NOT.} \text{src} \text{ .AND. } \text{dst} \rightarrow \text{dst})$
<b>Emulation</b>	BIC #2,SR
<b>Description</b>	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.
<b>Status Bits</b>	N: Not affected Z: Reset to 0 C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The zero bit in the SR is cleared.

CLRZ

Indirect, Auto-Increment mode: Call a subroutine at the 16-bit address contained in the word pointed to by register R5 (20-bit address) and increment the 16-bit address in R5 afterwards by 2. The next time the software uses R5 as a pointer, it can alter the program execution due to access to the next word address in the table pointed to by R5.

CALL @R5+ ; Start address at @R5. R5 + 2

Indexed mode: Call a subroutine at the 16-bit address contained in the 20-bit address pointed to by register (R5 + X); for example, a table with addresses starting at X. The address is within the lower 64 KB. X is within  $\pm 32$  KB.

CALL X(R5) ; Start address at @(R5+X). z16(R5)

### 6.6.2.14 CMP

<b>CMP[.W]</b>	Compare source word and destination word
<b>CMP.B</b>	Compare source byte and destination byte
<b>Syntax</b>	CMP src,dst or CMP.W src,dst CMP.B src,dst
<b>Operation</b>	(.not.src) + 1 + dst or dst – src
<b>Emulation</b>	BIC #2,SR
<b>Description</b>	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.
<b>Status Bits</b>	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).
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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.
	<pre> CMP      #01800h,EDE      ; Compare word EDE with 1800h JEQ      TONI             ; EDE contains 1800h ... </pre>
<b>Example</b>	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.
	<pre> CMP.W   10(R5),R7        ; Compare two signed numbers JL      TONI             ; R7 &lt; 10(R5) ... </pre>
<b>Example</b>	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.
	<pre> CMP.B   @R5+,&amp;P1OUT     ; Compare P1 bits with table. R5 + 1 JEQ      TONI             ; Equal contents ... </pre>

**6.6.2.15 DADC**

<b>* DADC[W]</b>	Add carry decimal to destination
<b>* DADC.B</b>	Add carry decimal to destination
<b>Syntax</b>	DADC dst or            DADC.W dst DADC.B dst
<b>Operation</b>	dst + C → dst (decimally)
<b>Emulation</b>	DADD #0, dst DADD.B #0, dst
<b>Description</b>	The carry bit (C) is added decimal to the destination.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

<b>Example</b>	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

```

### 6.6.2.16 DADD

<b>* DADD.W</b>	Add source word and carry decimal to destination word
<b>* DADD.B</b>	Add source byte and carry decimal to destination byte
<b>Syntax</b>	DADD src,dst or DADD.W src,dst  DADD.B src,dst
<b>Operation</b>	src + dst + C → dst (decimally)
<b>Description</b>	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 decimal 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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Decimal 10 is added to the 16-bit BCD counter DECCNTR.

```
DADD #10h,&DECCNTR ; Add 10 to 4-digit BCD counter
```

<b>Example</b>	The eight-digit BCD number contained in 16-bit RAM addresses BCD and BCD+2 is added decimal 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
```

<b>Example</b>	The two-digit BCD number contained in word BCD (16-bit address) is added decimal 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 decimal.
                    ; R4: 0,00ddh
```

**6.6.2.17 DEC**

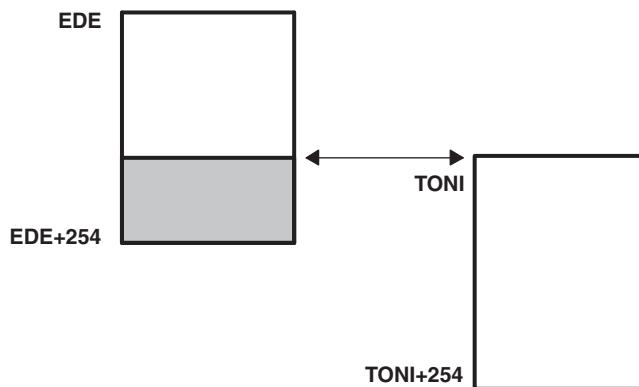
<b>* DEC[.W]</b>	Decrement destination
<b>* DEC.B</b>	Decrement destination
<b>Syntax</b>	DEC dst or        DEC.W dst DEC.B dst
<b>Operation</b>	dst – 1 → dst
<b>Emulation</b>	SUB #1,dst SUB.B #1,dst
<b>Description</b>	The destination operand is decremented by one. The original contents are lost.
<b>Status Bits</b>	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.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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 6-36](#).



**Figure 6-36. Decrement Overlap**

### **6.6.2.18 DECD**

<b>* DECD[W]</b>	Double-decrement destination
<b>* DECD.B</b>	Double-decrement destination
<b>Syntax</b>	DECD dst or DECD.W dst DECD.B dst
<b>Operation</b>	dst - 2 → dst
<b>Emulation</b>	SUB #2,dst SUB.B #2,dst
<b>Description</b>	The destination operand is decremented by two. The original contents are lost.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	Memory at location LEO is decremented by two.
----------------	---

```
DECD.B   LEO           ; Decrement MEM(LEO)
```

Decrement status byte STATUS by two

```
DECD.B   STATUS
```

**6.6.2.19 DINT**

<b>* DINT</b>	Disable (general) interrupts
<b>Syntax</b>	DINT
<b>Operation</b>	$0 \rightarrow GIE$ or $(0FFF7h .AND. SR \rightarrow SR / .NOT.src .AND. dst \rightarrow dst)$
<b>Emulation</b>	BIC #8,SR
<b>Description</b>	All interrupts are disabled. The constant 08h is inverted and logically ANDed with the SR. The result is placed into the SR.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	GIE is reset. OSCOFF and CPUOFF are not affected.
<b>Example</b>	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.

## 6.6.2.20 EINT

<b>* EINT</b>	Enable (general) interrupts
<b>Syntax</b>	EINT
<b>Operation</b>	1 → GIE or (0008h .OR. SR → SR / .src .OR. dst → dst)
<b>Emulation</b>	BIS #8,SR
<b>Description</b>	All interrupts are enabled. The constant #08h and the SR are logically ORed. The result is placed into the SR.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	GIE is set. OSCOFF and CPUOFF are not affected.
<b>Example</b>	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.

---

*Instruction Set Description*[www.ti.com](http://www.ti.com)**6.6.2.21 INC**

<b>* INC[W]</b>	Increment destination
<b>* INC.B</b>	Increment destination
<b>Syntax</b>	INC dst or           INC.W dst INC.B dst
<b>Operation</b>	dst + 1 → dst
<b>Emulation</b>	ADD #1,dst
<b>Description</b>	The destination operand is incremented by one. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFFFh, reset otherwise Set if dst contained OFFh, reset otherwise C: Set if dst contained 0FFFFh, reset otherwise Set if dst contained OFFh, reset otherwise V: Set if dst contained 07FFFh, reset otherwise Set if dst contained 07Fh, reset otherwise
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

### 6.6.2.22 INCD

<b>* INCD[.W]</b>	Double-increment destination
<b>* INCD.B</b>	Double-increment destination
<b>Syntax</b>	INCD dst or            INCD.W dst INCD.B dst
<b>Operation</b>	dst + 2 → dst
<b>Emulation</b>	ADD #2,dst
<b>Description</b>	The destination operand is incremented by two. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFEh, reset otherwise Set if dst contained 0FEh, reset otherwise C: Set if dst contained 0FFEh 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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	The byte on the top of the stack is incremented by two.
INCD.B	0(SP) ; Byte on TOS is increment by two

**6.6.2.23 INV**

<b>* INV[.W]</b>	Invert destination
<b>* INV.B</b>	Invert destination
<b>Syntax</b>	INV dst or            INV.W dst INV.B dst
<b>Operation</b>	.not.dst → dst
<b>Emulation</b>	XOR #0FFFFh,dst XOR.B #0FFh,dst
<b>Description</b>	The destination operand is inverted. The original contents are lost.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

<b>Example</b>	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	

```

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

```

### 6.6.2.24 JC, JHS

<b>JC</b>	Jump if carry
<b>JHS</b>	Jump if higher or same (unsigned)
<b>Syntax</b>	JC label JHS label
<b>Operation</b>	If C = 1: PC + (2 × Offset) → PC If C = 0: execute the following instruction
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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,R5        ; 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
```

**6.6.2.25 JEQ, JZ**

<b>JEQ</b>	Jump if equal
<b>JZ</b>	Jump if zero
<b>Syntax</b>	JEQ label JZ label
<b>Operation</b>	If Z = 1: PC + (2 × Offset) → PC If Z = 0: execute following instruction
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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.
```

## 6.6.2.26 JGE

<b>JGE</b>	Jump if greater or equal (signed)
<b>Syntax</b>	JGE label
<b>Operation</b>	If (N .xor. V) = 0: PC + (2 × Offset) → PC If (N .xor. V) = 1: execute following instruction
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	If the content of R6 is greater than or equal to the memory pointed to by R7, the program continues at 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	

<b>Example</b>	If R5 $\geq$ 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	

**6.6.2.27 JL**

<b>JL</b>	Jump if less (signed)
<b>Syntax</b>	JL label
<b>Operation</b>	If (N .xor. V) = 1: PC + (2 × Offset) → PC If (N .xor. V) = 0: execute following instruction
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

### 6.6.2.28 JMP

<b>JMP</b>	Jump unconditionally
<b>Syntax</b>	JMP label
<b>Operation</b>	PC + (2 × Offset) → PC
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
JMP    IHCCR1          ; No Timer_A interrupt pending
JMP    IHCCR2          ; Timer block 1 caused interrupt
JMP    IHCCR2          ; Timer block 2 caused interrupt
RETI              ; No legal interrupt, return
```

**6.6.2.29 JN**

<b>JN</b>	Jump if negative
<b>Syntax</b>	JN label
<b>Operation</b>	If N = 1: PC + (2 × Offset) → PC If N = 0: execute following instruction
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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.
```

<b>Example</b>	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.
```

### 6.6.2.30 JNC, JLO

<b>JNC</b>	Jump if no carry
<b>JLO</b>	Jump if lower (unsigned)
<b>Syntax</b>	JNC label JLO label
<b>Operation</b>	If C = 0: PC + (2 × Offset) → PC If C = 1: execute following instruction
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

**6.6.2.31 JNZ, JNE**

<b>JNZ</b>	Jump if not zero
<b>JNE</b>	Jump if not equal
<b>Syntax</b>	JNZ label JNE label
<b>Operation</b>	If Z = 0: PC + (2 × Offset) → PC If Z = 1: execute following instruction
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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.
```

### 6.6.2.32 MOV

<b>MOV[.W]</b>	Move source word to destination word
<b>MOV.B</b>	Move source byte to destination byte
<b>Syntax</b>	MOV src,dst or MOV.W src,dst MOV.B src,dst
<b>Operation</b>	src → dst
<b>Description</b>	The source operand is copied to the destination. The source operand is not affected.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Move a 16-bit constant 1800h to absolute address-word EDE (lower 64 K)

```
MOV      #01800h,&EDE           ; Move 1800h to EDE
```

<b>Example</b>	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.
----------------	--

```
Loop    MOV      #EDE,R10          ; Prepare pointer (16-bit address)
        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
        ...
```

<b>Example</b>	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.
----------------	--

```
Loop    MOVA   #EDE,R10          ; Prepare pointer (20-bit)
        MOV     #20h,R9            ; Prepare counter
        MOV.B  @R10+,TOM-EDE-1(R10); R10 points to both tables.
                                    ; R10+1
        DEC     R9                ; Decrement counter
        JNZ     Loop               ; Not yet done
        ...
```

*Instruction Set Description*[www.ti.com](http://www.ti.com)**6.6.2.33 NOP**

<b>* NOP</b>	No operation
<b>Syntax</b>	NOP
<b>Operation</b>	None
<b>Emulation</b>	MOV #0, R3
<b>Description</b>	No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.
<b>Status Bits</b>	Status bits are not affected.

### 6.6.2.34 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.

**6.6.2.35 PUSH**

<b>PUSH[.W]</b>	Save a word on the stack
<b>PUSH.B</b>	Save a byte on the stack
<b>Syntax</b>	PUSH dst or           PUSH.W dst PUSH.B dst
<b>Operation</b>	SP – 2 → SP dst → @SP
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Save the two 16-bit registers R9 and R10 on the stack

```
PUSH      R9          ; Save R9 and R10 xxxxh
PUSH      R10         ; YYYYh
```

<b>Example</b>	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
```

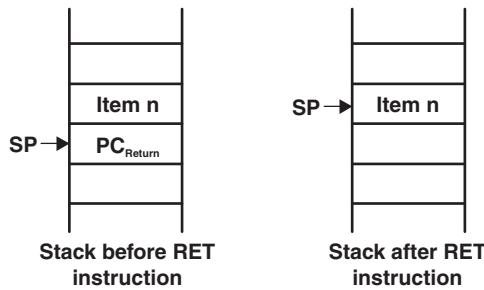
### 6.6.2.36 RET

<b>* RET</b>	Return from subroutine
<b>Syntax</b>	RET
<b>Operation</b>	$@SP \rightarrow PC.15:0$ Saved PC to PC.15:0. $PC.19:16 \leftarrow 0$ $SP + 2 \rightarrow SP$
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected. PC.19:16: Cleared
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
...
SUBR   PUSH R14    ; Save R14 (16 bit data)
...
POP    R14        ; Restore R14
RET     ; Return to lower 64 K

```



**Figure 6-37. Stack After a RET Instruction**

**6.6.2.37 RETI**

<b>RETI</b>	Return from interrupt
<b>Syntax</b>	RETI
<b>Operation</b>	$\begin{array}{ll} @SP \rightarrow SR.15:0 & \text{Restore saved SR with PC.19:16} \\ SP + 2 \rightarrow SP & \\ @SP \rightarrow PC.15:0 & \text{Restore saved PC.15:0} \\ SP + 2 \rightarrow SP & \text{Housekeeping} \end{array}$
<b>Description</b>	<p>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.</p> <p>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.</p>
<b>Status Bits</b>	<p>N: Restored from stack</p> <p>C: Restored from stack</p> <p>Z: Restored from stack</p> <p>V: Restored from stack</p>
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are restored from stack.
<b>Example</b>	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)
          ...
          POPM.A #2,R14      ; Restore R13 and R14 (20-bit data)
          RETI                ; Return to 20-bit address in full memory range

```

### 6.6.2.38 RLA

\* **RLA[.W]** Rotate left arithmetically

\* **RLA.B** Rotate left arithmetically

**Syntax**      RLA dst or      RLA.W dst

                  RLA.B dst

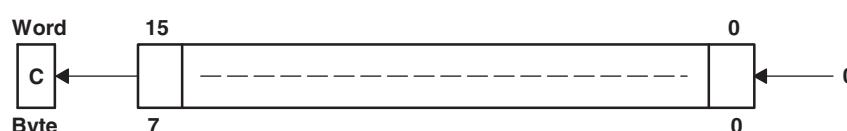
**Operation**     $C \leftarrow MSB \leftarrow MSB-1 \dots LSB+1 \leftarrow LSB \leftarrow 0$

**Emulation**    ADD dst,dst

                  ADD.B dst,dst

**Description**    The destination operand is shifted left one position as shown in [Figure 6-38](#). 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  $dst \geq 04000h$  and  $dst < 0C000h$  before operation is performed; the result has changed sign.



**Figure 6-38. Destination Operand—Arithmetic Shift Left**

An overflow occurs if  $dst \geq 040h$  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  $04000h \leq dst < 0C000h$ , reset otherwise

Set if an arithmetic overflow occurs; the initial value is  $040h \leq dst < 0C0h$ , 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
```

**6.6.2.39 RLC**

\* **RLC[.W]** Rotate left through carry

\* **RLC.B** Rotate left through carry

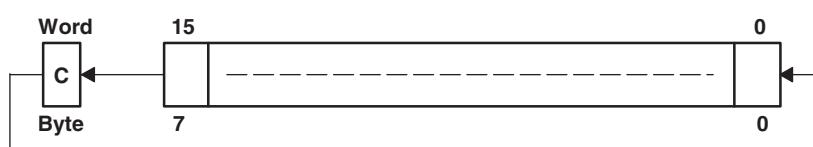
**Syntax**      RLC dst or      RLC.W dst

                  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 6-39](#). The carry bit (C) is shifted into the LSB, and the MSB is shifted into the carry bit (C).



**Figure 6-39. Destination Operand—Carry Left Shift**

<b>Status Bits</b>	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 $04000h \leq dst < 0C000h$ , reset otherwise Set if an arithmetic overflow occurs; the initial value is $040h \leq dst < 0C0h$ , reset otherwise
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

---

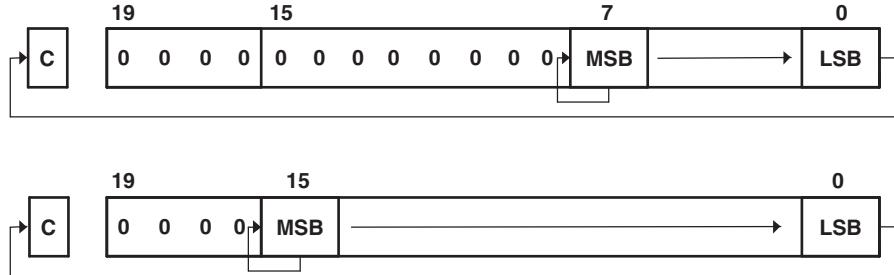
### 6.6.2.40 RRA

<b>RRA.[W]</b>	Rotate right arithmetically destination word
<b>RRA.B</b>	Rotate right arithmetically destination byte
<b>Syntax</b>	RRA.B dst or                  RRA.W dst
<b>Operation</b>	MSB → MSB → MSB-1 → ... LSB+1 → LSB → C
<b>Description</b>	The destination operand is shifted right arithmetically by one bit position as shown in <b>Figure 6-40</b> . 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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

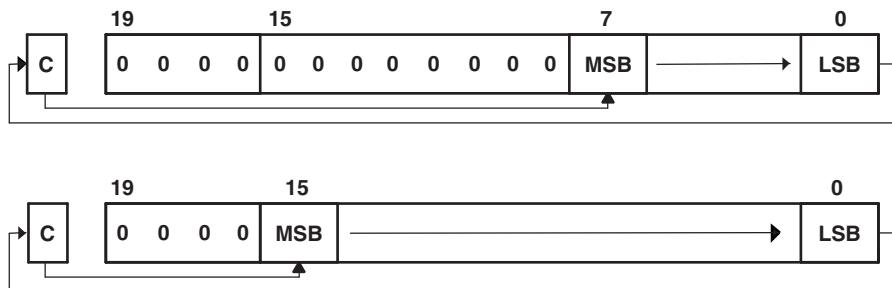


**Figure 6-40. Rotate Right Arithmetically RRA.B and RRA.W**

**6.6.2.41 RRC**

<b>RRC[.W]</b>	Rotate right through carry destination word
<b>RRC.B</b>	Rotate right through carry destination byte
<b>Syntax</b>	RRC dst or            RRC.W dst RRC.B dst
<b>Operation</b>	C → MSB → MSB-1 → ... LSB+1 → LSB → C
<b>Description</b>	The destination operand is shifted right by one bit position as shown in <a href="#">Figure 6-41</a> . The carry bit C is shifted into the MSB and the LSB is shifted into the carry bit C.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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 6-41. Rotate Right Through Carry RRC.B and RRC.W**

### 6.6.2.42 SBC

<b>* SBC[.W]</b>	Subtract borrow (.NOT. carry) from destination
<b>* SBC.B</b>	Subtract borrow (.NOT. carry) from destination
<b>Syntax</b>	SBC dst or            SBC.W dst SBC.B dst
<b>Operation</b>	dst + 0FFFFh + C → dst dst + 0FFh + C → dst
<b>Emulation</b>	SUBC #0, dst SUBC.B #0, dst
<b>Description</b>	The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.
	<pre>SUB.B   @R13,0(R12)    ; Subtract LSDs SBC.B   1(R12)        ; Subtract carry from MSD</pre>

---

**NOTE: Borrow implementation**

The borrow is treated as a .NOT. carry:

Borrow	Carry Bit
Yes	0
No	1

---

**6.6.2.43 SETC**

<b>* SETC</b>	Set carry bit
<b>Syntax</b>	SETC
<b>Operation</b>	$1 \rightarrow C$
<b>Emulation</b>	BIS #1,SR
<b>Description</b>	The carry bit (C) is set.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Set V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Emulation of the decimal subtraction: Subtract R5 from R6 decimal. 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
DADD  R5,R6              ; Prepare carry = 1
              ; Emulate subtraction by addition of:
              ; (010000h - R5 - 1)
              ; R6 = R6 + R5 + 1
              ; R6 = 0150h

```

### 6.6.2.44 SETN

* <b>SETN</b>	Set negative bit
<b>Syntax</b>	SETN
<b>Operation</b>	$1 \rightarrow N$
<b>Emulation</b>	BIS #4,SR
<b>Description</b>	The negative bit (N) is set.
<b>Status Bits</b>	N: Set Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.

### 6.6.2.45 SETZ

* <b>SETZ</b>	Set zero bit
<b>Syntax</b>	SETZ
<b>Operation</b>	$1 \rightarrow N$
<b>Emulation</b>	BIS #2,SR
<b>Description</b>	The zero bit (Z) is set.
<b>Status Bits</b>	N: Not affected Z: Set C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.

### 6.6.2.46 SUB

<b>SUB[.W]</b>	Subtract source word from destination word
<b>SUB.B</b>	Subtract source byte from destination byte
<b>Syntax</b>	SUB src,dst or SUB.W src,dst SUB.B src,dst
<b>Operation</b>	(.not.src) + 1 + dst → dst or dst – src → dst
<b>Description</b>	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.
<b>Status Bits</b>	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)
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	A 16-bit constant 7654h is subtracted from RAM word EDE.

```
SUB #7654h, &EDE ; Subtract 7654h from EDE
```

<b>Example</b>	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)
```

<b>Example</b>	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
```

**6.6.2.47 SUBC**

<b>SUBC.W</b>	Subtract source word with carry from destination word
<b>SUBC.B</b>	Subtract source byte with carry from destination byte
<b>Syntax</b>	SUBC src,dst or SUBC.W src,dst SUBC.B src,dst
<b>Operation</b>	$(\text{.not}.src) + C + dst \rightarrow dst$ or $dst - (\text{src} - 1) + C \rightarrow dst$
<b>Description</b>	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.
<b>Status Bits</b>	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)
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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.
----------------	--

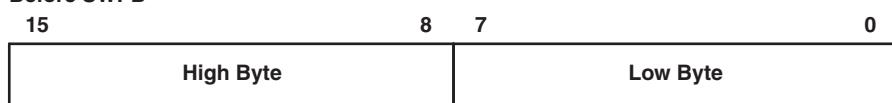
```
SUBC.B &CNT,0(R12) ; Subtract byte CNT from @R12
```

### 6.6.2.48 SWPB

<b>SWPB</b>	Swap bytes
<b>Syntax</b>	SWPB dst
<b>Operation</b>	dst.15:8 ↔ dst.7:0
<b>Description</b>	The high and the low byte of the operand are exchanged. PC.19:16 bits are cleared in register mode.
<b>Status Bits</b>	Status bits are not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Exchange the bytes of RAM word EDE (lower 64 K)

```
MOV    #1234h, &EDE      ; 1234h -> EDE
SWPB   &EDE             ; 3412h -> EDE
```

Before SWPB



After SWPB

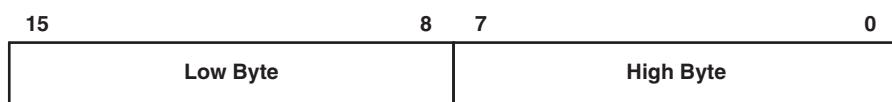
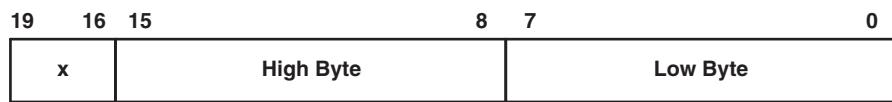


Figure 6-42. Swap Bytes in Memory

Before SWPB



After SWPB

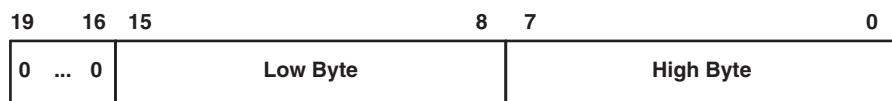


Figure 6-43. Swap Bytes in a Register

**6.6.2.49 SXT**

<b>SXT</b>	Extend sign
<b>Syntax</b>	SXT dst
<b>Operation</b>	dst.7 → dst.15:8, dst.7 → dst.19:8 (register mode)
<b>Description</b>	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
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

## 6.6.2.50 TST

<b>* TST[.W]</b>	Test destination
<b>* TST.B</b>	Test destination
<b>Syntax</b>	TST dst or            TST.W dst TST.B dst
<b>Operation</b>	dst + 0FFFFh + 1 dst + 0FFh + 1
<b>Emulation</b>	CMP #0,dst CMP.B #0,dst
<b>Description</b>	The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.
<b>Status Bits</b>	N: Set if destination is negative, reset if positive Z: Set if destination contains zero, reset otherwise C: Set V: Reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

<b>Example</b>	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

```

**6.6.2.51 XOR**

<b>XOR[.W]</b>	Exclusive OR source word with destination word
<b>XOR.B</b>	Exclusive OR source byte with destination byte
<b>Syntax</b>	<code>XOR src,dst</code> or <code>XOR.W src,dst</code> <code>XOR.B src,dst</code>
<b>Operation</b>	<code>src .xor. dst → dst</code>
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

### 6.6.3 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.

**6.6.3.1 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
```

### 6.6.3.2 ADDX

<b>ADDX.A</b>	Add source address-word to destination address-word
<b>ADDX.[W]</b>	Add source word to destination word
<b>ADDX.B</b>	Add source byte to destination byte
<b>Syntax</b>	ADDX.A src,dst ADDX src,dst or ADDX.W src,dst ADDX.B src,dst
<b>Operation</b>	$src + dst \rightarrow dst$
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

**6.6.3.3 ADDCX**

<b>ADDCX.A</b>	Add source address-word and carry to destination address-word
<b>ADDCX.[W]</b>	Add source word and carry to destination word
<b>ADDCX.B</b>	Add source byte and carry to destination byte
<b>Syntax</b>	ADDCX.A src,dst ADDCX src,dst or ADDCX.W src,dst ADDCX.B src,dst
<b>Operation</b>	$\text{src} + \text{dst} + C \rightarrow \text{dst}$
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

#### 6.6.3.4 ANDX

<b>ANDX.A</b>	Logical AND of source address-word with destination address-word
<b>ANDX.[W]</b>	Logical AND of source word with destination word
<b>ANDX.B</b>	Logical AND of source byte with destination byte
<b>Syntax</b>	ANDX.A src,dst ANDX src,dst OR ANDX.W src,dst ANDX.B src,dst
<b>Operation</b>	src .and. dst → dst
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

<b>Example</b>	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
```

**6.6.3.5 BICX**

<b>BICX.A</b>	Clear bits set in source address-word in destination address-word
<b>BICX.[W]</b>	Clear bits set in source word in destination word
<b>BICX.B</b>	Clear bits set in source byte in destination byte
<b>Syntax</b>	BICX.A src,dst BICX src,dst or BICX.W src,dst BICX.B src,dst
<b>Operation</b>	(.not. src) .and. dst → dst
<b>Description</b>	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.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

### 6.6.3.6 BISX

<b>BISX.A</b>	Set bits set in source address-word in destination address-word
<b>BISX.[W]</b>	Set bits set in source word in destination word
<b>BISX.B</b>	Set bits set in source byte in destination byte
<b>Syntax</b>	<b>BISX.A src,dst</b> <b>BISX src,dst OR BISX.W src,dst</b> <b>BISX.B src,dst</b>
<b>Operation</b>	<b>src .or. dst → dst</b>
<b>Description</b>	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.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

**6.6.3.7 BITX**

<b>BITX.A</b>	Test bits set in source address-word in destination address-word
<b>BITX.[W]</b>	Test bits set in source word in destination word
<b>BITX.B</b>	Test bits set in source byte in destination byte
<b>Syntax</b>	BITX.A src,dst BITX src,dst OR BITX.W src,dst BITX.B src,dst
<b>Operation</b>	src .and. dst → dst
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

### 6.6.3.8 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**

$0 \rightarrow \text{dst}$

**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
```

## 6.6.3.9 CMPX

<b>CMPX.A</b>	Compare source address-word and destination address-word
<b>CMPX.[W]</b>	Compare source word and destination word
<b>CMPX.B</b>	Compare source byte and destination byte
<b>Syntax</b>	CMPX.A src,dst CMPX src,dst or CMPX.W src,dst CMPX.B src,dst
<b>Operation</b>	(.not. src) + 1 + dst or dst - src
<b>Description</b>	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.
<b>Status Bits</b>	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)
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

### 6.6.3.10 DADCX

- \* **DADCX.A** Add carry decimal to destination address-word
- \* **DADCX.[W]** Add carry decimal to destination word
- \* **DADCX.B** Add carry decimal to destination byte

**Syntax**

```
DADCX.A dst
DADCX dst or      DADCX.W dst
DADCX.B dst
```

**Operation**

$dst + C \rightarrow dst$  (decimally)

**Emulation**

DADDX.A #0,dst

DADDX #0,dst

DADDX.B #0,dst

**Description**

The carry bit (C) is added decimal 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 decimal.

```
DADDX.A #1,0(R12)      ; Increment lower 20 bits
DADCX.A 0(R13)        ; Add carry to upper 20 bits
```

**6.6.3.11 DADDX**

<b>DADDX.A</b>	Add source address-word and carry decimal to destination address-word
<b>DADDX.[W]</b>	Add source word and carry decimal to destination word
<b>DADDX.B</b>	Add source byte and carry decimal to destination byte
<b>Syntax</b>	DADDX.A src,dst DADDX src,dst or DADDX.W src,dst DADDX.B src,dst
<b>Operation</b>	$src + dst + C \rightarrow dst$ (decimally)
<b>Description</b>	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 decimal 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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	The eight-digit BCD number contained in 20-bit addresses BCD and BCD+2 is added decimal 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
```

<b>Example</b>	The two-digit BCD number contained in 20-bit address BCD is added decimal to a two-digit BCD number contained in R4.
----------------	--

```
CLRC ; Clear carry
DADDX.B BCD,R4 ; Add BCD to R4 decimal.
; R4: 000ddh
```

### 6.6.3.12 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
```

*Instruction Set Description*[www.ti.com](http://www.ti.com)**6.6.3.13 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**

$dst - 2 \rightarrow dst$

**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
```

### 6.6.3.14 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**

$dst + 1 \rightarrow dst$

**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 0FFFFh, reset otherwise
  - 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 0FFFh, 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)
```

*Instruction Set Description*

www.ti.com

**6.6.3.15 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 0FFEh, reset otherwise
  - Set if dst contained 0FEh, reset otherwise
- C: Set if dst contained 0FFFFEh or 0FFFFFFh, reset otherwise
  - Set if dst contained 0FFEh or 0FFh, reset otherwise
  - Set if dst contained 0FEh or 0Fh, reset otherwise
- V: Set if dst contained 07FFFEh or 07FFFFFFh, 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
```

### 6.6.3.16 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 #0FFFFFh,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 0FFFh, 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
```

**6.6.3.17 MOVX**

<b>MOVX.A</b>	Move source address-word to destination address-word
<b>MOVX.[W]</b>	Move source word to destination word
<b>MOVX.B</b>	Move source byte to destination byte
<b>Syntax</b>	MOVX.A src,dst MOVX src,dst or MOVX.W src,dst MOVX.B src,dst
<b>Operation</b>	src → dst
<b>Description</b>	The source operand is copied to the destination. The source operand is not affected. Both operands may be located in the full address space.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Move a 20-bit constant 18000h to absolute address-word EDE

```
MOVX.A #018000h,&EDE ; Move 18000h to EDE
```

<b>Example</b>	The contents of table EDE (word data, 20-bit addresses) are copied to table TOM. The length of the table is 030h words.
----------------	---

```
Loop MOVA #EDE,R10 ; Prepare pointer (20-bit address)
      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
```

<b>Example</b>	The contents of table EDE (byte data, 20-bit addresses) are copied to table TOM. The length of the table is 020h bytes.
----------------	---

```
Loop MOVA #EDE,R10 ; Prepare pointer (20-bit)
      MOV #20h,R9 ; Prepare counter
      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

**6.6.3.18 POPM**

<b>POPM.A</b>	Restore n CPU registers (20-bit data) from the stack
<b>POPM.[W]</b>	Restore n CPU registers (16-bit data) from the stack
<b>Syntax</b>	POPM.A #n,Rdst $1 \leq n \leq 16$ POPM.W #n,Rdst or POPM #n,Rdst $1 \leq n \leq 16$
<b>Operation</b>	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.
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected, except SR is included in the operation.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

### 6.6.3.19 PUSHM

<b>PUSHM.A</b>	Save n CPU registers (20-bit data) on the stack
<b>PUSHM.[W]</b>	Save n CPU registers (16-bit words) on the stack
<b>Syntax</b>	PUSHM.A #n,Rdst $1 \leq n \leq 16$ PUSHM.W #n,Rdst Or PUSHM #n,Rdst $1 \leq n \leq 16$
<b>Operation</b>	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.
<b>Description</b>	PUSHM.A: The n CPU registers, starting with Rdst backwards, are stored on the stack. The SP is decremented by $(n \times 4)$ 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 $(n \times 2)$ 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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Save the five 20-bit registers R9, R10, R11, R12, R13 on the stack

PUSHM.A #5,R13 ; Save R13, R12, R11, R10, R9

<b>Example</b>	Save the five 16-bit registers R9, R10, R11, R12, R13 on the stack
	PUSHM.W #5,R13 ; Save R13, R12, R11, R10, R9

**6.6.3.20 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
```

### 6.6.3.21 PUSHX

<b>PUSHX.A</b>	Save single address-word to the stack
<b>PUSHX.[W]</b>	Save single word to the stack
<b>PUSHX.B</b>	Save single byte to the stack
<b>Syntax</b>	PUSHX.A src PUSHX src or           PUSHX.W src PUSHX.B src
<b>Operation</b>	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.
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Save the byte at the 20-bit address &EDE on the stack

```
PUSHX.B    &EDE      ; Save byte at address EDE
```

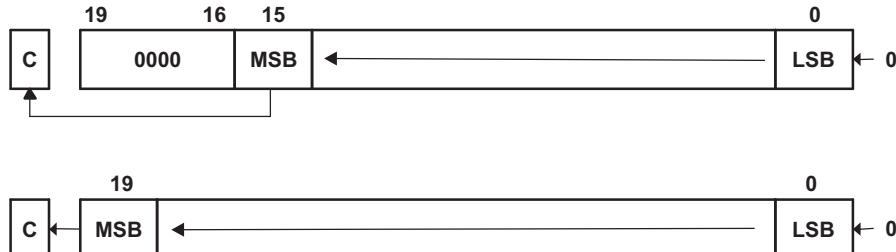
**Example**      Save the 20-bit value in R9 on the stack.

```
PUSHX.A    R9      ; Save address-word in R9
```

**6.6.3.22 RLAM**

<b>RLAM.A</b>	Rotate left arithmetically the 20-bit CPU register content
<b>RLAM.[W]</b>	Rotate left arithmetically the 16-bit CPU register content
<b>Syntax</b>	RLAM.A #n,Rdst $1 \leq n \leq 4$ RLAM.W #n,Rdst or RLAM #n,Rdst $1 \leq n \leq 4$
<b>Operation Description</b>	$C \leftarrow MSB \leftarrow MSB-1 \dots LSB+1 \leftarrow LSB \leftarrow 0$ The destination operand is shifted arithmetically left one, two, three, or four positions as shown in <a href="#">Figure 6-44</a> . 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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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 6-44. Rotate Left Arithmetically—RLAM.[W] and RLAM.A**

### 6.6.3.23 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**       $C \leftarrow MSB \leftarrow MSB-1 \dots LSB+1 \leftarrow LSB \leftarrow 0$

**Emulation**

```
ADDX.A dst,dst
ADDX dst,dst
ADDX.B dst,dst
```

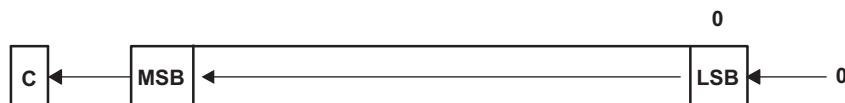
**Description**      The destination operand is shifted left one position as shown in [Figure 6-45](#). The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLAX instruction acts as a signed multiplication by 2.

- 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  $040000h \leq dst < 0C0000h$ ; reset otherwise
  - Set if an arithmetic overflow occurs: the initial value is  $04000h \leq dst < 0C00h$ ; reset otherwise
  - Set if an arithmetic overflow occurs: the initial value is  $040h \leq dst < 0C0h$ ; 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 6-45. Destination Operand-Arithmetic Shift Left**

**6.6.3.24 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****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 6-46](#). 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  $040000h \leq dst < 0C0000h$ ; reset otherwise  
     Set if an arithmetic overflow occurs: the initial value is  $04000h \leq dst < 0C000h$ ; reset otherwise  
     Set if an arithmetic overflow occurs: the initial value is  $040h \leq dst < 0C0h$ ; reset otherwise

**Mode Bits**

OSCOFF, CPUOFF, and GIE are not affected.

**Example**

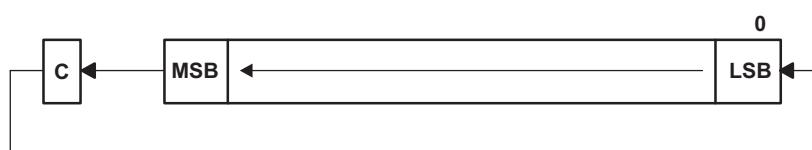
The 20-bit value in R5 is shifted left one position.

```
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 6-46. Destination Operand-Carry Left Shift**

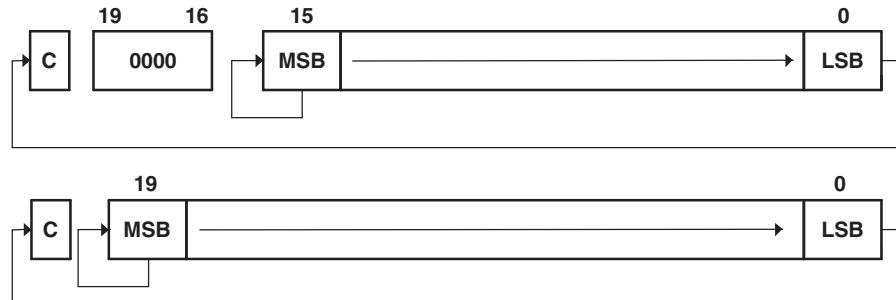
### **6.6.3.25 RRAM**

<b>RRAM.A</b>	Rotate right arithmetically the 20-bit CPU register content	
<b>RRAM.[W]</b>	Rotate right arithmetically the 16-bit CPU register content	
<b>Syntax</b>	RRAM.A #n,Rdst RRAM.W #n,Rdst OR RRAM #n,Rdst	1 ≤ n ≤ 4 1 ≤ n ≤ 4
<b>Operation Description</b>	MSB → MSB → MSB-1 ... LSB+1 → LSB → C	
	The destination operand is shifted right arithmetically by one, two, three, or four bit positions as shown in Figure 6-47. 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.	
<b>Status Bits</b>	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	
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.	
<b>Example</b>	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.  $(0.5 + 0.25) \times R15$ .

```
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 6-47.** Rotate Right Arithmetically RRAM[.W] and RRAM.A

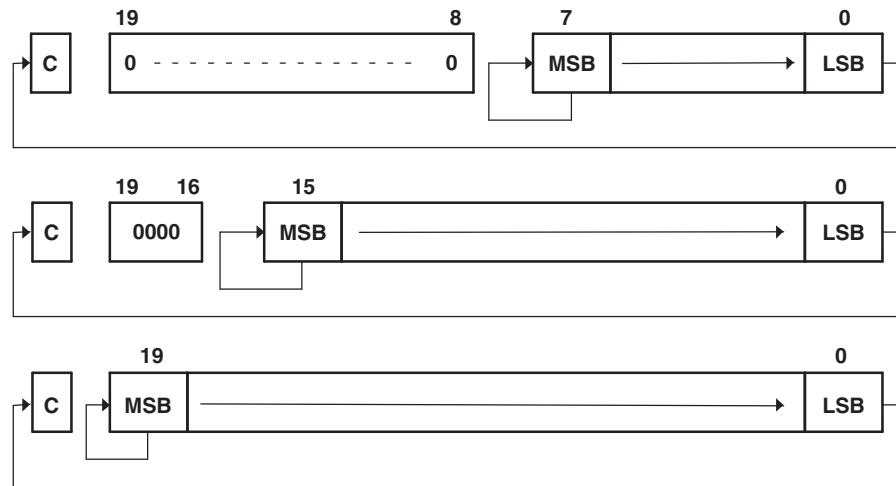
**6.6.3.26 RRAX**

<b>RRAX.A</b>	Rotate right arithmetically the 20-bit operand
<b>RRAX.[W]</b>	Rotate right arithmetically the 16-bit operand
<b>RRAX.B</b>	Rotate right arithmetically the 8-bit operand
<b>Syntax</b>	RRAX.A Rdst RRAX.W Rdst RRAX Rdst RRAX.B Rdst  RRAX.A dst RRAX dst or            RRAX.W dst RRAX.B dst
<b>Operation Description</b>	MSB → MSB → MSB-1 ... LSB+1 → LSB → C Register mode for the destination: the destination operand is shifted right by one bit position as shown in <a href="#">Figure 6-48</a> . 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 <a href="#">Figure 6-49</a> . 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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The signed 20-bit number in R5 is shifted arithmetically right four positions.

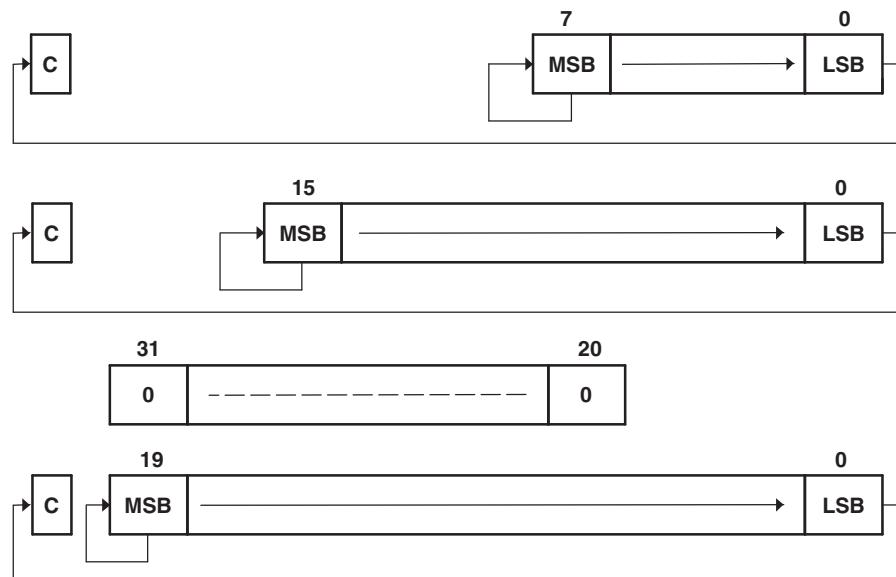
```
RPT      #4
RRAX.A  R5      ; R5/16 -> R5
```

**Example**      The signed 8-bit value in EDE is multiplied by 0.5.

RRAX.B      &EDE      ; EDE / 2 -> EDE



**Figure 6-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode**



**Figure 6-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode**

**6.6.3.27 RRCM**

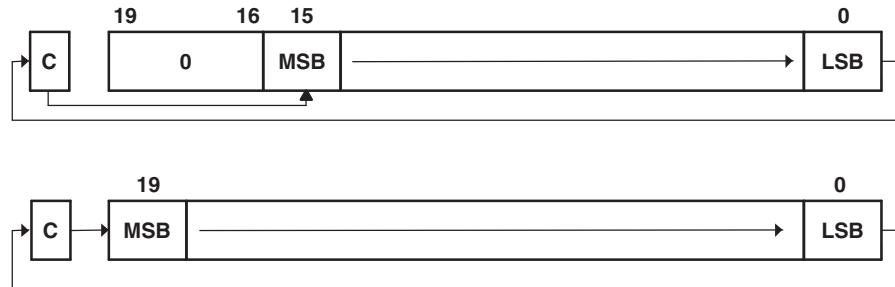
<b>RRCM.A</b>	Rotate right through carry the 20-bit CPU register content
<b>RRCM.[W]</b>	Rotate right through carry the 16-bit CPU register content
<b>Syntax</b>	RRCM.A #n,Rdst $1 \leq n \leq 4$ RRCM.W #n,Rdst or RRCM #n,Rdst $1 \leq n \leq 4$
<b>Operation Description</b>	C → MSB → MSB-1 ... LSB+1 → LSB → C The destination operand is shifted right by one, two, three, or four bit positions as shown in <a href="#">Figure 6-50</a> . 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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	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 6-50. Rotate Right Through Carry RRCM[W] and RRCM.A**

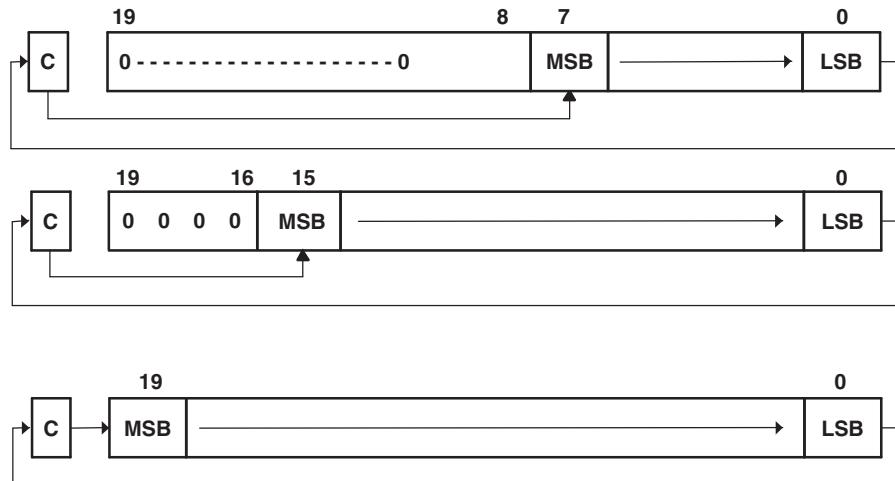
**6.6.3.28 RRCX**

<b>RRCX.A</b>	Rotate right through carry the 20-bit operand
<b>RRCX.[W]</b>	Rotate right through carry the 16-bit operand
<b>RRCX.B</b>	Rotate right through carry the 8-bit operand
<b>Syntax</b>	<pre>RRCX.A Rdst RRCX.W Rdst RRCX Rdst RRCX.B Rdst  RRCX.A dst RRCX dst or      RRCX.W dst RRCX.B dst</pre>
<b>Operation</b>	$C \rightarrow \text{MSB} \rightarrow \text{MSB-1} \dots \text{LSB+1} \rightarrow \text{LSB} \rightarrow C$
<b>Description</b>	Register mode for the destination: the destination operand is shifted right by one bit position as shown in <a href="#">Figure 6-51</a> . 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 <a href="#">Figure 6-52</a> . 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.
<b>Status Bits</b>	<p>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</p> <p>Z: Set if result is zero, reset otherwise</p> <p>C: Loaded from the LSB</p> <p>V: Reset</p>
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The 20-bit operand at address EDE is shifted right by one position. The MSB is loaded with 1.

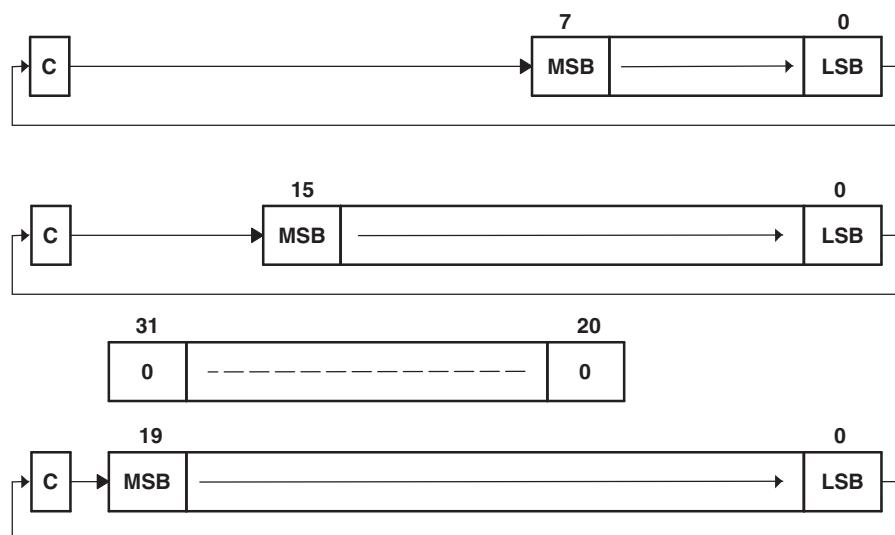
```
SETC          ; Prepare carry for MSB
RRCX.A       EDE          ; EDE = EDE >> 1 + 80000h
```

**Example**      The word in R6 is shifted right by 12 positions.

RPT #12  
 RRCX.W R6 ; R6 = R6 » 12. R6.19:16 = 0



**Figure 6-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode**



**Figure 6-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode**

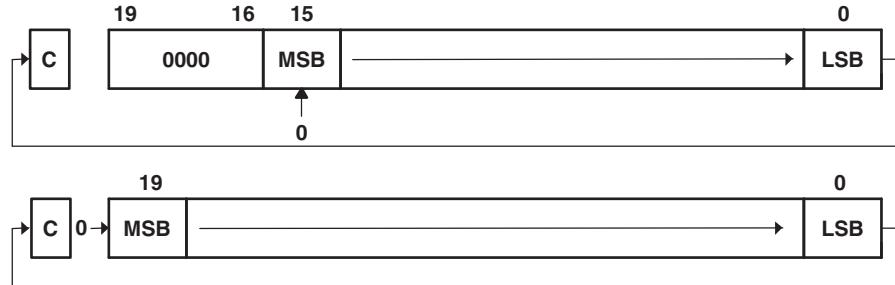
**6.6.3.29 RRUM**

<b>RRUM.A</b>	Rotate right through carry the 20-bit CPU register content
<b>RRUM.[W]</b>	Rotate right through carry the 16-bit CPU register content
<b>Syntax</b>	RRUM.A #n,Rdst $1 \leq n \leq 4$ RRUM.W #n,Rdst or RRUM #n,Rdst $1 \leq n \leq 4$
<b>Operation Description</b>	0 → MSB → MSB-1 ... LSB+1 → LSB → C The destination operand is shifted right by one, two, three, or four bit positions as shown in <a href="#">Figure 6-53</a> . 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.
<b>Status Bits</b>	Note : This instruction does not use the extension word. 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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

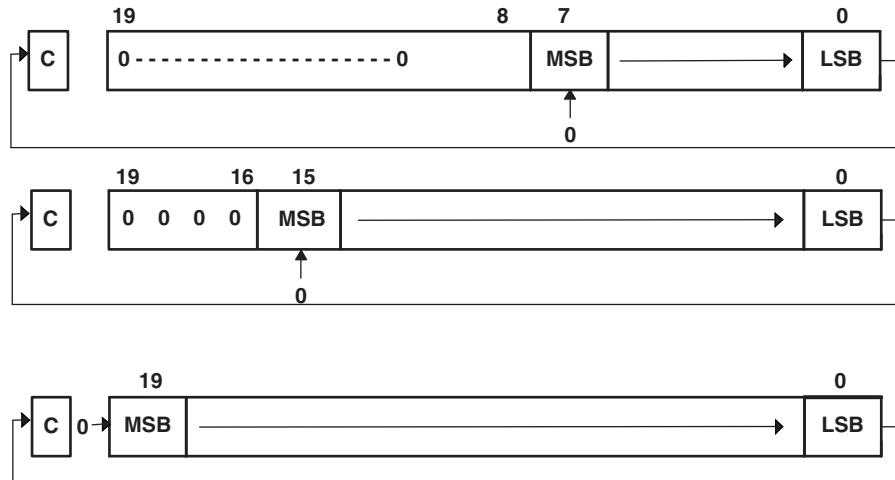


**Figure 6-53. Rotate Right Unsigned RRUM.[W] and RRUM.A**

### 6.6.3.30 RRUX

<b>RRUX.A</b>	Shift right unsigned the 20-bit CPU register content
<b>RRUX.[W]</b>	Shift right unsigned the 16-bit CPU register content
<b>RRUX.B</b>	Shift right unsigned the 8-bit CPU register content
<b>Syntax</b>	RRUX.A Rdst RRUX.W Rdst RRUX Rdst RRUX.B Rdst
<b>Operation</b>	$C=0 \rightarrow MSB \rightarrow MSB-1 \dots LSB+1 \rightarrow LSB \rightarrow C$
<b>Description</b>	RRUX is valid for register mode only: the destination operand is shifted right by one bit position as shown in <a href="#">Figure 6-54</a> . 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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The word in R6 is shifted right by 12 positions.

```
RPT      #12
RRUX.W  R6      ; R6 = R6 >> 12. R6.19:16 = 0
```



**Figure 6-54. Rotate Right Unsigned RRUX(.B.,A) – Register Mode**

**6.6.3.31 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 + 0FFFFh + 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

### 6.6.3.32 SUBX

<b>SUBX.A</b>	Subtract source address-word from destination address-word
<b>SUBX.[W]</b>	Subtract source word from destination word
<b>SUBX.B</b>	Subtract source byte from destination byte
<b>Syntax</b>	<code>SUBX.A src,dst</code> <code>SUBX src,dst or SUBX.W src,dst</code> <code>SUBX.B src,dst</code>
<b>Operation</b>	$(\text{not. src}) + 1 + \text{dst} \rightarrow \text{dst}$ or $\text{dst} - \text{src} \rightarrow \text{dst}$
<b>Description</b>	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.
<b>Status Bits</b>	N: Set if result is negative ( $\text{src} > \text{dst}$ ), reset if positive ( $\text{src} \leq \text{dst}$ ) Z: Set if result is zero ( $\text{src} = \text{dst}$ ), reset otherwise ( $\text{src} \neq \text{dst}$ ) 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)
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	A 20-bit constant 87654h is subtracted from EDE (LSBs) and EDE+2 (MSBs).

```
SUBX.A #87654h,EDE ; Subtract 87654h from EDE+2 | EDE
```

<b>Example</b>	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)
```

<b>Example</b>	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
```

**6.6.3.33 SUBCX**

<b>SUBCX.A</b>	Subtract source address-word with carry from destination address-word
<b>SUBCX.[W]</b>	Subtract source word with carry from destination word
<b>SUBCX.B</b>	Subtract source byte with carry from destination byte
<b>Syntax</b>	SUBCX.A src,dst SUBCX src,dst or SUBCX.W src,dst SUBCX.B src,dst
<b>Operation</b>	$(\text{not. src}) + C + \text{dst} \rightarrow \text{dst}$ or $\text{dst} - (\text{src} - 1) + C \rightarrow \text{dst}$
<b>Description</b>	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.
<b>Status Bits</b>	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).
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
```

<b>Example</b>	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
```

<b>Example</b>	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
```

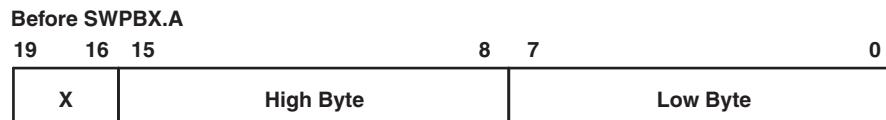
### 6.6.3.34 SWPBX

<b>SWPBX.A</b>	Swap bytes of lower word
<b>SWPBX.[W]</b>	Swap bytes of word
<b>Syntax</b>	SWPBX.A dst SWPBX dst or SWPBX.W dst
<b>Operation</b>	dst.15:8 ↔ dst.7:0
<b>Description</b>	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.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

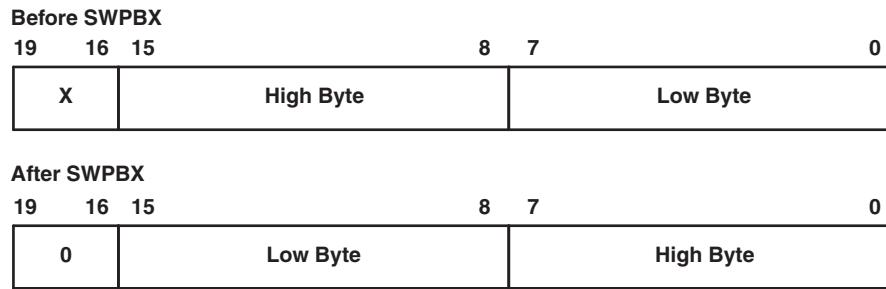
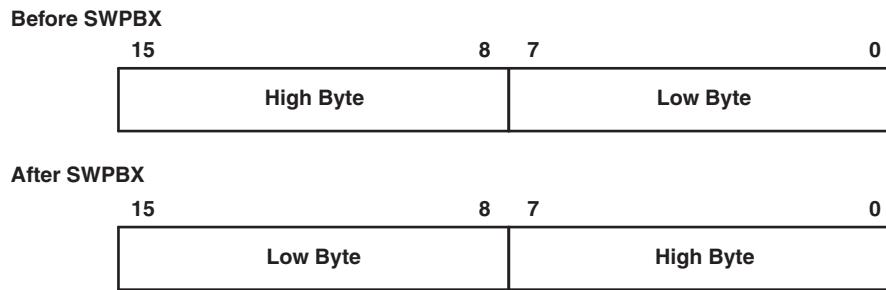
```
MOVA      #23456h, R5        ; 23456h -> R5
SWPBX.W  R5                  ; 05634h -> R5
```



**Figure 6-55. Swap Bytes SWPBX.A Register Mode**



**Figure 6-56. Swap Bytes SWPBX.A In Memory**

**Figure 6-57. Swap Bytes SWPBX[.W] Register Mode****Figure 6-58. Swap Bytes SWPBX[.W] In Memory**

### 6.6.3.35 SXTX

<b>SXTX.A</b>	Extend sign of lower byte to address-word
<b>SXTX.[W]</b>	Extend sign of lower byte to word
<b>Syntax</b>	SXTX.A dst
	SXTX dst OR SXTX.W dst
<b>Operation</b>	dst.7 → dst.15:8, Rdst.7 → Rdst.19:8 (Register mode)
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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.

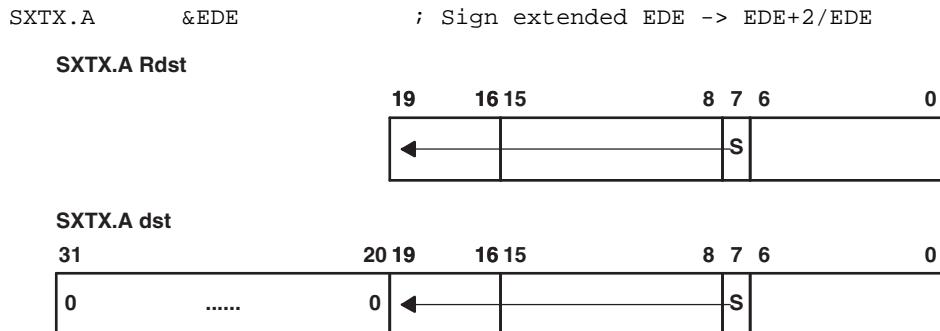


Figure 6-59. Sign Extend SXTX.A

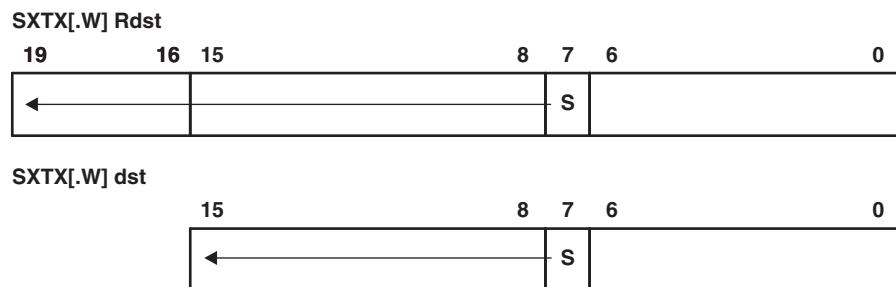


Figure 6-60. Sign Extend SXTX.[.W]

**6.6.3.36 TSTX**

<b>* TSTX.A</b>	Test destination address-word	
<b>* TSTX.[W]</b>	Test destination word	
<b>* TSTX.B</b>	Test destination byte	
<b>Syntax</b>	TSTX.A dst TSTX dst or            TSTX.W dst TSTX.B dst	
<b>Operation</b>	TSTX.A	dst + 0FFFFh + 1
	TSTX.W	dst + 0FFFFh + 1
	TSTX.B	dst + 0FFh + 1
<b>Emulation</b>	CMPX.A #0,dst	
	CMPX #0,dst	
	CMPX.B #0,dst	
<b>Description</b>	The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.	
<b>Status Bits</b>	N: Set if destination is negative, reset if positive Z: Set if destination contains zero, reset otherwise C: Set V: Reset	
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.	
<b>Example</b>	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

```

### 6.6.3.37 XORX

<b>XORX.A</b>	Exclusive OR source address-word with destination address-word
<b>XORX.[W]</b>	Exclusive OR source word with destination word
<b>XORX.B</b>	Exclusive OR source byte with destination byte
<b>Syntax</b>	<code>XORX.A src,dst</code> <code>XORX src,dst or XORX.W src,dst</code> <code>XORX.B src,dst</code>
<b>Operation</b>	<code>src .xor. dst → dst</code>
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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.
```

#### 6.6.4 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.

### 6.6.4.1 ADDA

<b>ADDA</b>	Add 20-bit source to a 20-bit destination register
<b>Syntax</b>	ADDA Rsrc,Rdst ADDA #imm20,Rdst
<b>Operation</b>	$\text{src} + \text{Rdst} \rightarrow \text{Rdst}$
<b>Description</b>	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.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

**6.6.4.2 BRA**

<b>* BRA</b>	Branch to destination
<b>Syntax</b>	BRA dst
<b>Operation</b>	dst → PC
<b>Emulation</b>	MOVA dst,PC
<b>Description</b>	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).

<b>Status Bits</b>	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
```

**6.6.4.3 CALLA**

<b>CALLA</b>	Call a subroutine
<b>Syntax</b>	CALLA dst
<b>Operation</b>	<p>dst → tmp 20-bit dst is evaluated and stored</p> <p>SP – 2 → SP</p> <p>PC.19:16 → @SP updated PC with return address to TOS (MSBs)</p> <p>SP – 2 → SP</p> <p>PC.15:0 → @SP updated PC to TOS (LSBs)</p> <p>tmp → PC saved 20-bit dst to PC</p>
<b>Description</b>	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.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Examples</b>	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)
```

#### 6.6.4.4 CLRA

* <b>CLRA</b>	Clear 20-bit destination register
<b>Syntax</b>	CLRA Rdst
<b>Operation</b>	$0 \rightarrow \text{Rdst}$
<b>Emulation</b>	MOVA #0, Rdst
<b>Description</b>	The destination register is cleared.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	The 20-bit value in R10 is cleared.

```
CLRA    R10      ; 0 -> R10
```

### 6.6.4.5 CMPA

<b>CMPA</b>	Compare the 20-bit source with a 20-bit destination register
<b>Syntax</b>	CMPA Rsrc,Rdst CMPA #imm20,Rdst
<b>Operation</b>	(.not. src) + 1 + Rdst or Rdst – src
<b>Description</b>	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.
<b>Status Bits</b>	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
```

### 6.6.4.6 DECDA

<b>* DECDA</b>	Double-decrement 20-bit destination register
<b>Syntax</b>	DECDA Rdst
<b>Operation</b>	Rdst – 2 → Rdst
<b>Emulation</b>	SUBA #2,Rdst
<b>Description</b>	The destination register is decremented by two. The original contents are lost.
<b>Status Bits</b>	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
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The 20-bit value in R5 is decremented by 2.

```
DECDA    R5      ; Decrement R5 by two
```

### 6.6.4.7 INCDA

<b>* INCDA</b>	Double-increment 20-bit destination register
<b>Syntax</b>	INCDA Rdst
<b>Operation</b>	Rdst + 2 → Rdst
<b>Emulation</b>	ADDA #2,Rdst
<b>Description</b>	The destination register is incremented by two. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if Rdst contained 0FFFFEh, reset otherwise Set if Rdst contained 0FFEh, reset otherwise Set if Rdst contained 0FEh, reset otherwise C: Set if Rdst contained 0FFFFEh or 0FFFFFh, reset otherwise Set if Rdst contained 0FFEh or 0FFFh, reset otherwise Set if Rdst contained 0FEh or 0FFh, reset otherwise V: Set if Rdst contained 07FFEh or 07FFFFh, reset otherwise Set if Rdst contained 07FEh or 07FFFh, reset otherwise Set if Rdst contained 07Eh or 07Fh, reset otherwise
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The 20-bit value in R5 is incremented by two.

```
INCDA    R5      ; Increment R5 by two
```

**6.6.4.8 MOVA**

<b>MOVA Syntax</b>	Move the 20-bit source to the 20-bit destination
	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
<b>Operation</b>	src → Rdst
	Rsrc → dst
<b>Description</b>	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.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Examples</b>	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  $\pm$  32 K.

MOVA R13,EDE ; R13 -> EDE. 2 words transferred

**6.6.4.9 RETA**

<b>* RETA</b>	Return from subroutine
<b>Syntax</b>	RETA
<b>Operation</b>	<p><math>@SP \rightarrow PC.15:0</math> LSBs (15:0) of saved PC to PC.15:0</p> <p><math>SP + 2 \rightarrow SP</math></p> <p><math>@SP \rightarrow PC.19:16</math> MSBs (19:16) of saved PC to PC.19:16</p> <p><math>SP + 2 \rightarrow SP</math></p>
<b>Emulation</b>	MOVA @SP+,PC
<b>Description</b>	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.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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
...
SUBR      PUSHM.A #2,R14    ; Save R14 and R13 (20 bit data)
          ...
          POPM.A #2,R14    ; Restore R13 and R14 (20 bit data)
          RETA            ; Return (to full address space)

```

### 6.6.4.10 SUBA

<b>SUBA</b>	Subtract 20-bit source from 20-bit destination register
<b>Syntax</b>	SUBA Rsrc,Rdst SUBA #imm20,Rdst
<b>Operation</b>	(.not.src) + 1 + Rdst → Rdst or Rdst – src → Rdst
<b>Description</b>	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.
<b>Status Bits</b>	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)
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

**6.6.4.11 TSTA**

<b>* TSTA</b>	Test 20-bit destination register
<b>Syntax</b>	TSTA Rdst
<b>Operation</b>	dst + 0FFFFFh + 1 dst + 0FFFFh + 1 dst + 0FFh + 1
<b>Emulation</b>	CMPA #0,Rdst
<b>Description</b>	The destination register is compared with zero. The status bits are set according to the result. The destination register is not affected.
<b>Status Bits</b>	N: Set if destination register is negative, reset if positive Z: Set if destination register contains zero, reset otherwise C: Set V: Reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	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

```

## **Flash Memory Controller**

This chapter describes the operation of the flash memory controller.

<b>Topic</b>		<b>Page</b>
7.1	<b>Flash Memory Introduction</b> .....	342
7.2	<b>Flash Memory Segmentation</b> .....	343
7.3	<b>Flash Memory Operation</b> .....	345
7.4	<b>FCTL Registers</b> .....	360

## 7.1 Flash Memory Introduction

The flash memory is byte, word, and long-word addressable and programmable. The flash memory module has an integrated controller that controls programming and erase operations. The module contains three registers, a timing generator, and a voltage generator to supply program and erase voltages. The cumulative high-voltage time must not be exceeded, and each 32-bit word can be written not more than four times (in byte, word, or long word write modes) before another erase cycle (see device-specific data sheet for details).

The flash memory features include:

- Internal programming voltage generation
- Byte, word (2 bytes), and long (4 bytes) programmable
- Ultralow power operation
- Segment erase, bank erase (device specific), and mass erase
- Marginal 0 and marginal 1 read modes
- Each bank (device specific) can be erased individually while program execution can proceed in a different flash bank.

---

**NOTE:** Bank operations are not supported on all devices. See the device-specific data sheet for banks supported and their respective sizes.

---

The block diagram of the flash memory and controller is shown in Figure 7-1.

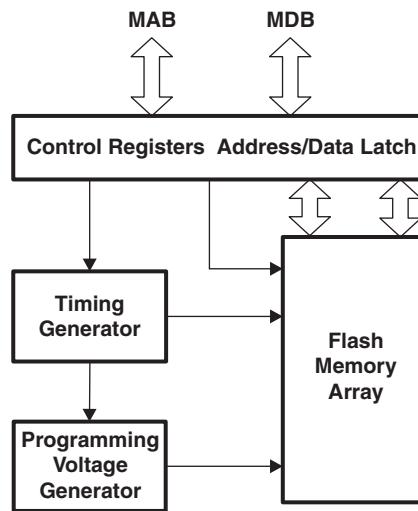


Figure 7-1. Flash Memory Module Block Diagram

## 7.2 Flash Memory Segmentation

The flash main memory is partitioned into 512-byte segments. Single bits, bytes, or words can be written to flash memory, but a segment is the smallest size of the flash memory that can be erased.

The flash memory is partitioned into main and information memory sections. There is no difference in the operation of the main and information memory sections. Code and data can be located in either section. The difference between the sections is the segment size.

There are four information memory segments, A through D. Each information memory segment contains 128 bytes and can be erased individually.

The bootstrap loader (BSL) memory consists of four segments, A through D. Each BSL memory segment contains 512 bytes and can be erased individually.

The main memory segment size is 512 byte. See the device-specific data sheet for the start and end addresses of each bank, when available, and for the complete memory map of a device.

Figure 7-2 shows the flash segmentation using an example of 256-KB flash that has four banks of 64 KB (segments A through D) and information memory.

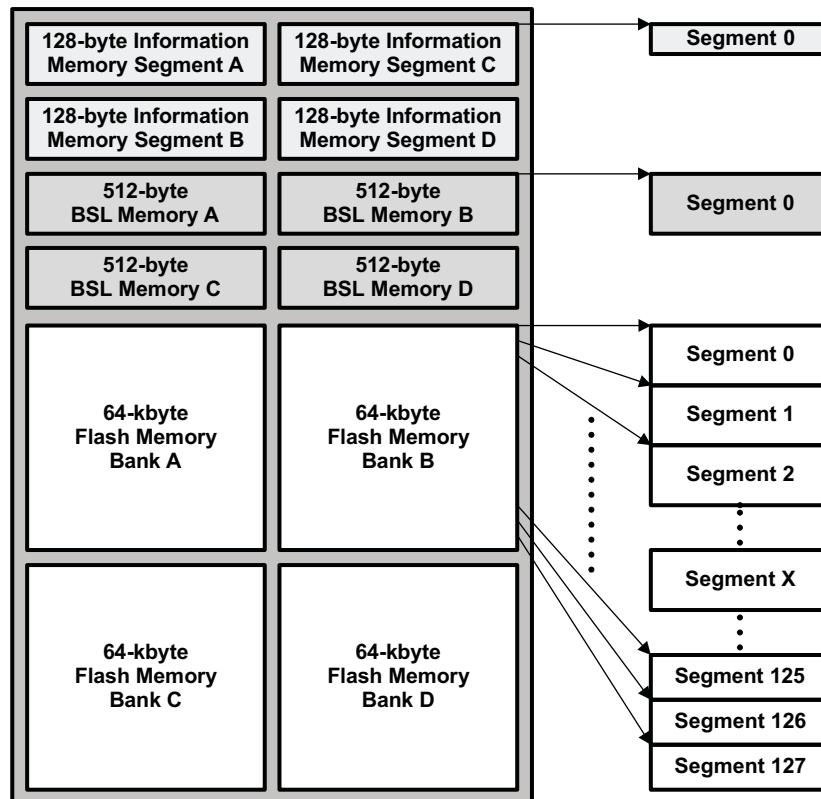


Figure 7-2. 256-KB Flash Memory Segments Example

### 7.2.1 Segment A

Segment A of the information memory is locked separately from all other segments with the LOCKA bit. If LOCKA = 1, segment A cannot be written or erased, and all information memory is protected from being segment erased. If LOCKA = 0, segment A can be erased and written like any other flash memory segment.

The state of the LOCKA bit is toggled when a 1 is written to it. Writing a 0 to LOCKA has no effect. This allows existing flash programming routines to be used unchanged.

```
; Unlock Info Memory
    MOV      #FWPW,&FCTL4           ; Clear LOCKINFO, if set
; Unlock SegmentA
    BIT      #LOCKA,&FCTL3          ; Test LOCKA
    JZ       SEGA_UNLOCKED         ; Already unlocked?
    MOV      #FWPW+LOCKA,&FCTL3     ; No, unlock SegmentA
SEGA_UNLOCKED
; SegmentA is unlocked

; Lock SegmentA
    BIT      #LOCKA,&FCTL3          ; Test LOCKA
    JNZ     SEGA_LOCKED           ; Already locked?
    MOV      #FWPW+LOCKA,&FCTL3     ; No, lock SegmentA
SEGA_LOCKED
; SegmentA is locked
; Lock Info Memory
    MOV      #FWPW+LOCKINFO,&FCTL4   ; Set LOCKINFO
```

## 7.3 Flash Memory Operation

The default mode of the flash memory is read mode. In read mode, the flash memory is not being erased or written, the flash timing generator and voltage generator are off, and the memory operates identically to ROM.

**Read and fetch while erase** – The flash memory allows execution of a program from flash while a different flash bank is erased. Data reads are also possible from any flash bank not being erased.

---

**NOTE: Read and fetch while erase**

The read and fetch while erase feature is available in flash memory configurations where more than one flash bank is available. If there is one flash bank available, holding the complete flash program memory, the read from the program memory and information memory and BSL memory during the erase is not provided. [Table 7-1](#) summarizes which flash operations are supported for devices that support read and fetch while erasing.

**Table 7-1. Supported Simultaneous Code Execution and Flash Operations**

Flash Operation	Simultaneous Code Execution	
	Within Flash	Within RAM
Bank Erase	Supported Executed code must not reside in the bank to be erased	Supported
Segment Erase	Not Supported	Supported
Byte, word, long-word write	Not supported	Supported

Flash memory is in-system programmable (ISP) without the need for additional external voltage. The CPU can program the flash memory. The flash memory write and erase modes are selected by the BLKWRT, WRT, MERAS, and ERASE bits and are:

- Byte, word, or long-word (32-bit) write
- Block write
- Segment erase
- Bank erase (only main memory)
- Mass erase (all main memory banks)
- Read during bank erase (except for the one currently read from)

Reading or writing to flash memory while it is busy programming or erasing (page, mass, or bank) from the same bank is prohibited. Any flash erase or programming can be initiated from within flash memory or RAM.

### 7.3.1 Erasing Flash Memory

The logical value of an erased flash memory bit is 1. Each bit can be programmed from 1 to 0 individually, but to reprogram from 0 to 1 requires an erase cycle. The smallest amount of flash that can be erased is one segment. There are three erase modes selected by the ERASE and MERAS bits listed in [Table 7-2](#).

**Table 7-2. Erase Modes**

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Bank erase (of one bank) selected by the dummy write address <sup>(1)</sup>
1	1	Mass erase (all memory banks are erased. Information memory A to D and BSL segments A to D are not erased)

<sup>(1)</sup> Bank operations are not supported on all devices. See the device-specific data sheet for support of bank operations.

### 7.3.1.1 Erase Cycle

An erase cycle is initiated by a dummy write to the address range of the segment to be erased. The dummy write starts the erase operation and is required for all erase operations including mass erase. Figure 7-3 shows the erase cycle timing. The BUSY bit is set immediately after the dummy write and remains set throughout the erase cycle. BUSY, MERAS, and ERASE are automatically cleared when the cycle completes. No additional dummy write access should be made while the control bits are cleared, otherwise, ACCVIFG is set. The mass erase cycle timing is not dependent on the amount of flash memory present on a device. Erase cycle times are equivalent for all devices.

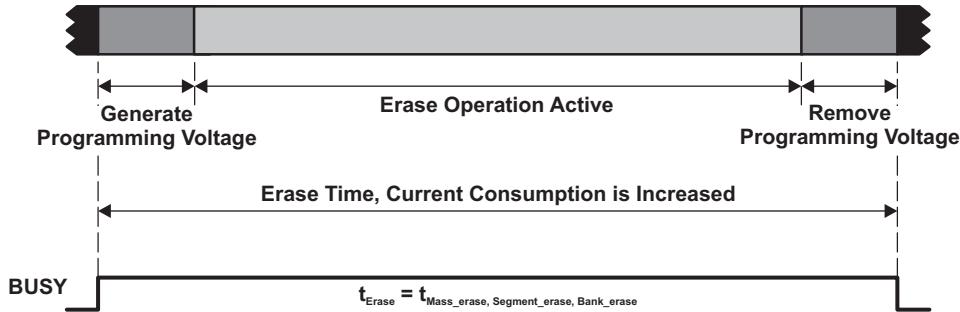


Figure 7-3. Erase Cycle Timing

### 7.3.1.2 Erasing Main Memory

The main memory consists of one or more banks. Each bank can be erased individually (bank erase). All main memory banks can be erased in the mass erase mode.

### 7.3.1.3 Erasing Information Memory or BSL Flash Segments

The information memory A to D and the BSL segments A to D can only be erased in segment erase mode. They are not erased during a bank erase or a mass erase. Erasing is only possible by first clearing the LOCKINFO bit.

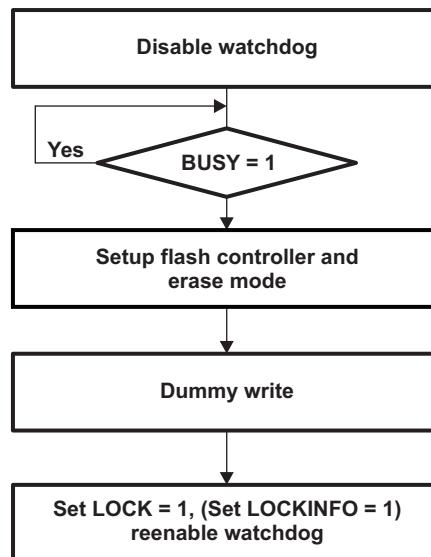
### 7.3.1.4 Initiating Erase From Flash

An erase cycle can be initiated from within flash memory. During a bank erase, code can be executed from flash or RAM. The executed code cannot be located in a bank to be erased.

For any segment erase, the CPU is held until the erase cycle completes regardless of the bank the code resides in. After the segment erase cycle ends, the CPU resumes code execution with the instruction following the dummy write.

When initiating an erase cycle from within flash memory, it is possible to erase the code needed for execution after the erase operation. If this occurs, CPU execution is unpredictable after the erase cycle.

The flow to initiate an erase from flash is shown in [Figure 7-4](#).



**Figure 7-4. Erase Cycle From Flash**

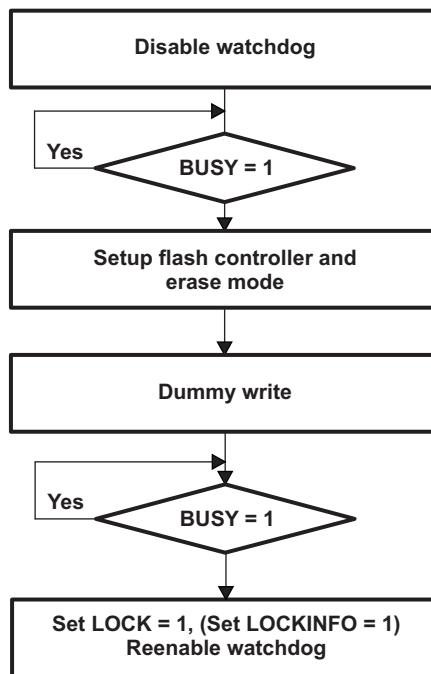
```

; Segment Erase from flash.
; Assumes Program Memory. Information memory or BSL
; requires LOCKINFO to be cleared as well.
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV    #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1   BIT    #BUSY,&FCTL3            ; Test BUSY
    JNZ    L1                  ; Loop while busy
    MOV    #FWPW,&FCTL3            ; Clear LOCK
    MOV    #FWPW+ERASE,&FCTL1        ; Enable segment erase
    CLR    &0FC10h              ; Dummy write
L2   BIT    #BUSY,&FCTL3            ; Test BUSY
    JNZ    L2                  ; Loop while busy
    MOV    #FWPW+LOCK,&FCTL3        ; Done, set LOCK
    ...                          ; Re-enable WDT?
  
```

### 7.3.1.5 Initiating Erase From RAM

An erase cycle can be initiated from RAM. In this case, the CPU is not held and continues to execute code from RAM. The mass erase (all main memory banks) operation is initiated while executing from RAM. The BUSY bit is used to determine the end of the erase cycle. If the flash is busy completing a bank erase, flash addresses of a different bank can be used to read data or to fetch instructions. While the flash is BUSY, starting an erase cycle or a programming cycle causes an access violation, ACCIFG is set to 1, and the result of the erase operation is unpredictable.

The flow to initiate an erase from flash from RAM is shown in [Figure 7-5](#).



**Figure 7-5. Erase Cycle From RAM**

```

; segment Erase from RAM.
; Assumes Program Memory. Information memory or BSL
; requires LOCKINFO to be cleared as well.
; Assumes ACCVIE = NMIIIE = OFIE = 0.
    MOV    #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1   BIT    #BUSY,&FCTL3            ; Test BUSY
    JNZ    L1                      ; Loop while busy
    MOV    #FWPW,&FCTL3            ; Clear LOCK
    MOV    #FWPW+ERASE,&FCTL1        ; Enable page erase
    CLR    &0FC10h                  ; Dummy write
L2   BIT    #BUSY,&FCTL3            ; Test BUSY
    JNZ    L2                      ; Loop while busy
    MOV    #FWPW+LOCK,&FCTL3        ; Done, set LOCK
    ...
  
```

### 7.3.2 Writing Flash Memory

The write modes, selected by the WRT and BLKWRT bits, are listed in [Table 7-3](#).

**Table 7-3. Write Modes**

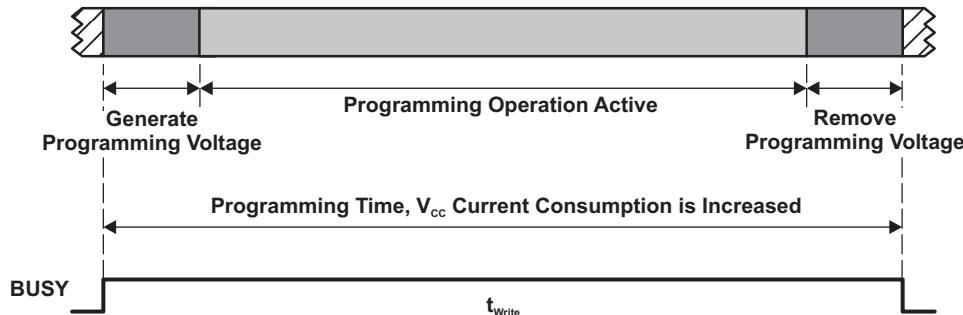
BLKWRT	WRT	Write Mode
0	1	Byte or word write
1	0	Long-word write
1	1	Long-word block write

The write modes use a sequence of individual write instructions. Using the long-word write mode is approximately twice as fast as the byte or word mode. Using the long-word block write mode is approximately four times faster than byte or word mode, because the voltage generator remains on for the complete block write, and long-words are written in parallel. Any instruction that modifies a destination can be used to modify a flash location in either byte or word write mode, long-word write mode, or block long-word write mode.

The BUSY bit is set while the write operation is active and cleared when the operation completes. If the write operation is initiated from RAM, the CPU must not access flash while BUSY is set to 1. Otherwise, an access violation occurs, ACCVIFG is set, and the flash write is unpredictable.

#### 7.3.2.1 Byte or Word Write

A byte or word write operation can be initiated from within flash memory or from RAM. When initiating from within flash memory, the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write access. The byte, word, and long-word write timing is shown in [Figure 7-6](#). Byte, word, and long-word write times are identical.



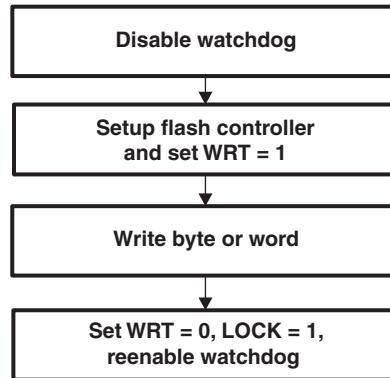
**Figure 7-6. Byte, Word, and Long-Word Write Timing**

When a byte or word write is executed from RAM, the CPU continues to execute code from RAM. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

In any write mode, the internally-generated programming voltage is applied to the complete 128-byte block. The cumulative programming time,  $t_{CPT}$ , must not be exceeded for any block. Each byte, word, or long-word write adds to the cumulative program time of a segment. If the maximum cumulative program time is reached or exceeded, the segment must be erased. Further programming or using the data returns unpredictable results (see the device-specific data sheet for specifications).

### 7.3.2.2 Initiating Byte or Word Write From Flash

The flow to initiate a byte or word write from flash is shown in [Figure 7-7](#).



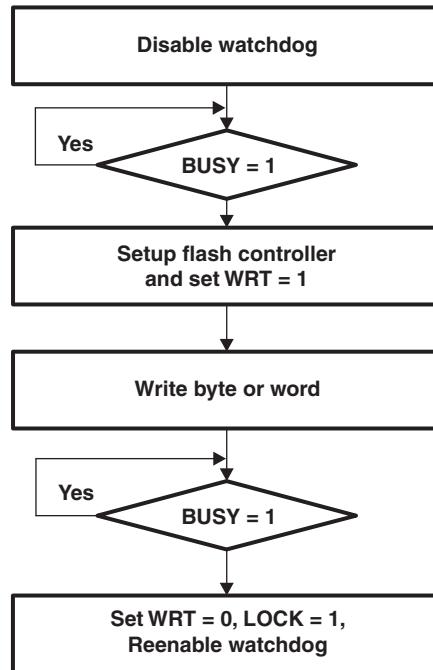
**Figure 7-7. Initiating a Byte or Word Write From Flash**

```

; Byte or word write from flash.
; Assumes 0x0FF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV #WDTPW+WDTHOLD,&WDTCTL ; Disable WDT
MOV #FWPW,&FCTL3 ; Clear LOCK
MOV #FWPW+WRT,&FCTL1 ; Enable write
MOV #0123h,&0FF1Eh ; 0123h -> 0x0FF1E
MOV #FWPW,&FCTL1 ; Done. Clear WRT
MOV #FWPW+LOCK,&FCTL3 ; Set LOCK
... ; Re-enable WDT?
  
```

### 7.3.2.3 Initiating Byte or Word Write From RAM

The flow to initiate a byte or word write from RAM is shown in [Figure 7-8](#).



**Figure 7-8. Initiating a Byte or Word Write From RAM**

```
; Byte or word write from RAM.  
; Assumes 0xFF1E is already erased  
; Assumes ACCVIE = NMIIIE = OFIE = 0.  
    MOV    #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT  
L1    BIT    #BUSY,&FCTL3            ; Test BUSY  
    JNZ    L1                          ; Loop while busy  
    MOV    #FWPW,&FCTL3              ; Clear LOCK  
    MOV    #FWPW+WRT,&FCTL1          ; Enable write  
    MOV    #0123h,&0FF1Eh            ; 0123h -> 0xFF1E  
L2    BIT    #BUSY,&FCTL3            ; Test BUSY  
    JNZ    L2                          ; Loop while busy  
    MOV    #FWPW,&FCTL1              ; Clear WRT  
    MOV    #FWPW+LOCK,&FCTL3          ; Set LOCK  
    ...                                ; Re-enable WDT?
```

### 7.3.2.4 Long-Word Write

A long-word write operation can be initiated from within flash memory or from RAM. The BUSY bit is set to 1 after 32 bits are written to the flash controller and the programming cycle starts. When initiating from within flash memory, the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write access. The long-word write timing is shown in [Figure 7-6](#).

A long-word consists of four consecutive bytes aligned to at 32-bit address (only the lower two address bits are different). The bytes can be written in any order or any combination of bytes and words. If a byte or word is written more than once, the last data written to the four bytes are stored into the flash memory.

If a write to a flash address outside of the 32-bit address happens before all four bytes are available, the data written so far is discarded, and the latest byte or word written defines the new 32-bit aligned address.

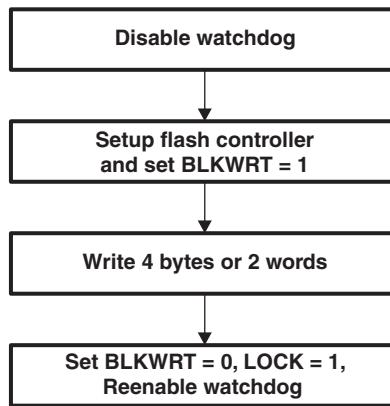
When 32 bits are available, the write cycle is executed. When executing from RAM, the CPU continues to execute code. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

In long-word write mode, the internally-generated programming voltage is applied to a complete 128-byte block. The cumulative programming time,  $t_{CPT}$ , must not be exceeded for any block. Each write adds to the cumulative program time of a segment. If the maximum cumulative program time is reached or exceeded, the segment must be erased. Further programming or using the data returns unpredictable results.

With each write, the amount of time the block is subjected to the programming voltage accumulates. If the cumulative programming time is reached or exceeded, the block must be erased before further programming or use (see the device-specific data sheet for specifications).

### 7.3.2.5 Initiating Long-Word Write From Flash

The flow to initiate a long-word write from flash is shown in [Figure 7-9](#).



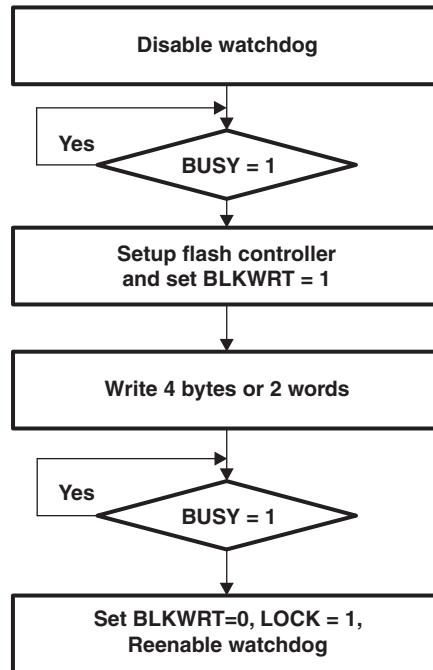
**Figure 7-9. Initiating Long-Word Write From Flash**

```

; Long-word write from flash.
; Assumes 0xFF1C and 0xFF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV #WDTPW+WDTHOLD,&WDTCTL ; Disable WDT
MOV #FWPW,&FCTL3 ; Clear LOCK
MOV #FWPW+BLKWRT,&FCTL1 ; Enable 2-word write
MOV #0123h,&0FF1Ch ; 0123h -> 0xFF1C
MOV #45676h,&0FF1Eh ; 04567h -> 0xFF1E
MOV #FWPW,&FCTL1 ; Done. Clear BLKWRT
MOV #FWPW+LOCK,&FCTL3 ; Set LOCK
MOV #FWPW+LOCK,&FCTL3 ; Re-enable WDT?
...
  
```

### 7.3.2.6 Initiating Long-Word Write From RAM

The flow to initiate a long-word write from RAM is shown in [Figure 7-10](#).



**Figure 7-10. Initiating Long-Word Write from RAM**

```

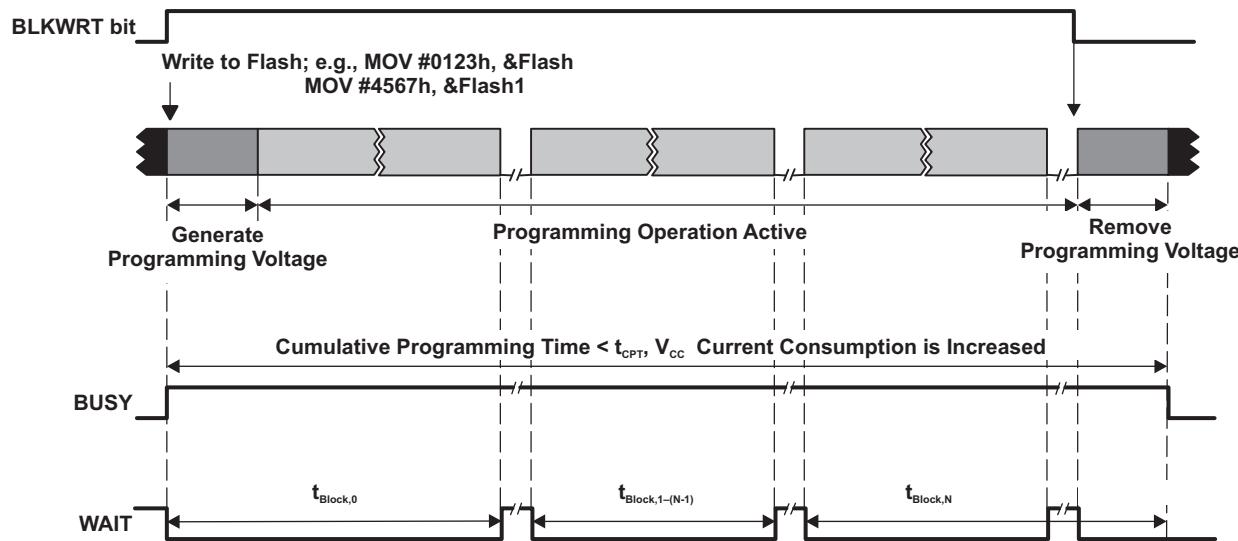
; Two 16-bit word writes from RAM.
; Assumes 0xFF1C and 0xFF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV    #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1    BIT    #BUSY,&FCTL3            ; Test BUSY
    JNZ    L1                          ; Loop while busy
    MOV    #FWPW,&FCTL3              ; Clear LOCK
    MOV    #FWPW+BLKWRT,&FCTL1        ; Enable write
    MOV    #0123h,&0FF1Ch             ; 0123h -> 0xFF1C
    MOV    #4567h,&0FF1Eh             ; 4567h -> 0xFF1E
L2    BIT    #BUSY,&FCTL3            ; Test BUSY
    JNZ    L2                          ; Loop while busy
    MOV    #FWPW,&FCTL1              ; Clear WRT
    MOV    #FWPW+LOCK,&FCTL3          ; Set LOCK
    ...                            ; Re-enable WDT?

```

### 7.3.2.7 Block Write

The block write can be used to accelerate the flash write process when many sequential bytes or words need to be programmed. The flash programming voltage remains on for the duration of writing the 128-byte row. The cumulative programming time,  $t_{CPT}$ , must not be exceeded for any row during a block write. Only long-word writes are possible using block write mode.

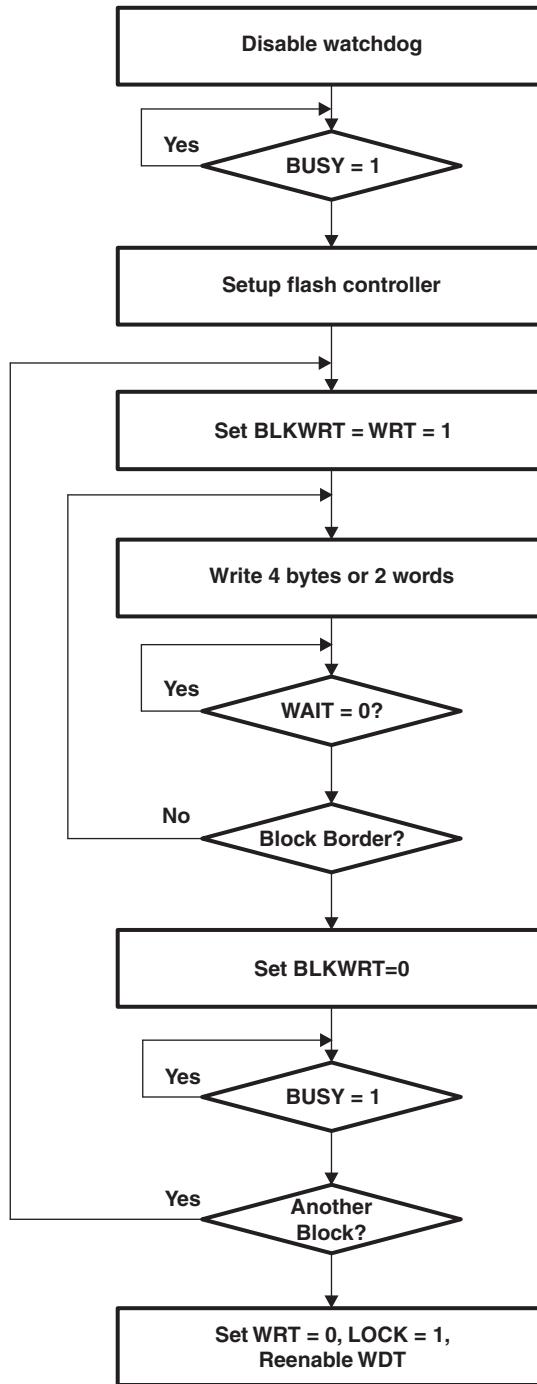
A block write cannot be initiated from within flash memory. The block write must be initiated from RAM. The BUSY bit remains set throughout the duration of the block write. The WAIT bit must be checked between writing four bytes, or two words, to the block. When WAIT is set, then four bytes, or two 16-bit words, of the block can be written. When writing successive blocks, the BLKWRT bit must be cleared after the current block is completed. BLKWRT can be set initiating the next block write after the required flash recovery time given by  $t_{END}$ . BUSY is cleared following each block write completion, indicating the next block can be written. [Figure 7-11](#) shows the block write timing. The first long-word write requires  $t_{Block,0}$  and the last long-write requires  $t_{Block,N}$ . All other blocks require  $t_{Block,1-(N-1)}$ .



**Figure 7-11. Block-Write Cycle Timing**

### 7.3.2.8 Block Write Flow and Example

A block write flow is shown in [Figure 7-12](#) and the following code example.



**Figure 7-12. Block Write Flow**

```

; Write one block starting at 0F000h.
; Must be executed from RAM, Assumes Flash is already erased.
; Assumes ACCVIE = NMIIIE = OFIE = 0.

    MOV      #32,R5                      ; Use as write counter
    MOV      #0F000h,R6                   ; Write pointer
    MOV      #WDTPW+WDTHOLD ,&WDTCTL   ; Disable WDT
L1     BIT      #BUSY,&FCTL3          ; Test BUSY
    JNZ      L1                         ; Loop while busy
    MOV      #FWPW,&FCTL3            ; Clear LOCK
    MOV      #FWPW+BLKWRT+WRT ,&FCTL1  ; Enable block write
L2     MOV      Write_Value1,0(R6)      ; Write 1st location
    MOV      Write_Value2,2(R6)          ; Write 2nd word
L3     BIT      #WAIT,&FCTL3          ; Test WAIT
    JZ       L3                         ; Loop while WAIT=0
    INCD    R6                         ; Point to next words
    INCD    R6                         ; Point to next words
    DEC     R5                         ; Decrement write counter
    JNZ      L2                         ; End of block?
    MOV      #FWPW,&FCTL1            ; Clear WRT, BLKWRT
L4     BIT      #BUSY,&FCTL3          ; Test BUSY
    JNZ      L4                         ; Loop while busy
    MOV      #FWPW+LOCK ,&FCTL3        ; Set LOCK
    ...                                ; Re-enable WDT if needed

```

### 7.3.3 Flash Memory Access During Write or Erase

When a write or an erase operation is initiated from RAM while BUSY = 1, the CPU may not write to any flash location. Otherwise, an access violation occurs, ACCVIFG is set, and the result is unpredictable. ACCVIFG is also set if a Flash write or erase access is attempted without any Flash write or erase mode selected first.

When a write operation is initiated from within flash memory, the CPU continues code execution with the next instruction fetch after the write cycle completed (BUSY = 0).

The op-code 3FFFh is the JMP PC instruction. This causes the CPU to loop until the flash operation is finished. When the operation is finished and BUSY = 0, the flash controller allows the CPU to fetch the op-code and program execution resumes.

The flash access conditions while BUSY = 1 are listed in [Table 7-4](#).

**Table 7-4. Flash Access While Flash is Busy (BUSY = 1)**

Flash Operation	Flash Access	WAIT	Result
Bank erase	Read	0	From the erased bank: ACCVIFG = 0. 03FFFh is the value read. From any other flash location: ACCVIFG = 0. Valid read.
	Write	0	ACCVIFG = 1. Write is ignored.
	Instruction fetch	0	From the erased bank: ACCVIFG = 0. CPU fetches 03FFFh. This is the JMP PC instruction. From any other flash location: ACCVIFG = 0. Valid instruction fetch.
Segment erase	Read	0	ACCVIFG = 0: 03FFFh is the value read.
	Write	0	ACCVIFG = 1: Write is ignored.
	Instruction fetch	0	ACCVIFG = 0: CPU fetches 03FFFh. This is the JMP PC instruction.
Word or byte write or long-word write	Read	0	ACCVIFG = 0: 03FFFh is the value read.
	Write	0	ACCVIFG = 1: Write is ignored.
	Instruction fetch	0	ACCVIFG = 0: CPU fetches 03FFFh. This is the JMP PC instruction.
Block write	Any	0	ACCVIFG = 1: LOCK = 1, block write is exited.
	Read	1	ACCVIFG = 0: 03FFFh is the value read.
	Write	1	ACCVIFG = 0: Valid write
	Instruction fetch	1	ACCVIFG = 1: LOCK = 1, block write is exited

Interrupts are automatically disabled during any flash operation.

The watchdog timer (in watchdog mode) should be disabled before a flash erase cycle. A reset aborts the erase and the result is unpredictable. After the erase cycle has completed, the watchdog may be reenabled.

### 7.3.4 Stopping Write or Erase Cycle

Any write or erase operation can be stopped before its normal completion by setting the emergency exit bit EMEX. Setting the EMEX bit stops the active operation and resets the flash controller. All flash operations cease, the flash returns to read mode, and all bits in the FCTL1 register are reset. The LOCK bit of FCTL3 is set. The result of the intended operation is unpredictable.

#### 7.3.4.1 EMEX With Single Bank Flash Memory

For devices with single bank flash memories, write and erase operations initiated from flash, the CPU is held until the flash operation completes. Therefore it is not possible to perform an emergency exit by the EMEX bit. The emergency exit of write or erase operations initiated from RAM can be performed using the EMEX bit. The BUSY bit is used to determine the end of the emergency exit cycle. The user must ensure that code execution does not continue until the BUSY bit is cleared by the flash controller.

#### 7.3.4.2 EMEX With Multiple Bank Flash Memory

For devices with multiple bank flash memories, write and segment erase operations initiated from flash, regardless of which bank the code resides in, the CPU is held until the flash operation completes.

Therefore it is not possible to perform an emergency exit by the EMEX bit. For bank erase, there is a possibility to perform an EMEX if the bank being erased is not where the code resides. The BUSY bit is used to determine the end of the emergency exit cycle. The user must ensure that code execution does not continue until the BUSY bit is cleared by the flash controller.

The emergency exit of write or any erase operations initiated from RAM can be performed using the EMEX bit. The BUSY bit is used to determine the end of the emergency exit cycle. The user must ensure that code execution does not continue until the BUSY bit is cleared by the flash controller.

### 7.3.5 Checking Flash Memory

The result of a programming cycle of the flash memory can be checked by calculating and storing a checksum (CRC) of parts or the complete flash memory content. The CRC module can be used for this purpose (see the device-specific data sheet). During the runtime of the system, the known checksums can be recalculated and compared with the expected values stored in the flash memory. The program checking the flash memory content is executed in RAM.

To get an early indication of weak memory cells, reading the flash can be done in combination with the device-specific marginal read modes. The marginal read modes are controlled by the FCTL4.MRG0 and FCTL4.MRG1 register bits if available (device specific). During marginal read mode, marginally programmed flash memory bit locations can be detected. One method for identifying such memory locations would be to periodically perform a checksum calculation over a section of flash memory (for example, a flash segment) and repeating this procedure with the marginal read mode enabled. If they do not match, it could indicate an insufficiently programmed flash memory location. It is possible to refresh the affected Flash memory segment by disabling marginal read mode, copying to RAM, erasing the flash segment, and writing back to it from RAM.

The program checking the flash memory contents must be executed from RAM. Executing code from flash automatically disables the marginal read mode. The marginal read modes are controlled by the MRG0 and MRG1 register bits. Setting MRG1 is used to detect insufficiently programmed flash cells containing a "1" (erased bits). Setting MRG0 is used to detect insufficiently programmed flash cells containing a "0" (programmed bits). Only one of these bits should be set at a time. Therefore, a full marginal read check requires two passes of checking the flash memory content's integrity. During marginal read mode, the flash access speed (MCLK) must be limited to 1 MHz (see the device-specific data sheet).

### 7.3.6 Configuring and Accessing the Flash Memory Controller

The FCTLx registers are 16-bit password-protected read and write registers. Any read or write access must use word instructions, and write accesses must include the write password 0A5h in the upper byte. Any write to any FCTLx register with a value other than 0A5h in the upper byte is a password violation, sets the KEYV flag, and triggers a PUC system reset. Any read of any FCTLx registers reads 096h in the upper byte.

Any write to FCTL1 during an erase or byte, word, double-word write operation is an access violation and sets ACCVIFG. Writing to FCTL1 is allowed in block write mode when WAIT = 1, but writing to FCTL1 in block write mode when WAIT = 0 is an access violation and sets ACCVIFG.

Any write to FCTL2 (this register is currently not implemented) when BUSY = 1 is an access violation.

Any FCTLx register may be read when BUSY = 1. A read does not cause an access violation.

### 7.3.7 Flash Memory Controller Interrupts

The flash controller has two interrupt sources, KEYV and ACCVIFG. ACCVIFG is set when an access violation occurs. When the ACCVIE bit is reenabled after a flash write or erase, a set ACCVIFG flag generates an interrupt request. The ACCVIE bit resides in the Special Function Register, SFRIE1 (see the [SYS chapter](#) for details). ACCVIFG sources the NMI interrupt vector, so it is not necessary for GIE to be set for ACCVIFG to request an interrupt. ACCVIFG may also be checked by software to determine if an access violation occurred. ACCVIFG must be reset by software.

The password violation flag, KEYV, is set when any of the flash control registers are written with an incorrect password. When this occurs, a PUC is generated immediately, resetting the device.

### 7.3.8 Programming Flash Memory Devices

There are three options for programming a flash device. All options support in-system programming.

- Program via JTAG
- Program via the BSL
- Program via a custom solution

#### 7.3.8.1 Programming Flash Memory Via JTAG

Devices can be programmed via the JTAG port. The JTAG interface requires four signals (five signals on 20- and 28-pin devices), ground, and optionally VCC and RST/NMI.

The JTAG port is protected with a fuse. Blowing the fuse completely disables the JTAG port and is not reversible. Further access to the device via JTAG is not possible. For more details, see the *MSP430 Programming Via the JTAG Interface User's Guide* ([SLAU320](#)).

#### 7.3.8.2 Programming Flash Memory Via Bootstrap Loader (BSL)

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

#### 7.3.8.3 Programming Flash Memory Via Custom Solution

The ability of the MSP430 CPU to write to its own flash memory allows for in-system and external custom programming solutions as shown in [Figure 7-13](#). The user can choose to provide data through any means available (for example, UART or SPI). User-developed software can receive the data and program the flash memory. Because this type of solution is developed by the user, it can be completely customized to fit the application needs for programming, erasing, or updating the flash memory.

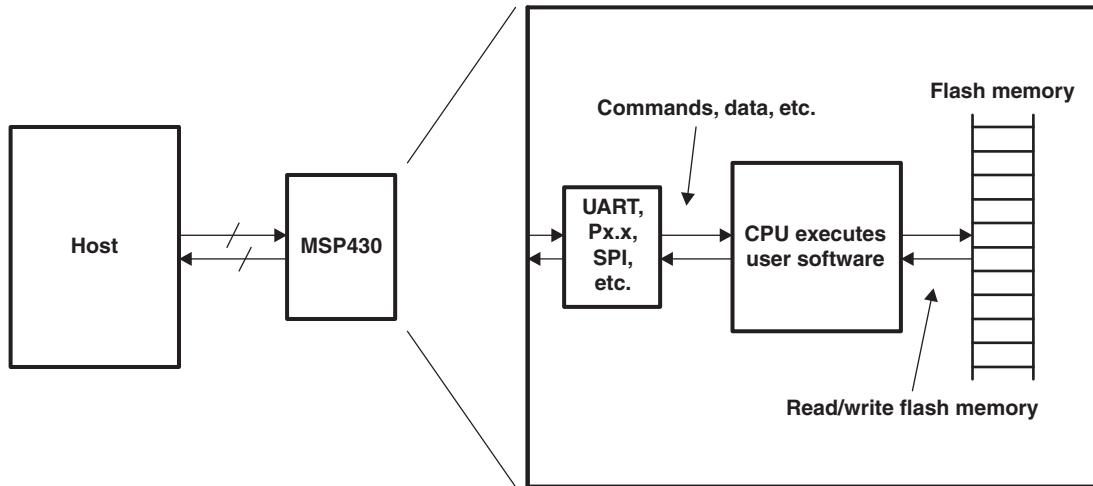


Figure 7-13. User-Developed Programming Solution

## 7.4 FCTL Registers

The flash memory controller (FCTL) registers are listed in [Table 7-5](#). The base address can be found in the device-specific data sheet. The address offset is given in [Table 7-5](#).

---

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

---

**Table 7-5. FCTL Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	FCTL1	Flash Memory Control 1	Read/write	Word	9600h	<a href="#">Section 7.4.1</a>
00h	FCTL1_L		Read/Write	Byte	00h	
01h	FCTL1_H		Read/Write	Byte	96h	
04h	FCTL3	Flash Memory Control 3	Read/write	Word	9658h	<a href="#">Section 7.4.2</a>
04h	FCTL3_L		Read/WWrite	Byte	58h	
05h	FCTL3_H		Read/Write	Byte	96h	
06h	FCTL4	Flash Memory Control 4	Read/write	Word	9600h	<a href="#">Section 7.4.3</a>
06h	FCTL4_L		Read/Write	Byte	00h	
07h	FCTL4_H		Read/WWrite	Byte	96h	

### **7.4.1 FCTL1 Register**

Flash Memory Control 1 Register

**Figure 7-14. FCTL1 Register**

15	14	13	12	11	10	9	8
FRPW/FWPW							
7	6	5	4	3	2	1	0
BLKWRT	WRT	SWRT	Reserved	MERAS	ERASE	Reserved	

rw-0            rw-0            rw-0            r-0            r-0            rw-0            rw-0            r-0

**Table 7-6. FCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-8	FRPW/FWPW	RW	96h	FCTL password. Always read as 096h. Must be written as 0A5h or a PUC is generated.
7	BLKWRT	RW	0h	Block write. BLKWRT and WRT are used together to select the write mode. The values shown below are for BLKWRT-WRT. 0-0 = Reserved 0-1 = Byte or word write 1-0 = Long-word write 1-1 = Long-word block write
6	WRT	RW	0h	Write. BLKWRT and WRT are used together to select the write mode. The values shown below are for BLKWRT-WRT. 0-0 = Reserved 0-1 = Byte or word write 1-0 = Long-word write 1-1 = Long-word block write
5	SWRT	RW	0h	Smart write. If this bit is set, the program time is shortened. The programming quality has to be checked by marginal read modes.
4-3	Reserved	R	0h	Reserved. Always reads as 0.
2	MERAS			Mass erase. MERAS and ERASE are used together to select the erase mode. MERAS and ERASE are automatically reset when EMEX is set or a flash erase operation has completed. The values shown below are for MERAS-ERASE. 0-0 = No erase 0-1 = Segment erase 1-0 = Bank erase (erase of one bank) 1-1 = Mass erase (erase all flash memory banks)
1	ERASE			Erase. MERAS and ERASE are used together to select the erase mode. MERAS and ERASE are automatically reset when EMEX is set or a flash erase operation has completed. The values shown below are for MERAS-ERASE. 0-0 = No erase 0-1 = Segment erase 1-0 = Bank erase (erase of one bank) 1-1 = Mass erase (erase all flash memory banks)
0	Reserved	R	0h	Reserved. Always reads as 0.

## 7.4.2 FCTL3 Register

Flash Memory Control 3 Register

**Figure 7-15. FCTL3 Register**

15	14	13	12	11	10	9	8
FRPW/FWPW							
7	6	5	4	3	2	1	0
Reserved	LOCKA	EMEX	LOCK	WAIT	ACCVIFG	KEYV	BUSY

Bit Descriptions:

r-0	rw-1	rw-0	rw-1	r-1	rw-0	rw-(0)	rw-0
-----	------	------	------	-----	------	--------	------

**Table 7-7. FCTL3 Register Description**

Bit	Field	Type	Reset	Description
15-8	FRPW/FWPW	RW	96h	FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC is generated.
7	Reserved	R	0h	Reserved. Always reads as 0.
6	LOCKA	RW	1h	Segment A lock. Write a 1 to this bit to change its state. Writing 0 has no effect. 0b = Segment A of the information memory is unlocked and can be written or erased in segment erase mode. 1b = Segment A of the information memory is locked and can not be written or erased in segment erase mode.
5	EMEX	RW	0h	Emergency exit. Setting this bit stops any erase or write operation. The LOCK bit is set. 0b = No emergency exit 1b = Emergency exit
4	LOCK	RW	1h	Lock. This bit unlocks the flash memory for writing or erasing. The LOCK bit can be set any time during a byte or word write or erase operation, and the operation completes normally. In the block write mode, if the LOCK bit is set while BLKWRT = WAIT = 1, BLKWRT and WAIT are reset and the mode ends normally. 0b = Unlocked 1b = Locked
3	WAIT	R	1h	Wait. Indicates the flash memory is being written to. 0b = Flash memory is not ready for the next byte or word write. 1b = Flash memory is ready for the next byte or word write.
2	ACCVIFG	RW	0h	Access violation interrupt flag 0b = No interrupt pending 1b = Interrupt pending
1	KEYV	RW	0h	Flash password violation. This bit indicates an incorrect FCTLx password was written to any flash control register and generates a PUC when set. KEYV must be reset with software. 0b = FCTLx password was written correctly. 1b = FCTLx password was written incorrectly.
0	BUSY	RW	0h	Busy. This bit indicates if the flash is currently busy erasing or programming. 0b = Not busy 1b = Busy

### 7.4.3 FCTL4 Register

Flash Memory Control 4 Register

**Figure 7-16. FCTL4 Register**

15	14	13	12	11	10	9	8
FRPW/FWPW							
7	6	5	4	3	2	1	0
LOCKINFO	Reserved	MRG1	MRG0	Reserved	Reserved	Reserved	VPE

rw-0            r-0            rw-0            rw-0            r-0            r-0            r-0            rw-0

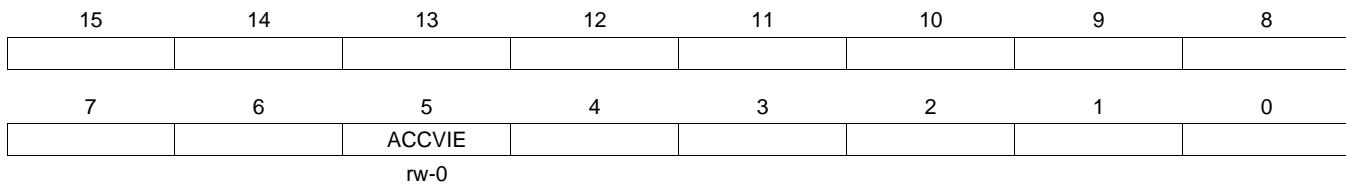
**Table 7-8. FCTL4 Register Description**

Bit	Field	Type	Reset	Description
15-8	FRPW/FWPW	RW	96h	FCTLx password. Always reads as 096h. Must be written as 0A5h or a PUC is generated.
7	LOCKINFO	RW	0h	Lock information memory. If set, the information memory cannot be erased in segment erase mode and cannot be written to.
6	Reserved	R	0h	Reserved. Always reads as 0.
5	MRG1	RW	0h	Marginal read 1 mode. This bit enables the marginal 1 read mode. The marginal read 1 bit is valid for reads from the flash memory only. During a fetch cycle, the marginal mode is turned off automatically. If both MRG1 and MRG0 are set, MRG1 is active and MRG0 is ignored. 0b = Marginal 1 read mode is disabled. 1b = Marginal 1 read mode is enabled.
4	MRG0	RW	0h	Marginal read 0 mode. This bit enables the marginal 0 read mode. The marginal read 1 bit is valid for reads from the flash memory only. During a fetch cycle, the marginal mode is turned off automatically. If both MRG1 and MRG0 are set, MRG1 is active and MRG0 is ignored. 0b = Marginal 0 read mode is disabled. 1b = Marginal 0 read mode is enabled.
3-1	Reserved	R	0h	Reserved. Always reads as 0.
0	VPE	RW	0h	Voltage changed during program error. This bit is set by hardware and can only be cleared by software. If DVCC changed significantly during programming, this bit is set to indicate an invalid result. The ACCVIFG bit is set if VPE is set.

#### 7.4.4 SFRIE1 Register

Interrupt Enable 1 Register

**Figure 7-17. SFRIE1 Register**



**Table 7-9. SFRIE1 Register Description**

Bit	Field	Type	Reset	Description
15-6				These bits may be used by other modules (see the device-specific data sheet and the <a href="#">SYS chapter</a> for details).
5	ACCVIE	RW	0h	<p>Flash memory access violation interrupt enable. This bit enables the ACCVIFG interrupt. Because other bits in SFRIE1 may be used for other modules, it is recommended to set or clear this bit using BIS or BIC instructions, rather than MOV or CLR instructions. See the <a href="#">SYS chapter</a> for more details.</p> <p>0b = Interrupt not enabled 1b = Interrupt enabled</p>
4-0				These bits may be used by other modules (see the device-specific data sheet and the <a href="#">SYS chapter</a> for details).

## ***Memory Integrity Detection (MID)***

---



---



---

Memory Integrity Detection (MID) is a program and data protection mechanism available on several device families (for example, MSP430F6659). It provides a high level of operation safety for fault critical application areas. This chapter explains how the firmware may be used to get the levels of operation safety and overall fault response that suits different applications.

Topic	Page
<b>8.1 MID Overview .....</b>	<b>366</b>
<b>8.2 Flash Memory With MID Support .....</b>	<b>367</b>
<b>8.3 MID Parity Check Logic .....</b>	<b>367</b>
<b>8.4 Detecting Unprogrammed Memory Accesses .....</b>	<b>368</b>
<b>8.5 MID ROM .....</b>	<b>368</b>
<b>8.6 MID Support Software Function .....</b>	<b>368</b>
<b>8.7 User's UNMI Interrupt Handler .....</b>	<b>372</b>

## 8.1 MID Overview

The MID is an add-on to the MSP430 flash memory controller. MID provides additional functionality over the regular flash operation methods as described in the [Flash Memory Controller](#) chapter.

The main purpose of the MID function is to help gain higher reliability of flash content and overall system integrity in harsh environments and application areas requiring such features. The additional level of security is reached by calculating parity information.

The complete MID solution consists of the blocks *Parity Generator and Parity Check* and *MID ROM*. The *Parity Generator and Parity Check* provides all necessary logic elements needed to identify bit errors in the whole memory array. The on-chip MID ROM contains the MID Support Software and it performs all the necessary tasks to operate MID. Those built-in MID functions provide all needed functionality that allow using MID features.

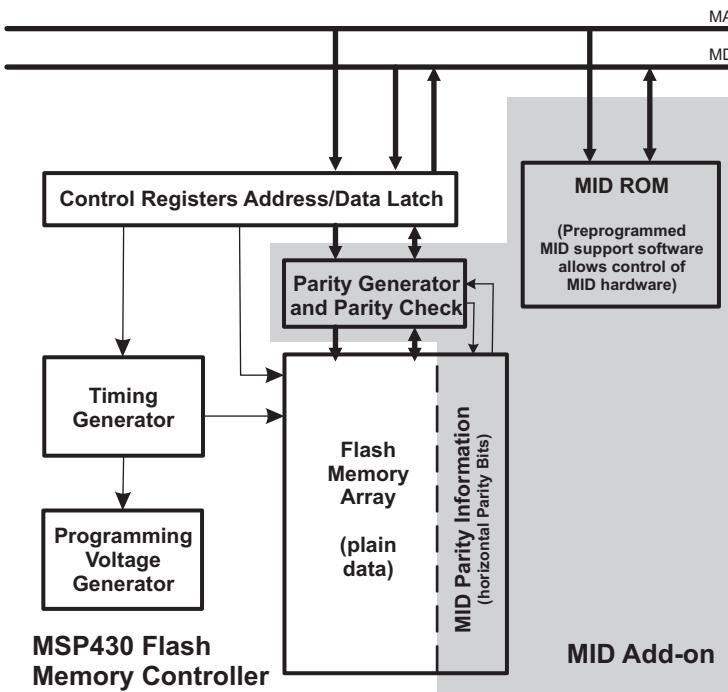
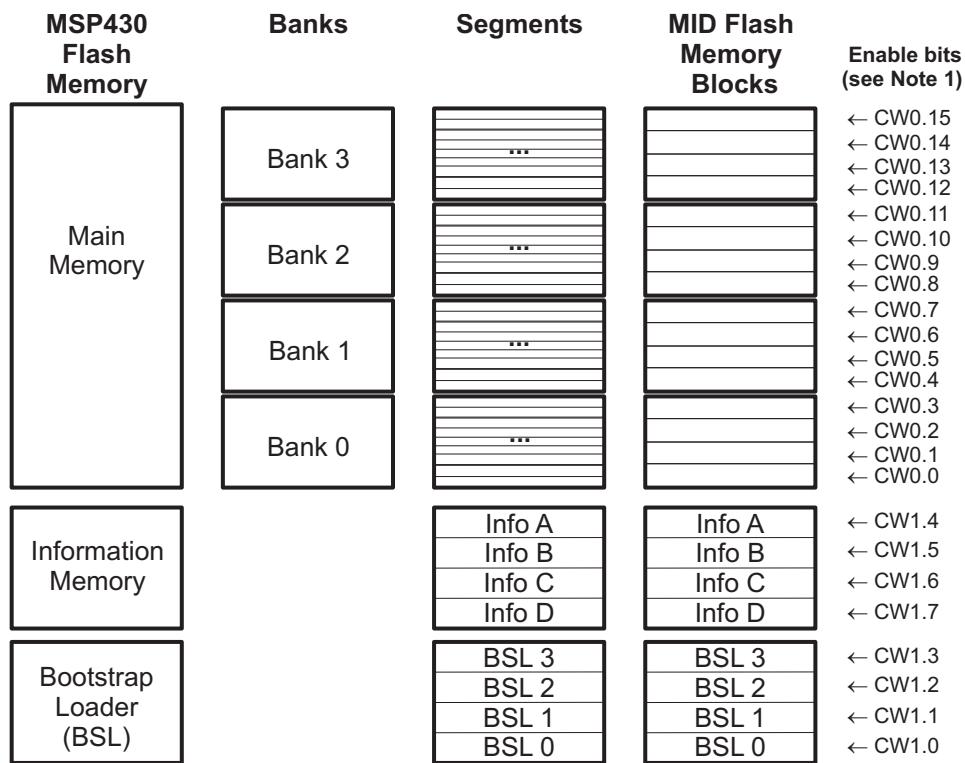


Figure 8-1. Block Diagram of MID Implementation

## 8.2 Flash Memory With MID Support

The MSP430 flash memory is partitioned into different memory areas—main and information memory, banks, segments, and memory blocks—where MID protection can be enabled. [Figure 8-2](#) shows an example for a typical MSP430 flash segmentation including the MID feature.



- (1) The cw0.x and cw1.x bits are used to enable MID functionality for the individual memory ranges. Further information can be found in [Section 8.6.1](#).

**Figure 8-2. Overview of MSP430 Flash Memory Segmentation**

The whole flash memory array consist of MID supported flash memory blocks. For a device with 512kB flash main memory, each such MID flash memory block has a size of 32kB (main memory divided by 16). Each row consists of one word of plain data (16-bits) and a horizontal parity bit (H-parity bit).

Erased segments show all ones in the data array field and horizontal parity. Writing to flash memory (MID after reset) automatically writes the horizontal parity bits along with the data bits. Writing to the plain data field can, of course, be interrupted and continued in any order. Adding content after the horizontal parity has been written is impracticable, as the horizontal parity information changes as well. The whole segment (not just a single MID memory block) would need to be erased before it can be written again. The shown method is excellent for data content of static nature like code, tables, and so on. For data acquisition into flash, other methods (for example, majority vote) are better suitable; whereas complete blocks of acquisition data can be protected with this method again.

## 8.3 MID Parity Check Logic

Any access to MID enabled flash memory causes a verification of its horizontal parity in the background. It does not matter if code or data is read from the flash memory. If a parity error is detected, the bus error event "parity error" is triggered and calls the user NMI exception handler. The application software can then react on the failure by, for example, showing an error message on the application's display.

## 8.4 Detecting Unprogrammed Memory Accesses

All bits are set after erasing the flash memory; this also includes the horizontal parity bit. If an erased memory range is accessed, the MID causes a NMI interrupt, because of a detected parity failure. Only programmed addresses are accessible without a MID failure interrupt; that is especially the case for the content 0xFFFF. If memory content should be 0xFFFF, it must be programmed. This ensures that the horizontal parity bit is cleared (0).

Enabling the MID functionality for nonprogrammed memory ranges allows detecting memory accesses to these nonprogrammed addresses.

## 8.5 MID ROM

The MID ROM is 1KB of read-only memory. The on-chip MID ROM is factory programmed with MID support software. These software functions are used to enable or disable the MID module. The start address of the MID ROM depends on the MSP430 device; see the device-specific data sheet for specifications.

## 8.6 MID Support Software Function

The MID is disabled by default after power-up of the device. To use the MID feature, it must be enabled within the application software. Enabling is done by calling the MidInit() function with parameters that define which MID memory blocks should be enabled or disabled.

[Table 8-1](#) list all existing MID functions. These functions are stored in the MID ROM; its start address is defined in the device-specific data sheet.

**Table 8-1. Overview of MID Support Software Functions**

Function	Address Offset	Description
Revision	0x00	Content of address: 2843h
	0x02	Content of address: 80xyh, xy is the revision word
MidEnable	0x04	Initialization and enabling of MID
MidDisable	0x08	MID is disabled
MidGetErrAdr	0x0C	Returns the error location
MidCheckMem	0x10	Memory check is performed
MidSetRaw	0x14	Writing a data word and parity bit into a defined address
MidGetParity	0x18	Read out horizontal parity bit
MidCalcVParity	0x1C	Calculating vertical parity
Reserved	0x20	
Reserved	0x24	
Reserved	0x28	
Reserved	0x2C	

### **8.6.1 MidEnable() Function**

#### **FUNCTION**

```
void MidEnable(unsigned short cw0,unsigned short cw1);
```

#### **FUNCTION DESCRIPTION**

This function initializes and enables MID. The argument *cw0* and *cw1* allow an explicit control what MID flash memory blocks are to be protected. MID feature is disabled after a power-up or BOR. *MidEnable()* function is expected to be called early after application start. Calling it again later reconfigures the settings.

#### **PARAMETERS**

Name	Type	Description
<i>cw0</i>	unsigned short R12.W	Configuration word 0. It is used to activate the MID feature for certain memory ranges. The main memory is split into 16 blocks. The LSB bit of <i>cw0</i> activates the lowest address range (see <a href="#">Figure 8-3</a> ).
<i>cw1</i>	unsigned short R13.W	Configuration word 1. This bit is used in the same way like <i>cw0</i> , the only difference is that instead of main memory the BSL, and Info memory is used (see <a href="#">Figure 8-4</a> ).

**Figure 8-3. *cw0* Parameter**

15	14	13	12	11	10	9	8
<i>cw0.15</i>	<i>cw0.14</i>	<i>cw0.13</i>	<i>cw0.12</i>	<i>cw0.11</i>	<i>cw0.10</i>	<i>cw0.9</i>	<i>cw0.8</i>
7	6	5	4	3	2	1	0
<i>cw0.7</i>	<i>cw0.6</i>	<i>cw0.5</i>	<i>cw0.4</i>	<i>cw0.3</i>	<i>cw0.2</i>	<i>cw0.1</i>	<i>cw0.0</i>

Bit	Field	Description
15-0	<i>cw0.x</i>	Main memory is split into MID flash memory blocks; each MID flash memory block has a size of main memory divided by 16 (for example, for a 512kB main memory, the MID memory block size is 32 KB). The <i>cw0.x</i> bits allow to enable MID support for the different flash memory blocks. For example, <i>cw0.0</i> is activating the lowest flash memory block and <i>cw0.15</i> activates the upper flash memory block. 0 = MID support is deactivated 1 = MID support is active

**Figure 8-4. *cw1* Parameter**

15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
<i>cw1.7</i>	<i>cw1.6</i>	<i>cw1.5</i>	<i>cw1.4</i>	<i>cw1.3</i>	<i>cw1.2</i>	<i>cw1.1</i>	<i>cw1.0</i>

Bit	Field	Description
15-8	Reserved	These bits are reserved. It is strongly recommended to reset (0) these bits
7	<i>cw1.7</i>	Enables or disables MID for the flash information memory segment D. 0 = MID support is deactivated 1 = MID support is active
6	<i>cw1.6</i>	Enables or disables MID for the flash information memory segment C. 0 = MID support is deactivated 1 = MID support is active
5	<i>cw1.5</i>	Enables or disables MID for the flash information memory segment B. 0 = MID support is deactivated 1 = MID support is active

Bit	Field	Description
4	cw1.4	Enables or disables MID for the flash information memory segment A. 0 = MID support is deactivated 1 = MID support is active
3	cw1.3	Enables or disables MID for the bootstrap loader memory 3. 0 = MID support is deactivated 1 = MID support is active
2	cw1.2	Enables or disables MID for the bootstrap loader memory 2. 0 = MID support is deactivated 1 = MID support is active
1	cw1.1	Enables or disables MID for the bootstrap loader memory 1. 0 = MID support is deactivated 1 = MID support is active
0	cw1.0	Enables or disables MID for the bootstrap loader memory 0. 0 = MID support is deactivated 1 = MID support is active

### 8.6.2 MidDisable() Function

#### FUNCTION

```
void MidDisable(void);
```

#### FUNCTION DESCRIPTION

This function clears the cw0 and cw1 parameters that were set during MidEnable() function call and it disables the MID hardware.

### 8.6.3 MidGetErrAdr() Function

#### FUNCTION

```
unsigned short * MidGetErrAdr(void);
```

#### FUNCTION DESCRIPTION

This function returns the error location is there was a memory integrity failure. If there is no valid failure address or error location, the function returns the value F'FFFFh.

Note that the MidGetErrAdr() function returns only the correct error address when this function is called prior to a read access of SYSBERRIV register. The code example in [Section 8.7](#) shows where the MidGetErrAdr() function call should be placed.

### **8.6.4 MidCheckMem() Function**

#### **FUNCTION**

```
void MidCheckMem(unsigned short * startAddr, unsigned short * endAddr);
```

#### **FUNCTION DESCRIPTION**

This function allows doing a memory integrity check. First, the MidEnable function should be called. This function enables MID and it defines the memory blocks that should be protected. After that the MidCheckMem function can be called. Its parameter list defines an address range that is accessed with wordwise reads. An UNMI interrupt (MID interrupt) is generated in case a parity error occurs and the read address is enabled for MID protection.

#### **PARAMETERS**

Name	Type	Description
startAddr	unsigned short *R12.W	Start address for the memory integrity check. The startAddr must be an even number.
endAddr	unsigned short R13.W	End address for the memory integrity check. The endAddr must be an even number. The address defined with endAddr is included in the memory integrity check.

### **8.6.5 MidSetRaw() Function**

#### **FUNCTION**

```
void MidSetRaw(unsigned short data, unsigned short parity, unsigned short * adr,  
               unsigned short flashKey);
```

#### **FUNCTION DESCRIPTION**

This function writes one word (data) and a separately definable parity bit (parity) to an MID memory address (adr). The Flash memory key is needed to allow access to flash control registers; this parameter is passed via the argument flashKey (see example below).

#### **PARAMETERS**

Name	Type	Description
data	unsigned short R12.W	Data to be written
parity	unsigned short R13.W	If parity = 0, the parity bit 0 is written. Otherwise, if parity >> 0, the parity bit 1 is written.
adr	unsigned short *R14.A	Destination address where raw information is written
flashKey	unsigned short R15.W	Flash memory key. This is needed to allow the MidSetRaw function to get access to the flash control registers. The passing parameter is usually defined in the standard MSP430 header files; therefore, "FWKEY" can be used here.

#### **EXAMPLE**

```
#include <msp430f6659.h>  
const unsigned short FlashAdr=0xFF00; // Flash memory address will be reprogrammed  
void main(void)  
{ static unsigned short Data; // variable for data the will be read  
  static unsigned short Parity; // H-parity bit that will be read  
  WDTCTL=WDTDPW+WDTHOLD; // disable Watchdog  
  Data=0x5A5A; // data that will be written into flash memory  
  Parity=0; // parity bit that will be written  
  MIDSetRaw(Data,Parity,&FlashAdr,FWKEY); // write data and parity bit  
  while(1);  
}
```

### 8.6.6 *MidGetParity()* Function

#### FUNCTION

```
unsigned short MidGetParity(unsigned short * adr);
```

#### FUNCTION DESCRIPTION

This function returns the parity bit of the appropriate address. Reading the parity bit works only when MID was enabled before calling MidGetParity() function and the appropriate MID memory block is enabled.

#### PARAMETERS

Name	Type	Description
adr	unsigned short *R12.A	Defines the address where the parity bit should be read.

### 8.6.7 *MidCalcVParity()* Function

#### FUNCTION

```
unsigned short MidCalcVParity(unsigned short * startAdr, unsigned short * endAdr);
```

#### FUNCTION DESCRIPTION

This function allows to calculate a vertical parity for a defined memory range.

#### PARAMETERS

Name	Type	Description
startAdr	unsigned short *R12.A	Defines the start address for calculating vertical parity. The startAdr must be an even number.
endAdr	unsigned short *R13.A	End address for calculating vertical parity. The endAdr must be an even number. The address defined with endAdr is included in the vertical parity calculation.

## 8.7 User's UNMI Interrupt Handler

If an error is detected, the on-chip MID generates an UNMI interrupt. The application software must manage error handling.

UNMI handler framework for MID error handling:

```
_interrupt void unmi_isr(void)
{
    switch(__even_in_range(SYSUNIV, 0x08))
    {
        case 0x00: break;
        case 0x02: break; // NMIIFG
        case 0x04: break; // OFIFG
        case 0x06: break; // ACCVIFG
        case 0x08: break; // BUSIFG
        // If needed, obtain the flash error location here.
        ErrorLocation = MidGetErrAddr();

        switch(__even_in_range(SYSBERRIV, 0x08))
        {
            case 0x00: break; // no bus error
            case 0x02: break; // USB bus error
            case 0x04: break; // reserved
            case 0x06: break; // MID error
            <place your MID error handler code here>
            break;
            case 0x08: break;
            default: break;
        }
        break;
    default: break;
}
```

## **RAM Controller (RAMCTL)**

The RAM controller (RAMCTL) allows control of the operation of the RAM.

<b>Topic</b>	<b>Page</b>
9.1 <b>RAM Controller (RAMCTL) Introduction .....</b>	<b>374</b>
9.2 <b>RAMCTL Operation .....</b>	<b>374</b>
9.3 <b>RAMCTL Registers .....</b>	<b>375</b>

## 9.1 RAM Controller (RAMCTL) Introduction

The RAMCTL provides access to the different power modes of the RAM. The RAMCTL allows the ability to reduce the leakage current while the CPU is off. The RAM can also be switched off. In retention mode, the RAM content is saved while the RAM content is lost in off mode. The RAM is partitioned in sectors, typically of 4KB (sector) size. See the device-specific data sheet for actual block allocation and size. Each sector is controlled by the RAM controller RAM Sector Off control bit (RCRSyOFF) of the RAMCTL Control 0 register (RCCTL0). The RCCTL0 register is protected with a key. Only if the correct key is written during a word write, the RCCTL0 register content can be modified. Byte write accesses or write accesses with a wrong key are ignored.

## 9.2 RAMCTL Operation

### Active mode

In active mode, the RAM can be read and written at any time. If a RAM address of a sector must hold data, the whole sector cannot be switched off.

### Low-power modes

In all low-power modes, the CPU is switched off. As soon as the CPU is switched off, the RAM enters retention mode to reduce the leakage current.

### RAM off mode

Each sector can be turned off independently of each other by setting the respective RCRSyOFF bit to 1. Reading from a switched off RAM sector returns 0 as data. All data previously stored into a switched off RAM sector is lost and cannot be read, even if the sector is turned on again.

### Stack pointer

The program stack is located in RAM. Sectors holding the stack must not be turned off if an interrupt has to be executed, or a low-power mode is entered.

### USB buffer memory

On devices with USB, the USB buffer memory is located in RAM. Sector 7 is used for this purpose. RCRS7OFF can be set to switch off this memory if it is not required for USB operation or is not being utilized in normal operation.

### 9.3 RAMCTL Registers

The RAMCTL module register is listed in [Table 9-1](#). The base address can be found in the device-specific data sheet. The address offset is given in [Table 9-1](#).

---

**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 9-1. RAMCTL Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	RCCTL0	RAM Controller Control 0	Read/write	Word	6900h	<a href="#">Section 9.3.1</a>
00h		RCCTL0_L	Read/write	Byte	00h	
01h		RCCTL0_H	Read/write	Byte	69h	

**9.3.1 RCCTL0 Register**

RAM Controller Control 0 Register

**Figure 9-1. RCCTL0 Register**

15	14	13	12	11	10	9	8
RCKEY							
rw-0	rw-1	rw-1	rw-0	rw-1	rw-0	rw-0	rw-1
7	6	5	4	3	2	1	0
RCRS7OFF	RCRS6OFF	RCRS5OFF	RCRS4OFF	RCRS3OFF	RCRS2OFF	RCRS1OFF	RCRS0OFF
rw-0							

**Table 9-2. RCCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-8	RCKEY	RW	69h	RAM controller key. Always read as 69h. Must be written as 5Ah, otherwise the RAMCTL write is ignored.
7	RCRS7OFF	RW	0h	RAM controller RAM sector 7 off. Setting the bit to 1 turns off the RAM sector 7. All data of the RAM sector 7 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.
6	RCRS6OFF	RW	0h	RAM controller RAM sector 6 off. Setting the bit to 1 turns off the RAM sector 6. All data of the RAM sector 6 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.
5	RCRS5OFF	RW	0h	RAM controller RAM sector 5 off. Setting the bit to 1 turns off the RAM sector 5. All data of the RAM sector 5 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.
4	RCRS4OFF	RW	0h	RAM controller RAM sector 4 off. Setting the bit to 1 turns off the RAM sector 4. All data of the RAM sector 4 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.
3	RCRS3OFF	RW	0h	RAM controller RAM sector 3 off. Setting the bit to 1 turns off the RAM sector 3. All data of the RAM sector 3 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.
2	RCRS2OFF	RW	0h	RAM controller RAM sector 2 off. Setting the bit to 1 turns off the RAM sector 2. All data of the RAM sector 2 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.
1	RCRS1OFF	RW	0h	RAM controller RAM sector 1 off. Setting the bit to 1 turns off the RAM sector 1. All data of the RAM sector 1 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.
0	RCRS0OFF	RW	0h	RAM controller RAM sector 0 off. Setting the bit to 1 turns off the RAM sector 0. All data of the RAM sector 0 is lost. See the device-specific data sheet to find the number of RAM sectors available along with their respective address ranges and sizes.

## ***Backup RAM***

The backup RAM is a (volatile) memory that is retained during LPMx.5 and operation from a backup supply (if supported by the device). This chapter describes the backup RAM.

Topic	Page
<b>10.1 Backup RAM Introduction and Operation .....</b>	<b>378</b>
<b>10.2 Battery Backup Registers .....</b>	<b>378</b>

## 10.1 Backup RAM Introduction and Operation

The backup RAM provides a limited number of bytes of RAM that are retained during LPMx.5 and during operation from a backup supply (if the device integrates the complete battery backup system). At least one byte or one word (one word is preferred) should be used to generate a checksum of the stored data or to store a signature to ensure that the data is still reliable when returning from LPMx.5 or from backup operation.

## 10.2 Battery Backup Registers

The backup RAM registers are listed in [Table 10-1](#). The base address for the backup RAM registers can be found in the device-specific data sheet. The address offsets are 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. Backup RAM Registers**

Offset	Acronym	Register Name	Type	Access	Reset	LPMx.5 Backup Operation
00h	BAKMEM0	Battery Backup Memory 0	Read/write	Word	Undefined	retained
00h	BAKMEM0_L		Read/write	Byte	Undefined	retained
01h	BAKMEM0_H		Read/write	Byte	Undefined	retained
02h	BAKMEM1	Battery Backup Memory 1	Read/write	Word	Undefined	retained
02h	BAKMEM1_L		Read/write	Byte	Undefined	retained
03h	BAKMEM1_H		Read/write	Byte	Undefined	retained
04h	BAKMEM2	Battery Backup Memory 2	Read/write	Word	Undefined	retained
04h	BAKMEM2_L		Read/write	Byte	Undefined	retained
05h	BAKMEM2_H		Read/write	Byte	Undefined	retained
06h	BAKMEM3	Battery Backup Memory 3	Read/write	Word	Undefined	retained
06h	BAKMEM3_L		Read/write	Byte	Undefined	retained
07h	BAKMEM3_H		Read/write	Byte	Undefined	retained

## **Direct Memory Access (DMA) Controller Module**

---

---

The direct memory access (DMA) controller module transfers data from one address to another without CPU intervention. This chapter describes the operation of the DMA controller.

Topic	Page
11.1 Direct Memory Access (DMA) Introduction .....	380
11.2 DMA Operation .....	382
11.3 DMA Registers .....	393

## 11.1 Direct Memory Access (DMA) Introduction

The DMA controller transfers data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can move data from the ADC conversion memory to RAM.

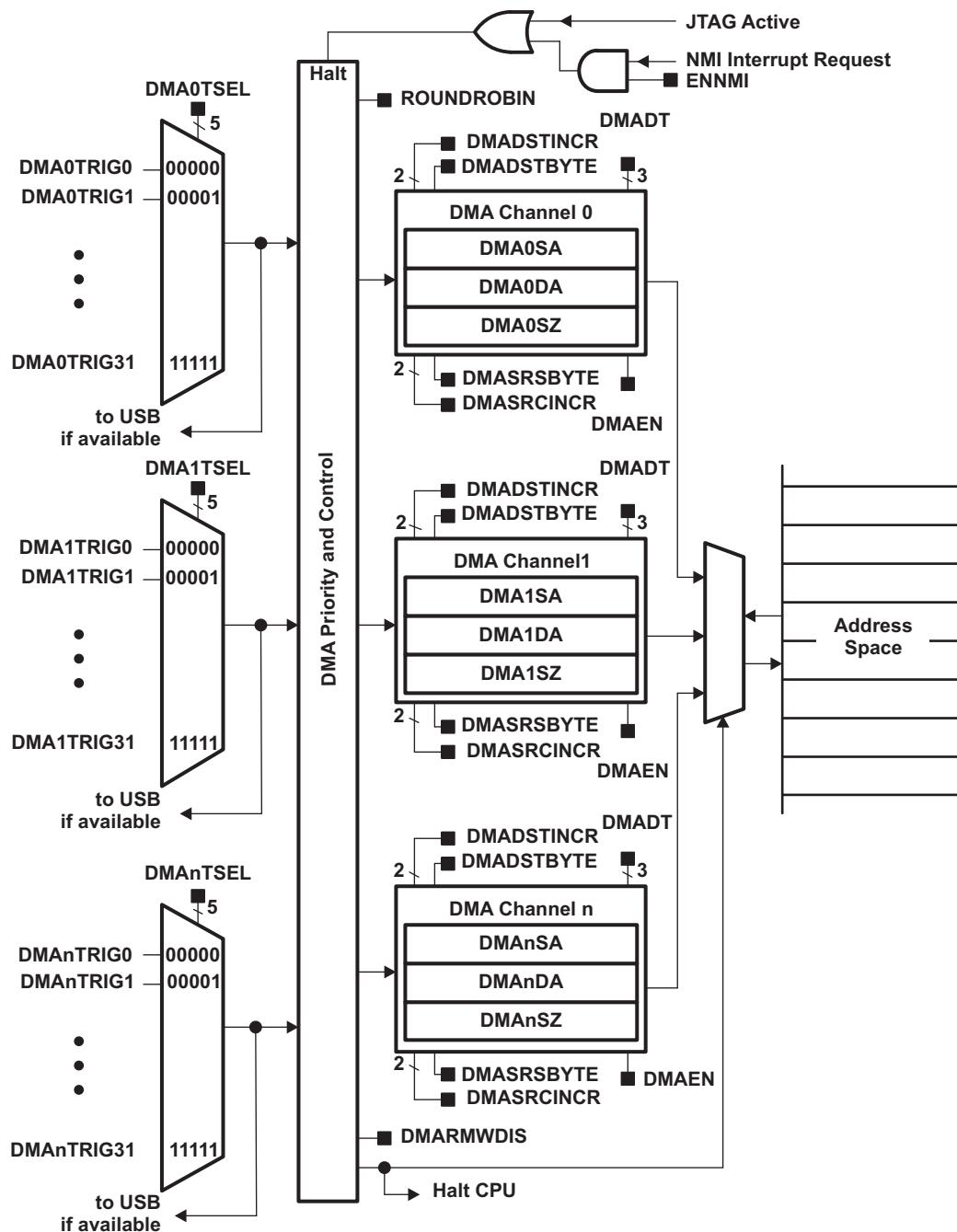
Devices that contain a DMA controller may have up to eight DMA channels available. Therefore, depending on the number of DMA channels available, some features described in this chapter are not applicable to all devices. See the device-specific data sheet for number of channels supported.

Using the DMA controller can increase the throughput of peripheral modules. It can also reduce system power consumption by allowing the CPU to remain in a low-power mode, without having to awaken to move data to or from a peripheral.

DMA controller features include:

- Up to eight independent transfer channels
- Configurable DMA channel priorities
- Requires only two MCLK clock cycles per transfer
- Byte or word and mixed byte/word transfer capability
- Block sizes up to 65535 bytes or words
- Configurable transfer trigger selections
- Selectable-edge or level-triggered transfer
- Four addressing modes
- Single, block, or burst-block transfer modes

The DMA controller block diagram is shown in [Figure 11-1](#).


**Figure 11-1. DMA Controller Block Diagram**

## 11.2 DMA Operation

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

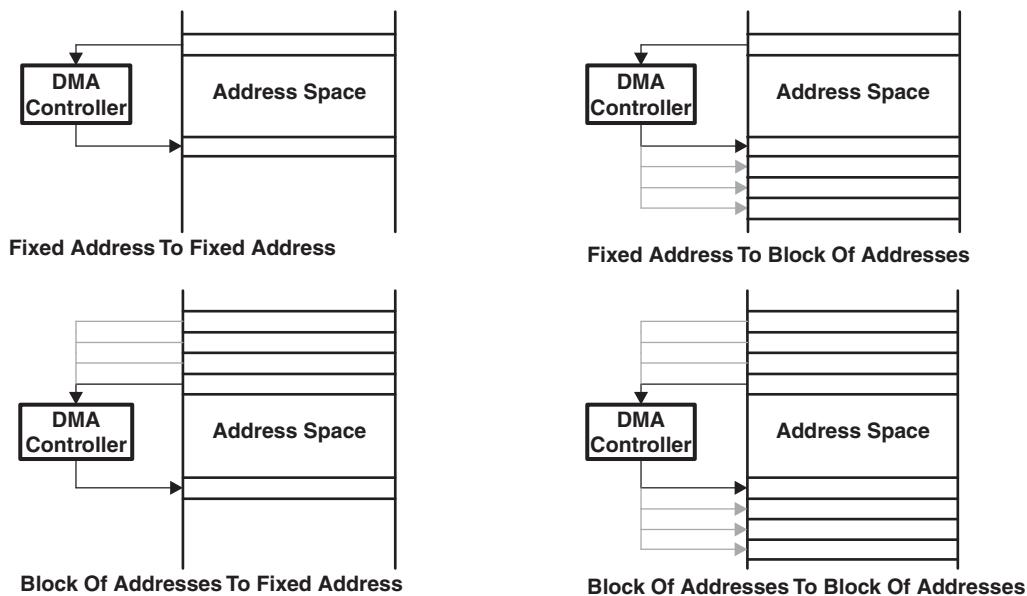
### 11.2.1 DMA Addressing Modes

The DMA controller has four addressing modes. The addressing mode for each DMA channel is independently configurable. For example, channel 0 may transfer between two fixed addresses, while channel 1 transfers between two blocks of addresses. The addressing modes are shown in [Figure 11-2](#). The addressing modes are:

- Fixed address to fixed address
- Fixed address to block of addresses
- Block of addresses to fixed address
- Block of addresses to block of addresses

The addressing modes are configured with the DMASRCINCR and DMADSTINCR control bits. The DMASRCINCR bits select if the source address is incremented, decremented, or unchanged after each transfer. The DMADSTINCR bits select if the destination address is incremented, decremented, or unchanged after each transfer.

Transfers may be byte to byte, word to word, byte to word, or word to byte. When transferring word to byte, only the lower byte of the source-word transfers. When transferring byte to word, the upper byte of the destination-word is cleared when the transfer occurs.



**Figure 11-2. DMA Addressing Modes**

### 11.2.2 DMA Transfer Modes

The DMA controller has six transfer modes selected by the DMADT bits as listed in [Table 11-1](#). Each channel is individually configurable for its transfer mode. For example, channel 0 may be configured in single transfer mode, while channel 1 is configured for burst-block transfer mode, and channel 2 operates in repeated block mode. The transfer mode is configured independently from the addressing mode. Any addressing mode can be used with any transfer mode.

Two types of data can be transferred selectable by the DMAxCTL DSTBYTE and SRCBYTE fields. The source and/or destination location can be either byte or word data. It is also possible to transfer byte to byte, word to word, or any combination.

**Table 11-1. DMA Transfer Modes**

<b>DMADT</b>	<b>Transfer Mode</b>	<b>Description</b>
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN remains enabled.
101	Repeated block transfer	A complete block is transferred with one trigger. DMAEN remains enabled.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.

### 11.2.2.1 Single Transfer

In single transfer mode, each byte/word transfer requires a separate trigger. The single transfer state diagram is shown in [Figure 11-3](#).

The DMAxSZ register is used to define the number of transfers to be made. The DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer. The DMAxSZ register is decremented after each transfer. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set. When DMADT = {0}, the DMAEN bit is cleared automatically when DMAxSZ decrements to zero and must be set again for another transfer to occur.

In repeated single transfer mode, the DMA controller remains enabled with DMAEN = 1, and a transfer occurs every time a trigger occurs.

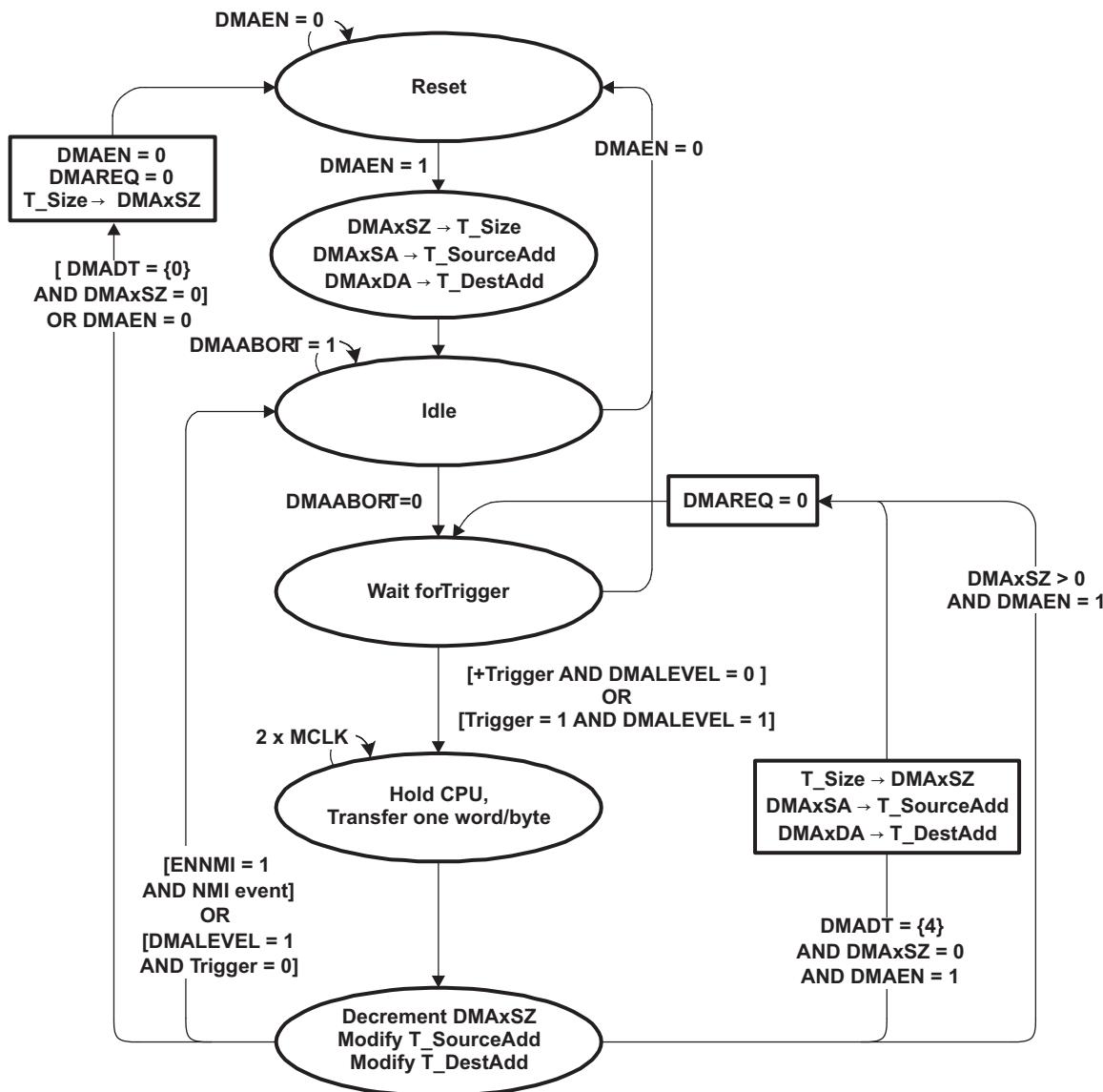


Figure 11-3. DMA Single Transfer State Diagram

### 11.2.2.2 Block Transfer

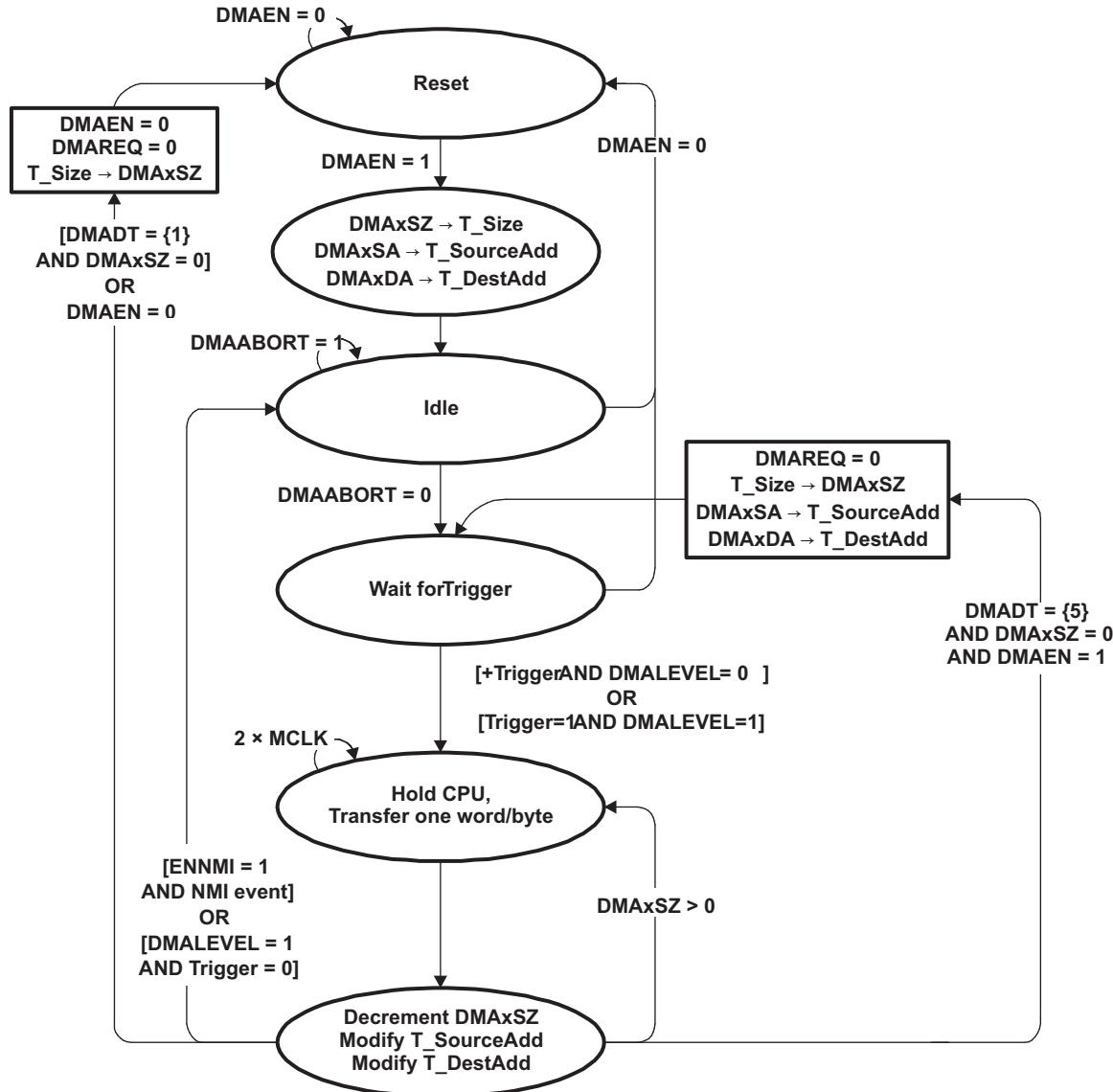
In block transfer mode, a transfer of a complete block of data occurs after one trigger. When  $\text{DMADT} = \{1\}$ , the DMAEN bit is cleared after the completion of the block transfer and must be set again before another block transfer can be triggered. After a block transfer has been triggered, further trigger signals occurring during the block transfer are ignored. The block transfer state diagram is shown in [Figure 11-4](#).

The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If  $\text{DMAxSZ} = 0$ , no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set.

During a block transfer, the CPU is halted until the complete block has been transferred. The block transfer takes  $2 \times \text{MCLK} \times \text{DMAxSZ}$  clock cycles to complete. CPU execution resumes with its previous state after the block transfer is complete.

In repeated block transfer mode, the DMAEN bit remains set after completion of the block transfer. The next trigger after the completion of a repeated block transfer triggers another block transfer.



**Figure 11-4. DMA Block Transfer State Diagram**

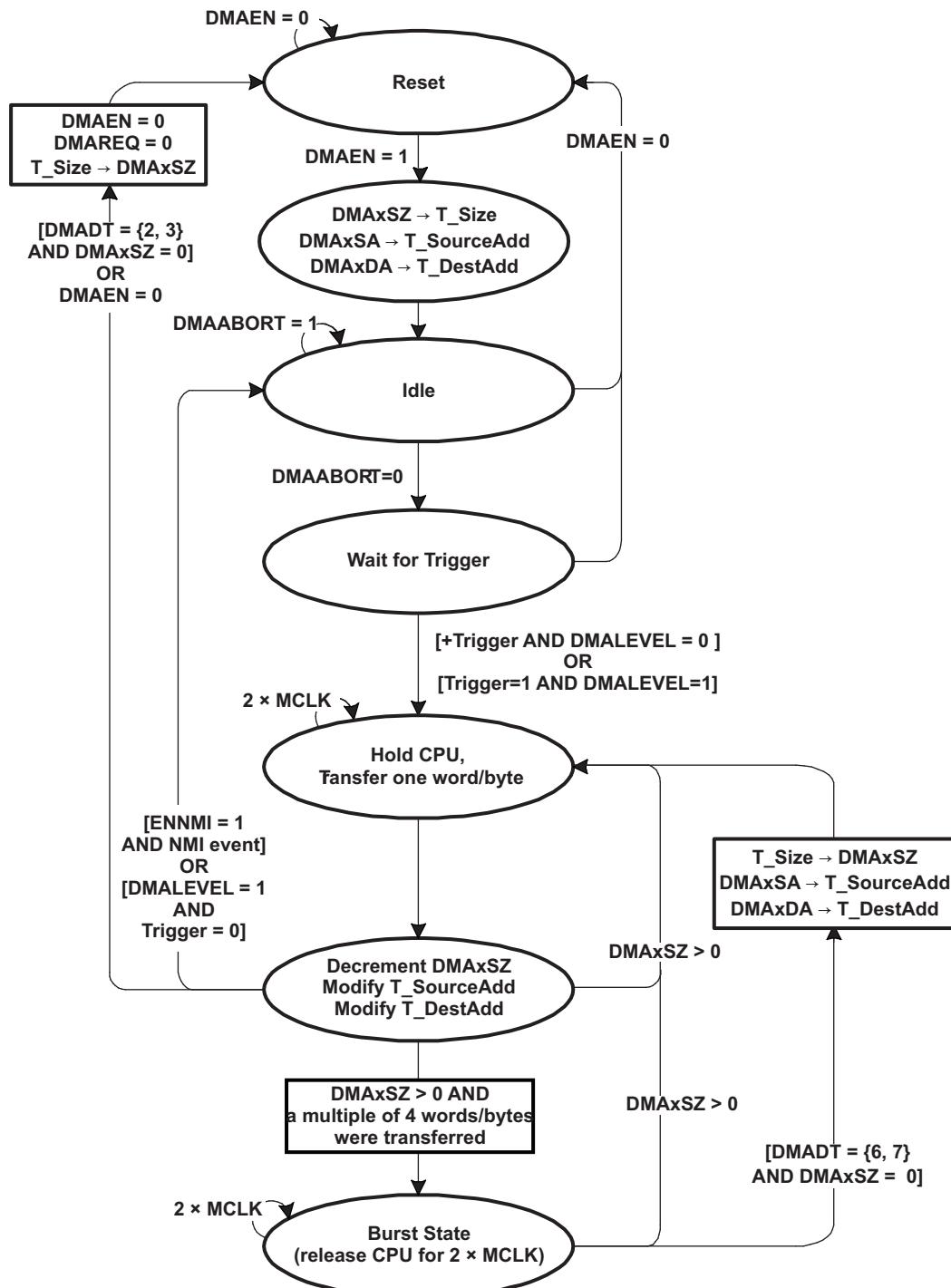
#### 11.2.2.3 Burst-Block Transfer

In burst-block mode, transfers are block transfers with CPU activity interleaved. The CPU executes two MCLK cycles after every four byte/word transfers of the block, resulting in 20% CPU execution capacity. After the burst-block, CPU execution resumes at 100% capacity and the DMAEN bit is cleared. DMAEN must be set again before another burst-block transfer can be triggered. After a burst-block transfer has been triggered, further trigger signals occurring during the burst-block transfer are ignored. The burst-block transfer state diagram is shown in [Figure 11-5](#).

The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set.

In repeated burst-block mode, the DMAEN bit remains set after completion of the burst-block transfer and no further trigger signals are required to initiate another burst-block transfer. Another burst-block transfer begins immediately after completion of a burst-block transfer. In this case, the transfers must be stopped by clearing the DMAEN bit, or by an (non)maskable interrupt (NMI) when ENNMI is set. In repeated burst-block mode the CPU executes at 20% capacity continuously until the repeated burst-block transfer is stopped.


**Figure 11-5. DMA Burst-Block Transfer State Diagram**

### 11.2.3 Initiating DMA Transfers

Each DMA channel is independently configured for its trigger source with the DMAxTSEL. The DMAxTSEL bits should be modified only when the DMACTLx DMAEN bit is 0. Otherwise, unpredictable DMA triggers may occur. [Table 11-2](#) describes the trigger operation for each type of module. See the device-specific data sheet for the list of triggers available, along with their respective DMAxTSEL values.

When selecting the trigger, the trigger must not have already occurred, or the transfer does not take place.

---

**NOTE: DMA trigger selection and USB**

On devices that contain a USB module, the triggers selection from DMA channels 0, 1, or 2 can be used for the USB time stamp event selection (see the USB module description for further details).

---

#### 11.2.3.1 Edge-Sensitive Triggers

When DMALEVEL = 0, edge-sensitive triggers are used, and the rising edge of the trigger signal initiates the transfer. In single-transfer mode, each transfer requires its own trigger. When using block or burst-block modes, only one trigger is required to initiate the block or burst-block transfer.

#### 11.2.3.2 Level-Sensitive Triggers

When DMALEVEL = 1, level-sensitive triggers are used. For proper operation, level-sensitive triggers can only be used when external trigger DMAE0 is selected as the trigger. DMA transfers are triggered as long as the trigger signal is high and the DMAEN bit remains set.

The trigger signal must remain high for a block or burst-block transfer to complete. If the trigger signal goes low during a block or burst-block transfer, the DMA controller is held in its current state until the trigger goes back high or until the DMA registers are modified by software. If the DMA registers are not modified by software, when the trigger signal goes high again, the transfer resumes from where it was when the trigger signal went low.

When DMALEVEL = 1, transfer modes selected when DMADT = {0, 1, 2, 3} are recommended because the DMAEN bit is automatically reset after the configured transfer.

### 11.2.4 Halting Executing Instructions for DMA Transfers

The DMARMWDIS bit controls when the CPU is halted for DMA transfers. When DMARMWDIS = 0, the CPU is halted immediately and the transfer begins when a trigger is received. In this case, it is possible that CPU read-modify-write operations can be interrupted by a DMA transfer. When DMARMWDIS = 1, the CPU finishes the currently executing read-modify-write operation before the DMA controller halts the CPU and the transfer begins (see [Table 11-2](#)).

**Table 11-2. DMA Trigger Operation**

<b>Module</b>	<b>Operation</b>
DMA	A transfer is triggered when the DMAREQ bit is set. The DMAREQ bit is automatically reset when the transfer starts. A transfer is triggered when the DMAxIFG flag is set. DMA0IFG triggers channel 1, DMA1IFG triggers channel 2, and DMA2IFG triggers channel 0. None of the DMAxIFG flags are automatically reset when the transfer starts. A transfer is triggered by the external trigger DMAEO.
Timer_A	A transfer is triggered when the TAxCRC0 CCIFG flag is set. The TAxCRC0 CCIFG flag is automatically reset when the transfer starts. If the TAxCRC0 CCIE bit is set, the TAxCRC0 CCIFG flag does not trigger a transfer. A transfer is triggered when the TAxCRC2 CCIFG flag is set. The TAxCRC2 CCIFG flag is automatically reset when the transfer starts. If the TAxCRC2 CCIE bit is set, the TAxCRC2 CCIFG flag does not trigger a transfer.
Timer_B	A transfer is triggered when the TBxCRC0 CCIFG flag is set. The TBxCRC0 CCIFG flag is automatically reset when the transfer starts. If the TBxCRC0 CCIE bit is set, the TBxCRC0 CCIFG flag does not trigger a transfer. A transfer is triggered when the TBxCRC2 CCIFG flag is set. The TBxCRC2 CCIFG flag is automatically reset when the transfer starts. If the TBxCRC2 CCIE bit is set, the TBxCRC2 CCIFG flag does not trigger a transfer.
USCI_Ax	A transfer is triggered when USCI_Ax receives new data. UCAXRXIFG is automatically reset when the transfer starts. If UCAXRXIE is set, the UCAXRXIFG does not trigger a transfer. A transfer is triggered when USCI_Ax is ready to transmit new data. UCAXTXIFG is automatically reset when the transfer starts. If UCAXTXIE is set, the UCAXTXIFG does not trigger a transfer.
USCI_Bx	A transfer is triggered when USCI_Bx receives new data. UCBXRXIFG is automatically reset when the transfer starts. If UCBXRXIE is set, the UCBXRXIFG does not trigger a transfer. A transfer is triggered when USCI_Bx is ready to transmit new data. UCBXTXIFG is automatically reset when the transfer starts. If UCBXTXIE is set, the UCBXTXIFG does not trigger a transfer.
DAC12_A	A transfer is triggered when the DAC12_xCTL0 DAC12IFG flag is set. The DAC12_xCTL0 DAC12IFG flag is automatically cleared when the transfer starts. If the DAC12_xCTL0 DAC12IE bit is set, the DAC12_xCTL0 DAC12IFG flag does not trigger a transfer.
ADC12_A	A transfer is triggered by an ADC12IFG flag. When single-channel conversions are performed, the corresponding ADC12IFG is the trigger. When sequences are used, the ADC12IFG for the last conversion in the sequence is the trigger. A transfer is triggered when the conversion is completed and the ADC12IFG is set. Setting the ADC12IFG with software does not trigger a transfer. All ADC12IFG flags are automatically reset when the associated ADC12MEMx register is accessed by the DMA controller.
MPY	A transfer is triggered when the hardware multiplier is ready for a new operand.
Reserved	No transfer is triggered.

### 11.2.5 Stopping DMA Transfers

There are two ways to stop DMA transfers in progress:

- A single, block, or burst-block transfer may be stopped with an NMI, if the ENNMI bit is set in register DMACTL1.
- A burst-block transfer may be stopped by clearing the DMAEN bit.

### 11.2.6 DMA Channel Priorities

The default DMA channel priorities are DMA0 through DMA7. If two or three triggers happen simultaneously or are pending, the channel with the highest priority completes its transfer (single, block, or burst-block transfer) first, then the second priority channel, then the third priority channel. Transfers in progress are not halted if a higher-priority channel is triggered. The higher-priority channel waits until the transfer in progress completes before starting.

The DMA channel priorities are configurable with the ROUNDROBIN bit. When the ROUNDROBIN bit is set, the channel that completes a transfer becomes the lowest priority. The order of the priority of the channels always stays the same, DMA0-DMA1-DMA2, for example, for three channels. When the ROUNDROBIN bit is cleared, the channel priority returns to the default priority.

<b>DMA Priority</b>	<b>Transfer Occurs</b>	<b>New DMA Priority</b>
DMA0-DMA1-DMA2	DMA1	DMA2-DMA0-DMA1
DMA2-DMA0-DMA1	DMA2	DMA0-DMA1-DMA2
DMA0-DMA1-DMA2	DMA0	DMA1-DMA2-DMA0

### 11.2.7 DMA Transfer Cycle Time

The DMA controller requires one or two MCLK clock cycles to synchronize before each single transfer or complete block or burst-block transfer. Each byte/word transfer requires two MCLK cycles after synchronization, and one cycle of wait time after the transfer. Because the DMA controller uses MCLK, the DMA cycle time is dependent on the MSP430 operating mode and clock system setup.

If the MCLK source is active but the CPU is off, the DMA controller uses the MCLK source for each transfer, without reenabling the CPU. If the MCLK source is off, the DMA controller temporarily restarts MCLK, sourced with DCOCLK, for the single transfer or complete block or burst-block transfer. The CPU remains off and after the transfer completes, MCLK is turned off. The maximum DMA cycle time for all operating modes is shown in [Table 11-3](#).

**Table 11-3. Maximum Single-Transfer DMA Cycle Time**

CPU Operating Mode Clock Source	Maximum DMA Cycle Time
Active mode MCLK = DCOCLK	4 MCLK cycles
Active mode MCLK = LFXT1CLK	4 MCLK cycles
Low-power mode LPM0/1 MCLK = DCOCLK	5 MCLK cycles
Low-power mode LPM3/4 MCLK = DCOCLK	5 MCLK cycles + 5 $\mu$ s <sup>(1)</sup>
Low-power mode LPM0/1 MCLK = LFXT1CLK	5 MCLK cycles
Low-power mode LPM3 MCLK = LFXT1CLK	5 MCLK cycles
Low-power mode LPM4 MCLK = LFXT1CLK	5 MCLK cycles + 5 $\mu$ s <sup>(1)</sup>

<sup>(1)</sup> The additional 5  $\mu$ s are needed to start the DCOCLK. It is the  $t_{(LPMx)}$  parameter in the data sheet.

### 11.2.8 Using DMA With System Interrupts

DMA transfers are not interruptible by system interrupts. System interrupts remain pending until the completion of the transfer. NMIs can interrupt the DMA controller if the ENNMI bit is set.

System interrupt service routines are interrupted by DMA transfers. If an interrupt service routine or other routine must execute with no interruptions, the DMA controller should be disabled prior to executing the routine.

### 11.2.9 DMA Controller Interrupts

Each DMA channel has its own DMAIFG flag. Each DMAIFG flag is set in any mode when the corresponding DMAxSZ register counts to zero. If the corresponding DMAIE and GIE bits are set, an interrupt request is generated.

All DMAIFG flags are prioritized, with DMA0IFG being the highest, and combined to source a single interrupt vector. The highest-priority enabled interrupt generates a number in the DMAIV register. This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled DMA interrupts do not affect the DMAIV value.

Any access, read or write, of the DMAIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, assume that DMA0 has the highest priority. If the DMA0IFG and DMA2IFG flags are set when the interrupt service routine accesses the DMAIV register, DMA0IFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the DMA2IFG generates another interrupt.

### 11.2.9.1 DMAIV Software Example

The following software example shows the recommended use of DMAIV and the handling overhead for an eight channel DMA controller. The DMAIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

		Cycles
DMA_HND	... ; Interrupt latency	6
ADD	&DMAIV, PC ; Add offset to Jump table	3
RETI	; Vector 0: No interrupt	5
JMP	DMA0_HND ; Vector 2: DMA channel 0	2
JMP	DMA1_HND ; Vector 4: DMA channel 1	2
JMP	DMA2_HND ; Vector 6: DMA channel 2	2
JMP	DMA3_HND ; Vector 8: DMA channel 3	2
JMP	DMA4_HND ; Vector 10: DMA channel 4	2
JMP	DMA5_HND ; Vector 12: DMA channel 5	2
JMP	DMA6_HND ; Vector 14: DMA channel 6	2
JMP	DMA7_HND ; Vector 16: DMA channel 7	2
DMA7_HND	... ; Task starts here	
RETI	; Back to main program	5
DMA6_HND	... ; Task starts here	
RETI	; Back to main program	5
DMA5_HND	... ; Task starts here	
RETI	; Back to main program	5
DMA4_HND	... ; Task starts here	
RETI	; Back to main program	5
DMA3_HND	... ; Task starts here	
RETI	; Back to main program	5
DMA2_HND	... ; Task starts here	
RETI	; Back to main program	5
DMA1_HND	... ; Task starts here	
RETI	; Back to main program	5
DMA0_HND	... ; Task starts here	
RETI	; Back to main program	5

### 11.2.10 Using the USCI\_B I<sup>2</sup>C Module With the DMA Controller

The USCI\_B I<sup>2</sup>C module provides two trigger sources for the DMA controller. The USCI\_B I<sup>2</sup>C module can trigger a transfer when new I<sup>2</sup>C data is received and the when the transmit data is needed.

### 11.2.11 Using ADC12 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data from any ADC12MEMx register to another location. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput of the ADC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

DMA transfers can be triggered from any ADC12IFG flag. When CONSEQx = {0,2}, the ADC12IFG flag for the ADC12MEMx used for the conversion can trigger a DMA transfer. When CONSEQx = {1,3}, the ADC12IFG flag for the last ADC12MEMx in the sequence can trigger a DMA transfer. Any ADC12IFG flag is automatically cleared when the DMA controller accesses the corresponding ADC12MEMx.

### 11.2.12 Using DAC12 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data to the DAC12\_xDAT register. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput to the DAC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

Applications requiring periodic waveform generation can benefit from using the DMA controller with the DAC12. For example, an application that produces a sinusoidal waveform may store the sinusoid values in a table. The DMA controller can continuously and automatically transfer the values to the DAC12 at specific intervals creating the sinusoid with zero CPU execution. The DAC12\_xCTL DAC12IFG flag is automatically cleared when the DMA controller accesses the DAC12\_xDAT register.

## 11.3 DMA Registers

The DMA module registers are listed in [Table 11-4](#). The base addresses can be found in the device-specific data sheet. Each channel starts at its respective base address. The address offsets are listed in [Table 11-4](#).

**Table 11-4. DMA Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	DMACTL0	DMA Control 0	Read/write	Word	0000h	<a href="#">Section 11.3.1</a>
02h	DMACTL1	DMA Control 1	Read/write	Word	0000h	<a href="#">Section 11.3.2</a>
04h	DMACTL2	DMA Control 2	Read/write	Word	0000h	<a href="#">Section 11.3.3</a>
06h	DMACTL3	DMA Control 3	Read/write	Word	0000h	<a href="#">Section 11.3.4</a>
08h	DMACTL4	DMA Control 4	Read/write	Word	0000h	<a href="#">Section 11.3.5</a>
0Eh	DMAIV	DMA Interrupt Vector	Read only	Word	0000h	<a href="#">Section 11.3.10</a>
00h	DMA0CTL	DMA Channel 0 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA0SA	DMA Channel 0 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA0DA	DMA Channel 0 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>
0Ah	DMA0SZ	DMA Channel 0 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>
00h	DMA1CTL	DMA Channel 1 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA1SA	DMA Channel 1 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA1DA	DMA Channel 1 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>
0Ah	DMA1SZ	DMA Channel 1 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>
00h	DMA2CTL	DMA Channel 2 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA2SA	DMA Channel 2 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA2DA	DMA Channel 2 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>
0Ah	DMA2SZ	DMA Channel 2 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>
00h	DMA3CTL	DMA Channel 3 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA3SA	DMA Channel 3 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA3DA	DMA Channel 3 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>
0Ah	DMA3SZ	DMA Channel 3 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>
00h	DMA4CTL	DMA Channel 4 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA4SA	DMA Channel 4 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA4DA	DMA Channel 4 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>
0Ah	DMA4SZ	DMA Channel 4 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>
00h	DMA5CTL	DMA Channel 5 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA5SA	DMA Channel 5 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA5DA	DMA Channel 5 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>
0Ah	DMA5SZ	DMA Channel 5 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>
00h	DMA6CTL	DMA Channel 6 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA6SA	DMA Channel 6 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA6DA	DMA Channel 6 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>

**Table 11-4. DMA Registers (continued)**

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Ah	DMA6SZ	DMA Channel 6 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>
00h	DMA7CTL	DMA Channel 7 Control	Read/write	Word	0000h	<a href="#">Section 11.3.6</a>
02h	DMA7SA	DMA Channel 7 Source Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.7</a>
06h	DMA7DA	DMA Channel 7 Destination Address	Read/write	Word, double word	undefined	<a href="#">Section 11.3.8</a>
0Ah	DMA7SZ	DMA Channel 7 Transfer Size	Read/write	Word	undefined	<a href="#">Section 11.3.9</a>

### 11.3.1 DMACTL0 Register

DMA Control 0 Register

**Figure 11-6. DMACTL0 Register**

15	14	13	12	11	10	9	8
Reserved		DMA1TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved		DMA0TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 11-5. DMACTL0 Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DMA1TSEL	RW	0h	DMA 1 trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA1TRIG0 00001b = DMA1TRIG1 00010b = DMA1TRIG2 ⋮ 11110b = DMA1TRIG30 11111b = DMA1TRIG31
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	DMA0TSEL	RW	0h	DMA 0 trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA0TRIG0 00001b = DMA0TRIG1 00010b = DMA0TRIG2 ⋮ 11110b = DMA0TRIG30 11111b = DMA0TRIG31

**11.3.2 DMACTL1 Register**

DMA Control 1 Register

**Figure 11-7. DMACTL1 Register**

15	14	13	12	11	10	9	8
Reserved		DMA3TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved		DMA2TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 11-6. DMACTL1 Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DMA3TSEL	RW	0h	DMA 3 trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA3TRIG0 00001b = DMA3TRIG1 00010b = DMA3TRIG2 ⋮ 11110b = DMA3TRIG30 11111b = DMA3TRIG31
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	DMA2TSEL	RW	0h	DMA 2 trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA2TRIG0 00001b = DMA2TRIG1 00010b = DMA2TRIG2 ⋮ 11110b = DMA2TRIG30 11111b = DMA2TRIG31

### 11.3.3 DMACTL2 Register

DMA Control 2 Register

**Figure 11-8. DMACTL2 Register**

15	14	13	12	11	10	9	8
Reserved		DMA5TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved		DMA4TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 11-7. DMACTL2 Register Description**

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

**11.3.4 DMACTL3 Register**

DMA Control 3 Register

**Figure 11-9. DMACTL3 Register**

15	14	13	12	11	10	9	8
Reserved		DMA7TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved		DMA6TSEL					
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 11-8. DMACTL3 Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DMA7TSEL	RW	0h	DMA 7 trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA7TRIG0 00001b = DMA7TRIG1 00010b = DMA7TRIG2 ⋮ 11110b = DMA7TRIG30 11111b = DMA7TRIG31
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	DMA6TSEL	RW	0h	DMA 6 trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA6TRIG0 00001b = DMA6TRIG1 00010b = DMA6TRIG2 ⋮ 11110b = DMA6TRIG30 11111b = DMA6TRIG31

### 11.3.5 DMACTL4 Register

DMA Control 4 Register

**Figure 11-10. DMACTL4 Register**

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

**Table 11-9. DMACTL4 Register Description**

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

**11.3.6 DMAxCTL Register**

DMA Channel x Control Register

**Figure 11-11. DMAxCTL Register**

15	14	13	12	11	10	9	8
Reserved	DMADT			DMADSTINCR		DMASRCINCR	
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DMADSTBYTE	DMASRCBYTE	DMALEVEL	DMAEN	DMAIFG	DMAIE	DMAABORT	DMAREQ
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 11-10. DMAxCTL Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-12	DMADT	RW	0h	DMA transfer mode 000b = Single transfer 001b = Block transfer 010b = Burst-block transfer 011b = Burst-block transfer 100b = Repeated single transfer 101b = Repeated block transfer 110b = Repeated burst-block transfer 111b = Repeated burst-block transfer
11-10	DMADSTINCR	RW	0h	DMA destination increment. This bit selects automatic incrementing or decrementing of the destination address after each byte or word transfer. When DMADSTBYTE = 1, the destination address increments/decrements by one. When DMADSTBYTE = 0, the destination address increments/decrements by two. The DMAxDA is copied into a temporary register and the temporary register is incremented or decremented. DMAxDA is not incremented or decremented. 00b = Destination address is unchanged. 01b = Destination address is unchanged. 10b = Destination address is decremented. 11b = Destination address is incremented.
9-8	DMASRCINCR	RW	0h	DMA source increment. This bit selects automatic incrementing or decrementing of the source address for each byte or word transfer. When DMASRCBYTE = 1, the source address increments/decrements by one. When DMASRCBYTE = 0, the source address increments/decrements by two. The DMAxSA is copied into a temporary register and the temporary register is incremented or decremented. DMAxSA is not incremented or decremented. 00b = Source address is unchanged. 01b = Source address is unchanged. 10b = Source address is decremented. 11b = Source address is incremented.
7	DMADSTBYTE	RW	0h	DMA destination byte. This bit selects the destination as a byte or word. 0b = Word 1b = Byte
6	DMASRCBYTE	RW	0h	DMA source byte. This bit selects the source as a byte or word. 0b = Word 1b = Byte
5	DMALEVEL	RW	0h	DMA level. This bit selects between edge-sensitive and level-sensitive triggers. 0b = Edge sensitive (rising edge) 1b = Level sensitive (high level)

**Table 11-10. DMAXCTL Register Description (continued)**

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

**11.3.7 DMAxSA Register**

DMA Channel x Source Address Register

**Figure 11-12. DMAxSA Register**

31	30	29	28	27	26	25	24
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
23	22	21	20	19	18	17	16
Reserved				DMAxSA			
r0	r0	r0	r0	rw	rw	rw	rw
15	14	13	12	11	10	9	8
DMAxSA							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
DMAxSA							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 11-11. DMAxSA Register Description**

Bit	Field	Type	Reset	Description
31-20	Reserved	R	0h	Reserved. Always reads as 0.
19-0	DMAxSA	RW	undefined	DMA source address. The source address register points to the DMA source address for single transfers or the first source address for block transfers. The source address register remains unchanged during block and burst-block transfers. There are two words for the DMAxSA register. Bits 31-20 are reserved and always read as zero. Reading or writing bits 19-16 requires the use of extended instructions. When writing to DMAxSA with word instructions, bits 19-16 are cleared.

### 11.3.8 DMAxDA Register

DMA Channel x Destination Address Register

**Figure 11-13. DMAxDA Register**

31	30	29	28	27	26	25	24
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
23	22	21	20	19	18	17	16
Reserved				DMAxDA			
r0	r0	r0	r0	rw	rw	rw	rw
15	14	13	12	11	10	9	8
DMAxDA							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
DMAxDA							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 11-12. DMAxDA Register Description**

Bit	Field	Type	Reset	Description
31-20	Reserved	R	0h	Reserved. Always reads as 0.
19-0	DMAxDA	RW	undefined	DMA destination address. The destination address register points to the DMA destination address for single transfers or the first destination address for block transfers. The destination address register remains unchanged during block and burst-block transfers. There are two words for the DMAxDA register. Bits 31-20 are reserved and always read as zero. Reading or writing bits 19-16 requires the use of extended instructions. When writing to DMAxDA with word instructions, bits 19-16 are cleared.

**11.3.9 DMAxSZ Register**

DMA Channel x Size Address Register

**Figure 11-14. DMAxSZ Register**

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

**Table 11-13. DMAxSZ Register Description**

Bit	Field	Type	Reset	Description
15-0	DMAxSZ	RW	undefined	<p>DMA size. The DMA size register defines the number of byte/word data per block transfer. DMAxSZ register decrements with each word or byte transfer. When DMAxSZ decrements to 0, it is immediately and automatically reloaded with its previously initialized value.</p> <p>00000h = Transfer is disabled.      00001h = One byte or word is transferred.      00002h = Two bytes or words are transferred.      :      0FFFFh = 65535 bytes or words are transferred.</p>

### 11.3.10 DMAIV Register

DMA Interrupt Vector Register

**Figure 11-15. DMAIV Register**

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

**Table 11-14. DMAIV Register Description**

Bit	Field	Type	Reset	Description
15-0	DMAIV	R	0h	<p>DMA interrupt vector value</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: DMA channel 0; Interrupt Flag: DMA0IFG; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: DMA channel 1; Interrupt Flag: DMA1IFG</p> <p>06h = Interrupt Source: DMA channel 2; Interrupt Flag: DMA2IFG</p> <p>08h = Interrupt Source: DMA channel 3; Interrupt Flag: DMA3IFG</p> <p>0Ah = Interrupt Source: DMA channel 4; Interrupt Flag: DMA4IFG</p> <p>0Ch = Interrupt Source: DMA channel 5; Interrupt Flag: DMA5IFG</p> <p>0Eh = Interrupt Source: DMA channel 6; Interrupt Flag: DMA6IFG</p> <p>10h = Interrupt Source: DMA channel 7; Interrupt Flag: DMA7IFG; Interrupt Priority: Lowest</p>

## **Digital I/O Module**

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

<b>Topic</b>		<b>Page</b>
12.1	Digital I/O Introduction .....	407
12.2	Digital I/O Operation .....	408
12.3	I/O Configuration and LPMx.5 Low-Power Modes .....	411
12.4	Digital I/O Registers .....	413

## 12.1 Digital I/O Introduction

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts. Some devices may include additional port interrupts.
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors

Devices within the family may have up to twelve digital I/O ports implemented (P1 to P11 and PJ). Most ports contain eight I/O lines; however, some ports may contain less (see the device-specific data sheet for ports available). Each I/O line is individually configurable for input or output direction, and each can be individually read or written. Each I/O line is individually configurable for pullup or pulldown resistors, as well as, configurable drive strength, full or reduced. PJ contains only four I/O lines.

Ports P1 and P2 always have interrupt capability. Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising or falling edge of an input signal. All P1 I/O lines source a single interrupt vector P1IV, and all P2 I/O lines source a different, single interrupt vector P2IV. On some devices, additional ports with interrupt capability may be available (see the device-specific data sheet for details) and contain their own respective interrupt vectors.

Individual ports can be accessed as byte-wide ports or can be combined into word-wide ports and accessed via word formats. Port pairs P1 and P2, P3 and P4, P5 and P6, and so on, are associated with the names PA, PB, PC, and so on, respectively. All port registers are handled in this manner with this naming convention except for the interrupt vector registers, for example, PAIV does not exist for P1IV and P2IV.

When writing to port PA with word operations, all 16 bits are written to the port. When writing to the lower byte of the PA port using byte operations, the upper byte remains unchanged. Similarly, writing to the upper byte of the PA port using byte instructions leaves the lower byte unchanged. When writing to a port that contains less than the maximum number of bits possible, the unused bits are a "don't care". Ports PB, PC, PD, PE, and PF behave similarly.

Reading of the PA port using word operations causes all 16 bits to be transferred to the destination. Reading the lower or upper byte of the PA port (P1 or P2) and storing to memory using byte operations causes only the lower or upper byte to be transferred to the destination, respectively. Reading of the PA port and storing to a general-purpose register using byte operations causes the byte transferred to be written to the least significant byte of the register. The upper significant byte of the destination register is cleared automatically. Ports PB, PC, PD, PE, and PF behave similarly. When reading from ports that contain less than the maximum bits possible, unused bits are read as zeros (similarly for port PJ).

## 12.2 Digital I/O Operation

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

### 12.2.1 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.

---

### 12.2.2 Output Registers (PxOUT)

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction.

- Bit = 0: Output is low
- Bit = 1: Output is high

If the pin is configured as I/O function, input direction and the pullup/pulldown resistor are enabled; the corresponding bit in the PxOUT register selects pullup or pulldown.

- Bit = 0: Pin is pulled down
- Bit = 1: Pin is pulled up

### 12.2.3 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

### 12.2.4 Pullup or Pulldown Resistor Enable Registers (PxREN)

Each bit in each PxREN register enables or disables the pullup/pulldown resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin contains a pullup or pulldown.

- Bit = 0: Pullup/pulldown resistor disabled
- Bit = 1: Pullup/pulldown resistor enabled

Table 12-1 summarizes the usage of PxDIR, PxREN, and PxOUT for proper I/O configuration.

**Table 12-1. 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

### 12.2.5 Output Drive Strength Registers (PxDS)

Each bit in each PxDS register selects either full drive or reduced drive strength. Default is reduced drive strength.

- Bit = 0: Reduced drive strength
- Bit = 1: Full drive strength

---

**NOTE: Drive strength and EMI**

All outputs default to reduced drive strength to reduce EMI. Using full drive strength can result in increased EMI.

---

### 12.2.6 Function Select Registers (PxSEL)

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PxSEL bit is used to select the pin function – I/O port or peripheral module function.

- Bit = 0: I/O Function is selected for the pin
- Bit = 1: Peripheral module function is selected for the pin

Setting PxSEL = 1 does not automatically set the pin direction. Other peripheral module functions may require the PxDIR bits to be configured according to the direction needed for the module function. See the pin schematics in the device-specific data sheet.

---

**NOTE: P1 and P2 interrupts are disabled when PxSEL = 1**

When any PxSEL bit is set, the corresponding pin's interrupt function is disabled. Therefore, signals on these pins does not generate P1 or P2 interrupts, regardless of the state of the corresponding P1IE or P2IE bit.

---

When a port pin is selected as an input to a peripheral, the input signal to the peripheral is a latched representation of the signal at the device pin. While its corresponding PxSEL = 1, the internal input signal follows the signal at the pin. However, if its PxSEL = 0, the input to the peripheral maintains the value of the input signal at the device pin before its corresponding PxSEL bit was reset.

### 12.2.7 Port Interrupts

Each pin in ports P1 and P2 has interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. On some devices, additional ports have interrupt capability (see the device-specific data sheet). All P1 interrupt flags are prioritized, with P1IFG.0 being the highest, and combined to source a single interrupt vector. The highest priority enabled interrupt generates a number in the P1IV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled P1 interrupts do not affect the P1IV value. The same functionality exists for P2. The PxIV registers are word access only. Some devices may contain additional port interrupts besides P1 and P2. See the device specific data sheet to determine which port interrupts are available.

Each PxIFG bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge occurs at the pin. All PxIFG interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Software can also set each PxIFG flag, providing a way to generate a software-initiated interrupt.

- Bit = 0: No interrupt is pending
- Bit = 1: An interrupt is pending

Only transitions, not static levels, cause interrupts. If any PxIFG flag becomes set during a Px interrupt service routine, or is set after the RETI instruction of a Px interrupt service routine is executed, the set PxIFG flag generates another interrupt. This ensures that each transition is acknowledged.

**NOTE: PxIFG flags when changing PxOUT, PxDIR, or PxREN**

Writing to P1OUT, P1DIR, P1REN, P2OUT, P2DIR, or P2REN can result in setting the corresponding P1IFG or P2IFG flags.

Any access (read or write) of the P1IV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, assume that P1IFG.0 has the highest priority. If the P1IFG.0 and P1IFG.2 flags are set when the interrupt service routine accesses the P1IV register, P1IFG.0 is reset automatically. After the RETI instruction of the interrupt service routine is executed, the P1IFG.2 generates another interrupt.

Port P2 interrupts behave similarly, and source a separate single interrupt vector and utilize the P2IV register.

**Port Interrupt Software Example**

The following software example shows the recommended use of P1IV and the handling overhead. The P1IV value is added to the PC to automatically jump to the appropriate routine. The P2IV is similar.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

		Cycles
P1_HND	... ADD &P1IV,PC RETI	6 3 5
JMP	P1_0_HND	2
JMP	P1_1_HND	2
JMP	P1_2_HND	2
JMP	P1_3_HND	2
JMP	P1_4_HND	2
JMP	P1_5_HND	2
JMP	P1_6_HND	2
JMP	P1_7_HND	2
P1_7_HND	... RETI	7 5
P1_6_HND	... RETI	6 5
P1_5_HND	... RETI	5 5
P1_4_HND	... RETI	4 5
P1_3_HND	... RETI	3 5
P1_2_HND	... RETI	2 5
P1_1_HND	... RETI	1 5
P1_0_HND	... RETI	0 5

## 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 with a low-to-high transition
- Bit = 1: Respective PxIFG flag is set with a high-to-low transition

---

### **NOTE: Writing to PxIES**

Writing to P1IES or P2IES for each corresponding I/O can result in setting the corresponding interrupt flags.

PxIES	PxIN	PxIFG
0 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 0	1	May be set

---

## 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

### **12.2.8 Configuring Unused Port Pins**

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

---

### **NOTE: Configuring port J and shared JTAG pins:**

Application should ensure that port PJ is configured properly to prevent a floating input. Because port PJ is shared with the JTAG function, floating inputs may not be noticed when in an emulation environment. Port J is initialized to high-impedance inputs by default.

## **12.3 I/O Configuration and LPMx.5 Low-Power Modes**

---

**NOTE:** The LPMx.5 low-power modes may not be available on all devices. The LPM4.5 power mode allows for lowest power consumption and no clocks are available. The LPM3.5 power mode allows for RTC mode operation at the lowest power consumption available. See the [SYS chapter](#) for details; also see the device-specific datasheet for LPMx.5 low-power modes that are available. With respect to the digital I/O, this section is applicable for both LPM3.5 and LPM4.5.

The regulator of the Power Management Module (PMM) is disabled upon entering LPMx.5 (LPM3.5 or LPM4.5), which causes all I/O register configurations to be lost. Because the I/O register configurations are lost, the configuration of I/O pins must be handled differently to ensure that all pins in the application behave in a controlled manner upon entering and exiting LPMx.5. Properly setting the I/O pins is critical to achieving the lowest possible power consumption in LPMx.5, as well as preventing any possible uncontrolled input or output I/O state in the application. The application has complete control of the I/O pin conditions preventing the possibility of unwanted spurious activity upon entry and exit from LPMx.5. The detailed flow for entering and exiting LPMx.5 with respect to the I/O operation is as follows:

- Set all I/Os to general purpose I/Os and configure as needed. Each I/O can be set to input high impedance, input with pulldown, input with pullup, output high (low or high drive strength), or output low (low or high drive strength). It is critical that no inputs are left floating in the application, otherwise excess current may be drawn in LPMx.5. Configuring the I/O in this manner ensures that each pin is in a safe condition prior to entering LPMx.5.

Optionally, configure input interrupt pins for wake-up from LPMx.5. To wake the device from LPMx.5, a general-purpose I/O port must contain an input port with interrupt capability. Not all devices include wakeup from LPMx.5 via I/O, and not all inputs with interrupt capability offer wakeup from LPMx.5. See the device-specific data sheet for availability. To configure a port to wake up the device, it should be configured properly prior to entering LPMx.5. Each port should be configured as general-purpose input. Pulldowns or pullups can be applied if required. Setting the PxIES bit of the corresponding register determines the edge transition that wakes the device. Lastly, the PxIE for the port must be enabled, as well as the general interrupt enable.

---

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

Although it is recommended to set GIE = 1 prior to entering LPMx.5, it is not required. Device wakeup from LPMx.5 with an enabled wakeup function will still cause the device to wake up from LPMx.5 even with GIE = 0. If GIE = 0 prior to LPMx.5, additional care may be required. Should the respective interrupt event should occur during LPMx.5 entry, the device may not recognize this or any future interrupt wakeup event on this function.

- Enter LPMx.5 with LPMx.5 entry sequence, enable general interrupts for wake-up:

```

MOV.B #PMMPW_H, &PMMCTL0_H           ; Open PMM registers for write
BIS.B #PMMREGOFF, &PMMCTL0_L         ;
BIS    #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR ; Enter LPMx.5 when PMMREGOFF is set

```

- Upon entry into LPMx.5, LOCKLPM5 residing in PM5CTL0 of the PMM module is set automatically. The I/O pin states are held and locked based on the settings prior to LPMx.5 entry. Note that only the pin conditions are retained. All other port configuration register settings such as PxDIR, PxREN, PxOUT, PxDS, PxIES, and PxIE contents are lost.
- An LPMx.5 wakeup event (for example, an edge on a configured wakeup input pin) starts the BOR entry sequence together with the regulator. All peripheral registers are set to their default conditions. Upon exit from LPMx.5, the I/O pins remain locked while LOCKLPM5 remains set. Keeping the I/O pins locked ensures that all pin conditions remain stable upon entering the active mode regardless of the default I/O register settings.
- Once in active mode, the I/O configuration and I/O interrupt configuration that was not retained during LPMx.5 should be restored to the values prior to entering LPMx.5. It is recommended to reconfigure the PxIES and PxIE to their previous settings to prevent a false port interrupt from occurring. The LOCKLPM5 bit can then be cleared, which releases the I/O pin conditions and I/O interrupt configuration. Any changes to the port configuration registers while LOCKLPM5 is set, have no effect on the I/O pins.
- After enabling the I/O interrupts, the I/O interrupt that caused the wakeup can be serviced indicated by the PxIFG flags. These flags can be used directly, or the corresponding PxIV register may be used. Note that the PxIFG flag cannot be cleared until the LOCKLPM5 bit has been cleared.

---

**NOTE:** It is possible that multiple events occurred on various ports. In these cases, multiple PxIFG flags will be set, and it cannot be determined which port has caused the I/O wakeup.

## 12.4 Digital I/O Registers

The digital I/O registers are listed in [Table 12-2](#). The base addresses can be found in the device-specific data sheet. Each port grouping begins at its base address. The address offsets are given in [Table 12-2](#).

---

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

---

**Table 12-2. Digital I/O Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Eh	P1IV	Port 1 Interrupt Vector	Read only	Word	0000h	<a href="#">Section 12.4.1</a>
0Eh	P1IV_L		Read only	Byte	00h	
0Fh	P1IV_H		Read only	Byte	00h	
1Eh	P2IV	Port 2 Interrupt Vector	Read only	Word	0000h	<a href="#">Section 12.4.2</a>
1Eh	P2IV_L		Read only	Byte	00h	
1Fh	P2IV_H		Read only	Byte	00h	
00h	P1IN or PAIN_L	Port 1 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
02h	P1OUT or PAOUT_L	Port 1 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
04h	P1DIR or PADIR_L	Port 1 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
06h	P1REN or PAREN_L	Port 1 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
08h	P1DS or PADS_L	Port 1 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Ah	P1SEL or PASEL_L	Port 1 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
18h	P1IES or PAIES_L	Port 1 Interrupt Edge Select	Read/write	Byte	undefined	<a href="#">Section 12.4.3</a>
1Ah	P1IE or PAIE_L	Port 1 Interrupt Enable	Read/write	Byte	00h	<a href="#">Section 12.4.4</a>
1Ch	P1IFG or PAIFG_L	Port 1 Interrupt Flag	Read/write	Byte	00h	<a href="#">Section 12.4.5</a>
01h	P2IN or PAIN_H	Port 2 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
03h	P2OUT or PAOUT_H	Port 2 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
05h	P2DIR or PADIR_H	Port 2 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
07h	P2REN or PAREN_H	Port 2 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
09h	P2DS or PADS_H	Port 2 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Bh	P2SEL or PASEL_H	Port 2 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
19h	P2IES or PAIES_H	Port 2 Interrupt Edge Select	Read/write	Byte	undefined	<a href="#">Section 12.4.6</a>
1Bh	P2IE or PAIE_H	Port 2 Interrupt Enable	Read/write	Byte	00h	<a href="#">Section 12.4.7</a>
1Dh	P2IFG or PAIFG_H	Port 2 Interrupt Flag	Read/write	Byte	00h	<a href="#">Section 12.4.8</a>
00h	P3IN or PBIN_L	Port 3 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
02h	P3OUT or PBOUT_L	Port 3 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>

**Table 12-2. Digital I/O Registers (continued)**

Offset	Acronym	Register Name	Type	Access	Reset	Section
04h	P3DIR or PBDIR_L	Port 3 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
06h	P3REN or PBREN_L	Port 3 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
08h	P3DS or PBDS_L	Port 3 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Ah	P3SEL or PBSEL_L	Port 3 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
01h	P4IN or PBIN_H	Port 4 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
03h	P4OUT or PBOUT_H	Port 4 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
05h	P4DIR or PBDIR_H	Port 4 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
07h	P4REN or PBREN_H	Port 4 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
09h	P4DS or PBDS_H	Port 4 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Bh	P4SEL or PBSEL_H	Port 4 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
00h	P5IN or PCIN_L	Port 5 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
02h	P5OUT or PCOUT_L	Port 5 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
04h	P5DIR or PCDIR_L	Port 5 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
06h	P5REN or PCREN_L	Port 5 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
08h	P5DS or PCDS_L	Port 5 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Ah	P5SEL or PCSEL_L	Port 5 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
01h	P6IN or PCIN_H	Port 6 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
03h	P6OUT or PCOUT_H	Port 6 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
05h	P6DIR or PCDIR_H	Port 6 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
07h	P6REN or PCREN_H	Port 6 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
09h	P6DS or PCDS_H	Port 6 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Bh	P6SEL or PCSEL_H	Port 6 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
00h	P7IN or PDIN_L	Port 7 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
02h	P7OUT or PDOUT_L	Port 7 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
04h	P7DIR or PDDIR_L	Port 7 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
06h	P7REN or PDREN_L	Port 7 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
08h	P7DS or PDDS_L	Port 7 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Ah	P7SEL or PDSEL_L	Port 7 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>

**Table 12-2. Digital I/O Registers (continued)**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Access</b>	<b>Reset</b>	<b>Section</b>
01h	P8IN or PDIN_H	Port 8 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
03h	P8OUT or PDOOUT_H	Port 8 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
05h	P8DIR or PDDIR_H	Port 8 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
07h	P8REN or PDREN_H	Port 8 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
09h	P8DS or PDDS_H	Port 8 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Bh	P8SEL or PDSEL_H	Port 8 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
00h	P9IN or PEIN_L	Port 9 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
02h	P9OUT or PEOUT_L	Port 9 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
04h	P9DIR or PEDIR_L	Port 9 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
06h	P9REN or PEREN_L	Port 9 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
08h	P9DS or PEDS_L	Port 9 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Ah	P9SEL or PESEL_L	Port 9 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
01h	P10IN or PEIN_H	Port 10 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
03h	P10OUT or PEOUT_H	Port 10 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
05h	P10DIR or PEDIR_H	Port 10 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
07h	P10REN or PEREN_H	Port 10 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
09h	P10DS or PEDS_H	Port 10 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Bh	P10SEL or PESEL_H	Port 10 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
00h	P11IN or PFIN_L	Port 11 Input	Read only	Byte		<a href="#">Section 12.4.9</a>
02h	P11OUT or PFOUT_L	Port 11 Output	Read/write	Byte	undefined	<a href="#">Section 12.4.10</a>
04h	P11DIR or PFDIR_L	Port 11 Direction	Read/write	Byte	00h	<a href="#">Section 12.4.11</a>
06h	P11REN or PFREN_L	Port 11 Resistor Enable	Read/write	Byte	00h	<a href="#">Section 12.4.12</a>
08h	P11DS or PFDS_L	Port 11 Drive Strength	Read/write	Byte	00h	<a href="#">Section 12.4.13</a>
0Ah	P11SEL or PFSEL_L	Port 11 Port Select	Read/write	Byte	00h	<a href="#">Section 12.4.14</a>
00h	PAIN	Port A Input	Read only	Word		
00h	PAIN_L		Read only	Byte		
01h	PAIN_H		Read only	Byte		
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	

**Table 12-2. Digital I/O Registers (continued)**

Offset	Acronym	Register Name	Type	Access	Reset	Section
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	
08h	PADS	Port A Drive Strength	Read/write	Word	0000h	
08h	PADS_L		Read/write	Byte	00h	
09h	PADS_H		Read/write	Byte	00h	
0Ah	PASEL	Port A Port Select	Read/write	Word	0000h	
0Ah	PASEL_L		Read/write	Byte	00h	
0Bh	PASEL_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	
00h	PBIN	Port B Input	Read only	Word		
00h	PBIN_L		Read only	Byte		
01h	PBIN_H		Read only	Byte		
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	
08h	PBDS	Port B Drive Strength	Read/write	Word	0000h	
08h	PBDS_L		Read/write	Byte	00h	
09h	PBDS_H		Read/write	Byte	00h	
0Ah	PBSEL	Port B Port Select	Read/write	Word	0000h	
0Ah	PBSEL_L		Read/write	Byte	00h	
0Bh	PBSEL_H		Read/write	Byte	00h	
00h	PCIN	Port C Input	Read only	Word		
00h	PCIN_L		Read only	Byte		
01h	PCIN_H		Read only	Byte		
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	

**Table 12-2. Digital I/O Registers (continued)**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Access</b>	<b>Reset</b>	<b>Section</b>
06h	PCREN	Port C Resistor Enable	Read/write	Word	0000h	
06h	PCREN_L		Read/write	Byte	00h	
07h	PCREN_H		Read/write	Byte	00h	
08h	PCDS	Port C Drive Strength	Read/write	Word	0000h	
08h	PCDS_L		Read/write	Byte	00h	
09h	PCDS_H		Read/write	Byte	00h	
0Ah	PCSEL	Port C Port Select	Read/write	Word	0000h	
0Ah	PCSEL_L		Read/write	Byte	00h	
0Bh	PCSEL_H		Read/write	Byte	00h	
00h	PDIN	Port D Input	Read only	Word		
00h	PDIN_L		Read only	Byte		
01h	PDIN_H		Read only	Byte		
02h	PDOUT	Port D Output	Read/write	Word	undefined	
02h	PDOUT_L		Read/write	Byte	undefined	
03h	PDOUT_H		Read/write	Byte	undefined	
04h	PDDR	Port D Direction	Read/write	Word	0000h	
04h	PDDR_L		Read/write	Byte	00h	
05h	PDDR_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	
08h	PDDS	Port D Drive Strength	Read/write	Word	0000h	
08h	PDDS_L		Read/write	Byte	00h	
09h	PDDS_H		Read/write	Byte	00h	
0Ah	PDSEL	Port D Port Select	Read/write	Word	0000h	
0Ah	PDSEL_L		Read/write	Byte	00h	
0Bh	PDSEL_H		Read/write	Byte	00h	
00h	PEIN	Port E Input	Read only	Word		
00h	PEIN_L		Read only	Byte		
01h	PEIN_H		Read only	Byte		
02h	PEOUT	Port E Output	Read/write	Word	undefined	
02h	PEOUT_L		Read/write	Byte	undefined	
03h	PEOUT_H		Read/write	Byte	undefined	
04h	PEDR	Port E Direction	Read/write	Word	0000h	
04h	PEDR_L		Read/write	Byte	00h	
05h	PEDR_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	
08h	PEDS	Port E Drive Strength	Read/write	Word	0000h	
08h	PEDS_L		Read/write	Byte	00h	
09h	PEDS_H		Read/write	Byte	00h	
0Ah	PESEL	Port E Port Select	Read/write	Word	0000h	
0Ah	PESEL_L		Read/write	Byte	00h	
0Bh	PESEL_H		Read/write	Byte	00h	
00h	PFIN	Port F Input	Read only	Word		
00h	PFIN_L		Read only	Byte		

**Table 12-2. Digital I/O Registers (continued)**

Offset	Acronym	Register Name	Type	Access	Reset	Section
01h	PFIN_H		Read only	Byte		
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	
08h	PFDS	Port F Drive Strength	Read/write	Word	0000h	
08h	PFDS_L		Read/write	Byte	00h	
09h	PFDS_H		Read/write	Byte	00h	
0Ah	PFSEL	Port F Port Select	Read/write	Word	0000h	
0Ah	PFSEL_L		Read/write	Byte	00h	
0Bh	PFSEL_H		Read/write	Byte	00h	
00h	PJIN	Port J Input	Read only	Word		
00h	PJIN_L		Read only	Byte		
01h	PJIN_H		Read only	Byte		
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	
08h	PJDS	Port J Drive Strength	Read/write	Word	0000h	
08h	PJDS_L		Read/write	Byte	00h	
09h	PJDS_H		Read/write	Byte	00h	

### 12.4.1 P1IV Register

Port 1 Interrupt Vector Register

**Figure 12-1. 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 12-3. P1IV Register Description**

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

**12.4.2 P2IV Register**

Port 2 Interrupt Vector Register

**Figure 12-2. 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 12-4. 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

### **12.4.3 P1IES Register**

Port 1 Interrupt Edge Select Register

**Figure 12-3. P1IES Register**

7	6	5	4	3	2	1	0
P1IES							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 12-5. P1IES Register Description**

Bit	Field	Type	Reset	Description
7-0	P1IES	RW	undefined	Port 1 interrupt edge select 0b = P1IFG flag is set with a low-to-high transition. 1b = P1IFG flag is set with a high-to-low transition.

### **12.4.4 P1IE Register**

Port 1 Interrupt Enable Register

**Figure 12-4. P1IE Register**

7	6	5	4	3	2	1	0
P1IE							
rw-0							

**Table 12-6. P1IE Register Description**

Bit	Field	Type	Reset	Description
7-0	P1IE	RW	0h	Port 1 interrupt enable 0b = Corresponding port interrupt disabled 1b = Corresponding port interrupt enabled

### **12.4.5 P1IFG Register**

Port 1 Interrupt Flag Register

**Figure 12-5. P1IFG Register**

7	6	5	4	3	2	1	0
P1IFG							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 12-7. P1IFG Register Description**

Bit	Field	Type	Reset	Description
7-0	P1IFG	RW	0h	Port 1 interrupt flag 0b = No interrupt is pending 1b = Interrupt is pending

**12.4.6 P2IES Register**

Port 2 Interrupt Edge Select Register

**Figure 12-6. P2IES Register**

7	6	5	4	3	2	1	0
P2IES							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 12-8. P2IES Register Description**

Bit	Field	Type	Reset	Description
7-0	P2IES	RW	undefined	Port 2 interrupt edge select 0b = P2IFG flag is set with a low-to-high transition. 1b = P2IFG flag is set with a high-to-low transition.

**12.4.7 P2IE Register**

Port 2 Interrupt Enable Register

**Figure 12-7. P2IE Register**

7	6	5	4	3	2	1	0
P2IE							
rw-0							

**Table 12-9. P2IE Register Description**

Bit	Field	Type	Reset	Description
7-0	P2IE	RW	0h	Port 2 interrupt enable 0b = Corresponding port interrupt disabled 1b = Corresponding port interrupt enabled

**12.4.8 P2IFG Register**

Port 2 Interrupt Flag Register

**Figure 12-8. P2IFG Register**

7	6	5	4	3	2	1	0
P2IFG							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 12-10. P2IFG Register Description**

Bit	Field	Type	Reset	Description
7-0	P2IFG	RW	0h	Port 2 interrupt flag 0b = No interrupt is pending 1b = Interrupt is pending

### 12.4.9 PxIN Register

Port x Input Register

**Figure 12-9. PxIN Register**

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

**Table 12-11. PxIN Register Description**

Bit	Field	Type	Reset	Description
7-0	PxIN	R	undefined	Port x input. Read only.

### 12.4.10 PxOUT Register

Port x Output Register

**Figure 12-10. PxOUT Register**

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

**Table 12-12. 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

### 12.4.11 PxDIR Register

Port x Direction Register

**Figure 12-11. 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 12-13. PxDIR 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

**12.4.12 PxREN Register**

Port x Pullup/Pulldown Resistor Enable Registers

**Figure 12-12. 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 12-14. PxREN Register Description**

Bit	Field	Type	Reset	Description
7-0	PxREN	RW	0h	Port x pullup/pulldown resistor enable. When respective port is configured as input, setting this bit will enable the pullup or pulldown. See <a href="#">Table 12-1</a> 0b = Pullup or pulldown disabled. 1b = Pullup or pulldown enabled.

**12.4.13 PxDS Register**

Port x Drive Strength Register

**Figure 12-13. PxDS Register**

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

**Table 12-15. PxDS Register Description**

Bit	Field	Type	Reset	Description
7-0	PxDS	RW	0h	Port x drive strength 0b = Reduced output drive strength 1b = Full output drive strength

**12.4.14 PxSEL Register**

REGISTERDESCRIPTION

**Figure 12-14. PxSEL Register**

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

**Table 12-16. PxSEL Register Description**

Bit	Field	Type	Reset	Description
7-0	PxSEL	RW	0h	Port x function selection 0b = I/O function is selected 1b = Peripheral module function is selected

## **Port Mapping Controller**

The port mapping controller allows a flexible mapping of digital functions to port pins. This chapter describes the port mapping controller.

Topic	Page
13.1 Port Mapping Controller Introduction .....	426
13.2 Port Mapping Controller Operation .....	426
13.3 Port Mapping Controller Registers .....	429

### 13.1 Port Mapping Controller Introduction

The port mapping controller allows the flexible and reconfigurable mapping of digital functions to port pins.

The port mapping controller features are:

- Configuration protected by write access key.
- Default mapping provided for each port pin (device-dependent, the device pinout in the device-specific data sheet).
- Mapping can be reconfigured during runtime.
- Each output signal can be mapped to several output pins.

### 13.2 Port Mapping Controller Operation

The port mapping is configured with user software. The setup is discussed in the following sections.

#### 13.2.1 Access

To enable write access to any of the port mapping controller registers, the correct key must be written into the PMAPKEYID register. The PMAPKEYID register always reads 096A5h. Writing the key 02D52h grants write access to all port mapping controller registers. Read access is always possible.

If an invalid key is written while write access is granted, any further write accesses are prevented. It is recommended that the application completes mapping configuration by writing an invalid key.

There is a timeout counter implemented that is incremented with each (assembler) instruction, and when it counts to 32, the write access is locked again. Any access to the port mapping controller registers resets the counter. Interrupts should be disabled during the configuration process or the application should take precautions that the execution of an interrupt service routine does not accidentally cause a permanent lock of the port mapping registers; for example, by using the reconfiguration capability (see [Section 13.2.2](#)).

The access status is reflected in the PMAPLOCK bit.

By default, the port mapping controller allows only one configuration after PUC. A second attempt to enable write access by writing the correct key is ignored, and the registers remain locked. A PUC is required to disable the permanent lock again. If it is necessary to reconfigure the mapping during runtime, the PMAPRECFG bit must be set during the first write access timeslot. If PMAPRECFG is cleared during later configuration sessions, no more configuration sessions are possible.

#### 13.2.2 Mapping

For each port pin, Px.y, on ports providing the mapping functionality, a mapping register, PxMAPy, is available. Setting this register to a certain value maps a module's input and output signals to the respective port pin Px.y. The port pin itself is switched from a general purpose I/O to the selected peripheral/secondary function by setting the corresponding PxSEL.y bit to 1. If the input or the output function of the module is used, it is typically defined by the setting the PxDIR.y bit. If PxDIR.y = 0, the pin is an input, if PxDIR.y = 1, the pin is an output. There are also peripherals (for example, the USCI module) that control the direction or even other functions of the pin (for example, open drain), and these options are documented in the mapping table.

With the port mapping functionality the output of a module can be mapped to multiple pins. Also the input of a module can receive inputs from multiple pins. When mapping multiple inputs onto one function, care needs to be taken because the input signals are logically ORed together without applying any priority; therefore, a logic one on any of the inputs results in a logic one at the module. If the PxSEL.y bit is 0, the corresponding input signal is a logic zero.

The mapping is device-dependent; see the device-specific data sheet for available functions and specific values. The use of mapping mnemonics to abstract the underlying PxMAPy values is recommended to allow simple portability between different devices. [Table 13-1](#) shows some examples for mapping mnemonics of some common peripherals.

All mappable port pins provide the function PM\_ANALOG (0FFh). Setting the port mapping register PxMAPy to PM\_ANALOG together with PxSEL.y = 1 disables the output driver and the input Schmitt-trigger, to prevent parasitic cross currents when applying analog signals.

**Table 13-1. Examples for Port Mapping Mnemonics and Functions**

PxMAPy Mnemonic	Input Pin Function With PxSEL.y = 1 and PxDIR.y = 0	Output Pin Function With PxSEL.y = 1 and PxDIR.y = 1
PM_NONE	None	DVSS
PM_ACLK	None	ACLK
PM_MCLK	None	MCLK
PM_SMCLK	None	SMCLK
PM_TA0CLK	Timer_A0 clock input	DVSS
PM_TA0CCR0A	Timer_A0 CCR0 capture input CCI0A	TA0 CCR0 compare output Out0
PM_TA0CCR1A	Timer_A0 CCR1 capture input CCI1A	TA0 CCR1 compare output Out1
PM_TA0CCR2A	Timer_A0 CCR2 capture input CCI2A	TA0 CCR2 compare output Out2
PM_TA0CCR3A	Timer_A0 CCR3 capture input CCI3A	TA0 CCR3 compare output Out3
PM_TA0CCR4A	Timer_A0 CCR4 capture input CCI4A	TA0 CCR4 compare output Out4
PM_TA1CLK	Timer_A1 clock input	DVSS
PM_TA1CCR0A	Timer_A1 CCR0 capture input CCI0A	TA1 CCR0 compare output Out0
PM_TA1CCR1A	Timer_A1 CCR1 capture input CCI1A	TA1 CCR1 compare output Out1
PM_TA1CCR2A	Timer_A1 CCR2 capture input CCI2A	TA1 CCR2 compare output Out2
PM_TBCLK	Timer_B clock input	DVSS
PM_TBOUTH	Timer_B outputs high impedance	DVSS
PM_TBCCR0A	Timer_B CCR0 capture input CCI0A	TB CCR0 compare output Out0 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR1A	Timer_B CCR1 capture input CCI1A	TB CCR1 compare output Out1 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR2A	Timer_B CCR2 capture input CCI2A	TB CCR2 compare output Out2 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR3A	Timer_B CCR3 capture input CCI3A	TB CCR3 compare output Out3 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR4A	Timer_B CCR4 capture input CCI4A	TB CCR4 compare output Out4 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR5A	Timer_B CCR5 capture input CCI5A	TB CCR5 compare output Out5 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR6A	Timer_B CCR6 capture input CCI6A	TB CCR6 compare output Out6 [direction controlled by Timer_B (TBOUTH)]
PM_UCA0RXD	USCI_A0 UART RXD (direction controlled by USCI - input)	
PM_UCA0SOMI	USCI_A0 SPI slave out master in (direction controlled by USCI)	
PM_UCA0TXD	USCI_A0 UART TXD (direction controlled by USCI - output)	
PM_UCA0SIMO	USCI_A0 SPI slave in master out (direction controlled by USCI)	
PM_UCA0CLK	USCI_A0 clock input/output (direction controlled by USCI)	
PM_UCA0STE	USCI_A0 SPI slave transmit enable (direction controlled by USCI)	
PM_UCB0SOMI	USCI_B0 SPI slave out master in (direction controlled by USCI)	
PM_UCB0SCL	USCI_B0 I2C clock (open drain and direction controlled by USCI)	
PM_UCB0SIMO	USCI_B0 SPI slave in master out (direction controlled by USCI)	
PM_UCB0SDA	USCI_B0 I2C data (open drain and direction controlled by USCI)	
PM_UCB0CLK	USCI_B0 clock input/output (direction controlled by USCI)	
PM_UCB0STE	USCI_B0 SPI slave transmit enable (direction controlled by USCI)	
PM_ANALOG	Disables the output driver and the input Schmitt-trigger to prevent parasitic cross currents when applying analog signals	



### 13.3 Port Mapping Controller Registers

The control register for the port mapping controller are listed in [Table 13-2](#). The mapping registers are listed in [Table 13-3](#). The mapping registers can also be accessed as words, as shown in [Table 13-4](#).

**Table 13-2. Port Mapping Control Registers**

Offset	Acronym	Register Name	Type	Reset
00h	PMAPKEYID	Port mapping key register	Read/write	Reset with PUC
02h	PMAPCTL	Port mapping control register	Read/write	Reset with PUC

**Table 13-3. Port Mapping Registers for Port Px – Byte Access**

Offset	Acronym	Register Name	Type	Reset
00h	PxMAP0	Port Px.0 mapping register	Read/write	Device dependent
01h	PxMAP1	Port Px.1 mapping register	Read/write	Device dependent
02h	PxMAP2	Port Px.2 mapping register	Read/write	Device dependent
03h	PxMAP3	Port Px.3 mapping register	Read/write	Device dependent
04h	PxMAP4	Port Px.4 mapping register	Read/write	Device dependent
05h	PxMAP5	Port Px.5 mapping register	Read/write	Device dependent
06h	PxMAP6	Port Px.6 mapping register	Read/write	Device dependent
07h	PxMAP7	Port Px.7 mapping register	Read/write	Device dependent

**Table 13-4. Port Mapping Registers for Port Px – Word Access**

Offset	Acronym	Register Name	Type	Reset
00h	PxMAP01	Port Px.0/Port Px.1 mapping register	Read/write	Device dependent
02h	PxMAP23	Port Px.2/Port Px.3 mapping register	Read/write	Device dependent
04h	PxMAP45	Port Px.4/Port Px.5 mapping register	Read/write	Device dependent
06h	PxMAP67	Port Px.6/Port Px.7 mapping register	Read/write	Device dependent

**13.3.1 PMAPKEYID Register**

Port Mapping Key Register

**Figure 13-1. PMAPKEYID Register**

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

**Table 13-5. PMAPKEYID Register Description**

Bit	Field	Type	Reset	Description
15-0	PMAPKEYx	RW	96A5h	Port write access key. Always reads 096A5h. Must be written 02D52h for write access to the port mapping registers.

**13.3.2 PMAPCTL Register**

Port Mapping Control Register

**Figure 13-2. PMAPCTL Register**

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

**Table 13-6. PMAPCTL Register Description**

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved. Always reads as 0.
1	PMAPRECFG	RW	0h	Port mapping reconfiguration control bit 0b = Configuration allowed only once 1b = Allow reconfiguration of port mapping
0	PMAPLOCKED	R	1h	Port mapping lock bit. Read only 0b = Access to mapping registers is granted 1b = Access to mapping registers is locked

**13.3.3 PxMAPy Register**

Port Px.y Mapping Register

**Figure 13-3. PxMAPy Register**

7	6	5	4	3	2	1	0
PMAPx							
rw-0 <sup>(1)</sup>							

<sup>(1)</sup> If not all bits are required to decode all provided functions, the unused bits are r0.**Table 13-7. PxMAPy Register Description**

Bit	Field	Type	Reset	Description
7-0	PMAPx	RW	0h	Selects secondary port function. Settings are device-dependent; see the device-specific data sheet.

## Cyclic Redundancy Check (CRC) Module

The cyclic redundancy check (CRC) module provides a signature for a given data sequence. This chapter describes the operation and use of the CRC module.

---

**NOTE:** The CRC module on the MSP430F543x and MSP430F541x non-A versions does not support the bit-wise reverse feature described in this module description. Registers CRCDIRB and CRCRESR, along with their respective functionality, are not available.

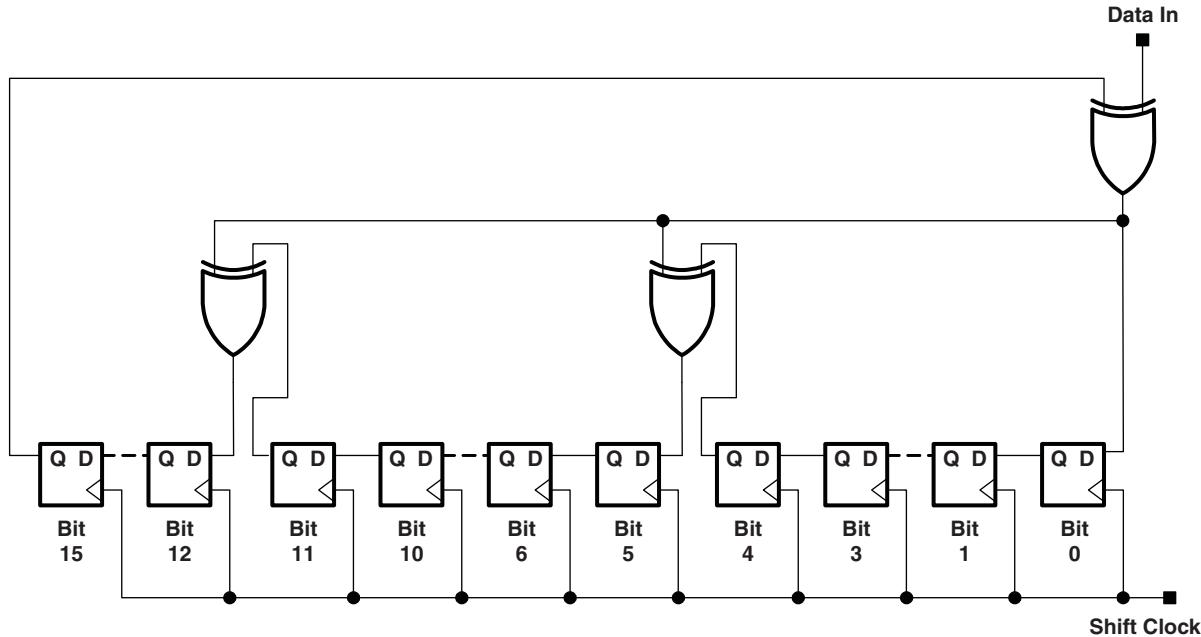
---

Topic	Page
<b>14.1 Cyclic Redundancy Check (CRC) Module Introduction .....</b>	<b>432</b>
<b>14.2 CRC Standard and Bit Order .....</b>	<b>432</b>
<b>14.3 CRC Checksum Generation .....</b>	<b>433</b>
<b>14.4 CRC Registers .....</b>	<b>436</b>

## 14.1 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 14-1). The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial (see Equation 10).

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



**Figure 14-1. 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.

## 14.2 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 14-1, the bit convention shown is as given in the original standards i.e. bit 0 is the MSB. The fact that bit 0 is treated for some as LSB, and for others as MSB, continues to cause confusion. The CRC16 module therefore provides a bit reversed register pair for CRC16 operations to support both conventions.

## 14.3 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 on how the result of a signature operation can be calculated. The calculated signature, which is computed by an external tool, is called checksum in the following text. The checksum is stored in the product's memory and is used to check the correctness of the CRC operation result.

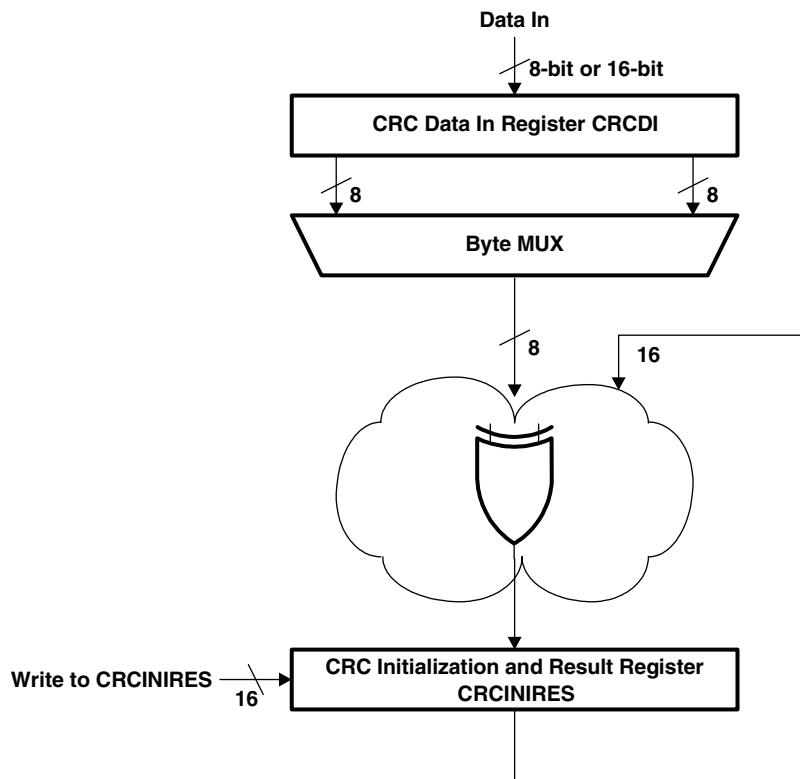
### 14.3.1 CRC Implementation

To allow parallel processing of the CRC, the linear feedback shift register (LFSR) functionality is implemented with an XOR tree. This implementation shows the identical behavior as the LFSR approach after 8 bits of data are shifted in when the LSB is 'shifted' in first. The generation of a signature calculation has to be started by writing a seed to the CRCINIRES register to initialize the register. Software or hardware (for example, DMA) can transfer data to the CRCDI or CRCDIRB register (for example, from memory). The value in CRCDI or CRCDIRB is then included into the signature, and the result is available in the signature result registers at the next read access (CRCINIRES and CRCRESR). The signature can be generated using word or byte data.

If a word data is processed, the lower byte at the even address is used at the first clock (MCLK) cycle. During the second clock cycle, the higher byte is processed. Thus, it takes two clock cycles to process word data, while it takes only one clock (MCLK) cycle to process byte data.

Data bytes written to CRCDIRB in word mode or the data byte in byte mode are bit-wise reversed before the CRC engine adds them to the signature. The bits among each byte are reversed. Data bytes written to CRCDI in word mode or the data byte in byte mode are not bit reversed before use by the CRC engine.

If the Check Sum itself (with reversed bit order) is included into the CRC operation (as data written to CRCDI or CRCDIRB), the result in the CRCINIRES and CRCRESR registers must be zero.



**Figure 14-2. Implementation of CRC-CCITT Using the CRCDI and CRCINIRES Registers**

### 14.3.2 Assembler Examples

#### 14.3.2.1 General Assembler Example

This example demonstrates the operation of the on-chip CRC:

```

...
PUSH  R4          ; Save registers
PUSH  R5
MOV   #StartAddress,R4  ; StartAddress < EndAddress
MOV   #EndAddress,R5
MOV   &INIT, &CRCINIRES ; INIT to CRCINIRES
L1   MOV   @R4+,&CRCDI  ; Item to Data In register
      CMP   R5,R4        ; End address reached?
      JLO   L1            ; No
      MOV   &Check_Sum,&CRCDI ; Yes, Include checksum
      TST   &CRCINIRES    ; Result = 0?
      JNZ   CRC_ERROR     ; No, CRCRES <> 0: error
      ...                ; Yes, CRCRES=0:
                          ; information ok.
      POP   R5            ; Restore registers
      POP   R4

```

#### **14.3.2.2 Reference Data Sequence**

The details of the implemented CRC algorithm is shown by the following data sequences using word or byte accesses and the CRC data-in as well as the CRC data-in reverse byte registers:

```

...
mov    #0FFFh,&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    #0FFFh,&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    #0FFFh,&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    #0FFFh,&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

```

## 14.4 CRC Registers

The CRC module registers are listed in [Table 14-1](#). The base address can be found in the device-specific data sheet. The address offset is given in [Table 14-1](#).

---

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

---

**Table 14-1. CRC Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
0000h	CRCIDI	CRC Data In	Read/write	Word	0000h	<a href="#">Section 14.4.1</a>
0000h	CRCIDI_L		Read/write	Byte	00h	
0001h	CRCIDI_H		Read/write	Byte	00h	
0002h	CRCDIRB	CRC Data In Reverse Byte <sup>(1)</sup>	Read/write	Word	0000h	<a href="#">Section 14.4.2</a>
0002h	CRCDIRB_L		Read/write	Byte	00h	
0003h	CRCDIRB_H		Read/write	Byte	00h	
0004h	CRCINIRES	CRC Initialization and Result	Read/write	Word	FFFFh	<a href="#">Section 14.4.3</a>
0004h	CRCINIRES_L		Read/write	Byte	FFh	
0005h	CRCINIRES_H		Read/write	Byte	FFh	
0006h	CRCRESR	CRC Result Reverse <sup>(1)</sup>	Read only	Word	FFFFh	<a href="#">Section 14.4.4</a>
0006h	CRCRESR_L		Read/write	Byte	FFh	
0007h	CRCRESR_H		Read/write	Byte	FFh	

<sup>(1)</sup> Not available on MSP430F543x and MSP430F541x non-A versions.

#### **14.4.1 CRCDI Register**

CRC Data In Register

**Figure 14-3. 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 14-2. 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.

#### **14.4.2 CRCDIRB Register**

CRC Data In Reverse Register

**Figure 14-4. 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 14-3. 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.

**14.4.3 CRCINIRES Register**

CRC Initialization and Result Register

**Figure 14-5. 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 14-4. 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.

**14.4.4 CRCRESR Register**

CRC Reverse Result Register

**Figure 14-6. 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 14-5. 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 reversed (for example, CRCINIRES[15] = CRCRESR[0]) compared to the order of bits in the CRCINIRES register (see example code).

## **AES Accelerator**

The AES accelerator module performs AES128 encryption or decryption in hardware. This chapter describes the AES accelerator.

Topic	Page
15.1 AES Accelerator Introduction .....	440
15.2 AES Accelerator Operation .....	441
15.3 AES_ACCEL Registers .....	446

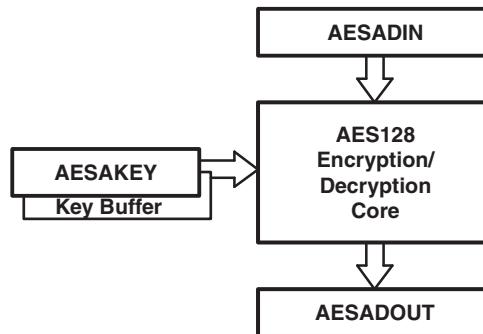
## 15.1 AES Accelerator Introduction

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

The AES accelerator features are:

- Encryption and decryption according to AES FIPS PUB 197 with 128-bit key
- On-the-fly key expansion for encryption and decryption
- Off-line key generation for decryption
- Byte and word access to key, input, and output data
- AES ready interrupt flag

The AES accelerator block diagram is shown in [Figure 15-1](#).

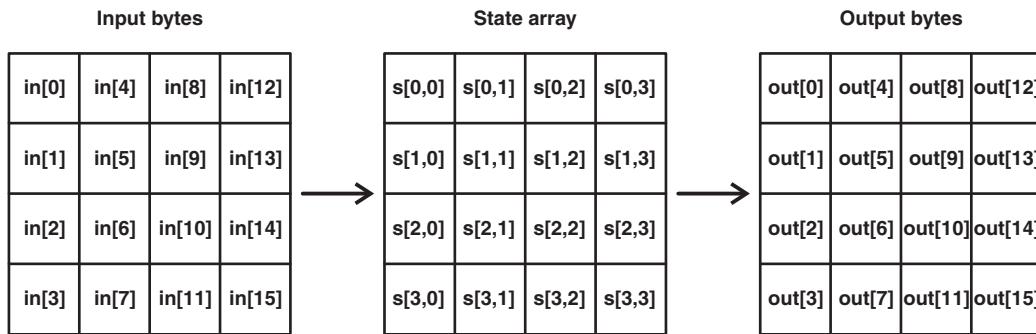


**Figure 15-1. AES Accelerator Block Diagram**

## 15.2 AES Accelerator Operation

The AES accelerator is configured with user software. The setup and operation is discussed in the following sections.

Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the State. For AES-128, the State consists of four rows of bytes, each containing four bytes. The input is assigned to the State array as illustrated in [Figure 15-2](#), with in[0] being the first data byte written into the AES accelerator data input register, AESADIN. The encrypt or decrypt operations are then conducted on the State array, after which its final values can be read from the output with out[0] being the first data byte read from the AES accelerator data output register, AESADOUT.



**Figure 15-2. AES State Array Input and Output**

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

---

**NOTE: Access Restrictions**

While the AES accelerator is busy (AESBUSY = 1), AESADOUT always reads as zero, the AESDOUTCNTx counter, the AESDOUTRD flag, and the AESDINWR flag are reset, any attempt to change AESOPx, AESDINWR, or AESKEYWR is ignored, and writing to AESAKEY or AESADIN aborts the current operation, the complete module is reset (except for AESRDYIE and AESOPx), and the AES error flag AESERRFG is set.

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

Writing data into AESADIN influences the content of the corresponding output data; for example, writing in[0] alters out[0], writing in[1] alters out[1], etc., but interleaved operation is possible; for example, first reading out[0], then writing in[0], and continuing with reading out[1], writing in[1], etc.

---

**NOTE:** When using a code debugger, the AES module does not stop its operation when program code is halted or single stepped.

---

### 15.2.1 Encryption

Figure 15-3 shows the encryption process with the cipher being a series of transformations that converts the plaintext written into the AESADIN register to a ciphertext that can be read from the AESADOUT register using the cipher key provided via the AESAKEY register.

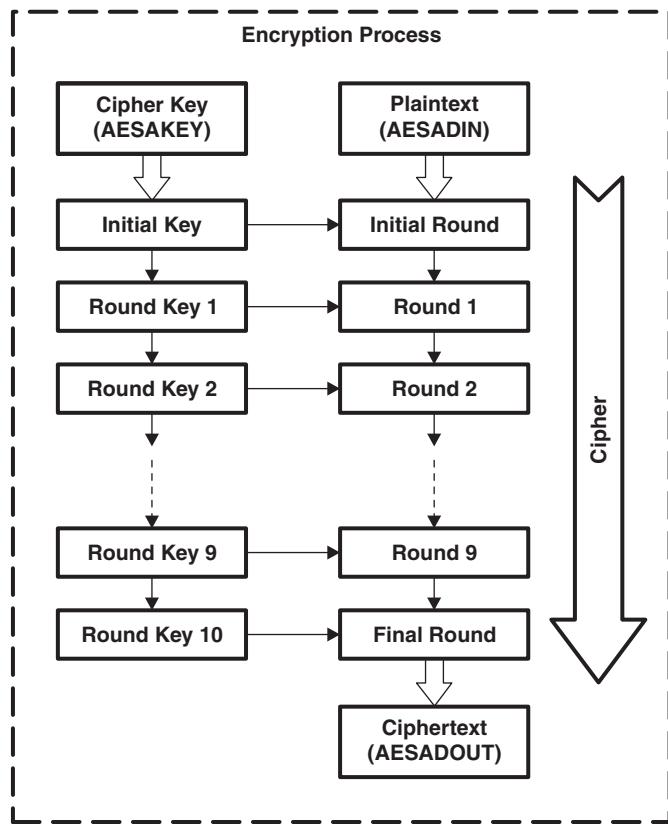


Figure 15-3. AES-128 Encryption Process

The steps to perform encryption are:

1. Set AESOPx = 00 to select encryption. Changing the AESOPx bits clears the AESKEYWR flag, and a new key must be loaded in the next step.
2. Load the 128-bit key into AESAKEY or set the AESKEYWR flag by software, if the key from a previous operation should be used. When all 16 bytes are written, the AESKEYWR flag indicates completion. If a key was loaded previously without changing AESOPx, the AESKEYWR flag is cleared with the first write access to AESAKEY. Loading the key must be completed before the next step is performed.
3. Load 128-bit data into AESADIN, or set the AESDINWR flag by software if the output data from a previous operation should be encrypted. When all 16 bytes are written, the AESDINWR flag indicates completion. The module starts encrypting the presented data when AESDINWR = 1.
4. While the AES module is performing encryption, the AESBUSY bit is 1. The encryption takes 167 MCLK clock cycles. After its completion, the AESRDYIFG is set, and the result can be read from AESADOUT. When all 16 bytes are read, the AESDOUTRD flag indicates completion.

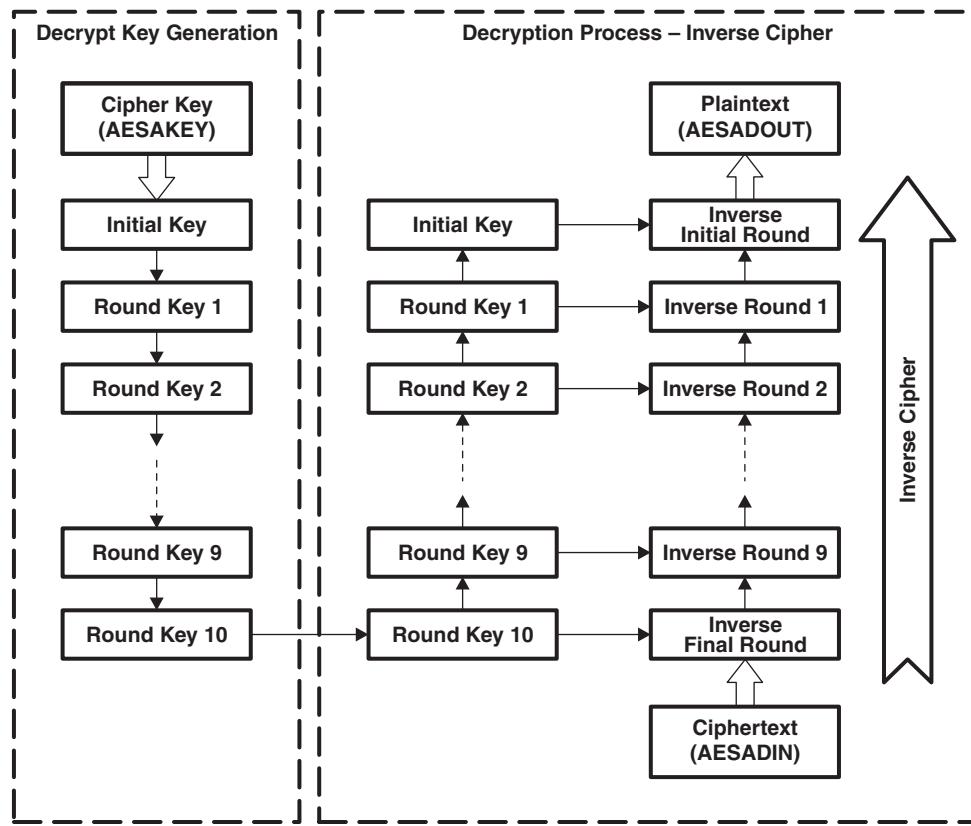
The AESRDYIFG flag is cleared when reading AESADOUT or writing to AESAKEY or AESADIN.

5. If additional data should be encrypted with the same key loaded in step 2, new data can be written into AESADIN after the results of the operation on the previous data were read from AESADOUT. When an additional 16 data bytes are written, the module automatically starts the encryption using the key loaded in step 2.

When using the output feedback (OFB) cipher block chaining mode, setting the AESDINWR flag is sufficient to trigger the next encryption, and the module starts the encryption automatically using the output data from the previous encryption as input data.

### 15.2.2 Decryption

Figure 15-4 shows the decryption process with the inverse cipher being a series of transformations that convert the ciphertext written into the AESADIN register to a plaintext that can be read from the AESADOUT register using the cipher key provided via the AESAKEY register.



**Figure 15-4. AES-128 Decryption Process using AESOPx = 01**

The steps to perform decryption are:

1. Set AESOPx = 01 to select decryption using the same key used for encryption. Set AESOPx = 11 if the first-round key required for decryption (round key 10) is already generated and is loaded in step 2. Changing the AESOPx bits clears the AESKEYWR flag, and a new key must be loaded in step 2.
  2. Load the 128-bit key into AESAKEY, or set the AESKEYWR flag by software, if the key from a previous operation should be used. When all 16 bytes are written, the AESKEYWR flag indicates completion.
- If a key was loaded previously without changing AESOPx, the AESKEYWR flag is cleared with the first write access to AESAKEY. Loading the key must be completed before the next step is performed.
3. Load 128-bit data into AESADIN or set the AESDINWR flag by software if the output data from a previous operation should be decrypted. When all 16 bytes are written, the AESDINWR flag indicates completion. The module starts decrypting the presented data as soon as AESDINWR = 1.
  4. While the AES module is performing decryption, the AESBUSY bit is 1. The decryption takes 214 MCLK clock cycles with AESOPx = 01 and 167 MCLK clock cycles with AESOPx = 11. After its completion, the AESRDYIFG is set, and the result can be read from AESADOUT. When all 16 bytes are read the AESDOUTRD flag indicates completion.

The AESRDYIFG flag is cleared when reading AESADOUT or writing to AESAKEY or AESADIN.

5. If additional data should be decrypted with the same key loaded in step 2, new data can be written into AESADIN after the results of the operation on the previous data were read from AESADOUT. When additional 16 data bytes are written, the module automatically starts the decryption using the key loaded in step 2.

### 15.2.3 Decryption Key Generation

Figure 15-5 shows the decryption process with a pregenerated decryption key. In this case, the decryption key is calculated first with AESOPx = 10, then the precalculated key can be used together with the decryption operation AESOPx = 11.

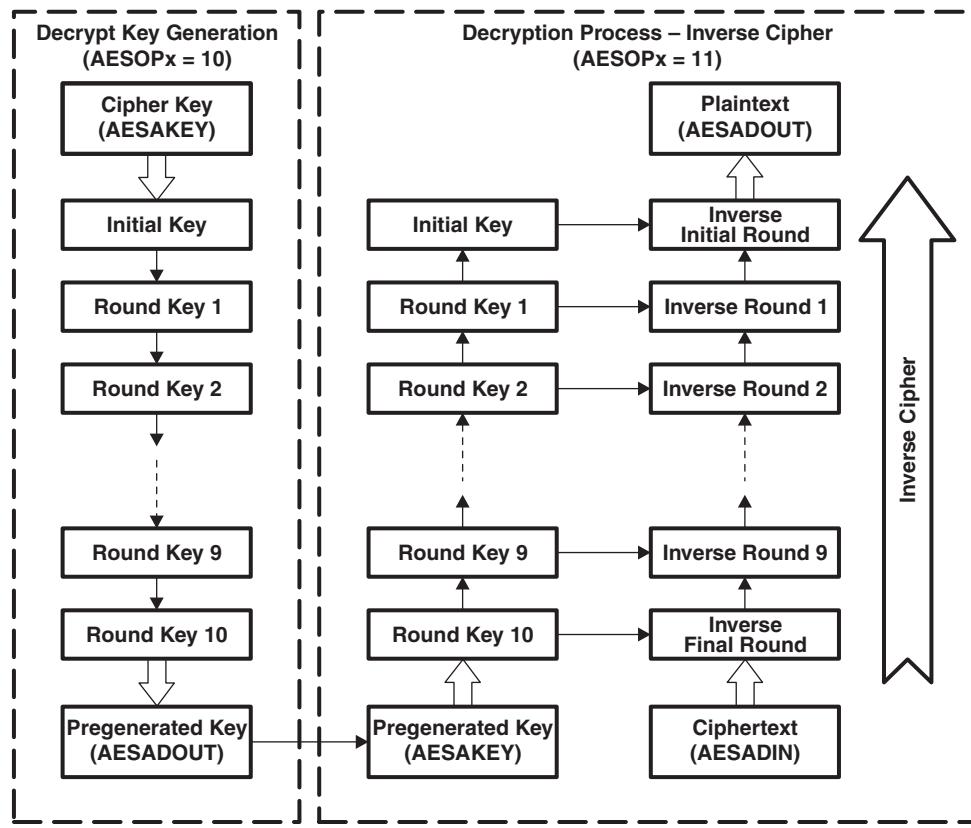


Figure 15-5. AES-128 Decryption Process using AESOPx = 10 and 11

To generate the decryption key independent from the actual decryption, the following steps are required:

1. Set AESOPx = 10 to select decryption key generation. Changing the AESOPx bits clears the AESKEYWR flag, and a new key must be loaded in step 2.
2. Load the 128-bit key into AESAKEY, or set the AESKEYWR flag by software if the key from a previous operation should be used. When all 16 bytes are written, the AESKEYWR flag indicates completion. The generation of the first round key required for decryption is started immediately.
3. While the AES module is performing the key generation, the AESBUSY bit is 1. It takes 52 CPU clock cycles to complete the key generation. After its completion, the AESRDYIFG is set, and the result can be read from AESADOUT. When all 16 bytes are read, the AESDOUTRD flag indicates completion. The AESRDYIFG flag is cleared when reading AESADOUT or writing to AESAKEY or AESADIN.
4. If data should be decrypted with the generated key, AESOPx must be set to 11. Then the generated key must be loaded or, if it was just generated with AESOPx = 10, it is sufficient to set the AESKEYWR flag by software to indicate that the key is already valid. Afterward, the steps described in Section 15.2.2 to load the data, etc., must be followed.

#### 15.2.4 Using the AES Accelerator With Low-Power Modes

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

#### 15.2.5 AES Accelerator Interrupts

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

#### 15.2.6 Implementing Block Cipher Modes

All block cipher modes can be implemented using the AES accelerator together with software. A separate application report describes the block cipher modes together with their implementation in software.

## 15.3 AES\_ACCEL Registers

The AES Accelerator registers are listed in [Table 15-1](#).

**Table 15-1. AES\_ACCEL Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
000h	AESACTL0	AES accelerator control register 0	Read/write	Word	00h	<a href="#">Section 15.3.1</a>
002h	AESACTL1	AES accelerator control register 1	Read/write	Word	00h	<a href="#">Section 15.3.2</a>
004h	AESASTAT	AES accelerator status register	Read only	Word	00h	<a href="#">Section 15.3.3</a>
006h	AESAKEY	AES accelerator key register	Read/write	Word	00h	<a href="#">Section 15.3.4</a>
008h	AESADIN	AES accelerator data in register	Read/write	Word	00h	<a href="#">Section 15.3.5</a>
00Ah	AESADOUT	AES accelerator data out register	Read/write	Word	00h	<a href="#">Section 15.3.6</a>
00Ch	AESAXDIN	AES accelerator XORed data in register	Read/write	Word	00h	<a href="#">Section 15.3.7</a>
00Eh	AESAXIN	AES accelerator XORed data in register (no trigger)	Read/write	Word	00h	<a href="#">Section 15.3.8</a>

### 15.3.1 AESACTL0 Register

AES accelerator control register 0

AESACTL0 is shown in [Figure 15-6](#) and described in [Table 15-2](#).

**Figure 15-6. AESACTL0 Register**

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

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

**Table 15-2. AESACTL0 Register Description**

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

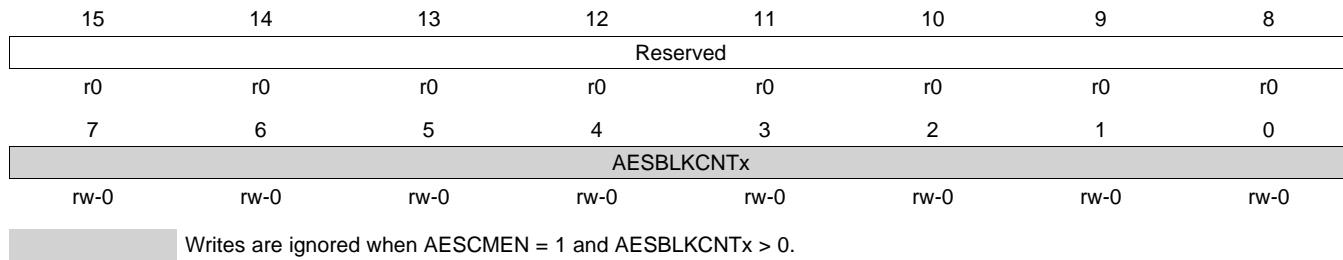
**Table 15-2. AESACTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
3-2	AESKLx	RW	0h	AES key length. These bits define which of the 3 AES standards is performed. The AESKLx bits are not reset by AECSRST = 1. Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0. 00 = AES128. The keysize is 128 bit. 01 = AES192. The keysize is 192 bit. 10 = AES256. The keysize is 256 bit. 11 = Reserved
1-0	AESOPx	RW	0h	AES operation. The AESOPx bits are not reset by AECSRST = 1. Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0. 00 = Encryption 01 = Decryption. The provided key is the same key used for encryption. 10 = Generate first round key required for decryption. 11 = Decryption. The provided key is the first round key required for decryption.

### 15.3.2 AESACTL1 Register

AES Accelerator Control Register 1

AESACTL1 is shown in [Figure 15-7](#) and described in [Table 15-3](#).

**Figure 15-7. AESACTL1 Register****Table 15-3. AESACTL1 Register Description**

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

### 15.3.3 AESASTAT Register

AES Accelerator Status Register

AESASTAT is shown in [Figure 15-8](#) and described in [Table 15-4](#).

**Figure 15-8. AESASTAT Register**

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

**Table 15-4. AESASTAT Register Description**

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

**15.3.4 AESAKEY Register**

AES Accelerator Key Register

AESAKEY is shown in [Figure 15-9](#) and described in [Table 15-5](#).**Figure 15-9. AESAKEY Register**

15	14	13	12	11	10	9	8
AESKEY1x (Key Byte n+1)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESKEY0x (Key Byte n)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

**Table 15-5. AESAKEY Register Description**

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

### 15.3.5 AESADIN Register

AES Accelerator Data In Register

AESADIN is shown in [Figure 15-10](#) and described in [Table 15-6](#).

**Figure 15-10. AESADIN Register**

15	14	13	12	11	10	9	8
AESDIN1x (DIN Byte n+1)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESDIN0x (DIN Byte n)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

**Table 15-6. AESADIN Register Description**

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

### 15.3.6 AESADOUT Register

AES Accelerator Data Out Register

AESADOUT is shown in [Figure 15-11](#) and described in [Table 15-7](#).

**Figure 15-11. AESADOUT Register**

15	14	13	12	11	10	9	8
AESDOUT1x (DOUT Byte n+1)							
7	6	5	4	3	2	1	0
AESDOUT0x (DOUT Byte n)							

**Table 15-7. AESADOUT Register Description**

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

### 15.3.7 AESAXDIN Register

AES accelerator XORed data in register

AESAXDIN is shown in [Figure 15-12](#) and described in [Table 15-8](#).

**Figure 15-12. AESAXDIN Register**

15	14	13	12	11	10	9	8
AESXDIN1x (XDIN Byte n+1)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESXDIN0x (XDIN Byte n)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

**Table 15-8. AESAXDIN Register Description**

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

### 15.3.8 AESAXIN Register

AES accelerator XORed data in register (no trigger)

AESAXIN is shown in [Figure 15-13](#) and described in [Table 15-9](#).

**Figure 15-13. AESAXIN Register**

15	14	13	12	11	10	9	8
AESXIN1x (DIN Byte n+1)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESXIN0x (DIN Byte n)							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

**Table 15-9. AESAXIN Register Description**

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

## **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
16.1 WDT_A Introduction .....	454
16.2 WDT_A Operation .....	456
16.3 WDT_A Registers .....	458

## 16.1 WDT\_A Introduction

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

Features of the watchdog timer module include:

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

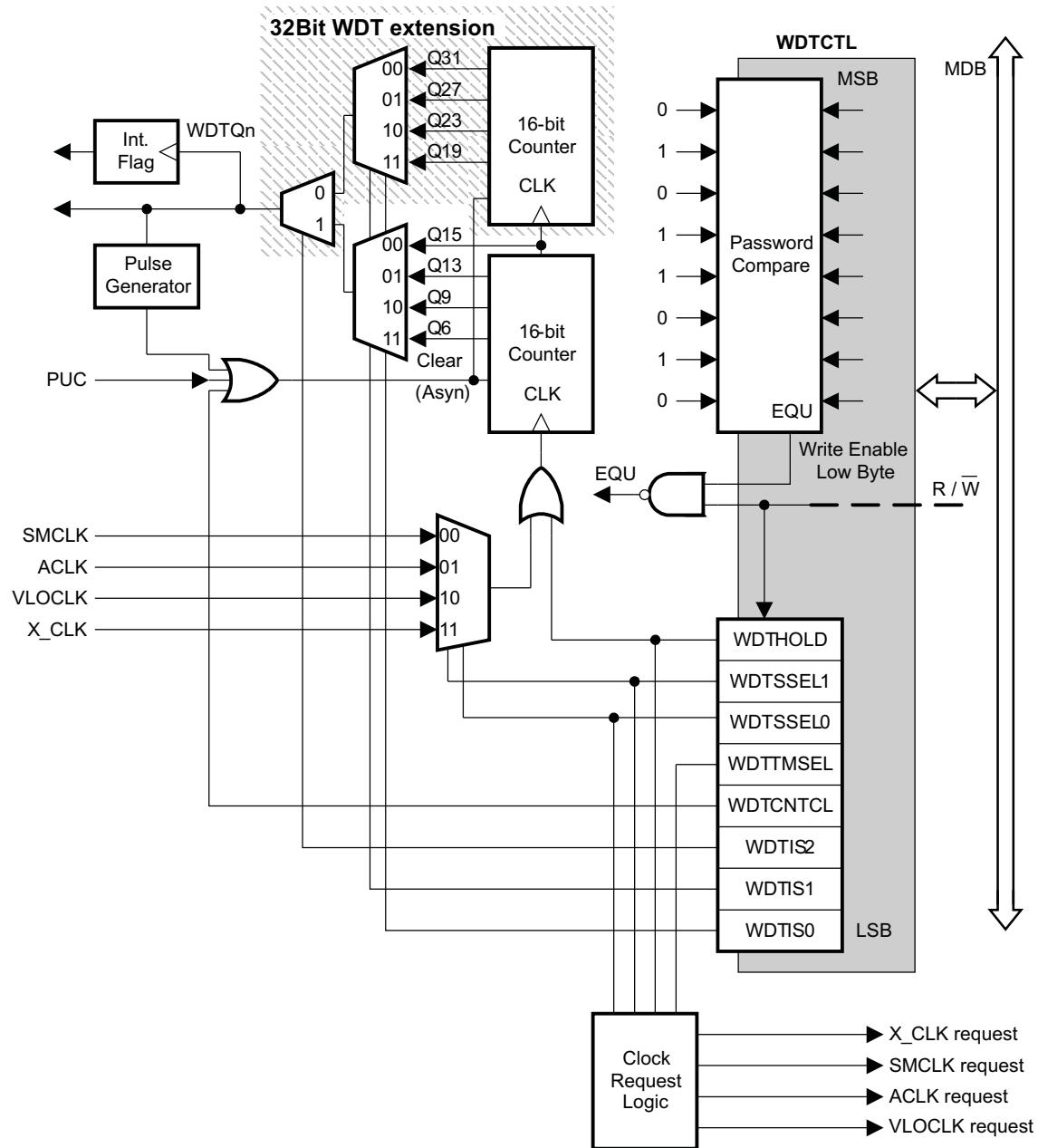
The watchdog timer block diagram is shown in [Figure 16-1](#).

---

**NOTE: Watchdog timer powers up active.**

After a PUC, the WDT\_A module is automatically configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK. The user must setup or halt the WDT\_A prior to the expiration of the initial reset interval.

---



**Figure 16-1. Watchdog Timer Block Diagram**

## 16.2 WDT\_A Operation

The watchdog timer module can be configured as either a watchdog or interval timer with the WDTCTL register. WDTCTL is a 16-bit password-protected read/write register. Any read or write access must use word instructions and write accesses must include the write password 05Ah in the upper byte. Any write to WDTCTL with any value other than 05Ah in the upper byte is a password violation and triggers a PUC system reset, regardless of timer mode. Any read of WDTCTL reads 069h in the upper byte. Byte reads on WDTCTL high or low part result in the value of the low byte. Writing byte wide to upper or lower parts of WDTCTL results in a PUC.

### 16.2.1 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 WDTSEL bits. The timer interval is selected with the WDTIS bits.

### 16.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK. The user must setup, halt, or clear the watchdog timer prior to the expiration of the initial reset interval or another PUC is generated. When the watchdog timer is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC. A PUC resets the watchdog timer to its default condition.

### 16.2.3 Interval Timer Mode

Setting the WDTMSEL 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 it 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.

---

### 16.2.4 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 will self clear upon a watchdog timeout event. The SYSRSTIV can be read to determine if the reset was caused by a watchdog timeout event.

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.

### 16.2.5 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 that 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.

### 16.2.6 Operation in Low-Power Modes

The devices have several low-power modes. Different clock signals are available in different low-power modes. The requirements of the application and the type of clocking that is used determine how the WDT\_A should be configured. For example, the WDT\_A should not be configured in watchdog mode with a clock source that is originally sourced from DCO, XT1 in high-frequency mode, or XT2 via SMCLK or ACLK, if the user wants to use low-power mode 3. In this case, SMCLK or ACLK would remain enabled, increasing the current consumption of LPM3. When the watchdog timer is not required, the WDTHOLD bit can be used to hold the WDTCNT, reducing power consumption.

### 16.2.7 Software Examples

Any write operation to WDTCTL must be a word operation with 05Ah (WDTPW) in the upper byte:

```
; Periodically clear an active watchdog
MOV #WDTPW+WDTIS2+WDTIS1+WDTCNTCL,&WDTCTL
;
; Change watchdog timer interval
MOV #WDTPW+WDTCNTCL+SSEL,&WDTCTL
;
; Stop the watchdog
MOV #WDTPW+WDTHOLD,&WDTCTL
;
; Change WDT to interval timer mode, clock/8192 interval
MOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS2+WDTIS0,&WDTCTL
```

## 16.3 WDT\_A Registers

The watchdog timer module registers are listed in [Table 16-1](#). The base address for the watchdog timer module registers and special function registers (SFRs) can be found in device-specific data sheets. The address offset is given in [Table 16-1](#).

---

**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 16-1. WDT\_A Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Ch	WDTCTL	Watchdog Timer Control	Read/write	Word	6904h	<a href="#">Section 16.3.1</a>
0Ch	WDTCTL_L		Read/write	Byte	04h	
0Dh	WDTCTL_H		Read/write	Byte	69h	

### 16.3.1 WDTCTL Register

Watchdog Timer Control Register

**Figure 16-2. WDTCTL Register**

15	14	13	12	11	10	9	8
WDTPW							
7	6	5	4	3	2	1	0
WDTHOLD	WDTSEL	WDTMSEL	WDTMSEL	WDTCNTCL	WDTIS		

rw-0            rw-0            rw-0            rw-0            r0(w)            rw-1            rw-0            rw-0

**Table 16-2. WDTCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	WDTPW	RW	69h	Watchdog timer password. Always read as 069h. Must be written as 5Ah; if any other value is written, 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	WDTSEL	RW	0h	Watchdog timer clock source select 00b = SMCLK 01b = ACLK 10b = VLOCLK 11b = X_CLK; VLOCLK in devices that do not support X_CLK
4	WDTMSEL	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 and/or generate a PUC. 000b = Watchdog clock source /(2^31) (18h:12m:16s at 32.768 kHz) 001b = Watchdog clock source /(2^27) (01h:08m:16s at 32.768 kHz) 010b = Watchdog clock source /(2^23) (00h:04m:16s at 32.768 kHz) 011b = Watchdog clock source /(2^19) (00h:00m:16s 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
17.1 Timer_A Introduction .....	461
17.2 Timer_A Operation .....	463
17.3 Timer_A Registers .....	475

## 17.1 Timer\_A Introduction

Timer\_A is a 16-bit timer/counter with up to seven capture/compare registers. Timer\_A can support multiple capture/comparisons, PWM outputs, and interval timing. Timer\_A also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer\_A features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with pulse width modulation (PWM) capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer\_A interrupts

The block diagram of Timer\_A is shown in [Figure 17-1](#).

---

**NOTE: Use of the word count**

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

---

**NOTE: Nomenclature**

There may be multiple instantiations of Timer\_A on a given device. The prefix TAx is used, where x is greater than or 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.

---

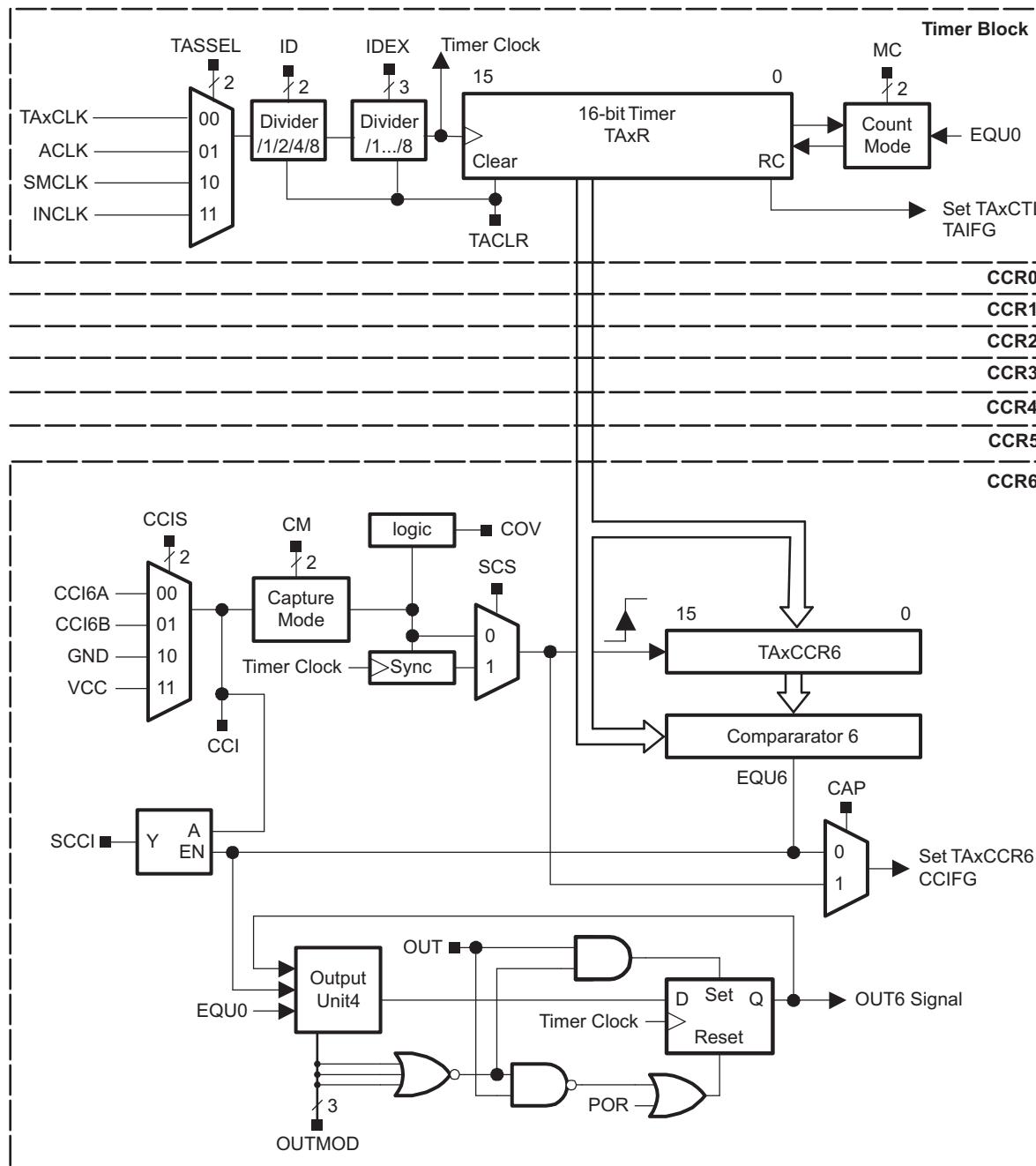


Figure 17-1. Timer\_A Block Diagram

## 17.2 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.

### 17.2.1 16-Bit Timer Counter

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

TAxR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider and count direction for up/down mode.

---

**NOTE: Modifying Timer\_A registers**

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable, interrupt flag, and TACLR) to avoid errant operating conditions.

When the timer clock is asynchronous to the CPU clock, any read from TAxR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TAxR takes effect immediately.

---

#### 17.2.1.1 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TAxClock or INCLK. The clock source is selected with the TASSEL bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the ID bits. The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the TAIDEX bits. The timer clock divider logic is reset when TACLR is set.

---

**NOTE: Timer\_A dividers**

After programming ID or TAIDEX bits, set the TACLR bit. This clears the contents of TAxR and resets the clock divider logic to a defined state. The clock dividers are implemented as down counters. Therefore, when the TACLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer\_A clock source selected with the TASSEL bits and continues clocking at the divider settings set by the ID and TAIDEX bits.

---

### 17.2.2 Starting the Timer

The timer may be started or restarted in the following ways:

- The timer counts when MC > { 0 } and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by writing 0 to TAxCCR0. The timer may then be restarted by writing a nonzero value to TAxCCR0. In this scenario, the timer starts incrementing in the up direction from zero.

### 17.2.3 Timer Mode Control

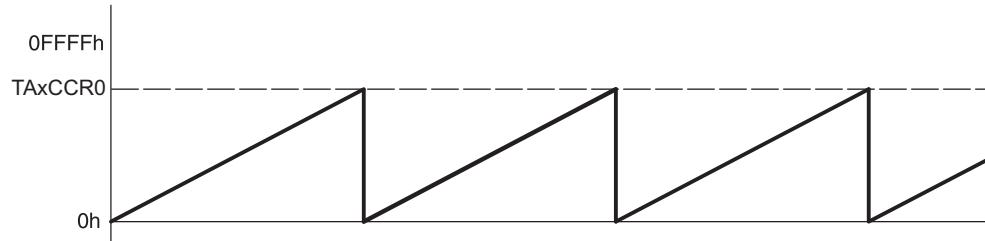
The timer has four modes of operation: stop, up, continuous, and up/down (see [Table 17-1](#)). The operating mode is selected with the MC bits.

**Table 17-1. 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.

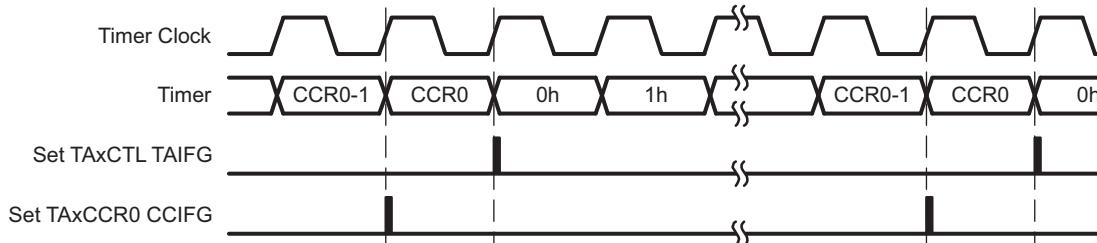
#### 17.2.3.1 Up Mode

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 17-2](#)). 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 17-2. 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 17-3](#) shows the flag set cycle.



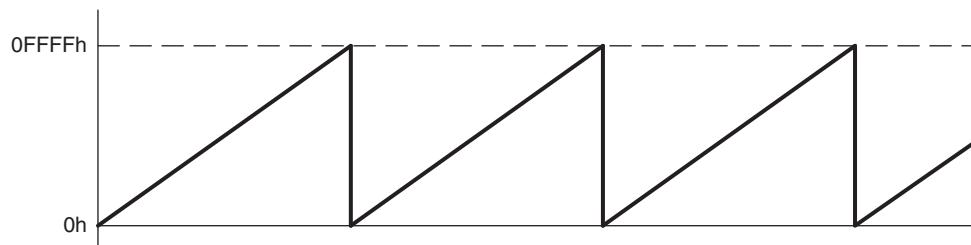
**Figure 17-3. Up Mode Flag Setting**

#### Changing Period Register TAxCCR0

When changing TAxCCR0 while the timer is running, if the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

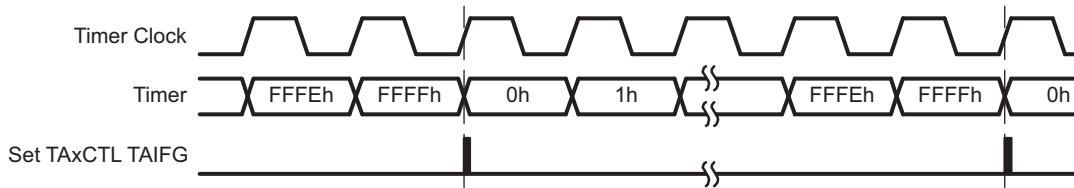
### 17.2.3.2 Continuous Mode

In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in [Figure 17-4](#). The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.



**Figure 17-4. Continuous Mode**

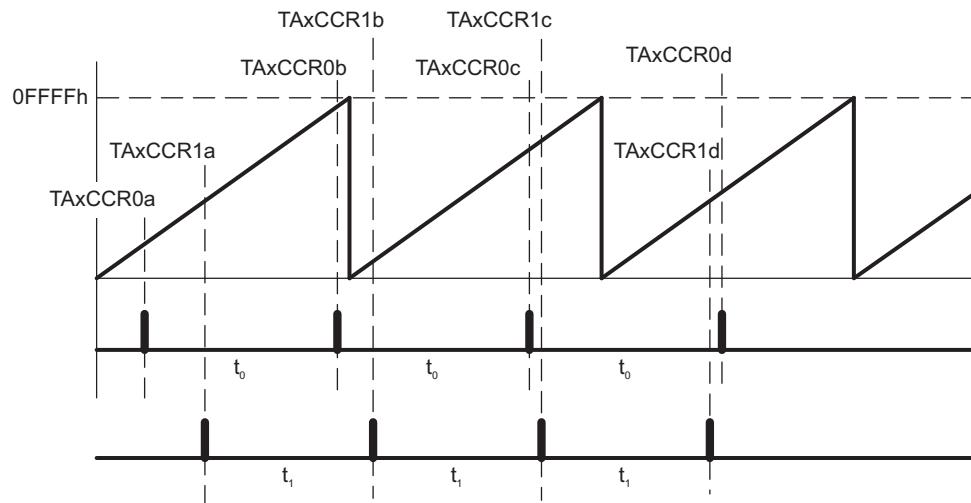
The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. [Figure 17-5](#) shows the flag set cycle.



**Figure 17-5. Continuous Mode Flag Setting**

### 17.2.3.3 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 17-6](#) 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.

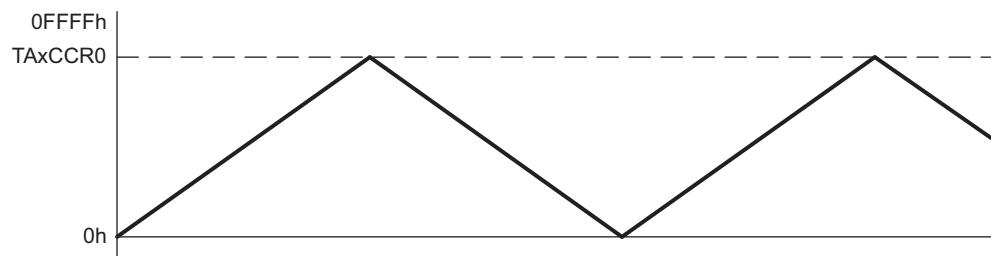


**Figure 17-6. Continuous Mode Time Intervals**

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

#### 17.2.3.4 Up/Down Mode

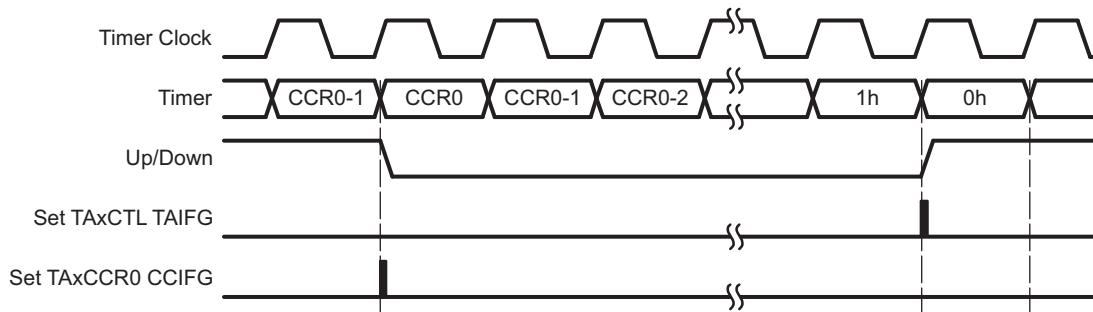
The up/down mode is used if the timer period must be different from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare register TAxCCR0 and back down to zero (see [Figure 17-7](#)). The period is twice the value in TAxCCR0.



**Figure 17-7. Up/Down Mode**

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

In up/down mode, the TAxCCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by one-half the timer period. The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* from TAxCCR0-1 to TAxCCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. [Figure 17-8](#) shows the flag set cycle.



**Figure 17-8. Up/Down Mode Flag Setting**

#### Changing Period Register TAxCCR0

When changing TAxCCR0 while the timer is running and counting in the down direction, the timer continues its descent until it reaches zero. The new period takes effect after the counter counts down to zero.

When the timer is counting in the up direction, and the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period before counting down.

When the timer is counting in the up direction and the new period is less than the current count value, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

### 17.2.3.5 Use of Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see section *Timer\_A Output Unit*). For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the example shown in [Figure 17-9](#), the  $t_{\text{dead}}$  is:

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

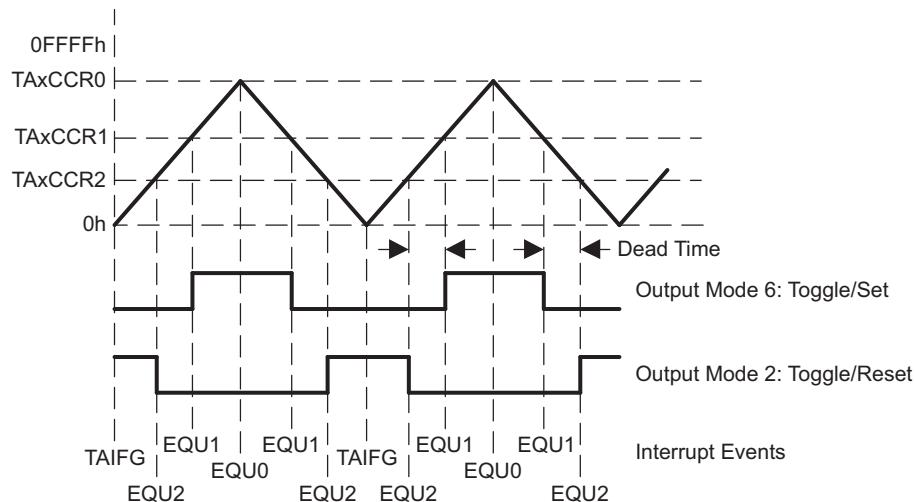
Where:

$t_{\text{dead}}$  = Time during which both outputs need to be inactive

$t_{\text{timer}}$  = Cycle time of the timer clock

TAxCCRn = Content of capture/compare register n

The TAxCCRn registers are not buffered. They update immediately when written to. Therefore, any required dead time is not maintained automatically.



**Figure 17-9. Output Unit in Up/Down Mode**

### 17.2.4 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.

#### 17.2.4.1 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 CCIA and CCIB are connected to external pins or internal signals and are selected with the CCIS bits. The CM bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture occurs:

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

The input signal level can be read at any time via the CCI bit. Devices may have different signals connected to CCIA and CCIB. See the device-specific data sheet for the connections of these signals.

The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit synchronizes the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended (see [Figure 17-10](#)).

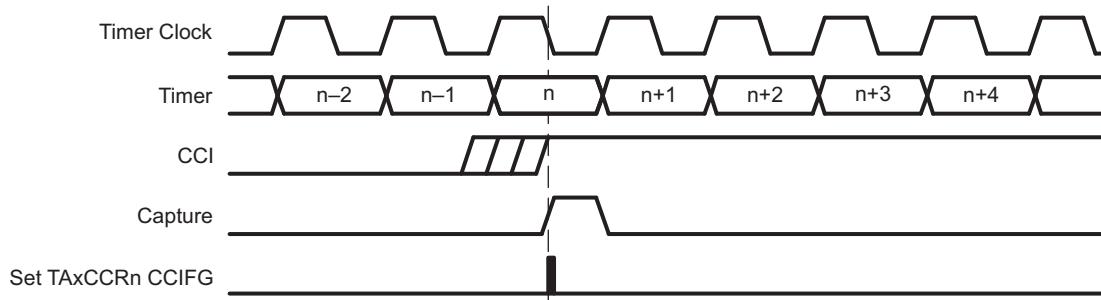


Figure 17-10. Capture Signal (SCS = 1)

**NOTE: Changing Capture Inputs**

Changing capture inputs while in capture mode may cause unintended capture events. To avoid this scenario, capture inputs should only be changed when capture mode is disabled ( $CM = \{0\}$  or  $CAP = 0$ ).

Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs as shown in [Figure 17-11](#). COV must be reset with software.

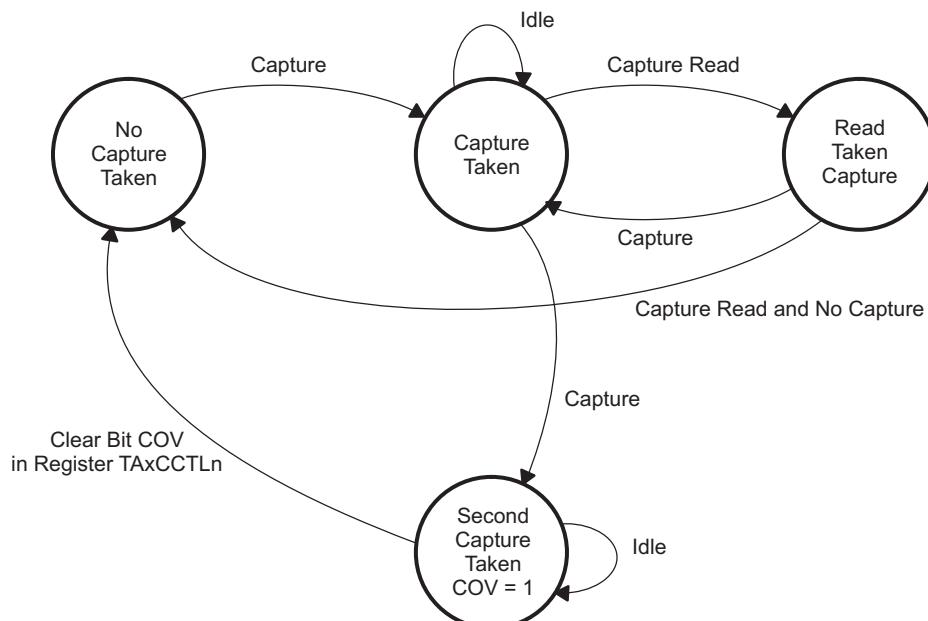


Figure 17-11. Capture Cycle

**Capture Initiated by Software**

Captures can be initiated by software. The CMx bits can be set for capture on both edges. Software then sets CCIS1 = 1 and toggles bit CCIS0 to switch the capture signal between V<sub>CC</sub> and GND, initiating a capture each time CCIS0 changes state:

```
MOV #CAP+SCS+CCIS1+CM_3,&TA0CCTL1 ; Setup TA0CCTL1, synch. capture mode  
; Event trigger on both edges of capture input.  
XOR #CCIS0,&TA0CCTL1 ; TA0CCR1 = TA0R
```

**NOTE: Capture Initiated by Software**

In general, changing capture inputs while in capture mode may cause unintended capture events. For this scenario, switching the capture input between VCC and GND, disabling the capture mode is not required.

**17.2.4.2 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 TA<sub>x</sub>R *counts* to the value in a TA<sub>x</sub>CCR<sub>n</sub>, 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.

**17.2.5 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.

**17.2.5.1 Output Modes**

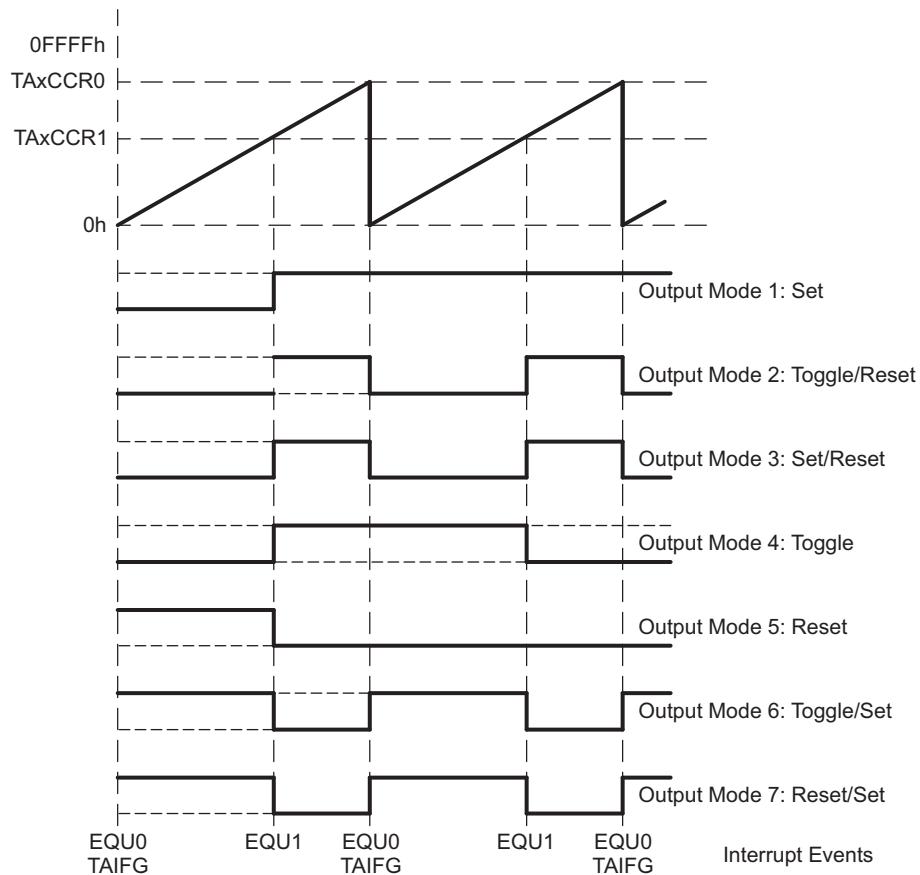
The output modes are defined by the OUTMOD bits and are described in [Table 17-2](#). The OUTn signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUn = EQU0.

**Table 17-2. Output Modes**

OUTMOD <sub>x</sub>	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 <i>counts</i> to the TA <sub>x</sub> CCR <sub>n</sub> 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 <i>counts</i> to the TA <sub>x</sub> CCR <sub>n</sub> value. It is reset when the timer <i>counts</i> to the TA <sub>x</sub> CCR0 value.
011	Set/Reset	The output is set when the timer <i>counts</i> to the TA <sub>x</sub> CCR <sub>n</sub> value. It is reset when the timer <i>counts</i> to the TA <sub>x</sub> CCR0 value.
100	Toggle	The output is toggled when the timer <i>counts</i> to the TA <sub>x</sub> CCR <sub>n</sub> value. The output period is double the timer period.
101	Reset	The output is reset when the timer <i>counts</i> to the TA <sub>x</sub> CCR <sub>n</sub> value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer <i>counts</i> to the TA <sub>x</sub> CCR <sub>n</sub> value. It is set when the timer <i>counts</i> to the TA <sub>x</sub> CCR0 value.
111	Reset/Set	The output is reset when the timer <i>counts</i> to the TA <sub>x</sub> CCR <sub>n</sub> value. It is set when the timer <i>counts</i> to the TA <sub>x</sub> CCR0 value.

**Output Example—Timer in Up Mode**

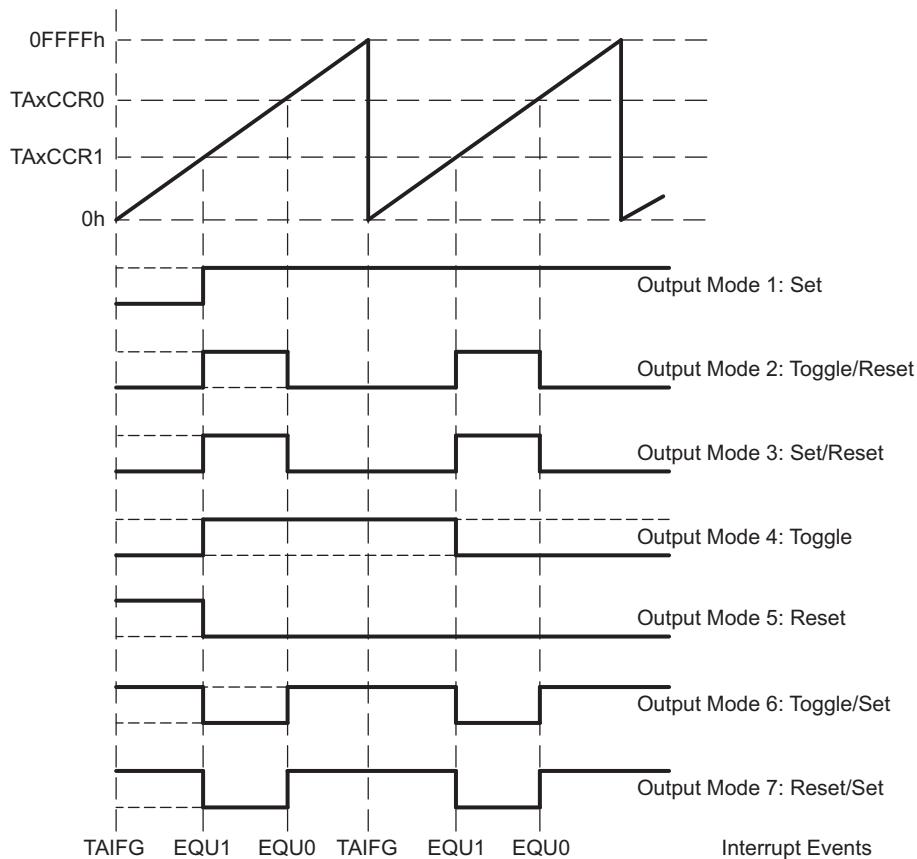
The OUTn signal is changed when the timer *counts* up to the TAxCRRn value and rolls from TAxCCR0 to zero, depending on the output mode. An example is shown in [Figure 17-12](#) using TAxCCR0 and TAxCCR1.



**Figure 17-12. Output Example – Timer in Up Mode**

**Output Example – Timer in Continuous Mode**

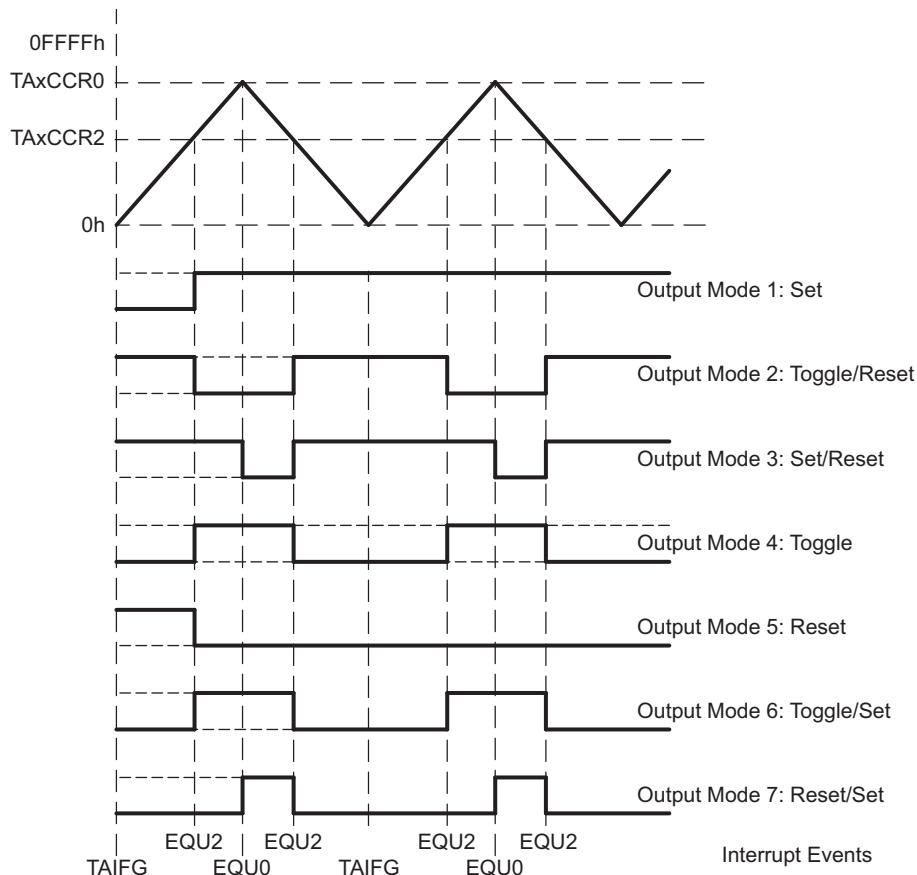
The OUTn signal is changed when the timer reaches the TAxCCRn and TAxCCR0 values, depending on the output mode. An example is shown in [Figure 17-13](#) using TAxCCR0 and TAxCCR1.



**Figure 17-13. Output Example – Timer in Continuous Mode**

**Output Example – Timer in Up/Down Mode**

The OUTn signal changes when the timer equals TAxCRRn in either count direction and when the timer equals TAxCCR0, depending on the output mode. An example is shown in Figure 17-14 using TAxCCR0 and TAxCCR2.

**Figure 17-14. Output Example – Timer in Up/Down Mode****NOTE: Switching between output modes**

When switching between output modes, one of the OUTMOD bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur, because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

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

## 17.2.6 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.

### 17.2.6.1 TAxCCR0 Interrupt

The TAxCCR0 CCIFG flag has the highest Timer\_A interrupt priority and has a dedicated interrupt vector as shown in [Figure 17-15](#). The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

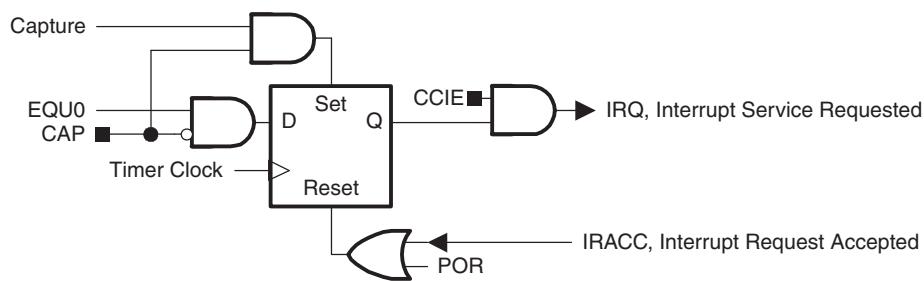


Figure 17-15. Capture/Compare TAxCCR0 Interrupt Flag

### 17.2.6.2 TAxIV, Interrupt Vector Generator

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

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

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

**TAxIV Software Example**

The following software example shows the recommended use of TAxIV and the handling overhead. The TAxIV value is added to the PC to automatically jump to the appropriate routine. The example assumes a single instantiation of the largest timer configuration available.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- Capture/compare block TA0CCR0: 11 cycles
- Capture/compare blocks TA0CCR1, TA0CCR2, TA0CCR3, TA0CCR4, TA0CCR5, TA0CCR6: 16 cycles
- Timer overflow TA0IFG: 14 cycles

```
; Interrupt handler for TA0CCR0 CCIFG.                                Cycles
CCIFG_0_HND
;           ...          ; Start of handler Interrupt latency      6
    RETI                                         5

; Interrupt handler for TA0IFG, TA0CCR1 through TA0CCR6 CCIFG.

TA0_HND     ...          ; Interrupt latency      6
    ADD      &TA0IV,PC   ; Add offset to Jump table      3
    RETI                                         5
    JMP      CCIFG_1_HND ; Vector  0: No interrupt      5
    JMP      CCIFG_2_HND ; Vector  2: TA0CCR1          2
    JMP      CCIFG_3_HND ; Vector  4: TA0CCR2          2
    JMP      CCIFG_4_HND ; Vector  6: TA0CCR3          2
    JMP      CCIFG_5_HND ; Vector  8: TA0CCR4          2
    JMP      CCIFG_6_HND ; Vector 10: TA0CCR5          2
    JMP      CCIFG_7_HND ; Vector 12: TA0CCR6          2

TA0IFG_HND
;           ...          ; Task starts here
    RETI                                         5

CCIFG_6_HND
;           ...          ; Task starts here
    RETI                                         5

CCIFG_5_HND
;           ...          ; Task starts here
    RETI                                         5

CCIFG_4_HND
;           ...          ; Task starts here
    RETI                                         5

CCIFG_3_HND
;           ...          ; Task starts here
    RETI                                         5

CCIFG_2_HND
;           ...          ; Task starts here
    RETI                                         5

CCIFG_1_HND
;           ...          ; Task starts here
    RETI                                         5
```

## 17.3 Timer\_A Registers

Timer\_A registers are listed in [Table 17-3](#) for the largest configuration available. The base address can be found in the device-specific data sheet.

**Table 17-3. Timer\_A Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TAxCTL	Timer_Ax Control	Read/write	Word	0000h	<a href="#">Section 17.3.1</a>
02h	TAxCCTL0	Timer_Ax Capture/Compare Control 0	Read/write	Word	0000h	<a href="#">Section 17.3.3</a>
04h	TAxCCTL1	Timer_Ax Capture/Compare Control 1	Read/write	Word	0000h	<a href="#">Section 17.3.3</a>
06h	TAxCCTL2	Timer_Ax Capture/Compare Control 2	Read/write	Word	0000h	<a href="#">Section 17.3.3</a>
08h	TAxCCTL3	Timer_Ax Capture/Compare Control 3	Read/write	Word	0000h	<a href="#">Section 17.3.3</a>
0Ah	TAxCCTL4	Timer_Ax Capture/Compare Control 4	Read/write	Word	0000h	<a href="#">Section 17.3.3</a>
0Ch	TAxCCTL5	Timer_Ax Capture/Compare Control 5	Read/write	Word	0000h	<a href="#">Section 17.3.3</a>
0Eh	TAxCCTL6	Timer_Ax Capture/Compare Control 6	Read/write	Word	0000h	<a href="#">Section 17.3.3</a>
10h	TAxR	Timer_Ax Counter	Read/write	Word	0000h	<a href="#">Section 17.3.2</a>
12h	TAxCCR0	Timer_Ax Capture/Compare 0	Read/write	Word	0000h	<a href="#">Section 17.3.4</a>
14h	TAxCCR1	Timer_Ax Capture/Compare 1	Read/write	Word	0000h	<a href="#">Section 17.3.4</a>
16h	TAxCCR2	Timer_Ax Capture/Compare 2	Read/write	Word	0000h	<a href="#">Section 17.3.4</a>
18h	TAxCCR3	Timer_Ax Capture/Compare 3	Read/write	Word	0000h	<a href="#">Section 17.3.4</a>
1Ah	TAxCCR4	Timer_Ax Capture/Compare 4	Read/write	Word	0000h	<a href="#">Section 17.3.4</a>
1Ch	TAxCCR5	Timer_Ax Capture/Compare 5	Read/write	Word	0000h	<a href="#">Section 17.3.4</a>
1Eh	TAxCCR6	Timer_Ax Capture/Compare 6	Read/write	Word	0000h	<a href="#">Section 17.3.4</a>
2Eh	TAxIV	Timer_Ax Interrupt Vector	Read only	Word	0000h	<a href="#">Section 17.3.5</a>
20h	TAxEX0	Timer_Ax Expansion 0	Read/write	Word	0000h	<a href="#">Section 17.3.6</a>

**17.3.1 TAxCTL Register**

Timer\_A Control Register

**Figure 17-16. 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 17-4. TAxCTL Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic, and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

### **17.3.2 TAxR Register**

Timer\_A Counter Register

**Figure 17-17. TAxR Register**

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

**Table 17-5. TAxR Register Description**

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

**17.3.3 TAxCCTLn Register**

Timer\_Ax Capture/Compare Control n Register

**Figure 17-18. 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 17-6. 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 TAxCCTL0 input signal. See the device-specific data sheet for specific signal connections. 00b = CCIA 01b = CCIB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	RW	0h	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit.
9	Reserved	R	0h	Reserved. Reads as 0.
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCTL0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high

**Table 17-6. 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

**17.3.4 TAxCRRn Register**

Timer\_A Capture/Compare n Register

**Figure 17-19. TAxCRRn Register**

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

**Table 17-7. TAxCRRn Register Description**

Bit	Field	Type	Reset	Description
15-0	TAxCCR0	RW	0h	Compare mode: TAxCRRn holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TAxCRRn register when a capture is performed.

**17.3.5 TAxIV Register**

Timer\_Ax Interrupt Vector Register

**Figure 17-20. 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 17-8. 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

### 17.3.6 TAxE0 Register

Timer\_Ax Expansion 0 Register

**Figure 17-21. TAxE0 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 <sup>(1)</sup>	
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> After programming TAIDEX bits and configuration of the timer, set TACLR bit to ensure proper reset of the timer divider logic.

**Table 17-9. TAxE0 Register Description**

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Reads as 0.
2-0	TAIDEX	RW	0h	<p>Input divider expansion. These bits along with the ID bits select the divider for the input clock.</p> <p>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</p>

## **Timer\_B**

Timer\_B is a 16-bit timer/counter with multiple capture/compare registers. There can be multiple Timer\_B modules on a given device (see the device-specific data sheet). This chapter describes the operaand use of the Timer\_B module.

Topic	Page
18.1 Timer_B Introduction .....	483
18.2 Timer_B Operation .....	485
18.3 Timer_B Registers .....	498

## 18.1 Timer\_B Introduction

Timer\_B is a 16-bit timer/counter with up to seven capture/compare registers. Timer\_B can support multiple capture/comparisons, PWM outputs, and interval timing. Timer\_B also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer\_B features include:

- Asynchronous 16-bit timer/counter with four operating modes and four selectable lengths
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with PWM capability
- Double-buffered compare latches with synchronized loading
- Interrupt vector register for fast decoding of all Timer\_B interrupts

The block diagram of Timer\_B is shown in [Figure 18-1](#).

---

**NOTE: Use of the word count**

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

---

**NOTE: Nomenclature**

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

---

### 18.1.1 Similarities and Differences From Timer\_A

Timer\_B is identical to Timer\_A with the following exceptions:

- The length of Timer\_B is programmable to be 8, 10, 12, or 16 bits.
- Timer\_B TB $x$ CCR $n$  registers are double-buffered and can be grouped.
- All Timer\_B outputs can be put into a high-impedance state.
- The SCCI bit function is not implemented in Timer\_B.

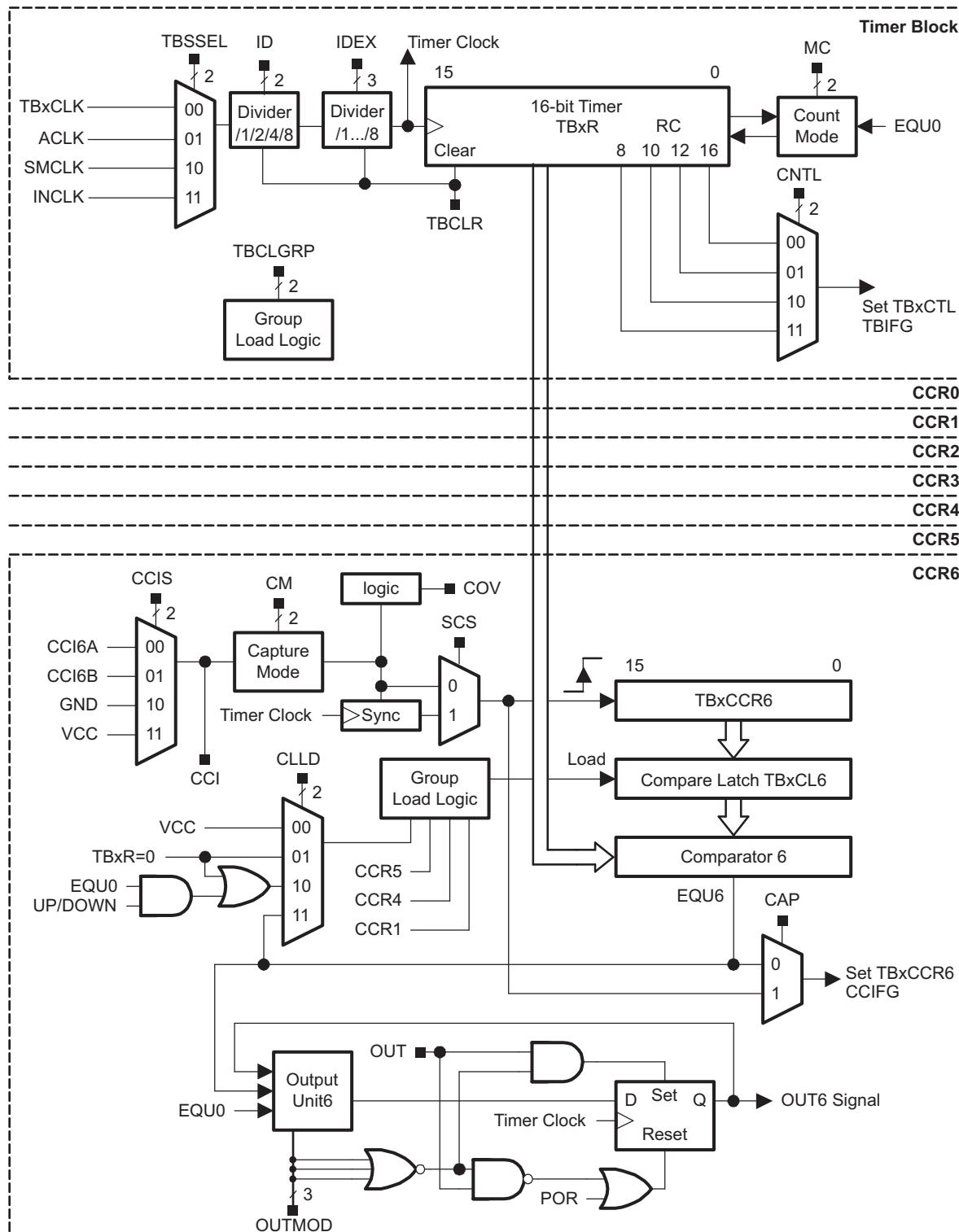


Figure 18-1. Timer\_B Block Diagram

## 18.2 Timer\_B Operation

The Timer\_B module is configured with user software. The setup and operation of Timer\_B is discussed in the following sections.

### 18.2.1 16-Bit Timer Counter

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

TBxR may be cleared by setting the TBCLR bit. Setting TBCLR also clears the clock divider and count direction for up/down mode.

---

**NOTE: Modifying Timer\_B registers**

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable, interrupt flag, and TBCLR) to avoid errant operating conditions.

When the timer clock is asynchronous to the CPU clock, any read from TBxR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TBxR takes effect immediately.

---

#### 18.2.1.1 TBxR Length

Timer\_B is configurable to operate as an 8-, 10-, 12-, or 16-bit timer with the CNTL bits. The maximum count value, TBxR<sub>(max)</sub>, for the selectable lengths is 0FFh, 03FFh, OFFFh, and 0FFFFh, respectively. Data written to the TBxR register in 8-, 10-, and 12-bit mode is right justified with leading zeros.

#### 18.2.1.2 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TBxCLK or INCLK. The clock source is selected with the TBSSEL bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the ID bits. The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the TBIDEX bits. The timer clock divider logic is reset when TBCLR is set.

---

**NOTE: Timer\_B dividers**

After programming ID or TBIDEX bits, set the TBCLR bit. This clears the contents of TBxR and resets the clock divider logic to a defined state. The clock dividers are implemented as down counters. Therefore, when the TBCLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer\_B clock source selected with the TBSSEL bits and continues clocking at the divider settings set by the ID and TBIDEX bits.

---

### 18.2.2 Starting the Timer

The timer may be started or restarted in the following ways:

- The timer counts when MC > { 0 } and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by loading 0 to TBxCL0. The timer may then be restarted by loading a nonzero value to TBxCL0. In this scenario, the timer starts incrementing in the up direction from zero.

### 18.2.3 Timer Mode Control

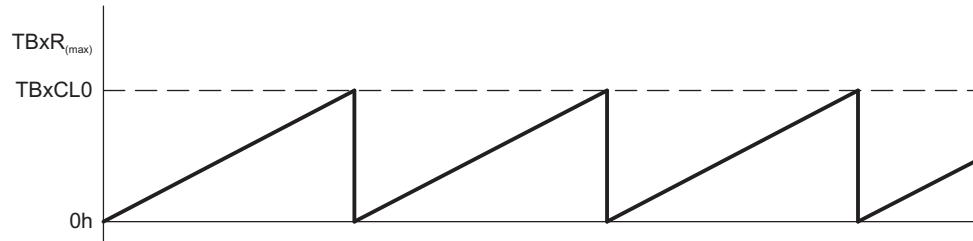
The timer has four modes of operation: stop, up, continuous, and up/down (see [Table 18-1](#)). The operating mode is selected with the MC bits.

**Table 18-1. Timer Modes**

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of compare register TBxCL0.
10	Continuous	The timer repeatedly counts from zero to the value selected by the CNTL bits.
11	Up/down	The timer repeatedly counts from zero up to the value of TBxCL0 and then back down to zero.

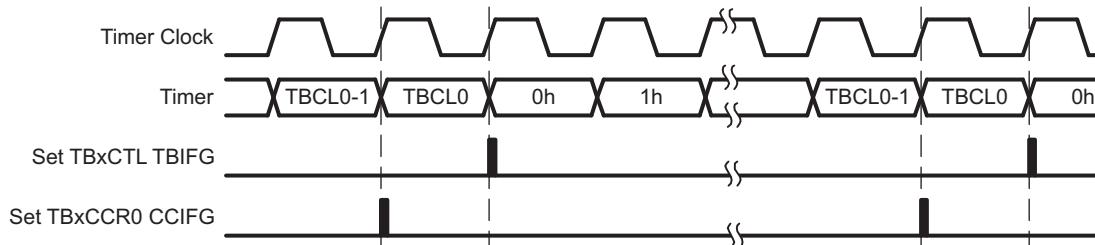
#### 18.2.3.1 Up Mode

The up mode is used if the timer period must be different from  $TBxR_{(max)}$  counts. The timer repeatedly counts up to the value of compare latch TBxCL0, which defines the period (see [Figure 18-2](#)). The number of timer counts in the period is  $TBxCL0 + 1$ . When the timer value equals TBxCL0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TBxCL0, the timer immediately restarts counting from zero.



**Figure 18-2. Up Mode**

The TBxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TBxCL0 value. The TBIFG interrupt flag is set when the timer *counts* from TBxCL0 to zero. [Figure 18-3](#) shows the flag set cycle.



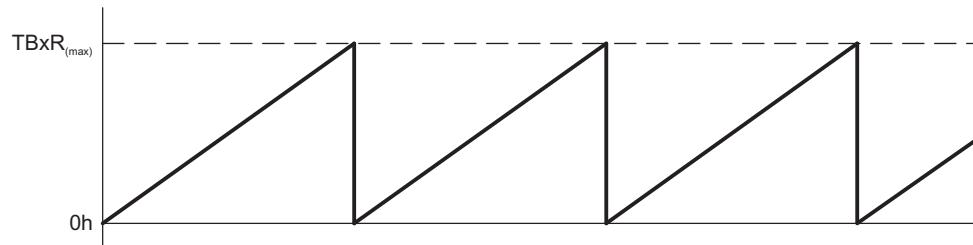
**Figure 18-3. Up Mode Flag Setting**

#### Changing Period Register TBxCL0

When changing TBxCL0 while the timer is running and when the TBxCL0 load mode is *immediate*, if the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

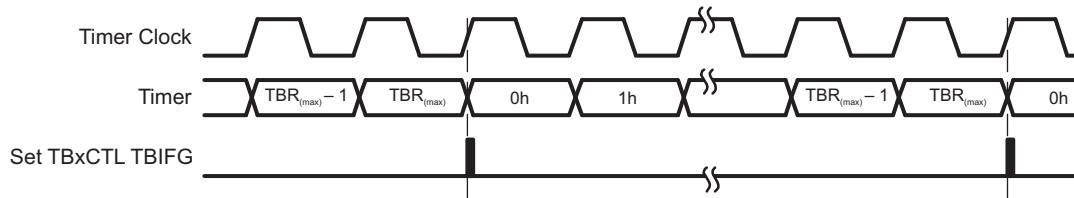
### 18.2.3.2 Continuous Mode

In continuous mode, the timer repeatedly counts up to  $TBxR_{(max)}$  and restarts from zero (see [Figure 18-4](#)). The compare latch  $TBxCL0$  works the same way as the other capture/compare registers.



**Figure 18-4. Continuous Mode**

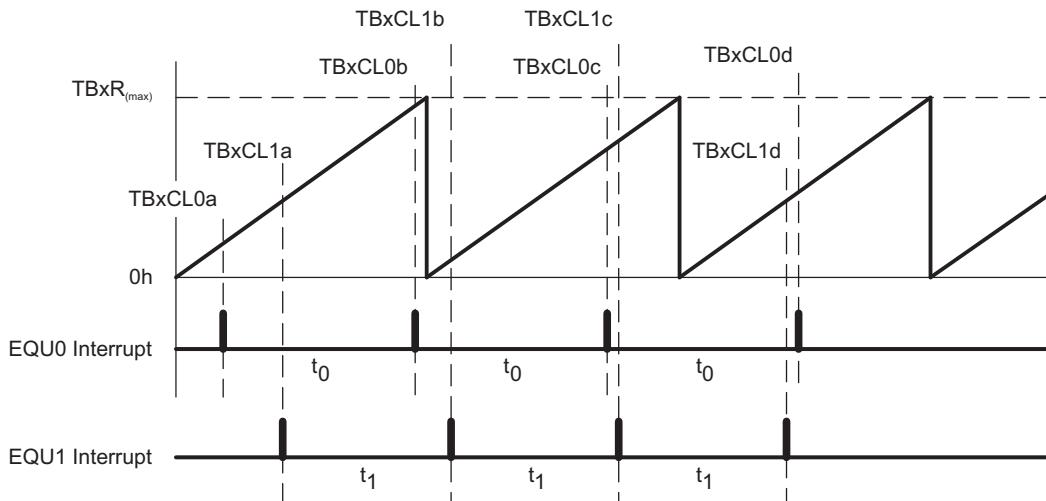
The  $TBIFG$  interrupt flag is set when the timer *counts* from  $TBxR_{(max)}$  to zero. [Figure 18-5](#) shows the flag set cycle.



**Figure 18-5. Continuous Mode Flag Setting**

### 18.2.3.3 Use of Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the  $TBxCLn$  latch in the interrupt service routine. [Figure 18-6](#) shows two separate time intervals,  $t_0$  and  $t_1$ , being added to the capture/compare registers. The time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where  $n = 0$  to 7), independent time intervals or output frequencies can be generated using capture/compare registers.



**Figure 18-6. Continuous Mode Time Intervals**

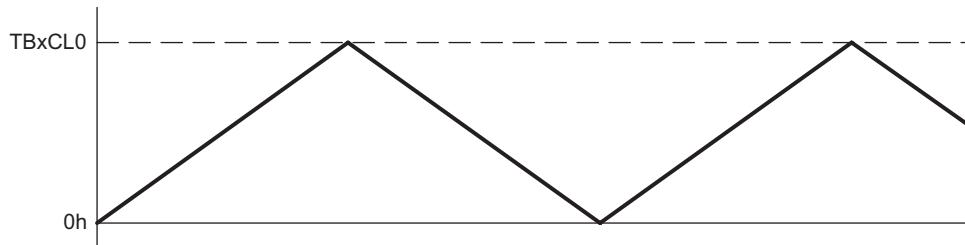
Time intervals can be produced with other modes as well, where TBxCL0 is used as the period register. Their handling is more complex, because the sum of the old TBxCLn data and the new period can be higher than the TBxCL0 value. When the sum of the previous TBxCLn value plus  $t_x$  is greater than the TBxCL0 data, the old TBxCL0 value must be subtracted to obtain the correct time interval.

#### 18.2.3.4 Up/Down Mode

The up/down mode is used if the timer period must be different from  $TBxR_{(max)}$  counts and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare latch TBxCL0, and back down to zero (see [Figure 18-7](#)). The period is twice the value in TBxCL0.

**NOTE:**  $TBxCL0 > TBxR_{(max)}$

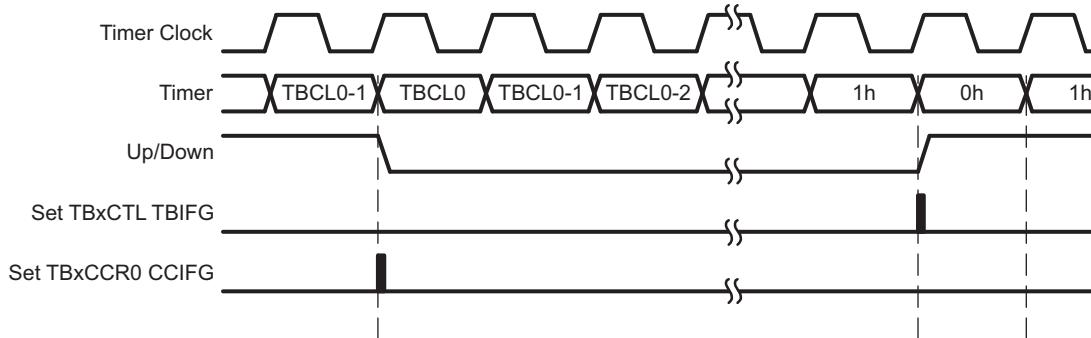
If  $TBxCL0 > TBxR_{(max)}$ , the counter operates as if it were configured for continuous mode. It does not count down from  $TBxR_{(max)}$  to zero.



**Figure 18-7. Up/Down Mode**

The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TBCLR bit must be used to clear the direction. The TBCLR bit also clears the TBxR value and the timer clock divider.

In up/down mode, the TBxCCR0 CCIFG interrupt flag and the TBIFG interrupt flag are set only once during the period, separated by one-half the timer period. The TBxCCR0 CCIFG interrupt flag is set when the timer *counts* from TBxCL0-1 to TBxCL0, and TBIFG is set when the timer completes *counting down* from 0001h to 0000h. [Figure 18-8](#) shows the flag set cycle.



**Figure 18-8. Up/Down Mode Flag Setting**

#### Changing the Value of Period Register TBxCL0

When changing TBxCL0 while the timer is running and counting in the down direction, and when the TBxCL0 load mode is *immediate*, the timer continues its descent until it reaches zero. The new period takes effect after the counter counts down to zero.

If the timer is counting in the up direction when the new period is latched into TBxCL0, and the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction, and the new period is less than the current count value when TBxCL0 is loaded, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

### 18.2.3.5 Use of Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see [Section 18.2.5](#)). 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 18-9](#), the  $t_{\text{dead}}$  is:

$$t_{\text{dead}} = t_{\text{timer}} \times (TBxCL1 - TBxCL3)$$

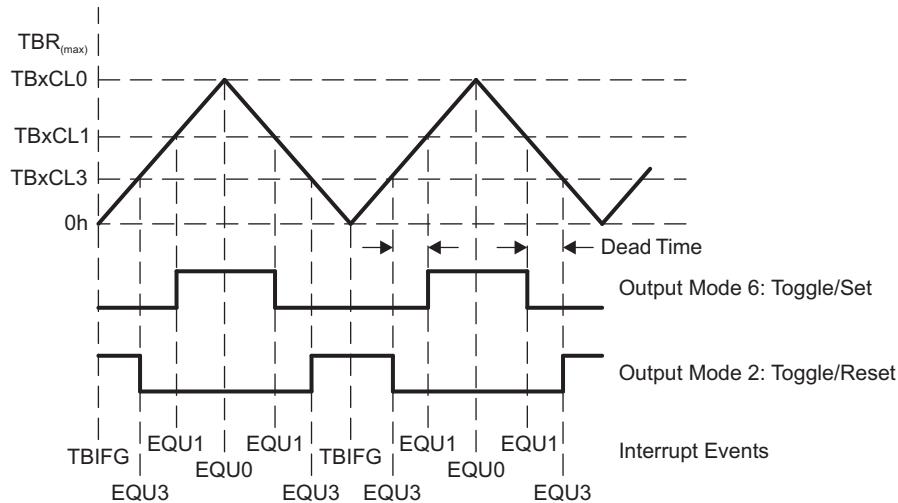
Where:

$t_{\text{dead}}$  = Time during which both outputs need to be inactive

$t_{\text{timer}}$  = Cycle time of the timer clock

TBxCLn = Content of compare latch n

The ability to simultaneously load grouped compare latches ensures the dead times.



**Figure 18-9. Output Unit in Up/Down Mode**

### 18.2.4 Capture/Compare Blocks

Up to seven identical capture/compare blocks, TBxCCRn (where n = 0 to 6), are present in Timer\_B. Any of the blocks may be used to capture the timer data or to generate time intervals.

#### 18.2.4.1 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 CCIA and CCIB are connected to external pins or internal signals and are selected with the CCIS bits. The CM bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture is performed:

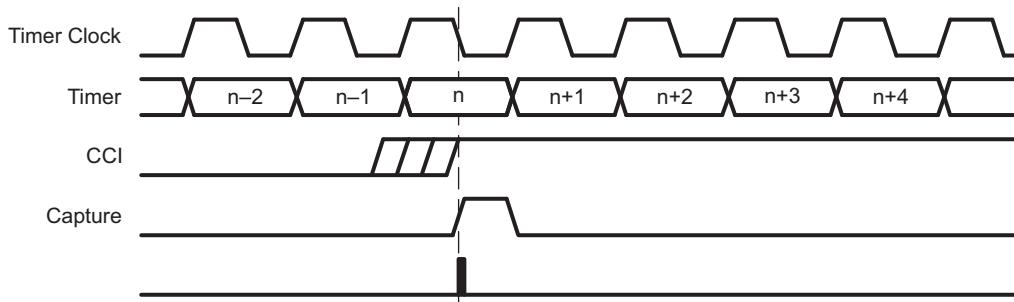
- The timer value is copied into the TBxCCRn register.
- The interrupt flag CCIFG is set.

The input signal level can be read at any time via the CCI bit. Devices may have different signals connected to CCIA and CCIB. See the device-specific data sheet for the connections of these signals.

*Timer\_B Operation*

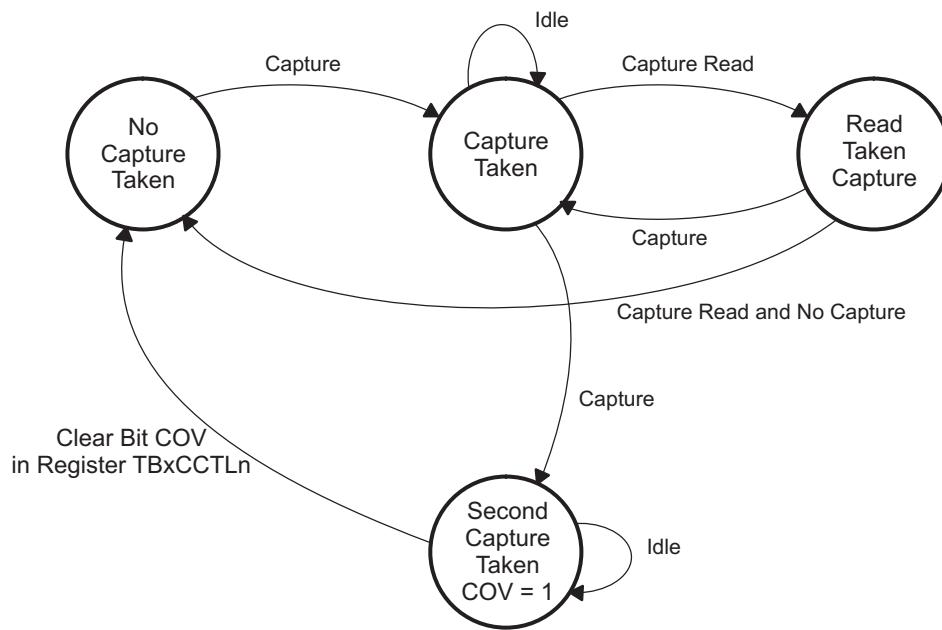
www.ti.com

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 18-10](#)).

**Figure 18-10. Capture Signal (SCS = 1)****NOTE: Changing Capture Inputs**

Changing capture inputs while in capture mode may cause unintended capture events. To avoid this scenario, capture inputs should only be changed when capture mode is disabled ( $CM = \{0\}$  or  $CAP = 0$ ).

Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs (see [Figure 18-11](#)). COV must be reset with software.

**Figure 18-11. Capture Cycle**

### Capture Initiated by Software

Captures can be initiated by software. The CM bits can be set for capture on both edges. Software then sets bit CCIS1 = 1 and toggles bit CCIS0 to switch the capture signal between V<sub>CC</sub> and GND, initiating a capture each time CCIS0 changes state:

```
MOV      #CAP+SCS+CCIS1+CM_3 ,&TB0CCTL1    ; Setup TB0CCTL1
XOR      #CCIS0 ,&TB0CCTL1                 ; TB0CCR1 = TB0R
```

**NOTE: Capture Initiated by Software**

In general, changing capture inputs while in capture mode may cause unintended capture events. For this scenario, switching the capture input between VCC and GND, disabling the capture mode is not required.

#### 18.2.4.2 Compare Mode

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

- Interrupt flag CCIFG is set.
- Internal signal EQUn = 1.
- EQUn affects the output according to the output mode.

#### Compare Latch TBxCn

The TBxCCRn compare latch, TBxCn, holds the data for the comparison to the timer value in compare mode. TBxCn is buffered by TBxCCRn. The buffered compare latch gives the user control over when a compare period updates. The user cannot directly access TBxCn. Compare data is written to each TBxCCRn and automatically transferred to TBxCn. The timing of the transfer from TBxCCRn to TBxCn is user selectable, with the CLLD bits as described in [Table 18-2](#).

**Table 18-2. TBxCn Load Events**

CLLD	Description
00	New data is transferred from TBxCCRn to TBxCn immediately when TBxCCRn is written to.
01	New data is transferred from TBxCCRn to TBxCn when TBxR <i>counts</i> to 0.
10	New data is transferred from TBxCCRn to TBxCn when TBxR <i>counts</i> to 0 for up and continuous modes. New data is transferred from TBxCCRn to TBxCn when TBxR <i>counts</i> to the old TBxCn value or to 0 for up/down mode.
11	New data is transferred from TBxCCRn to TBxCn when TBxR <i>counts</i> to the old TBxCn value.

### Grouping Compare Latches

Multiple compare latches may be grouped together for simultaneous updates with the TBCLGRPx bits. When using groups, the CLLD bits of the lowest numbered TBxCCRn in the group determine the load event for each compare latch of the group, except when TBCLGRP = 3 (see [Table 18-3](#)). The CLLD bits of the controlling TBxCCRn must not be set to zero. When the CLLD bits of the controlling TBxCCRn are set to zero, all compare latches update immediately when their corresponding TBxCCRn is written; no compare latches are grouped.

Two conditions must exist for the compare latches to be loaded when grouped. First, all TBxCCRn registers of the group must be updated, even when new TBxCCRn data = old TBxCCRn data. Second, the load event must occur.

**Table 18-3. Compare Latch Operating Modes**

TBCLGRPx	Grouping	Update Control
00	None	Individual
01	TBxCL1+TBxCL2 TBxCL3+TBxCL4 TBxCL5+TBxCL6	TBxCCR1 TBxCCR3 TBxCCR5
10	TBxCL1+TBxCL2+TBxCL3 TBxCL4+TBxCL5+TBxCL6	TBxCCR1 TBxCCR4
11	TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6	TBxCCR1

### 18.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals, such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUn signals. The TBOUTH pin function can be used to put all Timer\_B outputs into a high-impedance state. When the TBOUTH pin function is selected for the pin (corresponding PSEL bit is set, and port configured as input) and when the pin is pulled high, all Timer\_B outputs are in a high-impedance state.

#### 18.2.5.1 Output Modes

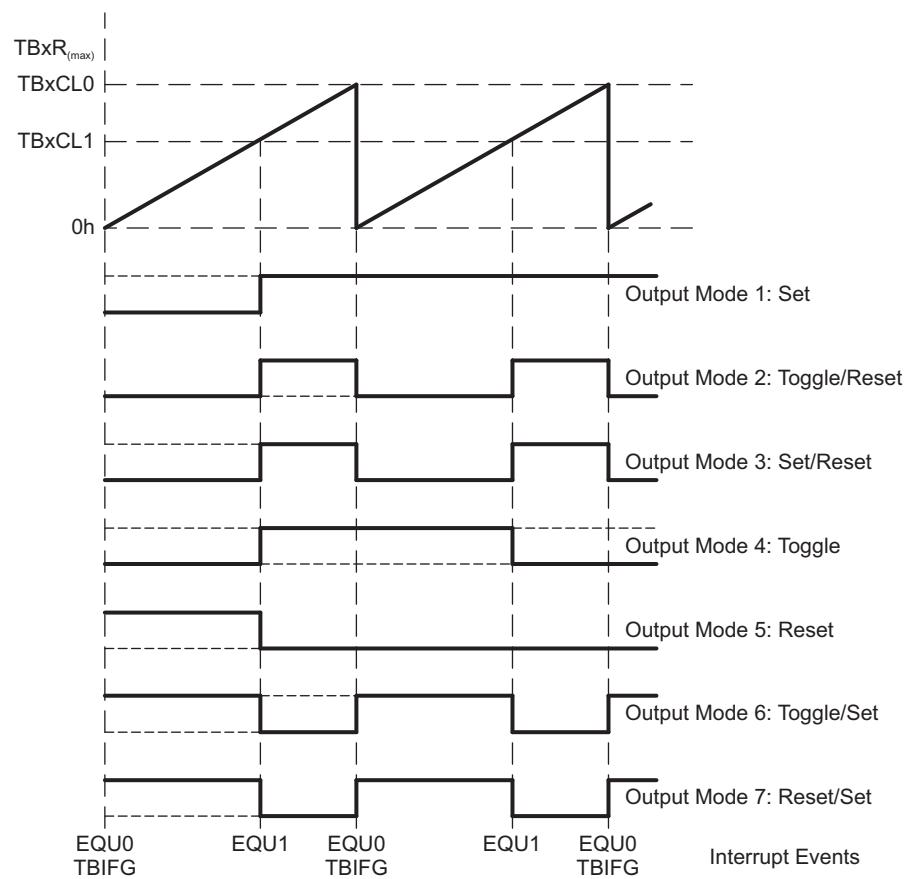
The output modes are defined by the OUTMOD bits and are described in [Table 18-4](#). The OUTn signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUn = EQU0.

**Table 18-4. Output Modes**

OUTMOD	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer <i>counts</i> to the TBxCLn value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer <i>counts</i> to the TBxCLn value. It is reset when the timer <i>counts</i> to the TBxCL0 value.
011	Set/Reset	The output is set when the timer <i>counts</i> to the TBxCLn value. It is reset when the timer <i>counts</i> to the TBxCL0 value.
100	Toggle	The output is toggled when the timer <i>counts</i> to the TBxCLn value. The output period is double the timer period.
101	Reset	The output is reset when the timer <i>counts</i> to the TBxCLn value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer <i>counts</i> to the TBxCLn value. It is set when the timer <i>counts</i> to the TBxCL0 value.
111	Reset/Set	The output is reset when the timer <i>counts</i> to the TBxCLn value. It is set when the timer <i>counts</i> to the TBxCL0 value.

**Output Example – Timer in Up Mode**

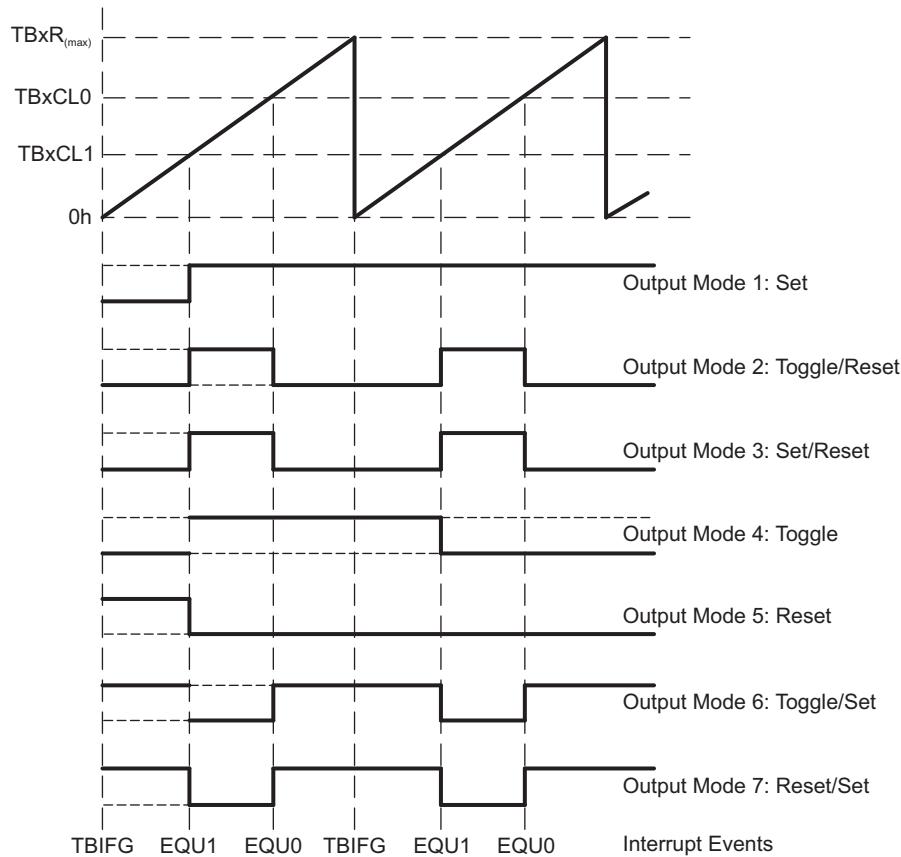
The OUTn signal is changed when the timer *counts* up to the TBxCLn value, and rolls from TBxCL0 to zero, depending on the output mode. An example is shown in [Figure 18-12](#) using TBxCL0 and TBxCL1.



**Figure 18-12. Output Example – Timer in Up Mode**

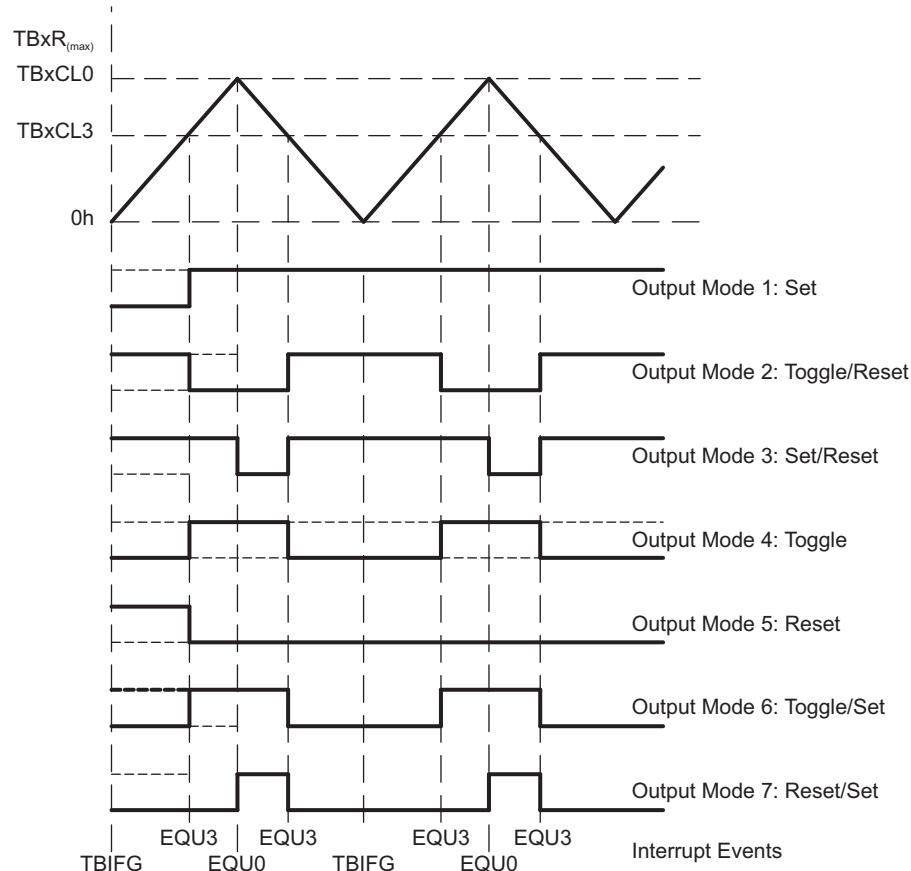
**Output Example – Timer in Continuous Mode**

The OUTn signal is changed when the timer reaches the TBxCLn and TBxCL0 values, depending on the output mode. An example is shown in Figure 18-13 using TBxCL0 and TBxCL1.

**Figure 18-13. Output Example – Timer in Continuous Mode**

**Output Example – Timer in Up/Down Mode**

The OUTn signal changes when the timer equals TBxCLn in either count direction and when the timer equals TBxCL0, depending on the output mode. An example is shown in [Figure 18-14](#) using TBxCL0 and TBxCL3.



**Figure 18-14. Output Example – Timer in Up/Down Mode**

**NOTE: Switching between output modes**

When switching between output modes, one of the OUTMOD bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

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

### 18.2.6 Timer\_B Interrupts

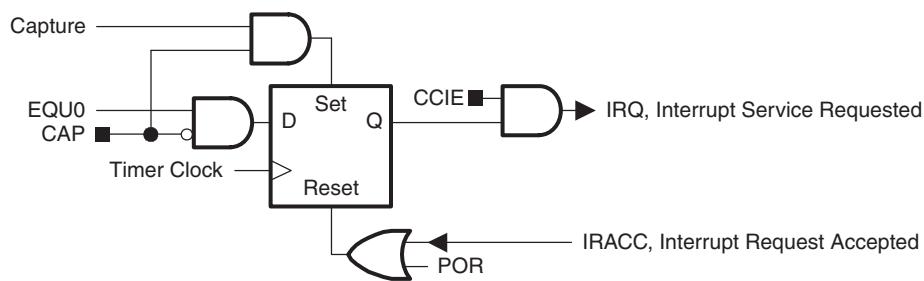
Two interrupt vectors are associated with the 16-bit Timer\_B module:

- TBxCCR0 interrupt vector for TBxCCR0 CCIFG
- TBIV interrupt vector for all other CCIFG flags and TBIFG

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

#### 18.2.6.1 TBxCCR0 Interrupt Vector

The TBxCCR0 CCIFG flag has the highest Timer\_B interrupt priority and has a dedicated interrupt vector (see [Figure 18-15](#)). The TBxCCR0 CCIFG flag is automatically reset when the TBxCCR0 interrupt request is serviced.



**Figure 18-15. Capture/Compare TBxCCR0 Interrupt Flag**

#### 18.2.6.2 TBxIV, Interrupt Vector Generator

The TBIFG flag and TBxCCRn CCIFG flags (excluding TBxCCR0 CCIFG) are prioritized and combined to source a single interrupt vector. The interrupt vector register TBxIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt (excluding TBxCCR0 CCIFG) generates a number in the TBxIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer\_B interrupts do not affect the TBxIV value.

Any access, read or write, of the TBxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TBxCCR1 and TBxCCR2 CCIFG flags are set when the interrupt service routine accesses the TBxIV register, TBxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TBxCCR2 CCIFG flag generates another interrupt.

#### 18.2.6.3 TBxIV, Interrupt Handler Examples

The following software example shows the recommended use of TBxIV and the handling overhead. The TBxIV value is added to the PC to automatically jump to the appropriate routine. The example assumes a single instantiation of the largest timer configuration available.

The numbers at the right margin show the necessary CPU clock cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- Capture/compare block CCR0: 11 cycles
- Capture/compare blocks CCR1 to CCR6: 16 cycles
- Timer overflow TBIFG: 14 cycles

The following software example shows the recommended use of TBxIV for Timer\_B3.

```

; Interrupt handler for TB0CCR0 CCIFG.                                Cycles
CCIFG_0_HND
;           ...          ; Start of handler Interrupt latency      6
    RETI                                         5

; Interrupt handler for TB0IFG, TB0CCR1 through TB0CCR6 CCIFG.

TB0_HND   ...
    ADD     &TB0IV,PC   ; Interrupt latency                      6
    RETI                                         3
    JMP     CCIFG_1_HND ; Vector 0: No interrupt                  5
    JMP     CCIFG_2_HND ; Vector 2: TB0CCR1                     2
    JMP     CCIFG_3_HND ; Vector 4: TB0CCR2                     2
    JMP     CCIFG_4_HND ; Vector 6: TB0CCR3                     2
    JMP     CCIFG_5_HND ; Vector 8: TB0CCR4                     2
    JMP     CCIFG_6_HND ; Vector 10: TB0CCR5                    2
    JMP    CCIFG_6_HND ; Vector 12: TB0CCR6                    2

TB0IFG_HND
...
    RETI                                         5

CCIFG_6_HND
...
    RETI                                         5

CCIFG_5_HND
...
    RETI                                         5

CCIFG_4_HND
...
    RETI                                         5

CCIFG_3_HND
...
    RETI                                         5

CCIFG_2_HND
...
    RETI                                         5

CCIFG_1_HND
...
    RETI                                         5

```

### 18.3 Timer\_B Registers

The Timer\_B registers are listed in [Table 18-5](#). The base address can be found in the device-specific data sheet. The address offset is listed in [Table 18-5](#).

**Table 18-5. Timer\_B Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TBxCTL	Timer_B Control	Read/write	Word	0000h	<a href="#">Section 18.3.1</a>
02h	TBxCCTL0	Timer_B Capture/Compare Control 0	Read/write	Word	0000h	<a href="#">Section 18.3.3</a>
04h	TBxCCTL1	Timer_B Capture/Compare Control 1	Read/write	Word	0000h	<a href="#">Section 18.3.3</a>
06h	TBxCCTL2	Timer_B Capture/Compare Control 2	Read/write	Word	0000h	<a href="#">Section 18.3.3</a>
08h	TBxCCTL3	Timer_B Capture/Compare Control 3	Read/write	Word	0000h	<a href="#">Section 18.3.3</a>
0Ah	TBxCCTL4	Timer_B Capture/Compare Control 4	Read/write	Word	0000h	<a href="#">Section 18.3.3</a>
0Ch	TBxCCTL5	Timer_B Capture/Compare Control 5	Read/write	Word	0000h	<a href="#">Section 18.3.3</a>
0Eh	TBxCCTL6	Timer_B Capture/Compare Control 6	Read/write	Word	0000h	<a href="#">Section 18.3.3</a>
10h	TBxR	Timer_B Counter	Read/write	Word	0000h	<a href="#">Section 18.3.2</a>
12h	TBxCCR0	Timer_B Capture/Compare 0	Read/write	Word	0000h	<a href="#">Section 18.3.4</a>
14h	TBxCCR1	Timer_B Capture/Compare 1	Read/write	Word	0000h	<a href="#">Section 18.3.4</a>
16h	TBxCCR2	Timer_B Capture/Compare 2	Read/write	Word	0000h	<a href="#">Section 18.3.4</a>
18h	TBxCCR3	Timer_B Capture/Compare 3	Read/write	Word	0000h	<a href="#">Section 18.3.4</a>
1Ah	TBxCCR4	Timer_B Capture/Compare 4	Read/write	Word	0000h	<a href="#">Section 18.3.4</a>
1Ch	TBxCCR5	Timer_B Capture/Compare 5	Read/write	Word	0000h	<a href="#">Section 18.3.4</a>
1Eh	TBxCCR6	Timer_B Capture/Compare 6	Read/write	Word	0000h	<a href="#">Section 18.3.4</a>
2Eh	TBxIV	Timer_B Interrupt Vector	Read only	Word	0000h	<a href="#">Section 18.3.5</a>
20h	TBxEX0	Timer_B Expansion 0	Read/write	Word	0000h	<a href="#">Section 18.3.6</a>

### 18.3.1 TBxCTL Register

Timer\_B x Control Register

**Figure 18-16. TBxCTL Register**

15	14	13	12	11	10	9	8
Reserved	TBCLGRPx			CNTL		Reserved	TBSSEL
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID	MC			Reserved	TBCLR	TBIE	TBIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

**Table 18-6. TBxCTL Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-13	TBCLGRP	RW	0h	TBxCLn group 00b = Each TBxCLn latch loads independently. 01b = TBxCL1+TBxCL2 (TBxCCR1 CLLD bits control the update); TBxCL3+TBxCL4 (TBxCCR3 CLLD bits control the update); TBxCL5+TBxCL6 (TBxCCR5 CLLD bits control the update); TBxCL0 independent 10b = TBxCL1+TBxCL2+TBxCL3 (TBxCCR1 CLLD bits control the update); TBxCL4+TBxCL5+TBxCL6 (TBxCCR4 CLLD bits control the update); TBxCL0 independent 11b = TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6 (TBxCCR1 CLLD bits control the update)
12-11	CNTL	RW	0h	Counter length 00b = 16-bit, TBxR(max) = 0FFFFh 01b = 12-bit, TBxR(max) = 0FFFh 10b = 10-bit, TBxR(max) = 03FFh 11b = 8-bit, TBxR(max) = 0FFh
10	Reserved	R	0h	Reserved. Always reads as 0.
9-8	TBSSEL	RW	0h	Timer_B clock source select 00b = TBxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits, along with the TBIDEX bits, select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MC = 00h when Timer_B is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TBxCL0 10b = Continuous mode: Timer counts up to the value set by CNTL 11b = Up/down mode: Timer counts up to TBxCL0 and down to 0000h
3	Reserved	R	0h	Reserved. Always reads as 0.
2	TBCLR	RW	0h	Timer_B clear. Setting this bit resets TBxR, the timer clock divider logic, and the count direction. The TBCLR bit is automatically reset and is always read as zero.
1	TBIE	RW	0h	Timer_B interrupt enable. This bit enables the TBIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled

**Table 18-6. TBxCTL Register Description (continued)**

Bit	Field	Type	Reset	Description
0	TBIFG	RW	0h	Timer_B interrupt flag 0b = No interrupt pending 1b = Interrupt pending

### **18.3.2 TBxR Register**

Timer\_B x Counter Register

**Figure 18-17. TBxR Register**

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

**Table 18-7. TBxR Register Description**

Bit	Field	Type	Reset	Description
15-0	TBxR	RW	0h	Timer_B register. The TBxR register is the count of Timer_B.

**18.3.3 TBxCCTLn Register**

Timer\_B x Capture/Compare Control Register n

**Figure 18-18. TBxCCTLn Register**

15	14	13	12	11	10	9	8
CM		CCIS		SCS		CLLD	CAP
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 18-8. TBxCCTLn Register Description**

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCIS	RW	0h	Capture/compare input select. These bits select the TBxCCRn input signal. See the device-specific data sheet for specific signal connections. 00b = CC <sub>I</sub> x A 01b = CC <sub>I</sub> x B 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10-9	CLLD	RW	0h	Compare latch load. These bits select the compare latch load event. 00b = TBxCLn loads on write to TBxCCRn 01b = TBxCLn loads when TBxR counts to 0 10b = TBxCLn loads when TBxR counts to 0 (up or continuous mode). TBxCLn loads when TBxR counts to TBxCL0 or to 0 (up/down mode). 11b = TBxCLn loads when TBxR counts to TBxCLn
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TBxCL0 because EQU <sub>n</sub> = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	Undef	Capture/compare input. The selected input signal can be read by this bit.

**Table 18-8. TBxCCTLn Register Description (continued)**

Bit	Field	Type	Reset	Description
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

**18.3.4 TBxCCRn Register**

Timer\_B x Capture/Compare Register n

**Figure 18-19. TBxCCRn Register**

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

**Table 18-9. TBxCCRn Register Description**

Bit	Field	Type	Reset	Description
15-0	TBxCCRn	RW	0h	<p>Timer_B capture/compare register.</p> <p>Compare mode: TBxCCRn holds the data for the comparison to the timer value in the Timer_B Register, TBR.</p> <p>Capture mode: The Timer_B Register, TBR, is copied into the TBxCCRn register when a capture is performed.</p>

### 18.3.5 TBxIV Register

Timer\_B x Interrupt Vector Register

**Figure 18-20. TBxIV Register**

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

**Table 18-10. TBxIV Register Description**

Bit	Field	Type	Reset	Description
15-0	TBIV	R	0h	<p>Timer_B interrupt vector value</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TBxCCR1 CCIFG; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TBxCCR2 CCIFG</p> <p>06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TBxCCR3 CCIFG</p> <p>08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TBxCCR4 CCIFG</p> <p>0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TBxCCR5 CCIFG</p> <p>0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TBxCCR6 CCIFG</p> <p>0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TBxCTL TBIFG; Interrupt Priority: Lowest</p>

**18.3.6 TBxEX0 Register**

Timer\_B x Expansion Register 0

**Figure 18-21. TBxEX0 Register**

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

<sup>(1)</sup> After programming TAIDEX bits and configuration of the timer, set TACLR bit to ensure proper reset of the timer divider logic.

**Table 18-11. TBxEX0 Register Description**

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	TBIDEX	RW	0h	Input divider expansion. These bits along with the ID bits select the divider for the input clock. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 8

Timer\_D is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes Timer\_D.

Topic	Page
19.1 Timer_D Introduction .....	508
19.2 Timer_D Operation .....	511
19.3 Timer_D Registers .....	534

## 19.1 Timer\_D Introduction

Timer\_D is a 16-bit timer/counter with multiple capture/compare registers. Timer\_D can support multiple capture/comparisons, interval timing, and PWM outputs both in general and high resolution modes. Timer\_D also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions, from each of the capture/compare registers.

Timer\_D features include:

- Asynchronous 16-bit timer/counter with four operating modes and four selectable lengths
- Selectable and configurable clock source
- Configurable capture/compare registers
- Controlling rising and falling PWM edges by combining two neighbor TDCCR registers in one compare channel output
- Configurable outputs with PWM capability
- High-resolution mode with a fine clock frequency up to 16 times the timer input clock frequency
- Double-buffered compare registers with synchronized loading
- Interrupt vector register for fast decoding of all Timer\_D interrupts

The block diagram of Timer\_D is shown in [Figure 19-1](#).

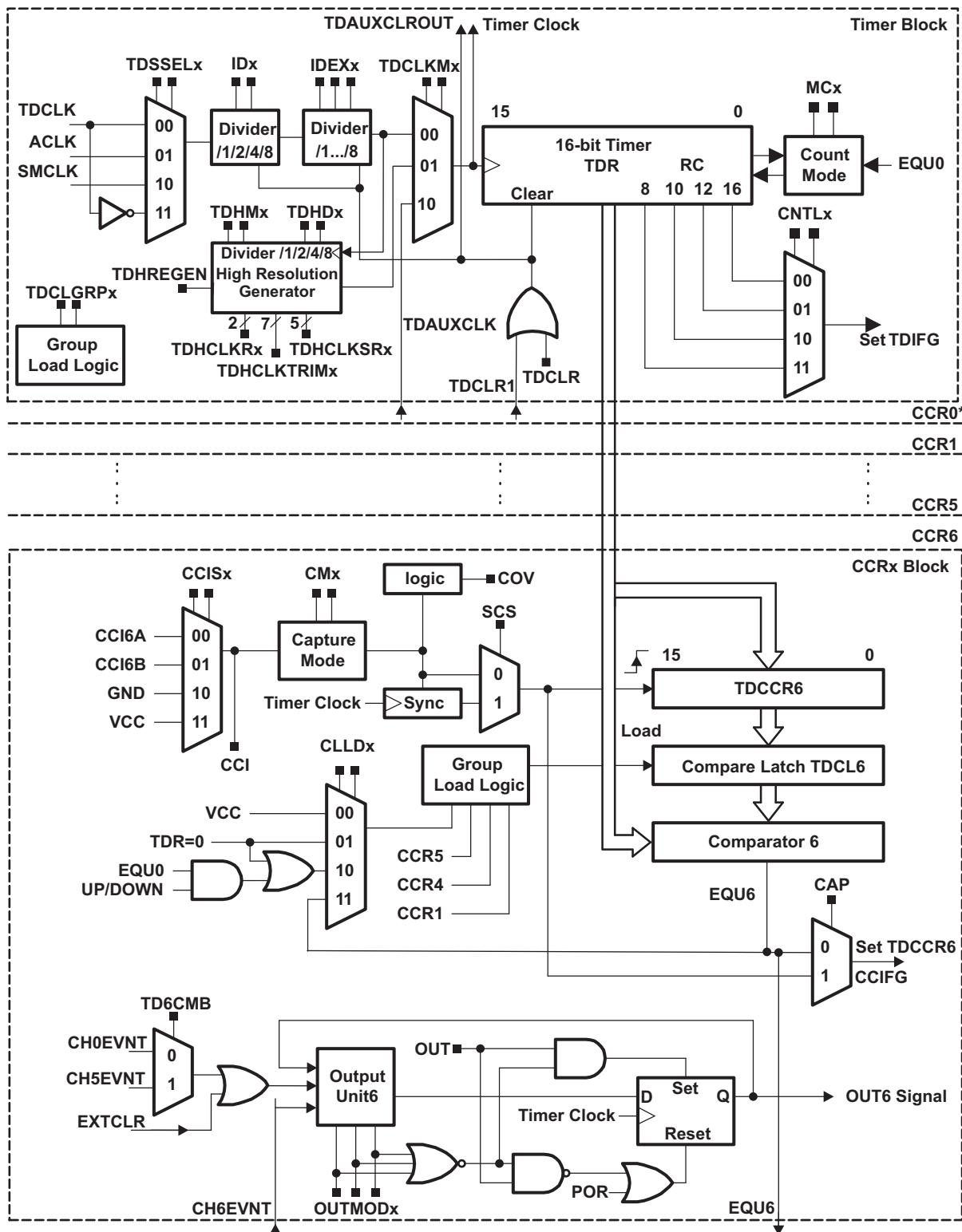
**NOTE: Use of the word *count***

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

### 19.1.1 Differences From Timer\_B

Timer\_D is identical to Timer\_B with the following exceptions:

- Timer\_D supports high-resolution mode.
- Timer\_D supports the combination of two adjacent TDCCR<sub>x</sub> registers in one capture/compare channel.
- Timer\_D supports the dual capture event mode.
- Timer\_D supports external fault input, external clear input, and signal. See the [TEC chapter](#) for detailed information.
- Timer\_D can synchronize with a second timer instance when available. See the [TEC chapter](#) for detailed information.


**Figure 19-1. Timer\_D Block Diagram**

---

**NOTE:** TDxCCR0 is slightly different from the other capture/compare blocks. The input signals to the output unit are:

- OUT0 signal
- EXTCLR signal
- Timer Counter overflow

This enables channel 0 to generate a PWM output signal in Continuous mode. The events to set and reset the PWM output 0 are the CCR0 match and the Timer Counter overflow (TDIFG).

---

**NOTE:** TDCCR<sub>x</sub> channels with odd channel numbers do not have the TDxCMB bit capabilities. Therefore, there is no MUX, and the CH0EVNT is ORed with EXTCLR.

---

## 19.2 Timer\_D Operation

The Timer\_D module is configured with user software. The setup and operation of Timer\_D is discussed in the following sections.

### 19.2.1 16-Bit Timer Counter

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

TDxR may be cleared by any of the following events:

- Writing 1 to TDCLR bit
- A logical high TDCLR1 signal. See the [TEC chapter](#) for details.

When the TDxR is cleared, the timer divider is restarted.

---

**NOTE: Modifying Timer\_D registers**

It is recommended to stop the timer before modifying its operation to avoid errant operating conditions. Stopping the timer is not required for the following cases:

- To enable interrupt
- Setting interrupt flags
- Setting TDCLR bit
- Enabling or disabling calibration in high-resolution mode

When the TDCLK is asynchronous to the CPU clock, any read from TDxR should occur while the timer is not operating, or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TDxR takes effect immediately.

---

#### 19.2.1.1 TDxR Length

Timer\_D is configurable to operate as an 8-, 10-, 12-, or 16-bit timer with the CNTLx bits. The maximum count value, TDR<sub>(max)</sub>, for the selectable lengths is 0FFh, 03FFh, 0FFFh, and 0FFFFh, respectively. Data written to the TDxR register in 8-, 10-, and 12-bit mode is right justified with leading zeros.

#### 19.2.1.2 Clock Source Select and Divider

The TDCLKMx bits of the control register 1 (TDxCTL1) select the timer clock between the timer input clock source, the high-resolution local clock, or the auxiliary clock of another timer instance.

The timer input clock can be sourced from ACLK, SMCLK, or externally via TDCLK or INCLK. The clock source is selected with the TDSSELx bits. The selected clock source may be passed directly to the timer or to a 2-stage division process by using the ID and IDEX bits. Whenever a timer clear event occurs, the clock divider is restarted. The clock divider is also restarted when passing the selected clock to the high-resolution generator.

### 19.2.2 High-Resolution Generator

In high-resolution mode, the high-resolution generator is used. Figure 19-2 shows the high resolution clock generator block diagram. It can be operated in free-running mode or in regulated mode.

In free-running mode ( $\text{TDHREGEN} = 0$ ), the high-resolution clock is selected by the high-resolution clock range selection bits  $\text{TDHCLKRx}$  in the register  $\text{TDxHCTL1}$ . Each clock range is divided into  $2^{\text{TDHCLKSRx}}$  sub-ranges. And in each sub-range, a total number of  $2^{\text{TDHCLKTRIMx}}$  slots can be chosen by configuring the  $\text{TDHCLKTRIMx}$  bits in the  $\text{TDxHCTL1}$  register.

In regulated mode ( $\text{TDHREGEN} = 1$ ), the selected high-resolution generator frequency is adjusted to the timer input clock frequency. The high-resolution clock frequency is 8x ( $\text{TDHMx} = 00$ ) or 16x ( $\text{TDHMx} = 01$ ) higher than the selected timer input clock. The timer input clock frequency can be within the range of 8 MHz to 16 MHz if 16x is selected ( $\text{TDHMx} = 00$ ) or 8 MHz to 25 MHz if 8x ( $\text{TDHMx} = 01$ ) is selected. When the timer input clock frequency is greater than 15 MHz, the  $\text{TDHCLKCR}$  bit in the  $\text{TDxHCTL1}$  register should be set. The  $\text{TDHCLKRx}$  bits are used to preset the frequency range of the high-resolution generator.

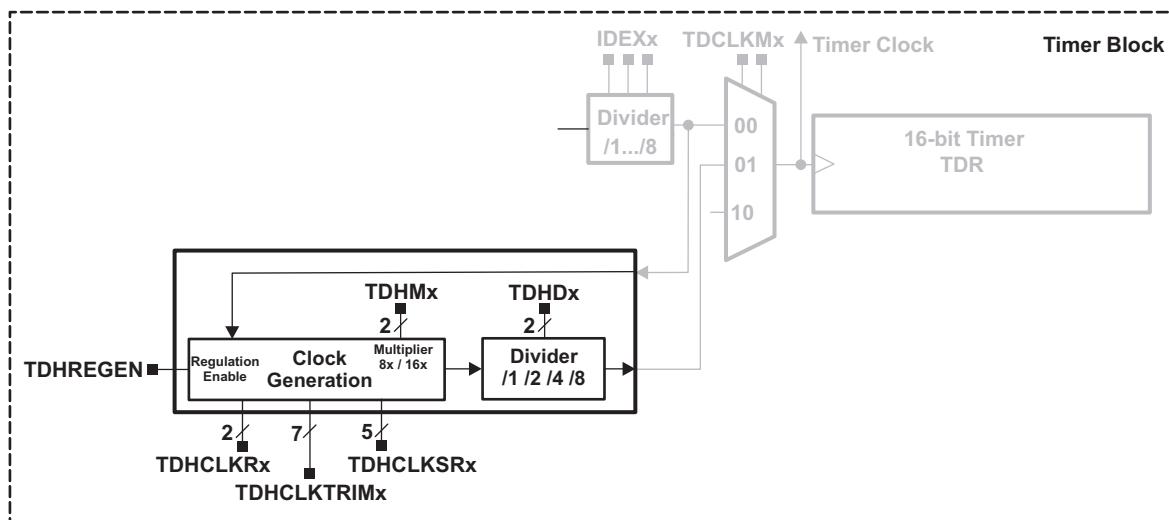


Figure 19-2. High Resolution Clock Generator

#### 19.2.2.1 Free-Running Mode of the High-Resolution Generator

When the high-resolution generator is used in free-running mode, the register  $\text{TDxHCTL1}$  is configured by the user software. Before the high-resolution generator is enabled, the bits  $\text{TDHCLKCR}$ ,  $\text{TDHCLKRx}$ ,  $\text{TDHSRx}$ , and  $\text{TDHCLKTRIMx}$  must be configured. See the device-specific data sheet for details about the frequency selection. After the high-resolution generator is turned on, only the  $\text{TDHCLKTRIMx}$  bits should be changed by +1 or -1 at a time. All other settings must not be altered manually. Using  $\text{TDHCLKTRIMx}$  allows changing the frequency by at least  $\pm 20\%$  from  $\text{TDHCLKTRIM} = 64$ .

There are factory preprogrammed values stored in the TLV structure of the flash memory for the  $\text{TDHCLKRx}$ ,  $\text{TDHSRx}$ , and  $\text{TDHCLKTRIMx}$  registers for specific frequencies. To use the calibrated settings, the register setting is copied into the  $\text{TDHCLKRx}$ ,  $\text{TDHSRx}$ , and  $\text{TDHCLKTRIMx}$  registers. In addition to these settings, a clock multiplication factor and coarse clock range selection must be applied to the  $\text{TDHMx}$  and  $\text{TDHCLKCR}$  bits of the  $\text{TDHCTL0}$  register (see Table 19-1).

Table 19-1. Factory Preprogrammed Frequency and  $\text{TDHMx}$ ,  $\text{TDHCLKCR}$  Bit Settings

Frequency	TLV Label	$\text{TDHMx}$	$\text{TDHCLKCR}$	Description
64	$\text{TDHxCTL1\_64}$	00	0	Local clock generator of Timer_Dx generates a frequency of 64 MHz = 8 x 8 MHz.
128	$\text{TDHxCTL1\_128}$	00	1	Local clock generator of Timer_Dx generates a frequency of 128 MHz = 8 x 16 MHz.

**Table 19-1. Factory Preprogrammed Frequency and TDHMX, TDHCLKCR Bit Settings (continued)**

Frequency	TLV Label	TDHMX	TDHCLKCR	Description
200	TDHxCTL1_200	00	1	Local clock generator of Timer_Dx generates a frequency of 200 MHz = 8 x 25 MHz.
256	TDHxCTL1_256	01	1	Local clock generator of Timer_Dx generates a frequency of 256 MHz = 16 x 16 MHz.

### 19.2.2.2 Change Frequency of High-Resolution Generator (Free-Running Mode)

To change the frequency of the high-resolution generator in free-running mode:

1. Increment or decrement TDHCLKTRIM by 1 until TDHCLKTRIM = 64 is reached. If the clock range must be changed, proceed to Step 1a for a higher clock range or to Step 1b for a lower clock range.
  - (a) If the clock range, TDHCLKRx, must be changed to a higher clock range, then the TDHCLKSR must be brought to TDHCLKSR = 31 by incrementing by 1.
  - (b) If the clock range, TDHCLKRx, must be changed to a lower clock range, then the TDHCLKSR must be brought to TDHCLKSR = 0 by decrementing by 1.
2. Change the clock range, TDHCLKR, after step 1a or 1b. Increment or decrement by 1 at a time.
3. Increment or decrement TDHCLKSR by 1 until the desired clock frequency is reached to an accuracy of approximately  $\pm 3\%$ .
4. Increment or decrement TDHCLKTRIM by 1 until the desired clock frequency is reached to an accuracy of approximately  $\pm 1.5\%$ .

Further changes to track the frequency because of changes in the environment can be done by changing only the TDHCLKTRIM values.

### 19.2.2.3 Regulated Mode of the High-Resolution Generator

In regulated mode, the high-resolution generator produces 8 or 16 equidistant events per timer input clock cycle. Regulation is enabled by setting the high-resolution calibration enable bit TDHREGEN. The high-resolution generator tracks changes of the timer input clock after locking to the timer input clock frequency. Locking is indicated by setting the lock interrupt flag TDHLKIFG. As long as the high-resolution generator is not locked, the interrupt flag TDHUNLKIFG is set.

If the timer input clock is out of the frequency range of the high-resolution generator, then the fail-high interrupt flag (TDHFHIFG) or the fail-low interrupt flag (TDHFLIFG) is set.

If the TDHREGEN bit is cleared, the continuous regulation is stopped and the high-resolution frequency enters free-running mode. The latest settings are kept.

[Example 20-1](#) shows how to set the timer to high-resolution mode.

#### Example 19-1. Set the Timer to High-Resolution Mode

```
; Example: TDCLK=12MHz ; Generate the high-resolution frequency 12x16MHz
;
MOV #TDHCLKCR_0+TDHCLKR_x+TDHSR_x+TDHCLKTRIM_x, &TDxHCTL1 ; pre-selected Clock Range
MOV #TDSSEL_0, &TDxCTL0 ; TDCLK, High-Res input
                      ; clock
MOV #TDCLKM_1, &TDxCTL1 ; Select High-Res Clock
MOV #TDHM_1+TDHREGEN+TDHEN+TDHRON, &TDxHCTL0 ; 16x, Regulation
                                                ; Waiting for TDHLKIFG
                                                ; Int handler Timer_D
                                                ; Add offset to Jump table
                                                ; Vector 0: No Interrupt
                                                ; Other vectors
                                                ; Vector 24: Locked handler
TD_HANDLER ...
ADD &TDxIV, PC
RETI
...
JMP TDHLK_HANDLER ; Begin of Clock Locked handler
...
TDHLK_HANDLER
```

**Example 19-1. Set the Timer to High-Resolution Mode (continued)**

```
MOV #MC_1, &TDxCTL0 ; Up Mode. Start Timer_D
...
RETI
```

**19.2.3 Starting the Timer**

The timer may be started or restarted by any of the following methods:

- The timer counts when MC<sub>x</sub> > 0 and the clock source is active.
- TDHEN = 0: When the timer mode is either up or up/down, the timer may be stopped by loading 0 to TDxCL0. The timer may then be restarted by loading a nonzero value to TDxCL0. In this case, the timer starts incrementing in the up direction from zero.
- TDHEN = 1: When the timer mode is in either up mode or up/down mode, the timer may be stopped by loading 0 to TDxCL0. The timer may then be restarted by loading a nonzero value to TDxCL0. In this case, the timer starts incrementing in the up direction from zero.

**19.2.4 Timer Mode Control**

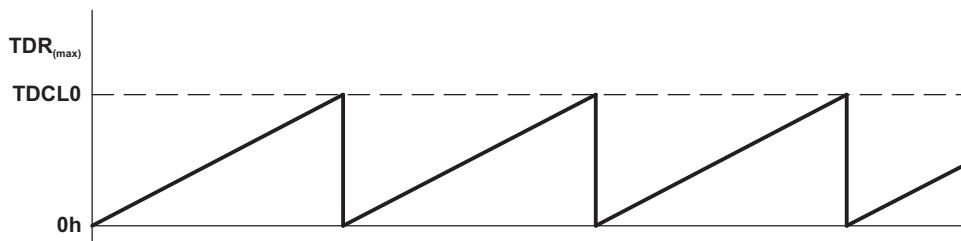
The timer has four modes of operation: stop, up, continuous, and up/down. The operating mode is selected with the MSC<sub>x</sub> bits (see [Table 19-2](#)).

**Table 19-2. Timer Modes**

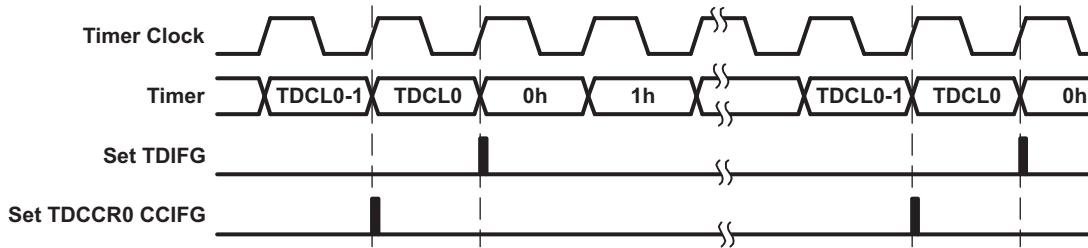
MC <sub>x</sub>	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of compare register TDxCL0.
10	Continuous	The timer repeatedly counts from zero to the value selected by the TDCNTL <sub>x</sub> bits.
11	Up/down	The timer repeatedly counts from zero up to the value of TDxCL0 and then back down to zero.

**19.2.4.1 Up Mode**

The up mode is used if the timer period must be different from TDR<sub>(max)</sub> counts. The timer repeatedly counts up to the value of compare latch TDxCL0, which defines the period (see [Figure 19-3](#)). The number of timer counts in the period is TDCL0 + 1. When the timer value equals TDCL0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TDCL0, the timer immediately restarts counting from zero.

**Figure 19-3. Up Mode**

The TDxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TDCL0 value. The TDIFG interrupt flag is set when the timer *counts* from TDCL0 to zero. [Figure 19-4](#) shows the flag set cycle.



**Figure 19-4. Up Mode Flag Setting**

#### **Changing Period Register TDxCL0**

When changing TDxCL0 while the timer is running and when the TDxCL0 load mode is *immediate*, if the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

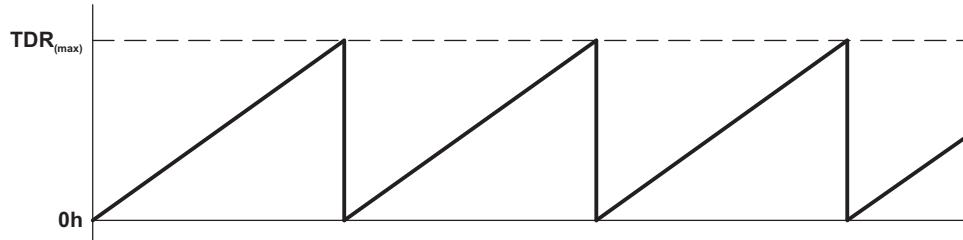
---

**NOTE:** When TDxCL0 is used as period register under high-resolution mode, the four LSB of the TDxCL0 in the 16x case or the three LSB in the 8x case are ignored.

---

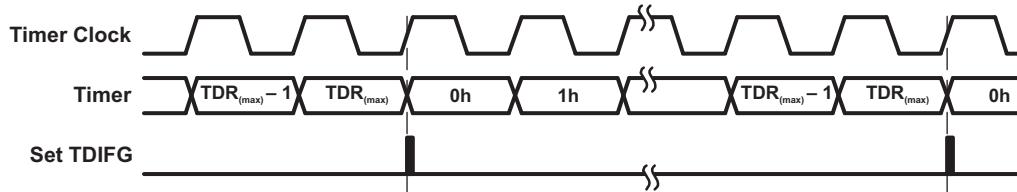
#### **19.2.4.2 Continuous Mode**

In continuous mode, the timer repeatedly counts up to  $TDR_{(max)}$  and restarts from zero (see [Figure 19-5](#)). The compare latch TDxCL0 works the same way as the other capture/compare registers.



**Figure 19-5. Continuous Mode**

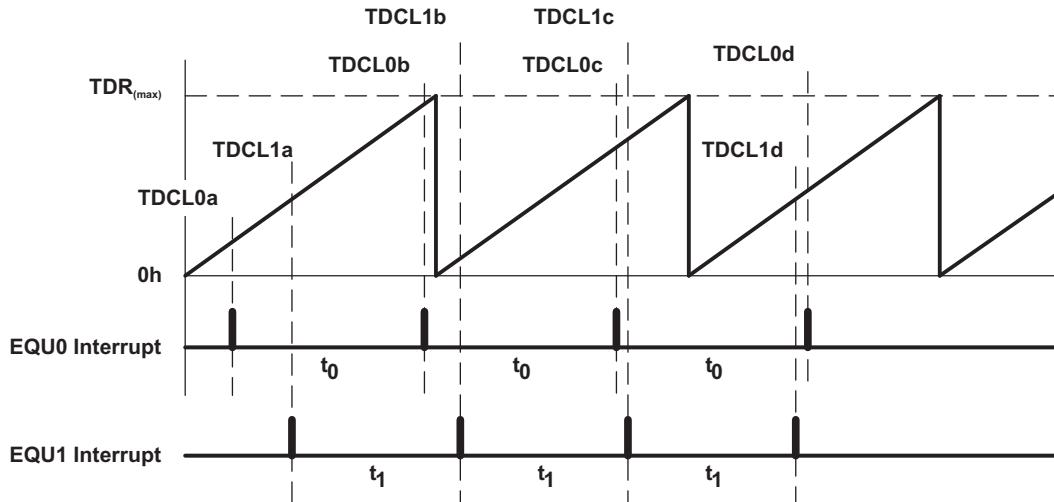
The TDIFG interrupt flag is set when the timer *counts* from  $TDR_{(max)}$  to zero. [Figure 19-6](#) shows the flag set cycle.



**Figure 19-6. Continuous Mode Flag Setting**

### 19.2.4.3 Use of Continuous Mode

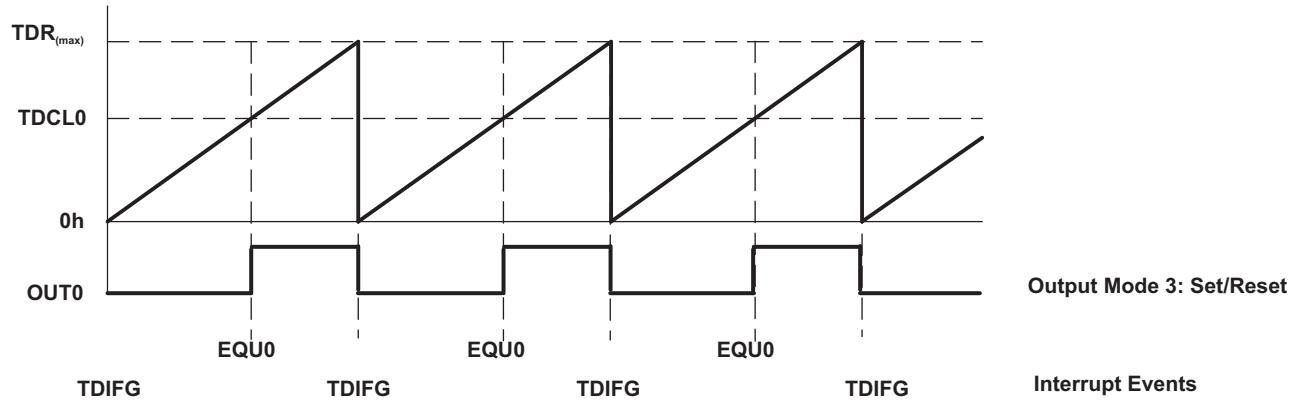
The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TDCLx latch in the interrupt service routine. Figure 19-7 shows two separate time intervals,  $t_0$  and  $t_1$ , being added to the capture/compare registers. The time interval is controlled by hardware, not software, without impact from interrupt latency. Up to three (for Timer\_D3) or 7 (for Timer\_D7) independent time intervals or output frequencies can be generated using capture/compare registers.



**Figure 19-7. Continuous Mode Time Intervals**

Time intervals can be produced with other modes as well, using TDxCL0 as the period register. Their handling is more complex, because the sum of the old TDCLx data and the new period can be higher than the TDxCL0 value. When the sum of the previous TDCLx value plus  $t_x$  is greater than the TDCL0 data, the old TDCL0 value must be subtracted to obtain the correct time interval.

PWM output can be generated in capture/compare channel 0 by using the timer overflow signal together with TDCL0. Figure 19-8 shows an example.



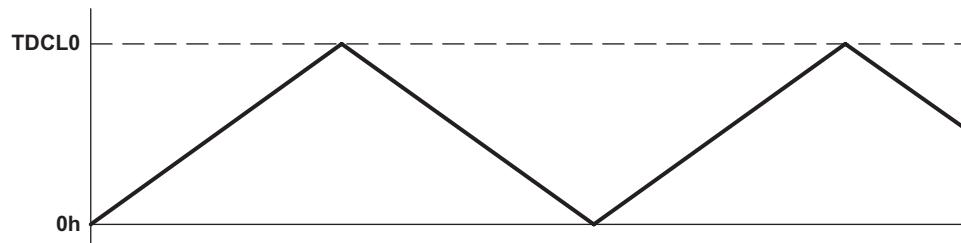
**Figure 19-8. TDxCCR0 PWM Generation Under Continuous Mode**

### 19.2.4.4 Up/Down Mode

The up/down mode is used if the timer period must be different from  $TDR_{(max)}$  counts and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare latch TDxCL0, and back down to zero (see Figure 19-9). The period is twice the value in TDCL0.

**NOTE: TDCL0 > TDR<sub>(max)</sub>**

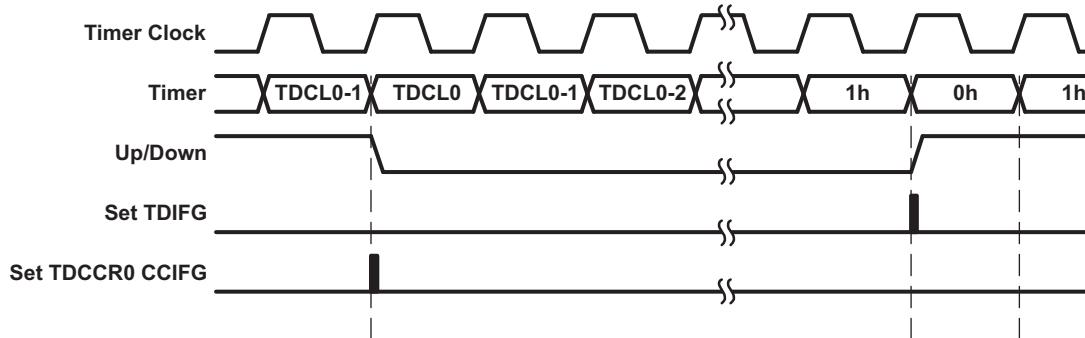
If TDCL0 > TDR<sub>(max)</sub>, the counter operates as if it were configured for continuous mode. It does not count down from TDR<sub>(max)</sub> to zero.



**Figure 19-9. 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 TDCLR bit must be used to clear the direction. The TDCLR bit also clears the TDR value and the TDCLK divider.

In up/down mode, the TDxCCR0 CCIFG interrupt flag and the TDIFG interrupt flag are set only once during the period, separated by one-half the timer period. The TDxCCR0 CCIFG interrupt flag is set when the timer counts from TDCL0 – 1 to TDCL0, and TDIFG is set when the timer completes *counting* down from 0001h to 0000h. [Figure 19-10](#) shows the flag set cycle.



**Figure 19-10. Up/Down Mode Flag Setting**

#### Changing the Value of Period Register TDxCL0

When changing TDxCL0 while the timer is running and counting in the down direction, and when the TDxCL0 load mode is *immediate*, the timer continues its descent until it reaches zero. The new period takes effect after the counter counts down to zero.

If the timer is counting in the up direction when the new period is latched into TDxCL0, and the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction and the new period is less than the current count value when TDxCL0 is loaded, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

##### 19.2.4.5 Use of Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see [Section 19.2.9](#)). 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 19-11](#), the t<sub>dead</sub> is:

$$t_{\text{dead}} = t_{\text{timer}} \times (TDCL1 - TDCL3)$$

Where:

$t_{\text{dead}}$  = Time during which both outputs need to be inactive

$t_{\text{timer}}$  = Cycle time of the timer clock

TDCLx = Content of compare latch x

The ability to simultaneously load grouped compare latches ensures the dead times.

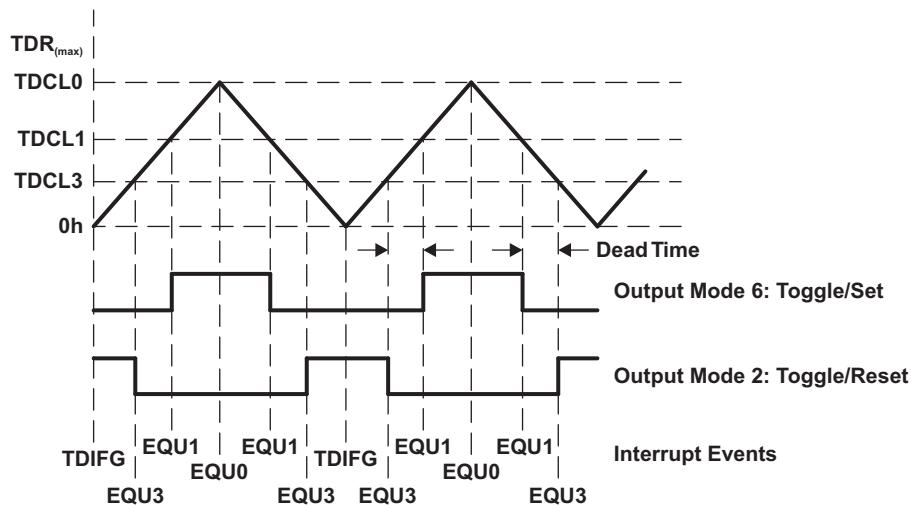


Figure 19-11. Output Unit in Up/Down Mode

### 19.2.5 PWM Generation

The previous examples have shown that PWM output can be produced by using a single TDCCR<sub>x</sub>. In TDxCCR0, for example, either the falling or the rising edge of the PWM signal is controlled by the TDxCCR0 or the period overflow (see [Figure 19-8](#)). In the other TDCCR<sub>x</sub> ( $x \neq 0$ ), PWM can be generated by setting the output to up/down mode and using the TDCCR<sub>x</sub> register to control both the rising and the falling edges of the signal (see [Figure 19-11](#)). However, this cannot always meet the application requirements; for example, in motor control applications.

The control bits, TDxCMB of the TDxCTL1 register, are implemented so that two neighbor TDCCR registers can be used for one channel to control both edges of one PWM output signal. If TDxCMB is set, TDCCR<sub>x-1</sub> and TDCCR<sub>x</sub> are combined to channel TD<sub>x</sub>; for example, TDxCCR1 and TDxCCR2 for TD2 output, TDxCCR3 and TDxCCR4 for TD4 output, and TDxCCR5 and TDxCCR6 for TD6. By configuring TDxCMB bit, a Timer\_D can then generate either two fully controlled PWM signals or four independent channels.

The PWM duty cycle can be zero. The smallest pulse width depends on the TDHEN and TDHM<sub>x</sub> settings.

- In normal timer mode, TDHEN = 0, TDHM<sub>x</sub> = don't care: CCR<sub>x</sub> = 1
- In high-resolution timer mode, 8x reference frequency, TDHEN = 1, TDHM<sub>x</sub> = 00: CCR<sub>x</sub> = 0x0010, 8 high-resolution cycles
- In high-resolution timer mode, 16x reference frequency, TDHEN = 1, TDHM<sub>x</sub> = 01: CCR<sub>x</sub> = 0x0020, 16 high-resolution cycles

[Figure 19-12](#) shows an example.

[Table 19-3](#) and [Table 19-4](#) summarize the duty cycle limitations of the Timer\_D. Duty cycles close to 0% and 100% should be avoided. [Table 19-3](#) and [Table 19-4](#) show how the output signal behaves in case the TDCCR<sub>x</sub> register is programmed to values beyond the recommended range of values dependent on the OUTMOD<sub>x</sub> setting.

**Table 19-3. High-Resolution Mode Limitation (TDHEN = 1) - Minimum Duty Cycle**

OUTMODx	Mode	Condition	Description
000	Output	n/a	n/a
001	Set	TDHMX = 0: TDCCR <sub>x</sub> < 0x0007 TDHMX = 1: TDCCR <sub>x</sub> < 0x000F	Output is not set.
010	Toggle/Reset	-	Not recommended.
011	Set/Reset	TDHMX = 0: TDCCR <sub>x</sub> < 0x0007 TDHMX = 1: TDCCR <sub>x</sub> < 0x000F	Result is 100% duty cycle.
100	Toggle	-	Not recommended.
101	Reset	TDHMX = 0: TDCCR <sub>x</sub> < 0x0007 TDHMX = 1: TDCCR <sub>x</sub> < 0x000F	Output is not reset.
110	Toggle/Set	-	Not recommended.
111	Reset/Set	TDHMX = 0: TDCCR <sub>x</sub> < 0x0007 TDHMX = 1: TDCCR <sub>x</sub> < 0x000F	Result is 0% duty cycle.

**Table 19-4. High-Resolution Mode Limitation (TDHEN = 1) - Maximum Duty Cycle**

OUTMODx	Mode	Condition	Description
000	Output	n/a	n/a
001	Set	TDCCR <sub>x</sub> > TDCCR <sub>0</sub> - 0x0008	Output is not set.
010	Toggle/Reset	-	Not recommended.
011	Set/Reset	TDCCR <sub>x</sub> > TDCCR <sub>0</sub> - 0x0008	Result is 100% duty cycle.
100	Toggle	-	Not recommended.
101	Reset	TDCCR <sub>x</sub> > TDCCR <sub>0</sub> - 0x0008	Output is not reset.
110	Toggle/Set	-	Not recommended.
111	Reset/Set	TDCCR <sub>x</sub> > TDCCR <sub>0</sub> - 0x0008	Result is 0% duty cycle.

In high-resolution compare mode there are limitations of the minimum and maximum duty cycles.

Minimum duty cycle:

TDHMX = 0: TDCCR<sub>x</sub> < 8 are set to TDCCR<sub>x</sub> = 0; that is, return 0 duty cycle.

TDHMX = 1: TDCCR<sub>x</sub> < 16 are set to TDCCR<sub>x</sub> = 0; that is, return 0 duty cycle.

Maximum duty cycle:

TDHMX = 0 or 1: The maximum duty cycle is limited to TDCCR<sub>x</sub> < TDCCR<sub>0</sub> - 0x000F. Values of TDCCR<sub>x</sub> greater than TDCCR<sub>0</sub> - 0x000F lead to 0 duty cycle.

---

**NOTE:** In high-resolution mode, when the two CCR<sub>x</sub> channels are combined by setting TDxCMB = 1, the minimum difference between the output compare registers (CCR<sub>x</sub> and CCR<sub>x</sub> + 1 must be:

if TDHMX = 00: | TDCCR<sub>x</sub> - TDCCR<sub>x-1</sub> | > 7

if TDHMX = 01: | TDCCR<sub>x</sub> - TDCCR<sub>x-1</sub> | > 15

---

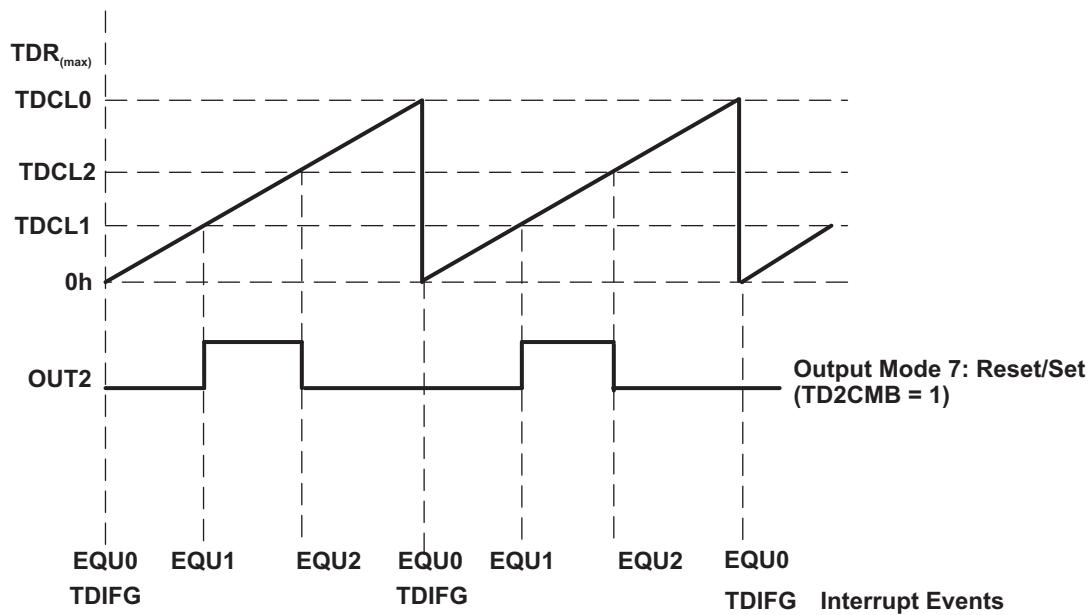


Figure 19-12. Controlling Rising and Falling Edge of PWM Output in Up Mode

Example 19-2 shows how to set up the timer for the case shown in Figure 19-12.

### Example 19-2.

```

MOV #TD2CMB+TDCLKM_0, &TDxCTL1           ; Combine TDxCCR1&TDxCCR2,
                                            ; External Clock
MOV #OUTMOD_7, &TDxCCTL2                 ; TDxCCR2 Reset/Set
MOV 0x80, &TDxCCTL0                     ; PWM Period TDxCCR0 128
MOV 0x28, &TDxCCTL1                     ; TDxCCR1 is 40, Set
MOV 0x50, &TDxCCTL2                     ; TDxCCR2 is 80, Reset
MOV #TDSSEL_1+MC_1, &TDxCTL0             ; ACLK, Up Mode
BIS #CPUOFF,SR                           ; Enter LPM0

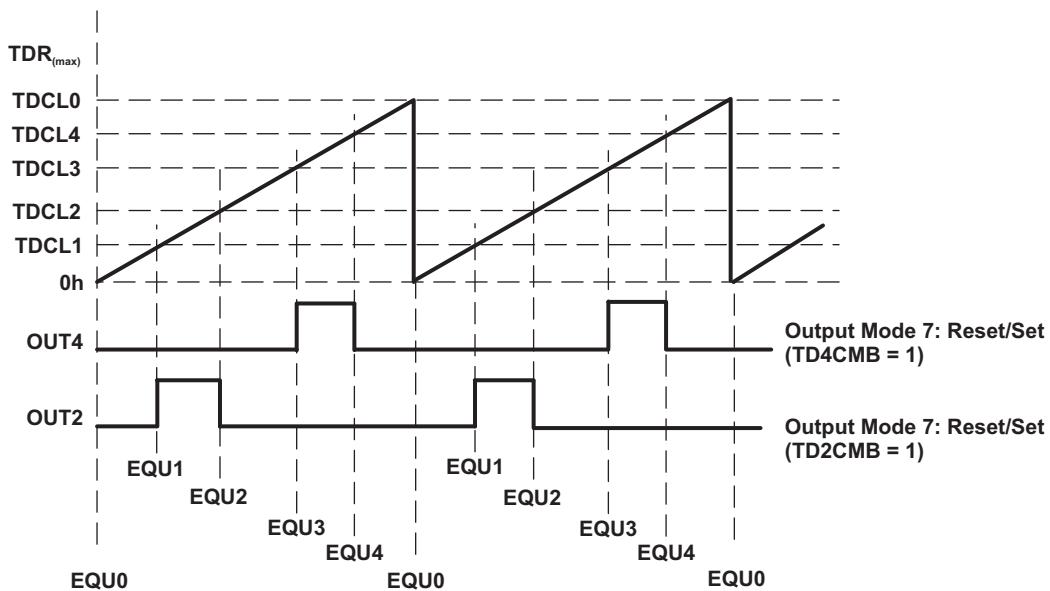
```

---

**NOTE:** Channels TD1/3/5 are still controlled by TDxCCR1/3/5 even when the TDxCCR<sub>x</sub> registers are paired.

---

Combining two pairs of CCR<sub>x</sub> channels to two PWM outputs can be used to generate non-overlapping PWM channels.



**Figure 19-13. Deadband Generation (TDxCMB = 1)**

### 19.2.6 Capture/Compare Blocks

Several identical capture/compare blocks, TDCCRx, are present in Timer\_D. Any of the blocks may be used to capture the timer data or to generate time intervals. See the device-specific data sheet for the number of capture/compare blocks available.

#### 19.2.6.1 Capture Mode

The capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CC<sub>IxA</sub> and CC<sub>IxB</sub> are connected to external pins or internal signals and are selected with the CCIS<sub>x</sub> bits. The CM<sub>x</sub> bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture is performed:

- The timer value is copied into the TDCCRx register.
- The TDCCRx register is copied into the TDCLx register.
- The interrupt flag CCIFG is set.
- The Capture Overflow bit is set at a capture overflow condition.

The input signal level can be read at any time via the CCI bit. Devices may have different signals connected to CC<sub>IxA</sub> and CC<sub>IxB</sub>. See the device-specific data sheet for the connections of these signals.

The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit synchronizes the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended (see [Figure 19-14](#)). In high-resolution mode, the SCS bit is ignored.

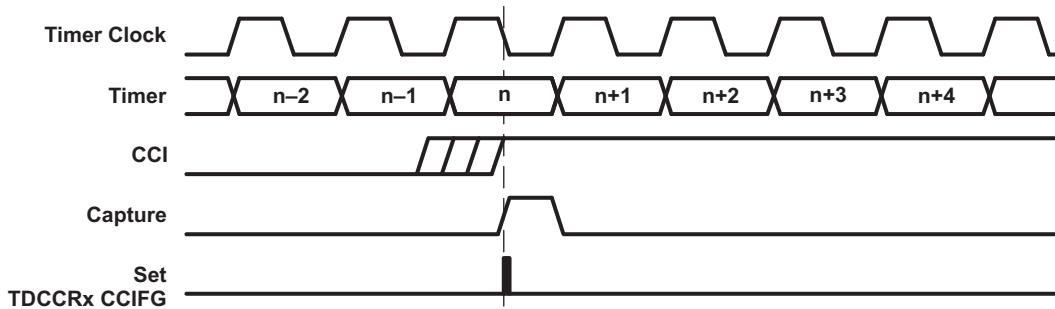


Figure 19-14. Capture Signal (SCS = 1)

Overflow logic is provided in each capture/compare register to indicate a capture overflow condition.

#### Single Capture Mode ( $TDCAPMx = 0$ )

Each input capture channel can be operated in single capture (compatibility) mode or in dual capture mode. The single capture mode is selected by  $TDCAPMx = 0$ . In single capture mode, the content of register TDCCR $x$  is shifted to TDCL $x$  register at each capture event. In single capture mode, the capture overflow bit (COV) is set if a capture events occurs before the TDCCR $x$  register has been read (see Figure 19-15). COV must be reset by software.

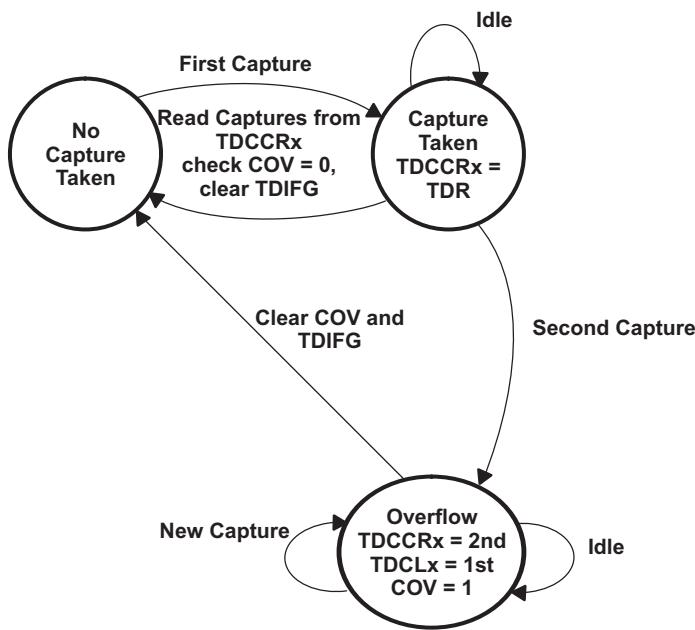
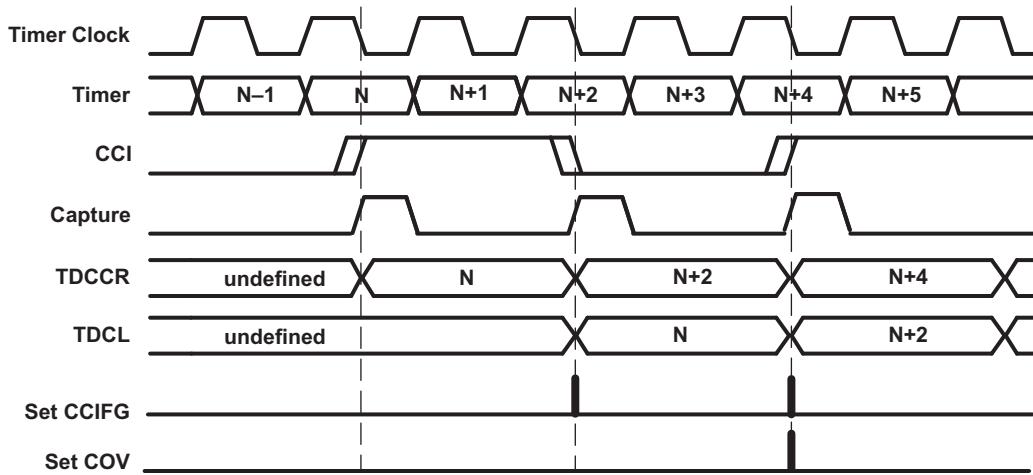


Figure 19-15. Single Capture Cycle

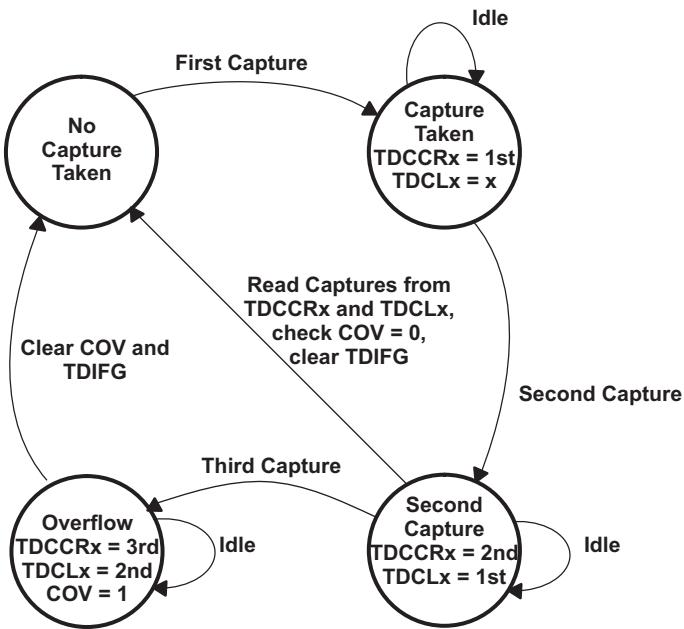
#### Dual Capture Mode ( $TDCAPMx = 1$ )

Each input capture channel can be operated in single capture (compatibility) mode or in dual capture mode. The dual capture mode is selected by  $TDCAPMx = 1$ . In dual capture mode, the register TDCCR $x$  content is shifted to the TDCL $x$  register per each capture event; that is, the second capture event shifts the first capture value from TDCCR $x$  into the TDCL $x$  register. Then TDCCR $x$  holds the second capture value while TDCL $x$  holds the first, and the CCIFG interrupt flag is set. To make sure there is no earlier captured value in the TDCCR $x$  and TDCL $x$  registers these registers should be read prior to using the dual capture mode. The COV bit is set at the third capture event if both TDCCR $x$  and TDCL $x$  have not been read prior to the third capture event (see Figure 19-16). The third capture event again shifts the TDCCR $x$  register content to the TDCL $x$  register. The first capture value is no longer available.



**Figure 19-16. Sequential Capture Events in Dual Capture Mode**

The COV bit is set at the third capture if TDCCRx and TDCLx are not read prior to the third capture event (see Figure 19-17). Checking for COV = 0 after reading the capture registers TDCCRx and TDCLx validates the register content.



**Figure 19-17. COV in Dual Capture Mode**

### Capture Initiated by Software

Captures can be initiated by software. The CM<sub>x</sub> bits can be set for capture on both edges. Software then sets bit CCIS1 = 1 and toggles bit CCIS0 to switch the capture signal between V<sub>CC</sub> and GND, initiating a capture each time CCIS0 changes state:

```
MOV #CAP+SCS+CCIS1+CM_3 ,&TDxCCTLx ; Set up TDxCCTLx
XOR #CCIS0 ,&TDCCRx ; TDCCRx = TDR
```

### 19.2.7 Compare Mode

The compare mode is selected when CAP = 0. Compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TDxR *counts* to the value in a TDCLx:

- Interrupt flag CCIFG is set.
- Internal signal EQUx = 1.
- EQUx affects the output according to the output mode.

#### 19.2.7.1 Compare Latch TDCLx

The TDCCR<sub>x</sub> compare latch, TDCL<sub>x</sub>, holds the data for the comparison to the timer value in compare mode. TDCL<sub>x</sub> is buffered by TDCCR<sub>x</sub>. The buffered compare latch gives the user control over when a compare period updates. Compare data is written to each TDCCR<sub>x</sub> and automatically transferred to TDCL<sub>x</sub>. The timing of the transfer from TDCCR<sub>x</sub> to TDCL<sub>x</sub> is user selectable by setting the CLLD<sub>x</sub> bits as described in [Table 19-5](#).

**Table 19-5. TDCLx Load Events**

CLLD <sub>x</sub>	Description
00	New data is transferred from TDCCR <sub>x</sub> to TDCL <sub>x</sub> immediately when TDCCR <sub>x</sub> is written to.
01	New data is transferred from TDCCR <sub>x</sub> to TDCL <sub>x</sub> when TDxR <i>counts</i> to 0.
10	New data is transferred from TDCCR <sub>x</sub> to TDCL <sub>x</sub> when TDxR <i>counts</i> to 0 for up and continuous modes. New data is transferred from TDCCR <sub>x</sub> to TDCL <sub>x</sub> when TDxR <i>counts</i> to the old TDCL0 value or to 0 for up/down mode.
11	New data is transferred from TDCCR <sub>x</sub> to TDCL <sub>x</sub> when TDxR <i>counts</i> to the old TDCL <sub>x</sub> value.

#### 19.2.7.2 Grouping Compare Latches

Multiple compare latches may be grouped together for simultaneous updates with the TDCLGRP<sub>x</sub> bits. When using groups, the CLLD<sub>x</sub> bits of the lowest numbered TDCCR<sub>x</sub> in the group determine the load event for each compare latch of the group, except when TDCLGRP = 3 (see [Table 19-6](#)). The CLLD<sub>x</sub> bits of the controlling TDCCR<sub>x</sub> must not be set to zero. When the CLLD<sub>x</sub> bits of the controlling TDCCR<sub>x</sub> are set to zero, all compare latches update immediately when their corresponding TDCCR<sub>x</sub> is written; no compare latches are grouped.

Two conditions must exist for the compare latches to be loaded when grouped. First, all TDCCR<sub>x</sub> registers of the group must be updated, even when new TDCCR<sub>x</sub> data equals the old TDCCR<sub>x</sub> data. Second, the load event must occur.

**Table 19-6. Compare Latch Operating Modes**

TDCLGRP <sub>x</sub>	Grouping	Update Control
00	None	Individual
01	TDxCL1+TDxCL2 TDxCL3+TDxCL4 TDxCL5+TDxCL6	TDxCCR1 TDxCCR3 TDxCCR5
10	TDxCL1+TDxCL2+TDxCL3 TDxCL4+TDxCL5+TDxCL6	TDxCCR1 TDxCCR4
11	TDxCL0+TDxCL1+TDxCL2+TDxCL3+TDxCL4+TDxCL5+TDxCL6	TDxCCR1

### **19.2.8 Switching From Capture to Compare Mode**

As discussed in [Section 19.2.7](#), the TDCLx register holds the first captured value when a second capture is taking place. Therefore, when the capture/compare block switches from capture mode to compare mode, TDCLx may have a non-zero initial value. It is important to clear the TDCLx before use or to update the TDCLx immediately when TDCCR<sub>x</sub> is written; that is, set CLLD<sub>x</sub> = 0x0.

### **19.2.9 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 EQUx signals. The TDOUTH pin function can be used to put all Timer\_D outputs into a high-impedance state. When the TDOUTH pin function is selected for the pin (corresponding PSEL bit is set, and port configured as input) and when the pin is pulled high, all Timer\_D outputs are in a high-impedance state.

#### **19.2.9.1 Output Modes**

The output modes are defined by the OUTMOD<sub>x</sub> bits and are described in [Table 19-7](#). The OUTx signal is changed with the rising edge of the timer clock for all modes except mode 0. In the term TDyR period match is used. The period match in continuous mode is defined by the counter length (TDyCNTL<sub>x</sub>), and in Up mode and Up/Down mode by TDyCL0 register.

**Table 19-7. Output Modes**

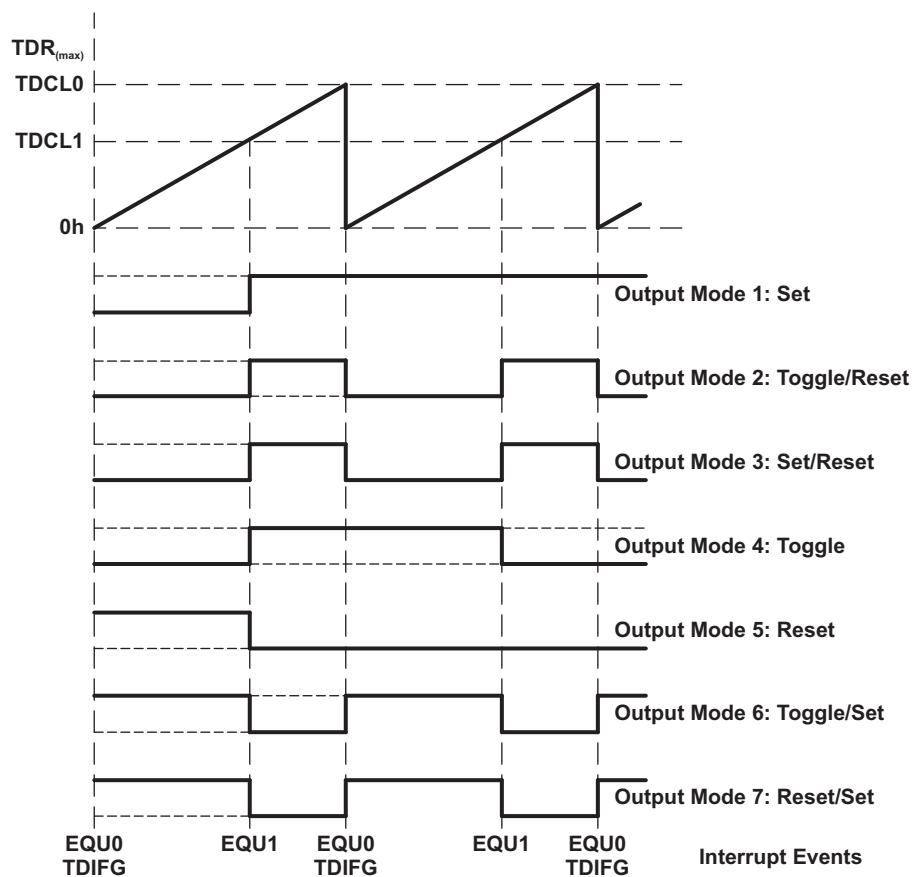
OUTMOD <sub>x</sub>	Mode	Description
000	Output	The output signal OUT <sub>x</sub> is defined by the OUT <sub>x</sub> bit. The OUT <sub>x</sub> signal updates immediately when OUT <sub>x</sub> is updated.
001	Set	Set events: Timer counts to TDyCL <sub>x</sub> , external fault (TECyFLTx). It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	TDxCMB=0: Toggle events: Timer counts to TDyCL <sub>x</sub> , external fault (TECyFLTx). Reset events: Timer counts to TDyCL <sub>x</sub> , TDyR period match, external fault (TECyFLT0), external clear (TECyCLR). TD2CMB=1: Toggle events: Timer counts to TDyCL2, external fault (TECyFLT2). Reset events: Timer counts to TDyCL1, external fault (TECyFLT1), external clear (TECyCLR). TD4CMB=1: Toggle events: Timer counts to TDyCL4, external fault (TECyFLT4). Reset events: Timer counts to TDyCL3, external fault (TECyFLT3), external clear (TECyCLR). TD6CMB=1: Toggle events: Timer counts to TDyCL6, external fault (TECyFLT6). Reset events: Timer counts to TDyCL5, external fault (TECyFLT5), external clear (TECyCLR).
011	Set/Reset	TDxCMB=0: Set events: Timer counts to TDyCL <sub>x</sub> , external fault (TECyFLTx). Reset events: Timer counts to TDyCL <sub>x</sub> , TDyR period match, external fault (TECyFLT0), external clear (TECyCLR). TD2CMB=1: Set events: Timer counts to TDyCL2, external fault (TECyFLT2). Reset events: Timer counts to TDyCL1, external fault (TECyFLT1),, external clear (TECyCLR). TD4CMB=1: Set events: Timer counts to TDyCL4, external fault (TECyFLT4). Reset events: Timer counts to TDyCL3, external fault (TECyFLT3), external clear (TECyCLR). TD6CMB=1: Set events: Timer counts to TDyCL6, external fault (TECyFLT6). Reset events: Timer counts to TDyCL5, external fault (TECyFLT5), external clear (TECyCLR).
100	Toggle	Toggle events: Timer counts to TDyCL <sub>x</sub> , external fault (TECyFLTx). The output period is twice the timer period.
101	Reset	Reset events: Timer counts to TDyCL <sub>x</sub> , external fault (TECyFLTx). It remains reset until another output mode is selected and affects the output.

**Table 19-7. Output Modes (continued)**

<b>OUTMODx</b>	<b>Mode</b>	<b>Description</b>
110	Toggle/Set	<p>TDxCMB=0:            Toggle events: Timer counts to TDyCLx, external fault (TECyFLT<sub>x</sub>).            Set events: Timer counts to TDyCLx, TDyR period match, external fault (TECyFLT0), external clear (TECyCLR).            TD2CMB=1:            Toggle events: Timer counts to TDyCL2, external fault (TECyFLT2).            Set events: Timer counts to TDyCL1, external fault (TECyFLT1), external clear (TECyCLR).            TD4CMB=1:            Toggle events: Timer counts to TDyCL4, external fault (TECyFLT4).            Set events: Timer counts to TDyCL3, external fault (TECyFLT3), external clear (TECyCLR).            TD6CMB=1:            Toggle events: Timer counts to TDyCL6, external fault (TECyFLT6).            Set events: Timer counts to TDyCL5, external fault (TECyFLT5), external clear (TECyCLR).TDxCMB=0:</p>
111	Reset/Set	<p>TDxCMB=0:            Reset events: Timer counts to TDyCLx, external fault (TECyFLT<sub>x</sub>).            Set events: Timer counts to TDyCLx, TDyR period match, external fault (TECyFLT0), external clear (TECyCLR).            TD2CMB=1:            Reset events: Timer counts to TDyCL2, external fault (TECyFLT2).            Set events: Timer counts to TDyCL1, external fault (TECyFLT1), external clear (TECyCLR).            TD4CMB=1:            Reset events: Timer counts to TDyCL4, external fault (TECyFLT4).            Set events: Timer counts to TDyCL3, external fault (TECyFLT3), external clear (TECyCLR).            TD6CMB=1:            Reset events: Timer counts to TDyCL6, external fault (TECyFLT6).            Set events: Timer counts to TDyCL5, external fault (TECyFLT5), external clear (TECyCLR).TDxCMB=0:</p>

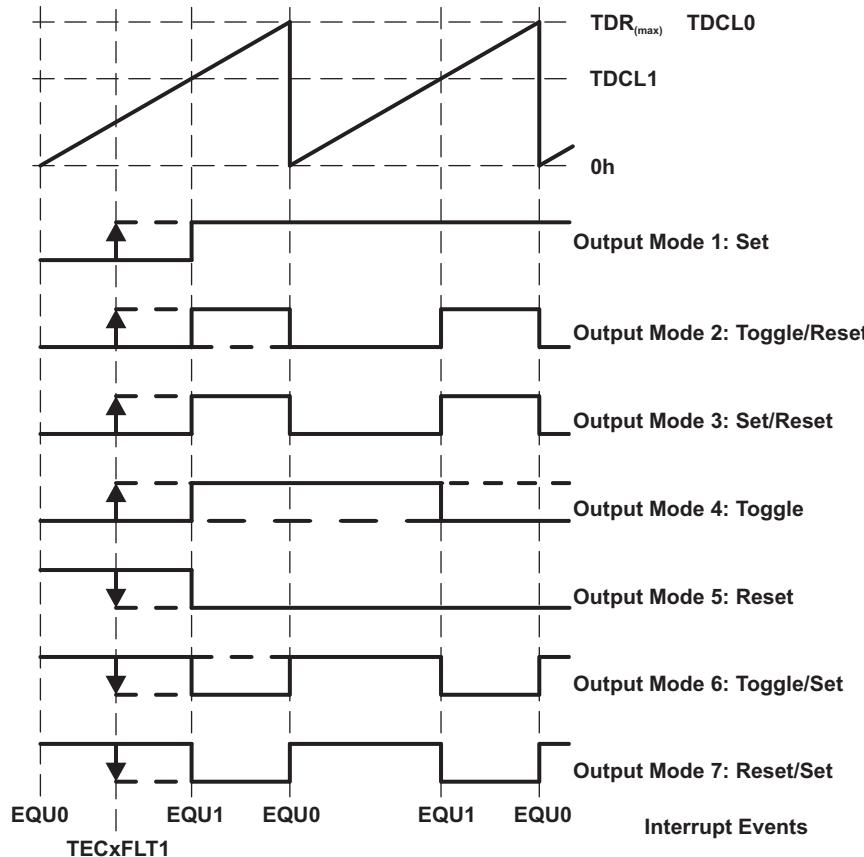
**Output Example – Timer in Up Mode**

The OUTx signal is changed when the timer *counts* up to the TDCLx value, and rolls from TDCL0 to zero, depending on the output mode. An example is shown in [Figure 19-18](#) using TDxCL0 and TDxCL1.



**Figure 19-18. Output Example, Channel 1 – Timer in Up Mode**

The OUTx signal is changed when the timer *counts* up to the TDCLx value, and rolls from TDCL0 to zero, depending on the output mode. The activity selected for the **TDCCR<sub>x</sub>** match event (TDR = TDCL<sub>x</sub>) occurs at the point in time where the **external fault** event happens. An example is shown in [Figure 19-19](#) using TDxCL0 and TDxCL1.



**Figure 19-19. Output Example, Channel 1 - Timer in Up Mode With External Fault Signal**

The OUTx signal is changed when the timer *counts* up to the TDCLx value, and rolls from TDCL0 to zero, depending on the output mode. The activity selected for the **period match** event (TDR = TDCL0) occurs at the point in time when the external **clear event** happens. The external clear event restarts the timer counter. As a consequence, the next period starts earlier, and all following events happen earlier as well. An example is shown in [Figure 19-20](#) using TDxCL0 and TDxCL1.

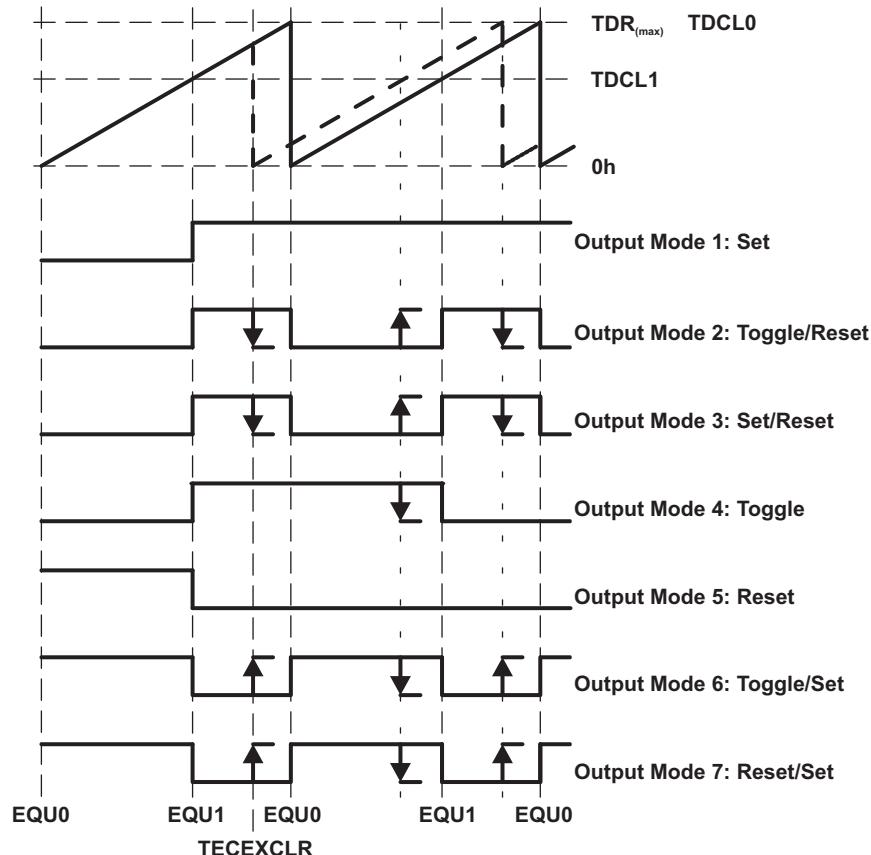
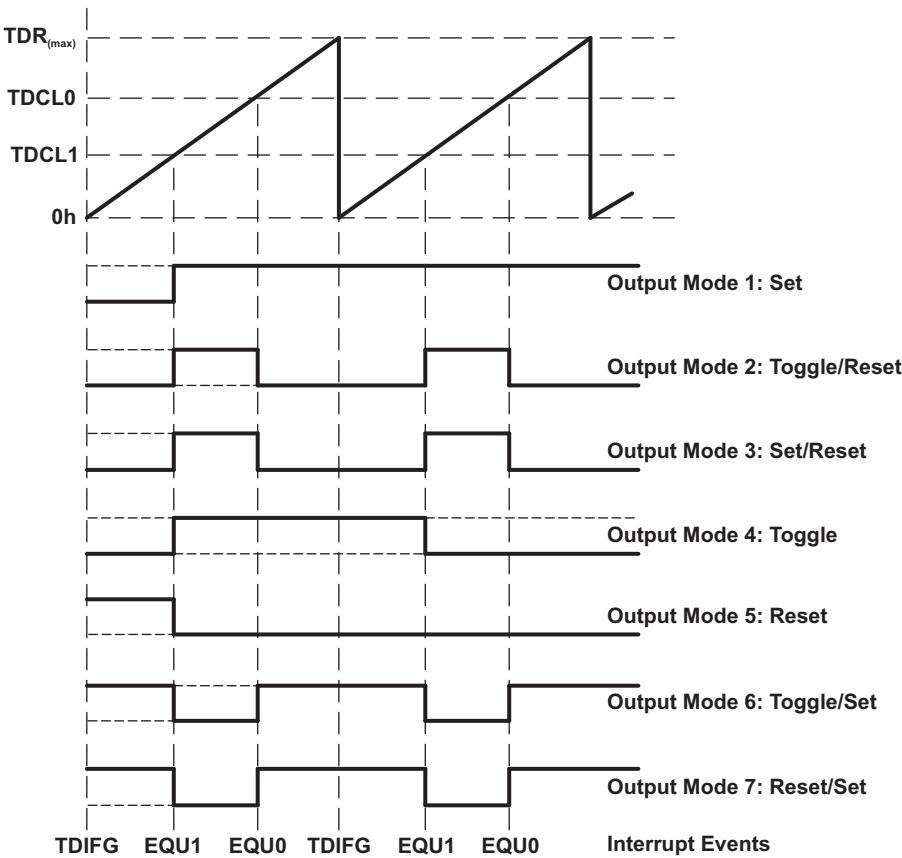


Figure 19-20. Output Example - Timer in Up Mode with External Timer Clear Signal

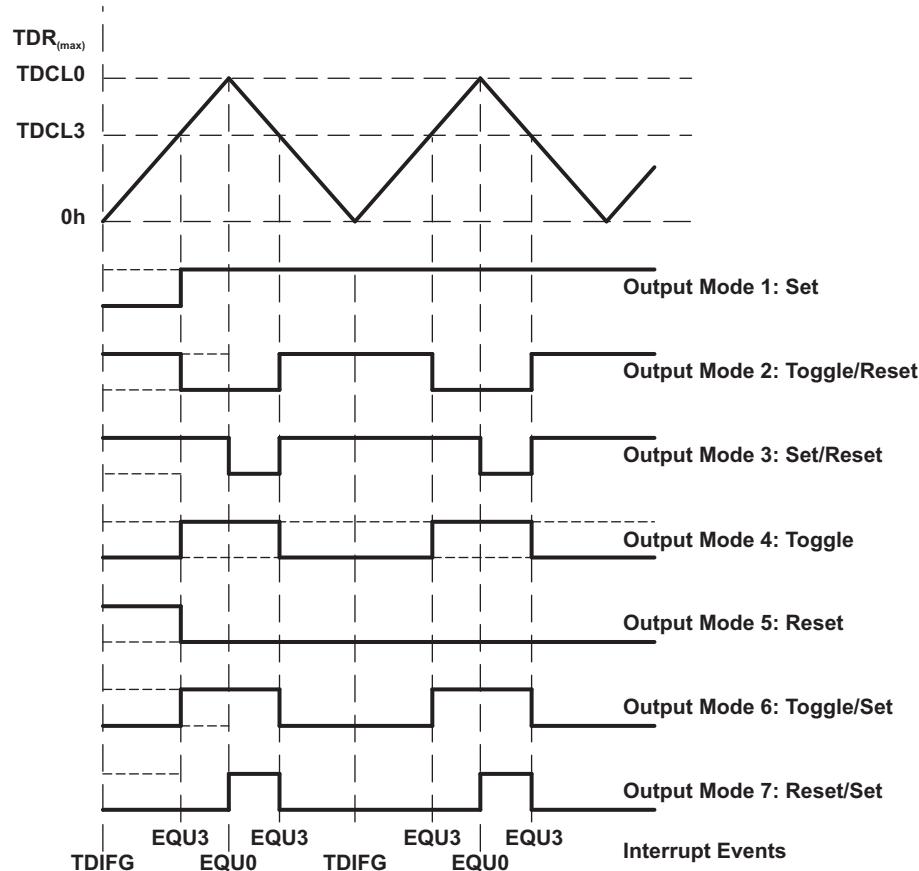
**Output Example – Timer in Continuous Mode**

The OUTx signal is changed when the timer reaches the TDCLx and TDCL0 values, depending on the output mode. An example is shown in Figure 19-21 using TDxCL0 and TDxCL1. The external fault and external clear signals have the same impact to the output signals as in Up mode as shown in and .

**Figure 19-21. Output Example – Timer in Continuous Mode**

***Output Example – Timer in Up/Down Mode***

The OUTx signal changes when the timer equals TDCLx in either count direction and when the timer equals TDCL0, depending on the output mode. An example is shown in [Figure 19-22](#) using TDxCL0 and TDxCL3.



**Figure 19-22. Output Example – Timer in Up/Down Mode**

**NOTE: Switching between output modes**

When switching between output modes, one of the OUTMODx 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 ,&TDCCTLx ; Set output mode=7
BIC #OUTMODx ,&TDCCTLx ; Clear unwanted bits
```

### 19.2.10 Synchronization Between Timer\_D Instances

For some devices that contain more than one Timer\_D instance, internal signals assist the synchronization between different timers. Those signals include Timer Clock, TDAUXCLK, TDAUXCLROUT, and some other signals that connect the Timer\_D module and the Timer Event Control module. See the [TEC chapter](#) for details.

### 19.2.11 Timer\_D Interrupts

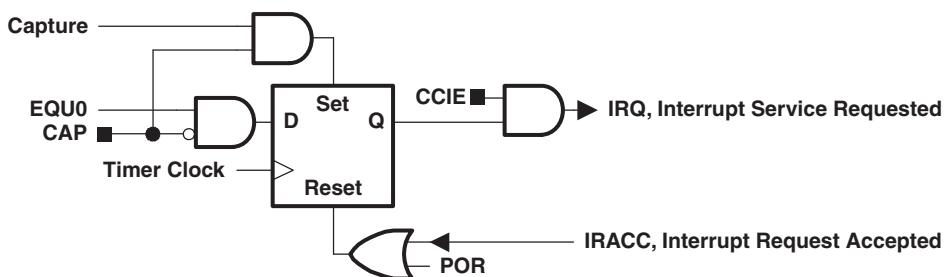
Two interrupt vectors are associated with the 16-bit Timer\_D module:

- TDxCCR0 interrupt vector for TDxCCR0 CCIFG
- TDIV interrupt vector for all other interrupt flags

In capture mode, any CCIFG flag is set when a timer value is captured in the associated TDCCR<sub>x</sub> register. In compare mode, any CCIFG flag is set when TDxR *counts* to the associated TDCL<sub>x</sub> 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.

#### 19.2.11.1 TDxCCR0 Interrupt Vector

The TDxCCR0 CCIFG flag has the highest Timer\_D interrupt priority and has a dedicated interrupt vector (see [Figure 19-23](#)). The TDxCCR0 CCIFG flag is automatically reset when the TDxCCR0 interrupt request is serviced.



**Figure 19-23. Capture/Compare TDxCCR0 Interrupt Flag**

#### 19.2.11.2 TDIV, Interrupt Vector Generator

The TDIFG flag, TDCCR<sub>x</sub> CCIFG flags (excluding TDxCCR0 CCIFG), and all of the high-resolution related interrupts are prioritized and combined to source a single interrupt vector. The interrupt vector register TDIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt (excluding TDxCCR0 CCIFG) generates a number in the TDxIV register (see [Section 19.3.11](#)). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer\_D interrupts do not affect the TDIV value.

Read access of the TDIV 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 TDxCCR1 and TDxCCR2 CCIFG flags are set when the interrupt service routine accesses the TDIV register, TDxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TDxCCR2 CCIFG flag generates another interrupt.

Write access of the TDIV register clears all pending interrupt conditions and flags.

### 19.2.11.3 TDIV, Interrupt Handler Examples

The following software example shows the recommended use of TDIV. The TDIV value is added to the PC to automatically jump to the appropriate routine.

Example 19-3 shows the recommended use of TDIV for Timer\_D3.

#### Example 19-3.

```
; Interrupt handler for TDxCCR0 CCIFG.  
CCIFG_0_HANDLER  
    ...                                ; Start of handler Interrupt latency  
    RETI  
; Interrupt handler for TDxIFG, TDxCCR1 and TDxCCR2 CCIFG.  
TD_HANDLER    ...                      ; Interrupt latency  
    ADD &TDxIV,PC                      ; Add offset to Jump table  
    RETI                                ; Vector 0: No interrupt  
    JMP CCIFG_1_HANDLER                ; Vector 2: Module 1  
    JMP CCIFG_2_HANDLER                ; Vector 4: Module 2  
    RETI                                ; Vector 6  
    RETI                                ; Vector 8  
    RETI                                ; Vector 10  
    RETI                                ; Vector 12  
    JMP TDxIFG_HANDLER                 ; Vector 14  
    RETI                                ; Vector 16  
    RETI                                ; Vector 18  
    RETI                                ; Vector 20  
    JMP TDHxUNLKIFG_HANDLER            ; Vector 22  
    ...  
  
TDHxUNLKIFG_HANDLER                    ; Vector 22: TDHUNLKIFG Flag  
    ...  
    RETI                                ; Task starts here  
  
TDxIFG_HANDLER                         ; Vector 14: TIMOV Flag  
    ...  
    RETI                                ; Task starts here  
    ...  
  
CCIFG_2_HANDLER                        ; Vector 4: Module 2  
    ...  
    RETI                                ; Task starts here  
    ; Back to main program  
  
; The Module 1 handler shows a way to look if any other  
; interrupt is pending: 5 cycles have to be spent, but  
; 9 cycles may be saved if another interrupt is pending  
CCIFG_1_HANDLER                        ; Vector 2: Module 1  
    ...  
    JMP TD_HANDLER                       ; Look for pending ints
```

## 19.3 Timer\_D Registers

The Timer\_D registers and address offsets are listed in [Table 19-8](#). The base address for each instance of Timer\_D can be found in the device-specific data sheet.

**Table 19-8. Timer\_D Registers**

Offset	Acronym	Register Name	Type	Reset	Section
0000h	TDxCTL0	Timer_D Control 0	Read/write	On POR	<a href="#">Section 19.3.1</a>
0002h	TDxCTL1	Timer_D Control 1	Read/write	On POR	<a href="#">Section 19.3.2</a>
0004h	TDxCTL2	Timer_D Control 2	Read/write	On POR	<a href="#">Section 19.3.3</a>
0006h	TDxR	Timer_D Counter	Read/write	On POR	<a href="#">Section 19.3.4</a>
0008h	TDxCCTL0	Timer_D Capture/Compare Control 0	Read/write	On POR	<a href="#">Section 19.3.5</a>
000Ah	TDxCCR0	Timer_D Capture/Compare 0	Read/write	On POR	<a href="#">Section 19.3.6</a>
000Ch	TDxCL0	Timer_D Capture/Compare Latch 0	Read only	On POR	<a href="#">Section 19.3.7</a>
000Eh	TDxCCTL1	Timer_D Capture/Compare Control 1	Read/write	On POR	<a href="#">Section 19.3.5</a>
0010h	TDxCCR1	Timer_D Capture/Compare 1	Read/write	On POR	<a href="#">Section 19.3.6</a>
0012h	TDxCL1	Timer_D Capture/Compare Latch 1	Read only	On POR	<a href="#">Section 19.3.7</a>
0014h	TDxCCTL2	Timer_D Capture/Compare Control 2	Read/write	On POR	<a href="#">Section 19.3.5</a>
0016h	TDxCCR2	Timer_D Capture/Compare 2	Read/write	On POR	<a href="#">Section 19.3.6</a>
0018h	TDxCL2	Timer_D Capture/Compare Latch 2	Read only	On POR	<a href="#">Section 19.3.7</a>
001Ah	TDxCCTL3	Timer_D Capture/Compare Control 3	Read/write	On POR	<a href="#">Section 19.3.5</a>
001Ch	TDxCCR3	Timer_D Capture/Compare 3	Read/write	On POR	<a href="#">Section 19.3.6</a>
001Eh	TDxCL3	Timer_D Capture/Compare Latch 3	Read only	On POR	<a href="#">Section 19.3.7</a>
0020h	TDxCCTL4	Timer_D Capture/Compare Control 4	Read/write	On POR	<a href="#">Section 19.3.5</a>
0022h	TDxCCR4	Timer_D Capture/Compare 4	Read/write	On POR	<a href="#">Section 19.3.6</a>
0024h	TDxCL4	Timer_D Capture/Compare Latch 4	Read only	On POR	<a href="#">Section 19.3.7</a>
0026h	TDxCCTL5	Timer_D Capture/Compare Control 5	Read/write	On POR	<a href="#">Section 19.3.5</a>
0028h	TDxCCR5	Timer_D Capture/Compare 5	Read/write	On POR	<a href="#">Section 19.3.6</a>
002Ah	TDxCL5	Timer_D Capture/Compare Latch 5	Read only	On POR	<a href="#">Section 19.3.7</a>
002Ch	TDxCCTL6	Timer_D Capture/Compare Control 6	Read/write	On POR	<a href="#">Section 19.3.5</a>
002Eh	TDxCCR6	Timer_D Capture/Compare 6	Read/write	On POR	<a href="#">Section 19.3.6</a>
0030h	TDxCL6	Timer_D Capture/Compare Latch 6	Read only	On POR	<a href="#">Section 19.3.7</a>
0032h		Reserved			
0034h		Reserved			
0036h		Reserved			
0038h	TDxHCTL0	Timer_D High-resolution Control 0	Read/write	On POR	<a href="#">Section 19.3.8</a>
003Ah	TDxHCTL1	Timer_D High-resolution Control 1	Read/write	On POR	<a href="#">Section 19.3.9</a>
003Ch	TDxHINT	Timer_D High-resolution Interrupt	Read/write	On POR	<a href="#">Section 19.3.10</a>
003Eh	TDxIV	Timer_D Interrupt Vector	Read only	On POR	<a href="#">Section 19.3.11</a>

### 19.3.1 TDxCTL0 Register

Timer\_D x Control Register 0

**Figure 19-24. TDxCTL0 Register**

15	14	13	12	11	10	9	8
Reserved	TDCLGRP <sub>x</sub>			CNTL <sub>x</sub>		Reserved	TDSSEL <sub>x</sub>
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID	MC <sub>x</sub>			Reserved	TDCLR	TDIE	TDIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0	w-(0)	rw-(0)	rw-(0)

**Table 19-9. TDxCTL0 Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-13	TDCLGRP <sub>x</sub>	RW	0h	TDCLx group 00b = Each TDCLx latch loads independently. 01b = TDxCL1+TDxCL2 (TDxCCR1 CLLDx bits control the update) TDxCL3+TDxCL4 (TDxCCR3 CLLDx bits control the update) TDxCL5+TDxCL6 (TDxCCR5 CLLDx bits control the update) TDxCL0 independent 10b = TDxCL1+TDxCL2+TDxCL3 (TDxCCR1 CLLDx bits control the update) TDxCL4+TDxCL5+TDxCL6 (TDxCCR4 CLLDx bits control the update) TDxCL0 independent 11b = TDxCL0+TDxCL1+TDxCL2+TDxCL3+TDxCL4+TDxCL5+TDxCL6 (TDxCCR1 CLLDx bits control the update)
12-11	CNTL <sub>x</sub>	RW	0h	Counter length 00b = 16-bit, TDR(max) = 0FFFFh 01b = 12-bit, TDR(max) = 0FFFh 10b = 10-bit, TDR(max) = 03FFh 11b = 8-bit, TDR(max) = 0FFh
10	Reserved	R	0h	Reserved. Always reads as 0.
9-8	TDSSEL <sub>x</sub>	RW	0h	Timer_D clock source select 00b = TDCLK 01b = ACLK 10b = SMCLK 11b = Inverted TDCLK
7-6	ID	RW	0h	Input divider. These bits, along with the INDEX bits in TDxCTL1, select the divider for the input clock. 00b = Divide by 1 01b = Divide by 2 10b = Divide by 4 11b = Divide by 8
5-4	MC <sub>x</sub>	RW	0h	Mode control. Setting MC <sub>x</sub> = 00h when Timer_D is not in use saves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TDCL0 10b = Continuous mode: Timer counts up to the value set by CNTL <sub>x</sub> (counter length) 11b = Up/down mode: Timer counts up to TDCL0 and down to 0000h
3	Reserved	R	0h	Reserved. Always reads as 0.
2	TDCLR	W	0h	Timer_D clear. Setting this bit resets TDR, the TDCLK divider, and the count direction. The TDCLR bit always read as zero.

**Table 19-9. TDxCTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
1	TDIE	RW	0h	Timer_D interrupt enable. This bit enables the TDIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TDIFG	RW	0h	Timer_D interrupt flag 0b = No interrupt pending 1b = Interrupt pending

### 19.3.2 TDxCTL1 Register

Timer\_D x Control Register 1

**Figure 19-25. TDxCTL1 Register**

15	14	13	12	11	10	9	8
Reserved						INDEX	
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved	TD6CMB	TD4CMB	TD2CMB	Reserved		TDCLKMx	
r0	rw-(0)	rw-(0)	rw-(0)	r0	r0	rw-(0)	rw-(0)

**Table 19-10. TDxCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	IDEX	RW	0h	<p>Input divider extension. These bits, along with the ID bits in TDxCTL0, select the divider for the input clock.</p> <p>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</p>
7	Reserved	R	0h	Reserved. Always reads as 0.
6	TD6CMB	RW	0h	<p>Control bit for TDCCR registers combination in TD6 (only available on Timer_D7)</p> <p>0b = TDxCCR5 and TDxCCR6 are not combined      1b = TDxCCR5 and TDxCCR6 are combined</p>
5	TD4CMB	RW	0h	<p>Control bit for TDCCR registers combination in TD4 (available on Timer_D5, Timer_D7)</p> <p>0b = TDxCCR3 and TDxCCR4 are not combined      1b = TDxCCR3 and TDxCCR4 are combined</p>
4	TD2CMB	RW	0h	<p>Control bit for TDCCR registers combination in TD2</p> <p>0b = TDxCCR1 and TDxCCR2 are not combined      1b = TDxCCR1 and TDxCCR2 are combined</p>
3-2	Reserved	R	0h	Reserved. Always reads as 0.
1-0	TDCLKMx	RW	0h	<p>Timer_D clocking mode register</p> <p>00b = External clock source is used      01b = High-resolution local clock is used      10b = Auxiliary clock source from another timer instance is used      11b = Reserved</p>

**19.3.3 TDxCTL2 Register**

Timer\_D x Control Register 2

**Figure 19-26. TDxCTL2 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	TDCAPM6	TDCAPM5	TDCAPM4	TDCAPM3	TDCAPM2	TDCAPM1	TDCAPM0
r0							

**Table 19-11. TDxCTL2 Register Description**

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6	TDCAPM6	RW	0h	Capture mode of channel 6 0b = Single capture mode 1b = Dual capture mode
5	TDCAPM5	RW	0h	Capture mode of channel 5 0b = Single capture mode 1b = Dual capture mode
4	TDCAPM4	RW	0h	Capture mode of channel 4 0b = Single capture mode 1b = Dual capture mode
3	TDCAPM3	RW	0h	Capture mode of channel 3 0b = Single capture mode 1b = Dual capture mode
2	TDCAPM2	RW	0h	Capture mode of channel 2 0b = Single capture mode 1b = Dual capture mode
1	TDCAPM1	RW	0h	Capture mode of channel 1 0b = Single capture mode 1b = Dual capture mode
0	TDCAPM0	RW	0h	Capture mode of channel 0 0b = Single capture mode 1b = Dual capture mode

### **19.3.4 TDxR Register**

Timer\_D x Counter Register

**Figure 19-27. TDxR Register**

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

**Table 19-12. TDxR Register Description**

Bit	Field	Type	Reset	Description
15-0	TDRx	RW	0h	Timer_D register. The TDxR register is the count of Timer_D. In high-resolution mode, the bits 0 to 3 return zero when the TDxR register is read.

**19.3.5 TDxCCTLn Register**

Timer\_D x Capture/Compare Control Register

**Figure 19-28. TDxCCTLn Register**

15	14	13	12	11	10	9	8
CMx		CCISx		SCS		CLLDx	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

**Table 19-13. TDxCCTLn Register Description**

Bit	Field	Type	Reset	Description
15-14	CMx	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	CCISx	RW	0h	Capture/compare input select. These bits select the TDCCR <sub>x</sub> input signal. See the device-specific data sheet for specific signal connections. 00b = CC <sub>l</sub> x A 01b = CC <sub>l</sub> x B 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. In high-resolution mode, the capture is always synchronous to the high-resolution clock, and this setting is ignored. 0b = Asynchronous capture 1b = Synchronous capture
10-9	CLLDx	RW	0h	Compare latch load. These bits select the compare latch load event. 00b = TDCL <sub>x</sub> loads on write to TDCCR <sub>x</sub> 01b = TDCL <sub>x</sub> loads when TDR counts to 0 10b = TDCL <sub>x</sub> loads when TDR counts to 0 (up or continuous mode). TDCL <sub>x</sub> loads when TDR counts to TDxCLO or to 0 (up/down mode). 11b = TDCL <sub>x</sub> loads when TDR counts to TDCL <sub>x</sub>
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMODx	RW	0h	Output mode 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. 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.

**Table 19-13. TDxCCTLn Register Description (continued)**

Bit	Field	Type	Reset	Description
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	RW	0h	Capture overflow. 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

**19.3.6 TDxCCRn Register**

Timer\_D x Capture/Compare Register

**Figure 19-29. TDxCCRn Register**

15	14	13	12	11	10	9	8
TDCCR <sub>x</sub>							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TDCCR <sub>x</sub>							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 19-14. TDxCCRn Register Description**

Bit	Field	Type	Reset	Description
15-0	TDCCR <sub>x</sub>	RW	0h	<p>Timer_D capture/compare register. The TDCCR<sub>x</sub> register is the count of capture/compare block x.</p> <p>Note: In high-resolution compare mode there are limitations of the minimum and maximum duty cycles. See <a href="#">Table 19-3</a> and <a href="#">Table 19-4</a> for details.</p>

**19.3.7 TDxCCLn Register**

Timer\_D x Capture/Compare Latch Register

**Figure 19-30. TDxCCLn Register**

15	14	13	12	11	10	9	8
TDCL <sub>x</sub>							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TDCL <sub>x</sub>							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 19-15. TDxCCLn Register Description**

Bit	Field	Type	Reset	Description
15-0	TDCL <sub>x</sub>	RW	0h	Timer_D capture/compare latch register. TDCL <sub>x</sub> register holds the comparison value in compare mode. In capture mode the register content of TDxCCR <sub>x</sub> is copied into TDxCCL <sub>x</sub> at each capture event.

### 19.3.8 TDxHCTL0 Register

Timer\_D x High-Resolution Control Register 0

**Figure 19-31. TDxHCTL0 Register**

15	14	13	12	11	10	9	8
Reserved							TDHFW
r0	r0	r0	r0	r0	r0	r0	rw-(0)
7	6	5	4	3	2	1	0
TDHDx	TDHMx	TDHRON	TDHEAEN	TDHREGEN	TDHEN		
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 19-16. TDxHCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	TDHFW	RW	0h	High-resolution generator fast wakeup enable 0b = High-resolution generator fast wakeup disabled 1b = High-resolution generator fast wakeup enable
7-6	TDHDx	RW	0h	High-resolution clock divider. These bits select the divider for the high resolution clock. 00b = Divide by 1 01b = Divide by 2 10b = Divide by 4 11b = Divide by 8
5-4	TDHMx	RW	0h	Timer_D high-resolution clock multiplication factor 00b = High-resolution clock 8x Timer_D clock 01b = High-resolution clock 16x Timer_D clock 10b = Reserved 11b = Reserved
3	TDHRON	RW	0h	Timer_D high-resolution generator forced on. 0b = High-resolution generator is on if the Timer_D counter MCx bits are 01, 10 or 11. 1b = High-resolution generator is on in all Timer_D MCx modes. The PMM remains in high-current mode.
2	TDHEAEN	RW	0h	Timer_D high-resolution clock enhanced accuracy enable bit. Setting this bit reduces the accumulated frequency offset of the high-resolution clock generator and the reference clock. 0b = Normal accuracy 1b = Enhanced accuracy enable
1	TDHREGEN	RW	0h	Timer_D regulation enable. Set this bit to synchronize the high-resolution clock to the Timer_D input clock defined by TDSSELx. 0b = Regulation disabled 1b = Regulation enabled
0	TDHEN	RW	0h	Timer_D high-resolution enable bit. This bit must be set to enable high-resolution operation mode. Whenever a high-resolution TDAUXCLK from another Timer_D instance is used, this bit must also be set. 0b = High-resolution mode disable 1b = High-resolution mode enable

**19.3.9 TDxHCTL1 Register**

Timer\_D x High-Resolution Control Register 1

**Figure 19-32. TDxHCTL1 Register**

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

**Table 19-17. TDxHCTL1 Register Description**

Bit	Field	Type	Reset	Description
15	TDHCLKCR	RW	0h	Timer_D high-resolution coarse clock range selection bits. For detailed frequency numbers, see the device-specific data sheet. 0b = Timer_D input clock to the high-resolution generator is <15 MHz. 1b = Timer_D input clock to the high-resolution generator is >15 MHz.
14-13	TDHCLKRx	RW	0h	Timer_D high-resolution clock range selection bits. These bits are used to define the coarse clock range of the high-resolution clock generator. If TDHREGEN = 1 these register bits are modified by hardware. 00b = Clock range 0. See data sheet for frequency details. 01b = Clock range 1. See data sheet for frequency details. 10b = Clock range 2. See data sheet for frequency details. 11b = Reserved
12-8	TDHCLKSRx	RW	0h	Timer_D high-resolution clock sub-range selection bits. These bits are used to define the sub-clock range of the high-resolution clock generator. The frequency change due to a change of the TDHCLKTRIMx bits is approximately half of the TDHCLKSRx bits in each clock range. If TDHREGEN = 1, these register bits are modified by hardware.
7-1	TDHCLKTRIMx	RW	40h	Timer_D high-resolution clock trim selection bits. These bits are used to change the clock frequency slightly. The frequency change due to a change of the TDHCLKTRIMx bits is approximately half of the TDHCLKSRx bits in each clock range. If TDHREGEN = 1, these register bits are modified by hardware.
0	Reserved	R	0h	Reserved. Always reads as 0.

### 19.3.10 TDxHINT Register

Timer\_D x High-Resolution Interrupt Register

**Figure 19-33. TDxHINT Register**

15	14	13	12	11	10	9	8
		Reserved		TDHUNLKIE	TDHLKIE	TDHFHIE	TDHFLIE
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
		Reserved		TDHUNLKIFG	TDHLKIFG	TDHFHIFG	TDHFLIFG
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 19-18. TDxHINT Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11	TDHUNLKIE	RW	0h	Timer_D interrupt enable. This bit enables the TDHUNLKIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
10	TDHLKIE	RW	0h	Timer_D interrupt enable. This bit enables the TDHLKIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
9	TDHFHIE	RW	0h	Timer_D interrupt enable. This bit enables the TDHFHIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
8	TDHFLIE	RW	0h	Timer_D interrupt enable. This bit enables the TDHFLIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
7-4	Reserved	R	0h	Reserved. Always reads as 0.
3	TDHUNLKIFG	RW	0h	Timer_D high-resolution frequency unlock interrupt flag if TDHREGEN is set to 1. This bit is set if the frequency is still unlocked and the calibration is not yet completed. If the bit is set until cleared by writing to it. 0b = No interrupt pending 1b = Interrupt pending
2	TDHLKIFG	RW	0h	Timer_D high-resolution frequency lock interrupt flag if TDHREGEN is set to 1. This bit is set if the frequency is locked and the calibration is completed. If the bit is set it remains set until cleared by writing to it. 0b = No interrupt pending 1b = Interrupt pending
1	TDHFHIFG	RW	0h	Timer_D high-resolution fail high interrupt flag. This bit is set if the high-resolution generator cannot generate the high-resolution frequency because the input frequency is too high. If the bit is set until cleared by writing to it. 0b = No interrupt pending 1b = Interrupt pending
0	TDHFLIFG	RW	0h	Timer_D high-resolution fail low interrupt flag. This bit is set if the high-resolution generator cannot generate the high-resolution frequency because the input frequency is too low. If the bit is set until cleared by writing to it. 0b = No interrupt pending 1b = Interrupt pending

**19.3.11 TDxIV Register**

Timer\_D x Interrupt Vector Register

**Figure 19-34. TDxIV Register**

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

**Table 19-19. TDxIV Register Description**

Bit	Field	Type	Reset	Description
15-0	TDIVx	R	0h	<p>Timer_D external interrupt vector value</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TDxCCR1 CCIFG; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TDxCCR2 CCIFG</p> <p>06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TDxCCR3 CCIFG</p> <p>08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TDxCCR4 CCIFG</p> <p>0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TDxCCR5 CCIFG</p> <p>0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TDxCCR6 CCIFG</p> <p>0Eh = Reserved</p> <p>10h = Interrupt Source: Timer overflow; Interrupt Flag: TDxIFG</p> <p>12h = Interrupt Source: Clock fail low; Interrupt Flag: TDxHINT TDHFLIFG</p> <p>14h = Interrupt Source: Clock fail high; Interrupt Flag: TDxHINT TDHFHIFG</p> <p>16h = Interrupt Source: High-resolution frequency locked; Interrupt Flag: TDxHINT TDHLKIFG</p> <p>18h = Interrupt Source: High-resolution frequency unlocked; Interrupt Flag: TDxHINT TDHUNLKIFG</p> <p>1Ah = Reserved</p> <p>1Ch = Reserved</p> <p>1Eh = Reserved ; Interrupt Priority: Lowest</p>

## **Timer Event Control (TEC)**

Timer Event Control (TEC) module is the interface between Timer modules and the external events. This chapter describes the TEC Module.

<b>Topic</b>	<b>Page</b>
<b>20.1 Timer Event Control Introduction .....</b>	<b>548</b>
<b>20.2 TEC Operation .....</b>	<b>549</b>
<b>20.3 TEC Registers .....</b>	<b>555</b>

## 20.1 Timer Event Control Introduction

TEC is a module that connects different Timer modules to each other and routes the external signals to the Timer modules. TEC contains the control registers to configure the routing between the Timer modules, and it also has the enable register bits and the interrupt enable and interrupt flags for external event inputs.

TEC features include:

- Enabling of internal and external clear signals
- Routing of internal signals (between Timer\_D instances) and external clear signals
- Support of external fault input signals
- Interrupt vector generation of external fault and clear signals.
- Generating feedback signals to the Timer capture/compare channels to affect the timer outputs

The block diagram of TEC is shown in [Figure 20-1](#).

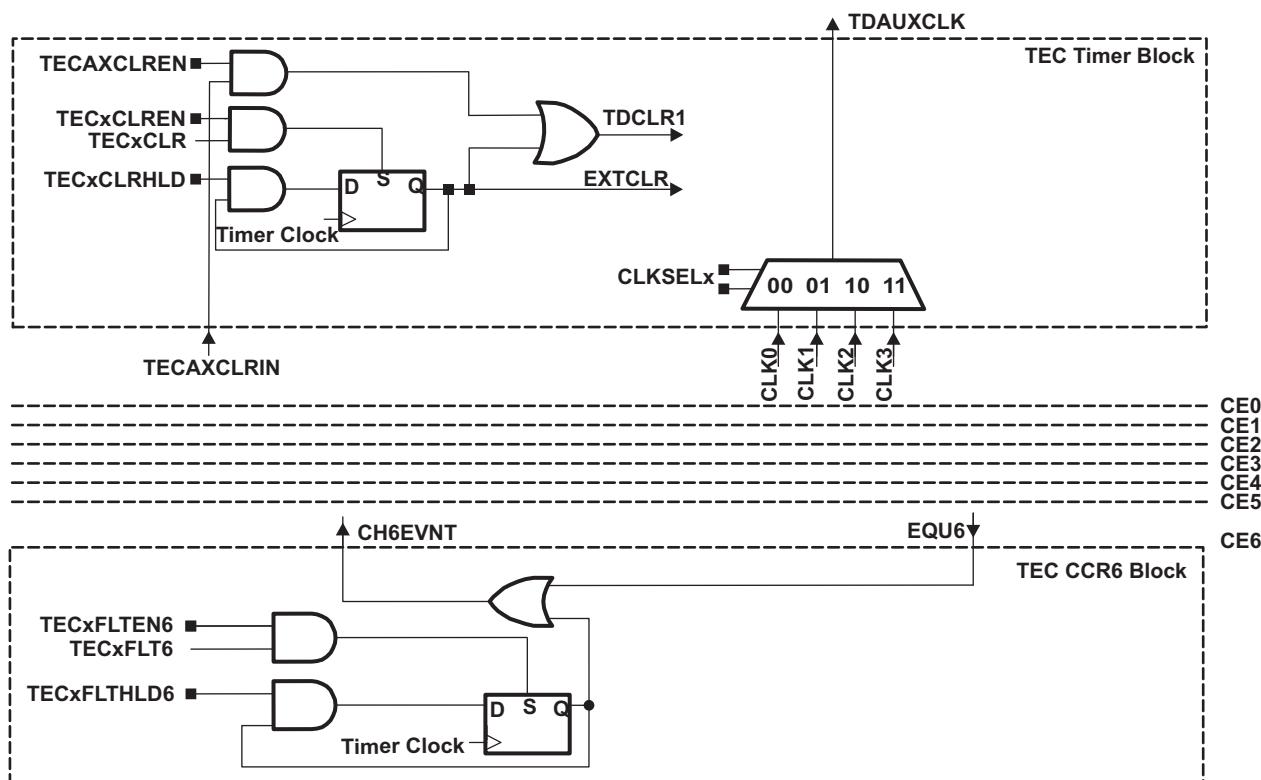


Figure 20-1. Timer Event Control Block Diagram

## 20.2 TEC Operation

The TEC module contains three sub-blocks:

- AUXCLK Selection sub-block
  - The AUXCLK Selection sub-block chooses the auxiliary clock source for the timer.
- External Clear sub-block
  - The External Clear sub-block controls the operation of the timer counter.
- Channel Event sub-block
  - The Channel Event sub-block controls the operation of a capture/compare channel.

This structure means that one Timer\_D3 module, for example, needs a TEC that contains one AUXCLK Selection sub-block, one External Clear sub-block, and three Channel Event sub-blocks.

### 20.2.1 AUXCLK Selection Sub-Block

The AUXCLK Selection sub-block provides the auxiliary clock to Timer\_D. By configuring the CLKSELx bits, one of the four clock sources can be chosen.

### 20.2.2 External Clear Sub-Block

The External Clear sub-block accepts external signals that can clear the Timer\_D and affect the timer output. The TDR register in Timer\_D module can be cleared by the following external events:

- External input TECEXCLR is enabled and is high.
- The auxiliary clear signal TECAXCLRIN is enabled and is high.

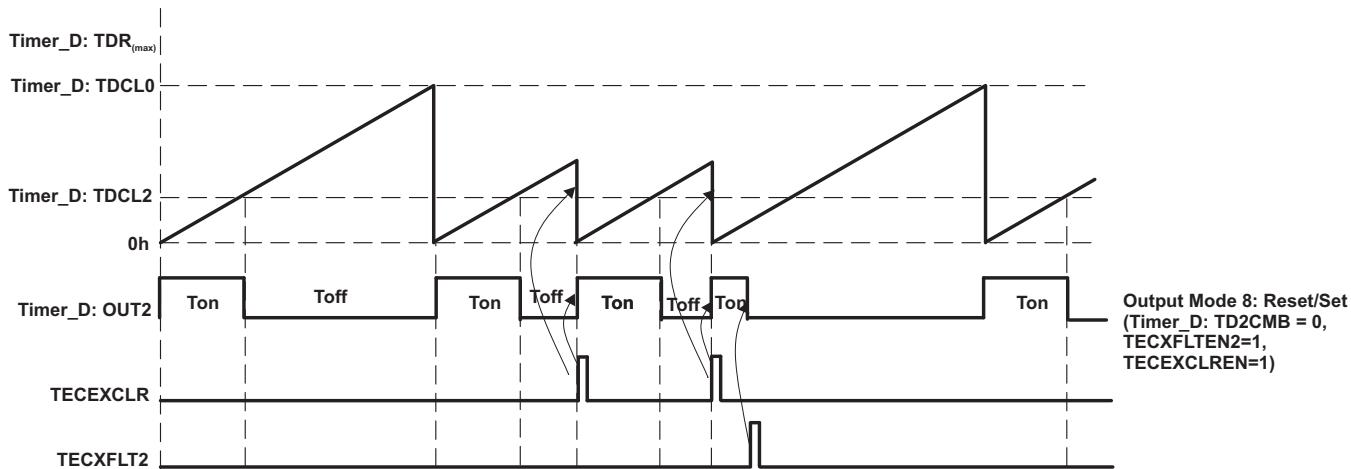
### 20.2.3 Channel Event Sub-Block

The Channel Event sub-block is responsible for timer event control and signaling. One Channel Event sub-block corresponds to one capture/compare channel in Timer\_D module. The output unit of a capture/compare channel accepts the external fault input as well as the EQux signal coming from the CCRx unit of the Timer\_D. Both events will affect the PWM output.

When a TECXFLTx event is enabled and occurs, the corresponding status bit TECXFLTxSTA in the TECSTA register is set. This event also sets the TECXFLTIFG interrupt flag. When TECXFLTIFG is set to logical 1, the TECXFLTxSTA status bits show which external fault signal or signals are active. The TECXFLTHLDx control bit holds the external fault signal. Clearing the TECXFLTxSTA status bits also resets the held signal back to zero.

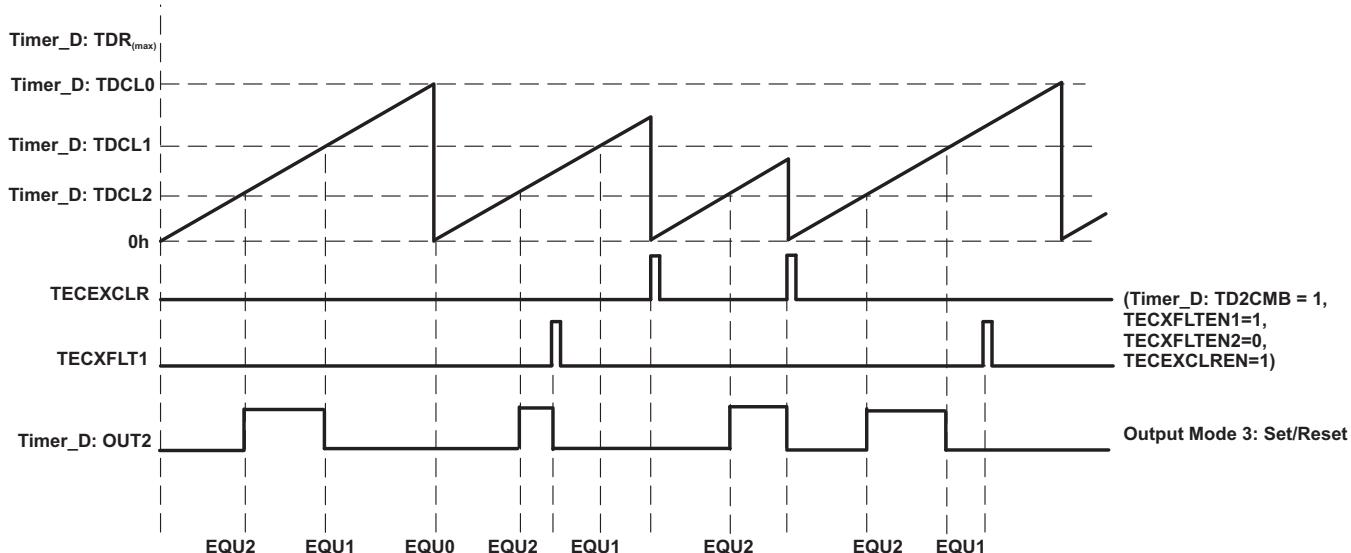
### 20.2.3.1 External Input Events Affect Timer Output

Figure 20-2 shows how the external signals affect the Timer\_D output in a Power Factor Corrector (PFC) application.



**Figure 20-2. External Input Events Affect Timer\_D Output**

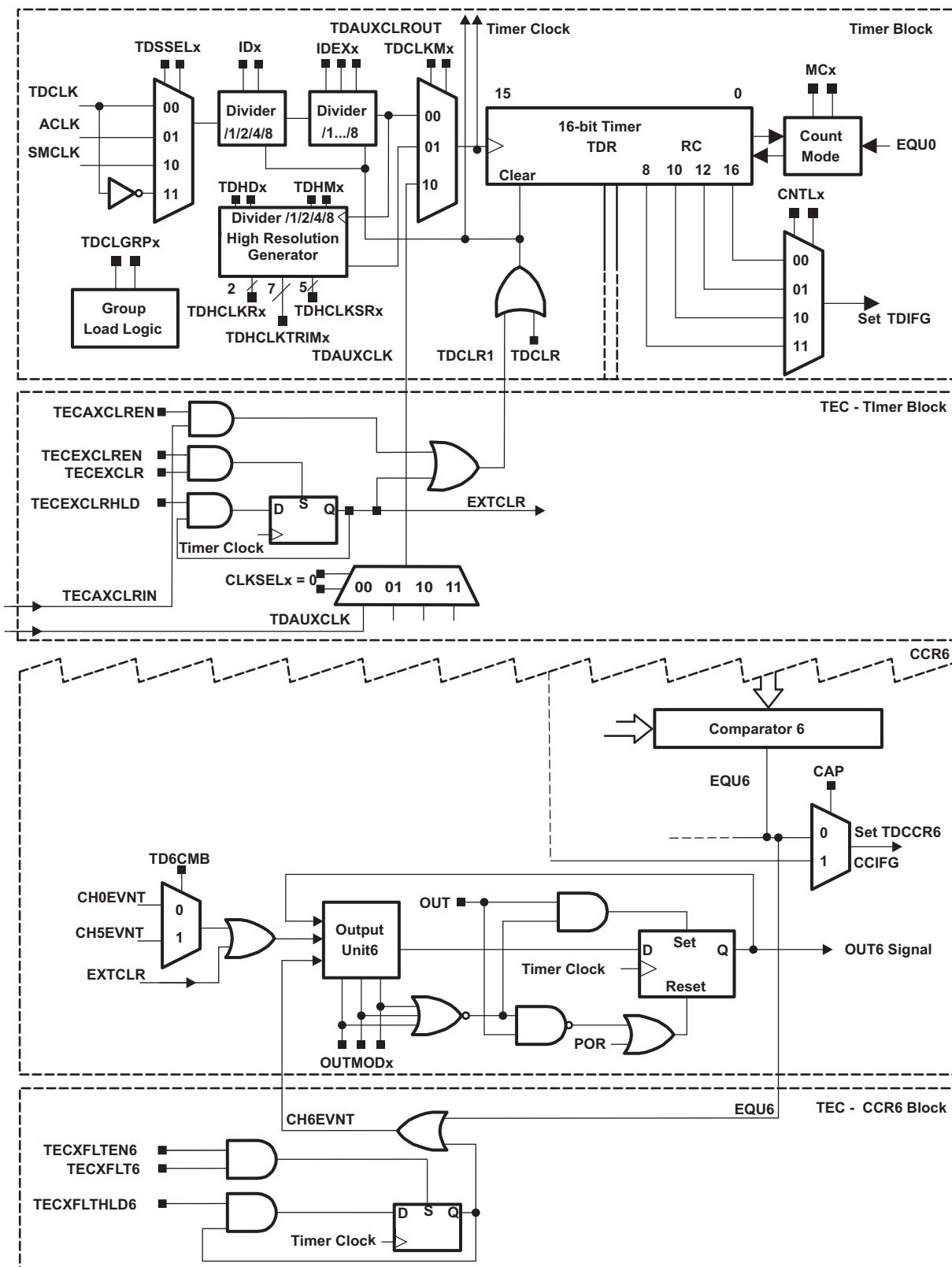
Figure 20-3 shows an example in which channels are combined.



**Figure 20-3. Timer\_D Output With Channel Combination**

### 20.2.4 Module Level Connection Between TEC and Timer\_D

The TEC and Timer\_D modules are connected through internal signals. Figure 20-4 shows the interconnection between TEC and Timer\_D module. See the [Timer\\_D chapter](#) for more Timer\_D module information.



**Figure 20-4. Module Level Connection Between TEC and Timer\_D**

## 20.2.5 Synchronization Mechanism Between Timer\_D Instances

Two or more Timer\_D modules can be synchronized to each other. One instance can be the master of the other Timer\_D instance (slave). The master is the instance that supplies the clock to the other instance. If two timer instances are synchronized to each other, then there is a device specific order; for example, Timer0\_D is the master of Timer1\_D. The slave uses the settings of the TDHEN bit, the CNTLx bits, and the MCx bits of the master. The CCR0 registers of the master and the slave must be programmed to equal values. [Figure 20-5](#) shows the combination of a Timer\_D master and a Timer\_D slave and the control signals routed through the TEC module of the slave. The clock generator of the Timer\_D slave is powered down by selecting the TDCLKMx to accept the clock input from the Timer\_D master. The TECEXCLR signals of the slave timers do not propagate to the master but the clear events of the master timers are signaled to the slave timers. See [Figure 20-5](#) for the detailed signal routing between master and slave. The divider and clock selector settings of the slave timers are ignored. The high resolution generator of the slave timer is disabled.

The [Example 20-1](#) shows how to program the master timer and the slave timer to synchronize them.

### Example 20-1.

```
// Example: TD0 is master timer, TD1 is slave timer
//           Period is 200. TD0 channel 1: 20% duty cycle. TD0 channel 2: 40% duty cycle
//           TD1 channel 1: 60% duty cycle. TD1 channel 2: 80% duty cycle
// Configure Master TD0
    TD0CTL0 = TDSSEL_2;                                // TDCLK = SMCLK = Hi-Res input clk
    TD0CTL1 |= TDCLKM_1;                                // TD0 clock = Hi-res local clock
    TD0HCTL0 = TDHM_0 + TDHREGEN + TDHEN;              // Hi-res clock 8x TDCLK,
                                                       //

Regulation and Hi-res mode enable
// TD0HINT |= TDHLKIE;                                // Set TDH Lock IFG -
not used in the example

// Configure Slave TD1
    TD1CTL1 = TDCLKM_2;                                // TD1 clock = Auxiliary clock source from
master
                                                       //

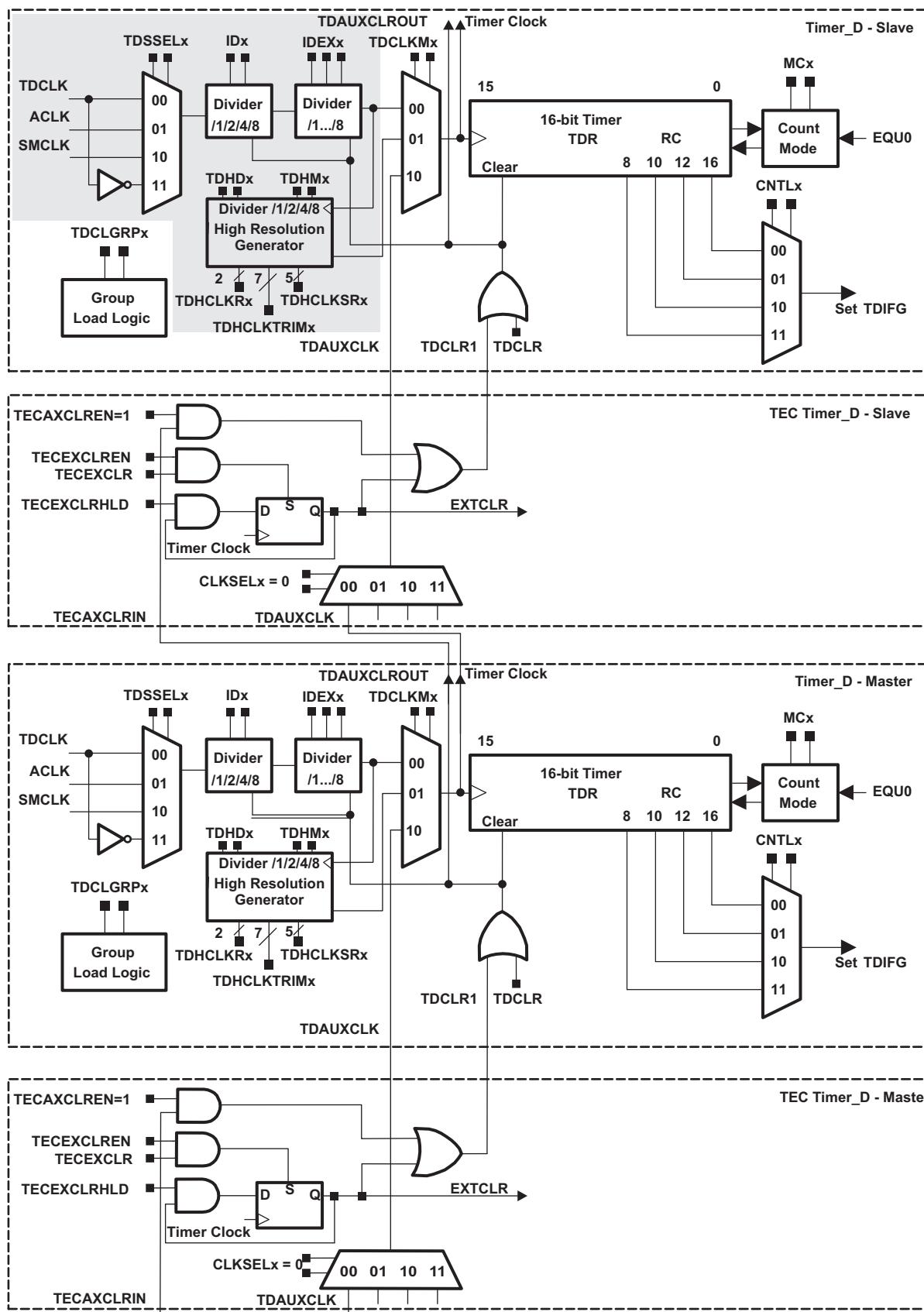
timer instance
// Configure Slave TEC1
    TEC1XCTL2 |= TECAEXCLREN;                         // Enable synchronized clear

// Configure TD0 and its PWM outputs
    TD0CTL0 |= TDCLR;                                 // Clear timer counter master+slave
                                                       //

    TD0CCR0 = 200;                                    // TD0CCR0
    TD0CCR1 = 40;                                     // 20% dutycycle
    TD0CCTL1 |= OUTMOD_7;                            // TD0CCR1, Reset/Set
    TD0CCR2 = 80;                                     // 40% dutycycle
    TD0CCTL2 |= OUTMOD_7;                            // TD0CCR2, Reset/Set

// Configure TD1 PWM outputs
    TD1CCR0 = 200;                                    // TD1CCR0
    TD1CCR1 = 120;                                    // 60% dutycycle
    TD1CCTL1 |= OUTMOD_7;                            // TD1CCR1, Reset/Set
    TD1CCR2 = 160;                                    // 80% dutycycle
    TD1CCTL2 |= OUTMOD_7;                            // TD1CCR2, Reset/Set

// Start timers and select Up-mode
    TD0CTL0 |= MC_1 + TDCLR;                         // up-mode, start timer
```


**Figure 20-5. Synchronization Between Timer Instances**

## 20.2.6 Timer Event Control Interrupts

The TEC module has only one interrupt vector, TECxIV.

### 20.2.6.1 TECxIV, Interrupt Vector Generator

The external fault interrupt TECXFLTIFG, the external clear interrupt TECEXCLRIFG, and the auxiliary clear interrupt TECAXCLRIFG are prioritized and combined to source a single interrupt vector. The interrupt vector register TECxIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt generates a number in the TECxIV register (see [Section 20.3.6](#)). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled TEC interrupts do not affect the TECxIV value.

Read access of the TECxIV 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 TECxIV register clears all pending interrupt conditions and flags.

## 20.3 TEC Registers

The Timer Event Control registers are listed in [Table 20-1](#). The base address can be found in the device-specific data sheet. The address offset is listed in [Table 20-1](#).

**Table 20-1. TEC Registers**

Offset	Acronym	Register Name	Type	Reset	Section
0000h	TECxCTL0	Timer Event Control External Control 0	Read/write	0000h	<a href="#">Section 20.3.1</a>
0002h	TECxCTL1	Timer Event Control External Control 1	Read/write	0000h	<a href="#">Section 20.3.2</a>
0004h	TECxCTL2	Timer Event Control External Control 2	Read/write	0000h	<a href="#">Section 20.3.3</a>
0006h	TECxSTA	Timer Event Control Status	Read/write	0000h	<a href="#">Section 20.3.4</a>
0008h	TECxINT	Timer Event Control External Interrupt	Read/write	0000h	<a href="#">Section 20.3.5</a>
000Ah	TECxIV	Timer Event Control Interrupt Vector	Read only	0000h	<a href="#">Section 20.3.6</a>

**20.3.1 TECxCTL0 Register**

Timer Event Control External Control Register 0

**Figure 20-6. TECxCTL0 Register**

15	14	13	12	11	10	9	8
Reserved	TECXFLTEN6	TECXFLTEN5	TECXFLTEN4	TECXFLTEN3	TECXFLTEN2	TECXFLTEN1	TECXFLTEN0
r0	rw-(0)						
7	6	5	4	3	2	1	0
Reserved	TECXFLTHLD6	TECXFLTHLD5	TECXFLTHLD4	TECXFLTHLD3	TECXFLTHLD2	TECXFLTHLD1	TECXFLTHLD0
r0	rw-(0)						

**Table 20-2. TECxCTL0 Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14	TECXFLTEN6	RW	0h	External fault signal enable for channel event block 6 (only available on TEC7) 0b = External fault signal is disabled for CE6 1b = External fault signal is enabled for CE6
13	TECXFLTEN5	RW	0h	External fault signal enable for channel event block 5 (only available on TEC7) 0b = External fault signal is disabled for CE5 1b = External fault signal is enabled for CE5
12	TECXFLTEN4	RW	0h	External fault signal enable for channel event block 4 (only available on TEC5 or TEC7) 0b = External fault signal is disabled for CE4 1b = External fault signal is enabled for CE4
11	TECXFLTEN3	RW	0h	External fault signal enable for channel event block 3 (only available on TEC5 or TEC7) 0b = External fault signal is disabled for CE3 1b = External fault signal is enabled for CE3
10	TECXFLTEN2	RW	0h	External fault signal enable for channel event block 2 0b = External fault signal is disabled for CE2 1b = External fault signal is enabled for CE2
9	TECXFLTEN1	RW	0h	External fault signal enable for channel event block 1 0b = External fault signal is disabled for CE1 1b = External fault signal is enabled for CE1
8	TECXFLTEN0	RW	0h	External fault signal enable for channel event block 0 0b = External fault signal is disabled for CE0 1b = External fault signal is enabled for CE0
7	Reserved	R	0h	Reserved. Always reads as 0.
6	TECXFLTHLD6	RW	0h	External fault signal hold bit for CE6 0b = External fault signal is not held 1b = External fault signal is held
5	TECXFLTHLD5	RW	0h	External fault signal hold bit for CE5 0b = External fault signal is not held 1b = External fault signal is held
4	TECXFLTHLD4	RW	0h	External fault signal hold bit for CE4 0b = External fault signal is not held 1b = External fault signal is held
3	TECXFLTHLD3	RW	0h	External fault signal hold bit for CE3 0b = External fault signal is not held 1b = External fault signal is held

**Table 20-2. TECxCTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
2	TECXFLTHLD2	RW	0h	External fault signal hold bit for CE2 0b = External fault signal is not held 1b = External fault signal is held
1	TECXFLTHLD1	RW	0h	External fault signal hold bit for CE1 0b = External fault signal is not held 1b = External fault signal is held
0	TECXFLTHLD0	RW	0h	External fault signal hold bit for CE0 0b = External fault signal is not held 1b = External fault signal is held

**20.3.2 TECxCTL1 Register**

Timer Event Control External Control Register 1

**Figure 20-7. TECxCTL1 Register**

15	14	13	12	11	10	9	8
Reserved	TECXFLTLS6	TECXFLTLS5	TECXFLTLS4	TECXFLTLS3	TECXFLTLS2	TECXFLTLS1	TECXFLTLS0
r0	rw-(0)						
7	6	5	4	3	2	1	0
Reserved	TECXFLTPOL6	TECXFLTPOL5	TECXFLTPOL4	TECXFLTPOL3	TECXFLTPOL2	TECXFLTPOL1	TECXFLTPOL0
r0	rw-(0)						

**Table 20-3. TECxCTL1 Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14	TECXFLTLS6	RW	0h	Signal type of external fault 6 0b = Edge sensitive 1b = Level sensitive
13	TECXFLTLS5	RW	0h	Signal type of external fault 5 0b = Edge sensitive 1b = Level sensitive
12	TECXFLTLS4	RW	0h	Signal type of external fault 4 0b = Edge sensitive 1b = Level sensitive
11	TECXFLTLS3	RW	0h	Signal type of external fault 3 0b = Edge sensitive 1b = Level sensitive
10	TECXFLTLS2	RW	0h	Signal type of external fault 2 0b = Edge sensitive 1b = Level sensitive
9	TECXFLTLS1	RW	0h	Signal type of external fault 1 0b = Edge sensitive 1b = Level sensitive
8	TECXFLTLS0	RW	0h	Signal type of external fault 0 0b = Edge sensitive 1b = Level sensitive
7	Reserved	R	0h	Reserved. Always reads as 0.
6	TECXFLTPOL6	RW	0h	Polarity bit of external fault 6 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive
5	TECXFLTPOL5	RW	0h	Polarity bit of external fault 5 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive
4	TECXFLTPOL4	RW	0h	Polarity bit of external fault 4 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive
3	TECXFLTPOL3	RW	0h	Polarity bit of external fault 3 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive

**Table 20-3. TECxCTL1 Register Description (continued)**

Bit	Field	Type	Reset	Description
2	TECXFLTPOL2	RW	0h	Polarity bit of external fault 2 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive
1	TECXFLTPOL1	RW	0h	Polarity bit of external fault 1 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive
0	TECXFLTPOL0	RW	0h	Polarity bit of external fault 0 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive

**20.3.3 TECxCTL2 Register**

Timer Event Control External Control Register 2

**Figure 20-8. TECxCTL2 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	TECEXCLRLVS	TECEXCLRPO L	TECEXCLRHL D	TECEXCLREN	TECAXCLREN	CLKSELx	
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 20-4. TECxCTL2 Register Description**

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6	TECEXCLRLVS	RW	0h	Signal type of external clear 0b = Edge sensitive 1b = Level sensitive
5	TECEXCLRPOL	RW	0h	Polarity bit of external clear 0b = Selects falling edge in edge sensitive or low level in level sensitive 1b = Selects rising edge in edge sensitive or high level in level sensitive
4	TECEXCLRHLD	RW	0h	External clear signal hold bit 0b = External clear signal is not held 1b = External clear signal is held
3	TECEXCLREN	RW	0h	External clear signal control bit 0b = External clear signal disabled 1b = External clear signal enabled
2	TECAXCLREN	RW	0h	Auxiliary clear signal control bit 0b = Auxiliary clear signal disabled 1b = Auxiliary clear signal enabled
1-0	CLKSELx	RW	0h	Auxiliary clock source selection bits 00b = Clock input from a Timer_D master instance 01b = Reserved 10b = Reserved 11b = Reserved

### **20.3.4 TECxSTA Register**

Timer Event Control Status Register

**Figure 20-9. TECxSTA Register**

15	14	13	12	11	10	9	8
Reserved							TECXCLRSTA
r0	r0	r0	r0	r0	r0	r0	rw-(0)
7	6	5	4	3	2	1	0
Reserved	TECXFLT6STA	TECXFLT5STA	TECXFLT4STA	TECXFLT3STA	TECXFLT2STA	TECXFLT1STA	TECXFLT0STA
r0	rw-(0)						

**Table 20-5. TECxSTA Register Description**

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	TECXCLRSTA	RW	0h	External clear status flag. This bit is set if the external clear input is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external clear detected 1b = External clear detected
7	Reserved	R	0h	Reserved. Always reads as 0.
6	TECXFLT6STA	RW	0h	External fault status flag. This bit is set if the external fault signal in CE6 is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external fault detected 1b = External fault detected
5	TECXFLT5STA	RW	0h	External fault status flag. This bit is set if the external fault signal in CE5 is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external fault detected 1b = External fault detected
4	TECXFLT4STA	RW	0h	External fault status flag. This bit is set if the external fault signal in CE4 is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external fault detected 1b = External fault detected
3	TECXFLT3STA	RW	0h	External fault status flag. This bit is set if the external fault signal in CE3 is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external fault detected 1b = External fault detected
2	TECXFLT2STA	RW	0h	External fault status flag. This bit is set if the external fault signal in CE2 is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external fault detected 1b = External fault detected
1	TECXFLT1STA	RW	0h	External fault status flag. This bit is set if the external fault signal in CE1 is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external fault detected 1b = External fault detected
0	TECXFLT0STA	RW	0h	External fault status flag. This bit is set if the external fault signal in CE0 is detected. If the bit is set it remains set until cleared by writing 0 to it. 0b = No external fault detected 1b = External fault detected

**20.3.5 TECxINT Register**

Timer Event Control External Interrupt Register

**Figure 20-10. TECxINT Register**

15	14	13	12	11	10	9	8
		Reserved			TECXFLTIE	TECEXCLRIE	TECAXCLRIE
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
		Reserved			TECXFLTIFG	TECEXCLRIFG	TECAXCLRIFG
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

**Table 20-6. TECxINT Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10	TECXFLTIE	RW	0h	External fault interrupt enable. This bit enables the TECXFLTIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
9	TECEXCLRIE	RW	0h	External clear interrupt enable. This bit enables the TECEXCLRIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
8	TECAXCLRIE	RW	0h	Auxiliary interrupt enable. This bit enables the TECAXCLRIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
7-3	Reserved	R	0h	Reserved. Always reads as 0.
2	TECXFLTIFG	RW	0h	External fault interrupt flag. This bit is set if one of the external fault signal TECXFLTx is detected. Software has to look into the control register TECSTA to find out which one it is. If the bit is set it remains set until cleared by reading it or writing a 0 to it. 0b = No interrupt pending 1b = Interrupt pending
1	TECEXCLRIFG	RW	0h	External clear interrupt flag. This bit is set if the external clear signal is detected. If the bit is set it remains set until cleared by reading it or writing a 0 to it. 0b = No interrupt pending 1b = Interrupt pending
0	TECAXCLRIFG	RW	0h	Auxiliary clear interrupt flag. This bit is set if the auxiliary clear signal is detected. If the bit is set it remains set until cleared by reading it or writing a 0 to it. 0b = No interrupt pending 1b = Interrupt pending

### **20.3.6 TECxIV Register**

Timer Event Control Interrupt Vector Register

**Figure 20-11. TECxIV Register**

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

**Table 20-7. TECxIV Register Description**

Bit	Field	Type	Reset	Description
15-0	TECxIVx	R	0h	TEC external interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: External fault; Interrupt Flag: TECXFLTIFG; Interrupt Priority: Highest 04h = Interrupt Source: External clear; Interrupt Flag: TECEXCLRIFG 06h = Interrupt Source: Auxiliary clear; Interrupt Flag: TECAAXCLRIFG; Interrupt Priority: Lowest

## Real-Time Clock (RTC) Overview

### 21.1 RTC Overview

**Table 21-1. RTC Overview**

Feature	RTC_A	RTC_B LPM3.5, Calendar Mode Only	RTC_C Protection Plus Improved Calibration and Compensation
Calendar Mode	Yes	Yes	Yes
Counter Mode	Yes	No	Optional (device-dependent) <sup>(1)</sup>
Programmable Alarms	Yes	Yes	Yes
Password Protected Calendar Registers	No	No	Yes
Input Clocks	ALCK, SMCLK	32-kHz crystal oscillator	32-kHz crystal oscillator
LPM3.5 Support	No	Yes	Yes
Offset Calibration Register	Yes	Yes	Yes
Temperature Compensation Register	No	No	Yes
Frequency Adjustment Range	-2.035 ppm $\times$ 63 ≈ -128 ppm +4.069 ppm $\times$ 63 ≈ +256 ppm	-2.17 ppm $\times$ 59 ≈ -128 ppm +4.34 ppm $\times$ 59 ≈ +256 ppm	-240 ppm, +240 ppm <sup>(2)</sup>
Frequency Adjustment Steps	-2.035 ppm, +4.069 ppm	-2.17 ppm, +4.34 ppm	-1 ppm, +1 ppm
Temperature Compensation	With software, manipulating offset calibration value	With software, manipulating offset calibration value	With software using separate temperature compensation register
Calibration and Compensation Period	64 min	60 min	1 min
BCD to Binary Conversion	Integrated for Calendar Mode	Integrated for Calendar Mode plus separate conversion registers	Integrated for Calendar Mode plus separate conversion registers
Event/Tamper Detect With Time Stamp	No	No	Optional (device-dependent) <sup>(1)</sup>

<sup>(1)</sup> See the device-specific data sheet.

<sup>(2)</sup> Total adjustment range of offset calibration plus temperature compensation. See the RTC\_C chapter for details.

## **Real-Time Clock (RTC\_A)**

The Real-Time Clock (RTC\_A) module provides clock counters with a calendar, a flexible programmable alarm, and calibration. This chapter describes the RTC\_A module.

Topic	Page
22.1 RTC_A Introduction .....	566
22.2 RTC_A Operation .....	568
22.3 RTC_A Registers .....	574

## 22.1 RTC\_A Introduction

The RTC\_A module provides a real-time clock and calendar function that can also be configured as a general-purpose counter.

RTC\_A features include:

- Configurable for real-time clock with calendar function or general-purpose counter
- Provides seconds, minutes, hours, day of week, day of month, month, and year in real-time clock with calendar function
- Interrupt capability
- Selectable BCD or binary format in real-time clock mode
- Programmable alarms in real-time clock mode
- Calibration logic for time offset correction in real-time clock mode

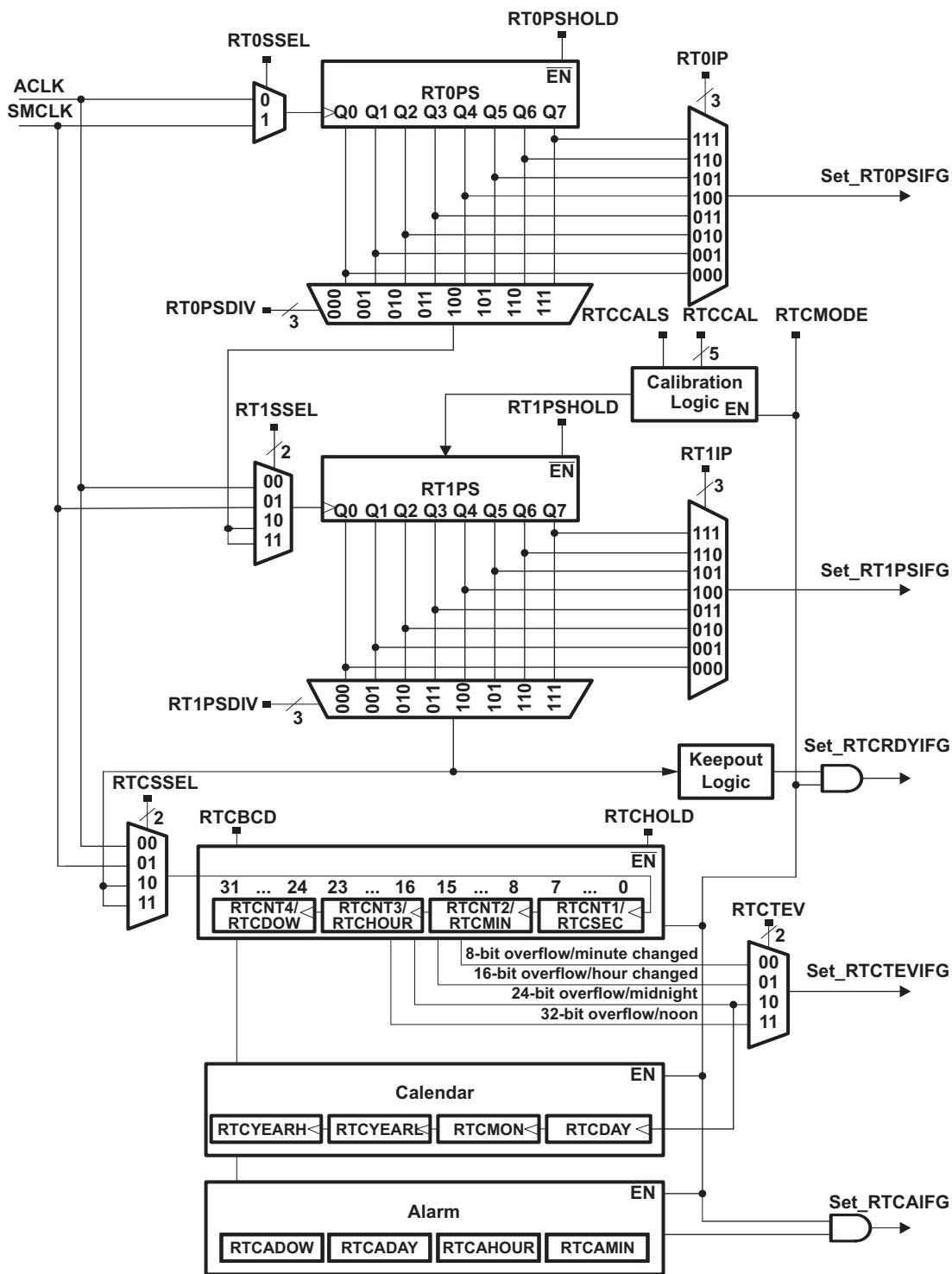
The RTC\_A block diagram is shown in [Figure 22-1](#).

---

**NOTE: Real-time clock initialization**

Most RTC\_A module registers have no initial condition. These registers must be configured by user software before use.

---


**Figure 22-1. RTC\_A**

## 22.2 RTC\_A Operation

The RTC\_A module can be configured as a real-time clock with calendar function (calendar mode) or as a 32-bit general purpose counter (counter mode) with the RTCMODE bit.

### 22.2.1 Counter Mode

Counter mode is selected when RTCMODE is reset. In this mode, a 32-bit counter is provided that is directly accessible by software. Switching from calendar mode to counter mode resets the count value (RTCNT1, RTCNT2, RTCNT3, RTCNT4), as well as the prescale counters (RT0PS, RT1PS).

The clock to increment the counter can be sourced from ACLK, SMCLK, or prescaled versions of ACLK or SMCLK. Prescaled versions of ACLK or SMCLK are sourced from the prescale dividers (RT0PS and RT1PS). RT0PS and RT1PS output /2, /4, /8, 16, /32, /64, /128, and /256 versions of ACLK and SMCLK, respectively. The output of RT0PS can be cascaded with RT1PS. The cascaded output can be used as a clock source input to the 32-bit counter.

Four individual 8-bit counters are cascaded to provide the 32-bit counter. This provides 8-bit, 16-bit, 24-bit, or 32-bit overflow intervals of the counter clock. The RTCTEV bits select the respective trigger event. An RTCTEV event can trigger an interrupt by setting the RTCTEVIE bit. Each counter, RTCNT1 through RTCNT4, is individually accessible and may be written to.

RT0PS and RT1PS can be configured as two 8-bit counters or cascaded into a single 16-bit counter. RT0PS and RT1PS can be halted on an individual basis by setting their respective RT0PSHOLD and RT1PSHOLD bits. When RT0PS is cascaded with RT1PS, setting RT0PSHOLD causes both RT0PS and RT1PS to be halted. The 32-bit counter can be halted several ways depending on the configuration. If the 32-bit counter is sourced directly from ACLK or SMCLK, it can be halted by setting RTCHOLD. If it is sourced from the output of RT1PS, it can be halted by setting RT1PSHOLD or RTCHOLD. Finally, if it is sourced from the cascaded outputs of RT0PS and RT1PS, it can be halted by setting RT0PSHOLD, RT1PSHOLD, or RTCHOLD.

---

**NOTE: Accessing the RTCNT1, RTCNT2, RTCNT3, RTCNT4, RT0PS, RT1PS registers**

When the counter clock is asynchronous to the CPU clock, any read from any RTCNT1, RTCNT2, RTCNT3, RTCNT4, RT0PS, or RT1PS register should occur while the counter is not operating. Otherwise, the results may be unpredictable. Alternatively, the counter may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to these registers takes effect immediately.

---

### 22.2.2 Calendar Mode

Calendar mode is selected when RTCMODE is set. In calendar mode, the RTC\_A module provides seconds, minutes, hours, day of week, day of month, month, and year in selectable BCD or hexadecimal format. The calendar includes a leap-year algorithm that considers all years evenly divisible by four as leap years. This algorithm is accurate from the year 1901 through 2099.

#### 22.2.2.1 Real-Time Clock and Prescale Dividers

The prescale dividers, RT0PS and RT1PS, are automatically configured to provide a 1-s clock interval for the RTC\_A. RT0PS is sourced from ACLK. ACLK must be set to 32768 Hz (nominal) for proper RTC\_A calendar operation. RT1PS is cascaded with the output ACLK/256 of RT0PS. The RTC\_A is sourced with the /128 output of RT1PS, thereby providing the required 1-s interval. Switching from counter to calendar mode clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1. In addition, RT0PS and RT1PS are cleared.

When RTCBCD = 1, BCD format is selected for the calendar registers. The format must be selected before the time is set. Changing the state of RTCBCD clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1. In addition, RT0PS and RT1PS are cleared.

In calendar mode, the RT0SSEL, RT1SSEL, RT0PSDIV, RT1PSDIV, RT0PSHOLD, RT1PSHOLD, and RTCSEL bits are don't care. Setting RTCHOLD halts the real-time counters and prescale counters, RT0PS and RT1PS.

### 22.2.2.2 Real-Time Clock Alarm Function

The RTC\_A module provides for a flexible alarm system. There is a single user-programmable alarm that can be programmed based on the settings contained in the alarm registers for minutes, hours, day of week, and day of month. The user-programmable alarm function is only available in the calendar mode of operation.

Each alarm register contains an alarm enable (AE) bit that can be used to enable the respective alarm register. By setting AE bits of the various alarm registers, a variety of alarm events can be generated.

- Example 1: A user wishes to set an alarm every hour at 15 minutes past the hour; that is, at 00:15:00, 01:15:00, 02:15:00, and so on. This is possible by setting RTCAMIN to 15. By setting the AE bit of the RTCAMIN and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the AF is set when the count transitions from 00:14:59 to 00:15:00, 01:14:59 to 01:15:00, 02:14:59 to 02:15:00, etc.
- Example 2: A user wishes to set an alarm every day at 04:00:00. This is possible by setting RTCAHOUR to 4. By setting the AE bit of the RTCHOUR and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the AF is set when the count transitions from 03:59:59 to 04:00:00.
- Example 3: A user wishes to set an alarm for 06:30:00. RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the AF is set when the time count transitions from 06:29:59 to 06:30:00. In this case, the alarm event occurs every day at 06:30:00.
- Example 4: A user wishes to set an alarm every Tuesday at 06:30:00. RTCADOW would be set to 2, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADOW, RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the AF is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCADOW transitions from 1 to 2.
- Example 5: A user wishes to set an alarm the fifth day of each month at 06:30:00. RTCADAY would be set to 5, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADAY, RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the AF is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCADAY equals 5.

---

**NOTE: Invalid alarm settings**

Invalid alarm settings are not checked via hardware. It is the user's responsibility to ensure that valid alarm settings are entered.

---

**NOTE: Invalid time and date values**

Writing of invalid date and/or time information or data values outside the legal ranges specified in the RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCDOW, RTCYEARH, RTCYEARL, RTCAMIN, RTCAHOUR, RTCADAY, and RTCADOW registers can result in unpredictable behavior.

---

**NOTE: Setting the alarm**

To prevent potential erroneous alarm conditions from occurring, the alarms should be disabled by clearing the RTCAIE, RTCAIFG, and AE bits prior to writing new time values to the RTC time registers.

---

### 22.2.2.3 Reading or Writing Real-Time Clock Registers in Calendar Mode

Because the system clock may be asynchronous to the RTC\_A clock source, special care must be taken when accessing the real-time clock registers.

In calendar mode, the real-time clock registers are updated once per second. To prevent reading any real-time clock register at the time of an update, which could result in an invalid time being read, a keepout window is provided. The keepout window is centered approximately -128/32768 s around the update transition. The read-only RTCRDY bit is reset during the keepout window period and set outside the keepout the window period. Any read of the clock registers while RTCRDY is reset is considered to be potentially invalid, and the time read should be ignored.

An easy way to safely read the real-time clock registers is to use the RTCRDYIFG interrupt flag. Setting RTCRDYIE enables the RTCRDYIFG interrupt. Once enabled, an interrupt is generated based on the rising edge of the RTCRDY bit, causing the RTCRDYIFG to be set. At this point, the application has nearly a complete second to safely read any or all of the real-time clock registers. This synchronization process prevents reading the time value during transition. The RTCRDYIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

In counter mode, the RTCRDY bit remains reset. RTCRDYIE is a don't care and RTCRDYIFG remains reset.

**NOTE: Reading or writing real-time clock registers**

When the counter clock is asynchronous to the CPU clock, any read from any RTCSEC, RTCMIN, RTCHOUR, RTCDOW, RTCDAY, RTCMON, RTCYEARL, or RTCYEARH register while the RTCRDY is reset may result in invalid data being read. To safely read the counting registers, either polling of the RTCRDY bit or the synchronization procedure previously described can be used. Alternatively, the counter register can be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Reading the RT0PS and RT1PS can only be handled by reading the registers multiple times and a majority vote taken in software to determine the correct reading or by halting the counters.

Any write to any counting register takes effect immediately. However, the clock is stopped during the write. In addition, RT0PS and RT1PS registers are reset. This could result in losing up to 1 s during a write. Writing of data outside the legal ranges or invalid time stamp combinations results in unpredictable behavior.

### 22.2.3 Real-Time Clock Interrupts

The RTC\_A module has five interrupt sources available, each with independent enables and flags.

#### 22.2.3.1 Real-Time Clock Interrupts in Calendar Mode

In calendar mode, five sources for interrupts are available, namely RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, and RTCAIFG. These flags are prioritized and combined to source a single interrupt vector. The interrupt vector register (RTCIV) is used to determine which flag requested an interrupt.

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

Any access, read or write, of the RTCIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. In addition, all flags can be cleared via software.

The user-programmable alarm event sources the real-time clock interrupt, RTCAIFG. Setting RTCAIE enables the interrupt. In addition to the user-programmable alarm, the RTC\_A module provides for an interval alarm that sources real-time clock interrupt, RTCTEVIFG. The interval alarm can be selected to cause an alarm event when RTCMIN changed or RTCHOUR changed, every day at midnight (00:00:00) or every day at noon (12:00:00). The event is selectable with the RTCTEV bits. Setting the RTCTEVIE bit enables the interrupt.

The RTCRDY bit sources the real-time clock interrupt, RTCRDYIFG, and is useful in synchronizing the read of time registers with the system clock. Setting the RTCRDYIE bit enables the interrupt.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. In calendar mode, RT0PS is sourced with ACLK at 32768 Hz, so intervals of 16384 Hz, 8192 Hz, 4096 Hz, 2048 Hz, 1024 Hz, 512 Hz, 256 Hz, or 128 Hz are possible. Setting the RT0PSIE bit enables the interrupt.

RT1PSIFG can generate interrupt intervals selectable by the RT1IP bits. In calendar mode, RT1PS is sourced with the output of RT0PS, which is 128 Hz (32768/256 Hz). Therefore, intervals of 64 Hz, 32 Hz, 16 Hz, 8 Hz, 4 Hz, 2 Hz, 1 Hz, or 0.5 Hz are possible. Setting the RT1PSIE bit enables the interrupt.

### **22.2.3.2 Real-Time Clock Interrupts in Counter Mode**

In counter mode, three interrupt sources are available: RT0PSIFG, RT1PSIFG, and RTCTEVIFG. RTCAIFG and RTCRDYIFG are cleared. RTCRDYIE and RTCAIE are don't care.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. In counter mode, RT0PS is sourced with ACLK or SMCLK, so divide ratios of /2, /4, /8, /16, /32, /64, /128, and /256 of the respective clock source are possible. Setting the RT0PSIE bit enables the interrupt.

RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. In counter mode, RT1PS is sourced with ACLK, SMCLK, or the output of RT0PS, so divide ratios of /2, /4, /8, /16, /32, /64, /128, and /256 of the respective clock source are possible. Setting the RT1PSIE bit enables the interrupt.

The RTC\_A module provides for an interval timer that sources real-time clock interrupt, RTCTEVIFG. The interval timer can be selected to cause an interrupt event when an 8-bit, 16-bit, 24-bit, or 32-bit overflow occurs within the 32-bit counter. The event is selectable with the RTCTEV bits. Setting the RTCTEVIE bit enables the interrupt.

### **RTCIV Software Example**

The following software example shows the recommended use of RTCIV and the handling overhead. The RTCIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

<i>; Interrupt handler for RTC interrupt flags.</i>		<i>Cycles</i>
RTC_HND	<i>; Interrupt latency</i>	6
ADD &RTCIV, PC	<i>; Add offset to Jump table</i>	3
RETI	<i>; Vector 0: No interrupt</i>	5
JMP RTCRDYIFG_HND	<i>; Vector 2: RTCRDYIFG</i>	2
JMP RTCTEVIFG_HND	<i>; Vector 4: RTCTEVIFG</i>	2
JMP RTCAIFG	<i>; Vector 6: RTCAIFG</i>	5
JMP RT0PSIFG	<i>; Vector 8: RT0PSIFG</i>	5
JMP RT1PSIFG	<i>; Vector A: RT1PSIFG</i>	5
RETI	<i>; Vector C: Reserved</i>	5
RTCRDYIFG_HND	<i>; Vector 2: RTCRDYIFG Flag</i>	
to	<i>; Task starts here</i>	
RETI		5
RTCTEVIFG_HND	<i>; Vector 4: RTCTEVIFG</i>	
to	<i>; Task starts here</i>	
RETI	<i>; Back to main program</i>	5
RTCAIFG_HND	<i>; Vector 6: RTCAIFG</i>	
to	<i>; Task starts here</i>	
RT0PSIFG_HND	<i>; Vector 8: RT0PSIFG</i>	
to	<i>; Task starts here</i>	
RT1PSIFG_HND	<i>; Vector A: RT1PSIFG</i>	
to	<i>; Task starts here</i>	

## 22.2.4 Real-Time Clock Calibration

The RTC\_A module has calibration logic that allows for adjusting the crystal frequency in approximately +4-ppm or -2-ppm steps, allowing for higher time keeping accuracy from standard crystals. The RTCCAL bits are used to adjust the frequency. When RTCCALS is set, each RTCCAL LSB causes a  $\approx$  +4-ppm adjustment. When RTCCALS is cleared, each RTCCAL LSB causes a  $\approx$  -2-ppm adjustment. Calibration is available only in calendar mode. In counter mode (RTCMODE = 0), the calibration logic is disabled.

Calibration is accomplished by periodically adjusting the RT1PS counter based on the RTCCALS and RTCCALx settings. In calendar mode, the RT0PS divides the nominal 37268-Hz low-frequency (LF) crystal clock input by 256. A 64-minute period has  $32768 \text{ cycles/sec} \times 60 \text{ sec/min} \times 64 \text{ min} = 125829120$  cycles. Therefore a -2-ppm reduction in frequency (down calibration) approximately equates to adding an additional 256 cycles every 125829120 cycles ( $256/125829120 = 2.035 \text{ ppm}$ ). This is accomplished by holding the RT1PS counter for one additional clock of the RT0PS output within a 64-minute period. Similarly, a +4-ppm increase in frequency (up calibration) approximately equates to removing 512 cycles every 125829120 cycle ( $512/125829120 = 4.069 \text{ ppm}$ ). This is accomplished by incrementing the RT1PS counter for two additional clocks of the RT0PS output within a 64-minute period. Each RTCCALx calibration bit causes either 256 LF crystal clock cycles to be added every 64 minutes or 512 LF crystal clock cycles to be subtracted every 64 minutes, giving a frequency adjustment of approximately -2 ppm or +4 ppm, respectively.

To calibrate the frequency, the RTCCLK output signal is available at a pin. The RTCCALF bits can be used to select the frequency rate of the RTCCLK output signal, either no signal, 512 Hz, 256 Hz, or 1 Hz.

The basic flow to calibrate the frequency is as follows:

1. Configure the RTCCLK pin.
2. Measure the RTCCLK output signal with an appropriate resolution frequency counter; that is, within the resolution required.
3. Compute the absolute error in ppm:  $\text{Absolute Error (ppm)} = |10^6 \times (f_{\text{MEASURED}} - f_{\text{RTCCLK}}) / f_{\text{RTCCLK}}|$ , where  $f_{\text{RTCCLK}}$  is the expected frequency of 512 Hz, 256 Hz, or 1 Hz.
4. Adjust the frequency, by performing the following:
  - (a) If the frequency is too low, set RTCALS = 1 and apply the appropriate RTCCALx bits, where  $\text{RTCCALx} = (\text{Absolute Error}) / 4.069$ , rounded to the nearest integer.
  - (b) If the frequency is too high, clear RTCALS = 0 and apply the appropriate RTCCALx bits, where  $\text{RTCCALx} = (\text{Absolute Error}) / 2.035$ , rounded to the nearest integer.

For example, assume that RTCCLK is output at a frequency of 512 Hz. The measured RTCCLK is 511.9658 Hz. The frequency error is approximately 66.8 ppm low. To increase the frequency by 66.8 ppm, RTCCALS would be set, and RTCCAL would be set to 16 ( $66.8/4.069$ ). Similarly, assume that the measured RTCCLK is 512.0125 Hz. The frequency error is approximately 24.4 ppm high. To decrease the frequency by 24.4 ppm, RTCCALS would be cleared, and RTCCAL would be set to 12 ( $24.4 / 2.035$ ).

The calibration corrects only initial offsets and does not adjust for temperature and aging effects. This can be handled by periodically measuring temperature and using the crystal's characteristic curve to adjust the ppm based on temperature as required. In counter mode (RTCMODE = 0), the calibration logic is disabled.

---

### NOTE: Minimum Possible Calibration

The minimal calibration possible is -4 ppm or +8 ppm. For example, setting RTCCALS = 0 and RTCCAL = 0h would result in a -4 ppm decrease in frequency. Similarly, setting RTCCALS = 1 and RTCCAL = 0h would result in a +8 ppm increase in frequency.

---

---

**NOTE: Calibration output frequency**

The 512-Hz and 256-Hz output frequencies observed at the RTCCLK pin are not affected by changes in the calibration settings since these output frequencies are generated prior to the calibration logic. The 1-Hz output frequency is affected by changes in the calibration settings. Because the frequency change is small and infrequent over a very long time interval, it can be difficult to observe.

---

## 22.3 RTC\_A Registers

The RTC\_A module registers are listed in and [Table 22-1](#). The base register for the RTC\_A module registers can be found in the device-specific data sheet. The address offsets are given in [Table 22-1](#).

---

**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 22-1. RTC\_A Registers**

Offset	Acronym	Register Name	Type	Access	Reset
00h	RTCCTL01	Real-Time Clock Control 0, 1	Read/write	Word	4000h
00h	RTCCTL0 or RTCCTL01_L	Real-Time Clock Control 0	Read/write	Byte	00h
01h	RTCCTL1 or RTCCTL01_H	Real-Time Clock Control 1	Read/write	Byte	40h
02h	RTCCTL23	Real-Time Clock Control 2, 3	Read/write	Word	0000h
02h	RTCCTL2 or RTCCTL23_L	Real-Time Clock Control 2	Read/write	Byte	00h
03h	RTCCTL3 or RTCCTL23_H	Real-Time Clock Control 3	Read/write	Byte	00h
08h	RTCP0SCTL	Real-Time Prescale Timer 0 Control	Read/write	Word	0100h
08h	RTCP0SCTL or RTCP0SCTL_L		Read/write	Byte	00h
09h	RTCP0SCTLH or RTCP0SCTL_H		Read/write	Byte	01h
0Ah	RTCP1SCTL	Real-Time Prescale Timer 1 Control	Read/write	Word	0100h
0Ah	RTCP1SCTL or RTCP1SCTL_L		Read/write	Byte	00h
0Bh	RTCP0SCTLH or RTCP0SCTL_H		Read/write	Byte	01h
0Ch	RTCP0S or RTCP0S_L	Real-Time Prescale Timer 0, 1 Counter	Read/write	Word	undefined
0Ch	RT0PS or RTCP0S_L	Real-Time Prescale Timer 0 Counter	Read/write	Byte	undefined
0Dh	RT1PS or RTCP0S_H	Real-Time Prescale Timer 1 Counter	Read/write	Byte	undefined
0Eh	RTCIV	Real Time Clock Interrupt Vector	Read	Word	0000h
0Eh	RTCIV_L		Read	Byte	00h
0Fh	RTCIV_H		Read	Byte	00h
10h	RTCTIM0 or RTCNT12	Real-Time Clock Seconds, Minutes	Read/write	Word	undefined
10h	RTCSEC RTCN1 or RTCTIM0_L	Real-Time Counter 1, 2	Read/write	Byte	undefined
11h	RTCMIN RTCN2 or RTCTIM0_H	Real-Time Clock Seconds	Read/write	Byte	undefined
11h	RTCN1 or RTCTIM0_L	Real-Time Counter 1	Read/write	Byte	undefined
12h	RTCTIM1 or RTCNT34	Real-Time Clock Minutes	Read/write	Word	undefined
12h	RTCHOUR	Real-Time Counter 2	Read/write	Byte	undefined
12h	RTCHOUR	Real-Time Clock Hour, Day of Week	Read/write	Word	undefined
12h	RTCHOUR	Real-Time Counter 3, 4	Read/write	Byte	undefined
12h	RTCHOUR	Real-Time Clock Hour	Read/write	Byte	undefined

**Table 22-1. RTC\_A Registers (continued)**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Access</b>	<b>Reset</b>
	RTCNT3 or RTCTIM1_L	Real-Time Counter 3			
13h	RTCDOW RTCNT4 or RTCTIM1_H	Real-Time Clock Day of Week Real-Time Counter 4	Read/write	Byte	undefined
14h	RTCDATE	Real-Time Clock Date	Read/write	Word	undefined
14h	RTCDAY or RTCDATE_L	Real-Time Clock Day of Month	Read/write	Byte	undefined
15h	RTCMON or RTCDATE_H	Real-Time Clock Month	Read/write	Byte	undefined
16h	RTCYEAR	Real-Time Clock Year	Read/write	Word	undefined
16h	RTCYEARL or RTCYEAR_L		Read/write	Byte	undefined
17h	RTCYEARH or RTCYEAR_H		Read/write	Byte	undefined
18h	RTCAMINHR	Real-Time Clock Minutes, Hour Alarm	Read/write	Word	undefined
18h	RTCAMIN or RTCAMINHR_L	Real-Time Clock Minutes Alarm	Read/write	Byte	undefined
19h	RTCAHOUR or RTCAMINHR_H	Real-Time Clock Hours Alarm	Read/write	Byte	undefined
1Ah	RTCADOWDAY	Real-Time Clock Day of Week, Day of Month Alarm	Read/write	Word	undefined
1Ah	RTCADOW or RTCADOWDAY_L	Real-Time Clock Day of Week Alarm	Read/write	Byte	undefined
1Bh	RTCADAY or RTCADOWDAY_H	Real-Time Clock Day of Month Alarm	Read/write	Byte	undefined

**22.3.1 RTCCTL0 Register**

Real-Time Clock Control 0 Register

**Figure 22-2. RTCCTL0 Register**

7	6	5	4	3	2	1	0
Reserved	RTCTEVIE	RTCAIE	RTCRDYIE	Reserved	RTCTEVIFG	RTCAIFG	RTCRDYIFG
r0	rw-0	rw-0	rw-0	r0	rw-(0)	rw-(0)	rw-(0)

**Table 22-2. RTCCTL0 Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	R	0h	Reserved. Always reads as 0.
6	RTCTEVIE	RW	0h	Real-time clock time event interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
5	RTCAIE	RW	0h	Real-time clock alarm interrupt enable. This bit remains cleared when in counter mode (RTCMODE = 0). 0b = Interrupt not enabled 1b = Interrupt enabled
4	RTCRDYIE	RW	0h	Real-time clock read ready interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
3	Reserved	R	0h	Reserved. Always reads as 0.
2	RTCTEVIFG	RW	0h	Real-time clock time event flag 0b = No time event occurred. 1b = Time event occurred.
1	RTCAIFG	RW	0h	Real-time clock alarm flag. This bit remains cleared when in counter mode (RTCMODE = 0). 0b = No time event occurred. 1b = Time event occurred.
0	RTCRDYIFG	RW	0h	Real-time clock read ready flag 0b = RTC cannot be read safely. 1b = RTC can be read safely.

### 22.3.2 RTCCTL1 Register

Real-Time Clock Control Register 1

**Figure 22-3. RTCCTL1 Register**

7	6	5	4	3	2	1	0
RTCBBCD	RTCHOLD	RTCMODE	RTCRDY	RTCSSEL		RTCTEV	
rw-(0)	rw-(1)	rw-(0)	r-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 22-3. RTCCTL1 Register Description**

Bit	Field	Type	Reset	Description
7	RTCBBCD	RW	0h	Real-time clock BCD select. Selects BCD counting for real-time clock. Applies to calendar mode (RTCMODE = 1) only; setting is ignored in counter mode. Changing this bit clears seconds, minutes, hours, day of week, and year to 0 and sets day of month and month to 1. The real-time clock registers must be set by software afterwards. 0b = Binary (hexadecimal) code selected 1b = Binary coded decimal (BCD) code selected
6	RTCHOLD	RW	1h	Real-time clock hold 0b = Real-time clock (32-bit counter or calendar mode) is operational. 1b = In counter mode (RTCMODE = 0), only the 32-bit counter is stopped. In calendar mode (RTCMODE = 1), the calendar is stopped as well as the prescale counters, RT0PS and RT1PS. RT0PSHOLD and RT1PSHOLD are don't care.
5	RTCMODE	RW	0h	Real-time clock mode 0b = 32-bit counter mode 1b = Calendar mode. Switching between counter and calendar mode resets the real-time clock counter registers. Switching to calendar mode clears seconds, minutes, hours, day of week, and year to 0 and sets day of month and month to 1. The real-time clock registers must be set by software afterwards. RT0PS and RT1PS are also cleared.
4	RTCRDY	RW	0h	Real-time clock ready 0b = RTC time values in transition (calendar mode only) 1b = RTC time values safe for reading (calendar mode only). This bit indicates when the real-time clock time values are safe for reading (calendar mode only). In counter mode, RTCRDY signal remains cleared.
3-2	RTCSSEL	RW	0h	Real-time clock source select. Selects clock input source to the RTC/32-bit counter. In calendar mode, these bits are don't care. The clock input is automatically set to the output of RT1PS. 00b = ACLK 01b = SMCLK 10b = Output from RT1PS 11b = Output from RT1PS
1-0	RTCTEV	RW	0h	Real-time clock time event Counter mode (RTCMODE = 0) 00b = 8-bit overflow 01b = 16-bit overflow 10b = 24-bit overflow 11b = 32-bit overflow Calendar mode (RTCMODE = 1) 00b = Minute changed 01b = Hour changed 10b = Every day at midnight (00:00) 11b = Every day at noon (12:00)

**22.3.3 RTCCTL2 Register**

Real-Time Clock Control 2 Register

**Figure 22-4. RTCCTL2 Register**

7	6	5	4	3	2	1	0
RTCCALS	Reserved	RTCCAL					
rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 22-4. RTCCTL2 Register Description**

Bit	Field	Type	Reset	Description
7	RTCCALS	RW	0h	Real-time clock calibration sign 0b = Frequency adjusted down 1b = Frequency adjusted up
6	Reserved	R	0h	Reserved. Always reads as 0.
5-0	RTCCAL	RW	0h	Real-time clock calibration. Each LSB represents approximately +4ppm (RTCCALS = 1) or a -2ppm (RTCCALS = 0) adjustment in frequency.

**22.3.4 RTCCTL3 Register**

Real-Time Clock Control 3 Register

**Figure 22-5. RTCCTL3 Register**

7	6	5	4	3	2	1	0
Reserved							RTCCALF
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)

**Table 22-5. RTCCTL3 Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1-0	RTCCALF	RW	0h	Real-time clock calibration frequency. Selects frequency output to RTCCLK pin for calibration measurement. The corresponding port must be configured for the peripheral module function. The RTCCLK is not available in counter mode and remains low, and the RTCCALF bits are don't care. 00b = No frequency output to RTCCLK pin 01b = 512 Hz 10b = 256 Hz 11b = 1 Hz

### 22.3.5 RTCNT1 Register

Real-Time Clock Counter 1 Register – Counter Mode

**Figure 22-6. RTCNT1 Register**

7	6	5	4	3	2	1	0
RTCNT1							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-6. RTCNT1 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT1	RW	undefined	The RTCNT1 register is the count of RTCNT1

### 22.3.6 RTCNT2 Register

Real-Time Clock Counter 2 Register – Counter Mode

**Figure 22-7. RTCNT2 Register**

7	6	5	4	3	2	1	0
RTCNT2							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-7. RTCNT2 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT2	RW	undefined	The RTCNT2 register is the count of RTCNT2

### 22.3.7 RTCNT3 Register

Real-Time Clock Counter 3 Register – Counter Mode

**Figure 22-8. RTCNT3 Register**

7	6	5	4	3	2	1	0
RTCNT3							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-8. RTCNT3 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT3	RW	undefined	The RTCNT3 register is the count of RTCNT3

### 22.3.8 RTCNT4 Register

Real-Time Clock Counter 4 Register – Counter Mode

**Figure 22-9. RTCNT4 Register**

7	6	5	4	3	2	1	0
RTCNT4							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-9. RTCNT4 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT4	RW	undefined	The RTCNT4 register is the count of RTCNT4.

**22.3.9 RTCSEC Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Seconds Register – Calendar Mode With Hexadecimal Format

**Figure 22-10. RTCSEC Register**

7	6	5	4	3	2	1	0
Seconds							
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 22-10. RTCSEC Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always 0
5-0	Seconds	RW	undefined	Seconds (0 to 59)

**22.3.10 RTCSEC Register – Calendar Mode With BCD Format**

Real-Time Clock Seconds Register – Calendar Mode With BCD Format

**Figure 22-11. RTCSEC Register**

7	6	5	4	3	2	1	0
Seconds – high digit				Seconds – low digit			
r-0	rw	rw	rw	rw	rw	rw	rw

**Table 22-11. RTCSEC Register Description**

Bit	Field	Type	Reset	Description
7	0	R	0h	Always 0
6-4	Seconds – high digit	RW	undefined	Seconds – high digit (0 to 5)
3-0	Seconds – low digit	RW	undefined	Seconds – low digit (0 to 9)

### 22.3.11 RTCMIN Register – Calendar Mode With Hexadecimal Format

Real-Time Clock Minutes Register – Calendar Mode With Hexadecimal Format

**Figure 22-12. RTCMIN Register**

7	6	5	4	3	2	1	0
0	Minutes						
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 22-12. RTCMIN Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always 0
5-0	Minutes	RW	undefined	Minutes (0 to 59)

### 22.3.12 RTCMIN Register – Calendar Mode With BCD Format

Real-Time Clock Minutes Register – Calendar Mode With BCD Format

**Figure 22-13. RTCMIN Register**

7	6	5	4	3	2	1	0
0	Minutes – high digit						
r-0	rw	rw	rw	rw	rw	rw	rw

**Table 22-13. RTCMIN Register Description**

Bit	Field	Type	Reset	Description
7	0	R	0h	Always 0
6-4	Minutes – high digit	RW	undefined	Minutes – high digit (0 to 5)
3-0	Minutes – low digit	RW	undefined	Minutes – low digit (0 to 9)

**22.3.13 RTCHOUR Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Hours Register – Calendar Mode With Hexadecimal Format

**Figure 22-14. RTCHOUR Register**

7	6	5	4	3	2	1	0
				Hours			
r-0	r-0	r-0	rw	rw	rw	rw	rw

**Table 22-14. RTCHOUR Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always 0
4-0	Hours	RW	undefined	Hours (0 to 23)

**22.3.14 RTCHOUR Register – Calendar Mode With BCD Format**

Real-Time Clock Hours Register – Calendar Mode With BCD Format

**Figure 22-15. RTCHOUR Register**

7	6	5	4	3	2	1	0
			Hours – high digit		Hours – low digit		
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 22-15. RTCHOUR Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always 0
5-4	Hours – high digit	RW	undefined	Hours – high digit (0 to 2)
3-0	Hours – low digit	RW	undefined	Hours – low digit (0 to 9)

### 22.3.15 RTCDOW Register – Calendar Mode

Real-Time Clock Day of Week Register – Calendar Mode

**Figure 22-16. RTCDOW Register**

7	6	5	4	3	2	1	0		
			0	Day of week					
r-0	r-0	r-0	r-0	r-0	rw	rw	rw		

**Table 22-16. RTCDOW Register Description**

Bit	Field	Type	Reset	Description
7-3	0	R	0h	Always 0
2-0	Day of week	RW	undefined	Day of week (0 to 6)

### 22.3.16 RTCDAY Register – Calendar Mode With Hexadecimal Format

Real-Time Clock Day of Month Register – Calendar Mode With Hexadecimal Format

**Figure 22-17. RTCDAY Register**

7	6	5	4	3	2	1	0		
		0		Day of month					
r-0	r-0	r-0	rw	rw	rw	rw	rw		

**Table 22-17. RTCDAY Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always 0
4-0	Day of month	RW	undefined	Day of month (1 to 28, 29, 30, 31)

### 22.3.17 RTCDAY Register – Calendar Mode With BCD Format

Real-Time Clock Day of Month Register – Calendar Mode With BCD Format

**Figure 22-18. RTCDAY Register**

7	6	5	4	3	2	1	0
	0	Day of month – high digit				Day of month – low digit	
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 22-18. RTCDAY Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	
5-4	Day of month – high digit	RW	undefined	Day of month – high digit (0 to 3)
3-0	Day of month – low digit	RW	undefined	Day of month – low digit (0 to 9)

**22.3.18 RTCMON Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Month Register – Calendar Mode With Hexadecimal Format

**Figure 22-19. RTCMON Register**

7	6	5	4	3	2	1	0
0				Month			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

**Table 22-19. RTCMON Register Description**

Bit	Field	Type	Reset	Description
7-4	0	R	0h	Always 0
3-0	Month	RW	undefined	Month (1 to 12)

**22.3.19 RTCMON Register – Calendar Mode With BCD Format**

Real-Time Clock Month Register – Calendar Mode With BCD Format

**Figure 22-20. RTCMON Register**

7	6	5	4	3	2	1	0
0				Month – high digit	Month – low digit		
r-0	r-0	r-0	rw	rw	rw	rw	rw

**Table 22-20. RTCMON Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always 0
4	Month – high digit	RW	undefined	Month – high digit (0 or 1)
3-0	Month – low digit	RW	undefined	Month – low digit (0 to 9)

### 22.3.20 RTCYEARL Register – Calendar Mode With Hexadecimal Format

Real-Time Clock Year Low-Byte Register – Calendar Mode With Hexadecimal Format

**Figure 22-21. RTCYEARL Register**

7	6	5	4	3	2	1	0
Year – low byte of 0 to 4095							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-21. RTCYEARL Register Description**

Bit	Field	Type	Reset	Description
7-0	Year	RW	undefined	Year – low byte of 0 to 4095

### 22.3.21 RTCYEARL Register – Calendar Mode With BCD Format

Real-Time Clock Year Low-Byte Register – Calendar Mode With BCD Format

**Figure 22-22. RTCYEARL Register**

7	6	5	4	3	2	1	0
Decade					Year – lowest digit		
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-22. RTCYEARL Register Description**

Bit	Field	Type	Reset	Description
7-4	Decade	RW	undefined	Decade (0 to 9)
3-0	Year – lowest digit	RW	undefined	Year – lowest digit (0 to 9)

**22.3.22 RTCYEARH Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Year High-Byte Register – Calendar Mode With Hexadecimal Format

**Figure 22-23. RTCYEARH Register**

7	6	5	4	3	2	1	0
0				Year – high byte of 0 to 4095			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

**Table 22-23. RTCYEARH Register Description**

Bit	Field	Type	Reset	Description
7-4	0	R	0h	Always 0
3-0	Year	RW	undefined	Year – high byte of 0 to 4095

**22.3.23 RTCYEARH Register – Calendar Mode With BCD Format**

Real-Time Clock Year High-Byte Register – Calendar Mode With BCD Format

**Figure 22-24. RTCYEARH Register**

7	6	5	4	3	2	1	0
0				Century – high digit			
r-0	rw	rw	rw	rw	rw	rw	rw

**Table 22-24. RTCYEARH Register Description**

Bit	Field	Type	Reset	Description
7	0	R	0h	Always 0
6-4	Century – high digit	RW	undefined	Century – high digit (0 to 4)
3-0	Century – low digit	RW	undefined	Century – low digit (0 to 9)

### **22.3.24 RTCAMIN Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Minutes Alarm Register – Calendar Mode With Hexadecimal Format

**Figure 22-25. RTCAMIN Register**

7	6	5	4	3	2	1	0
AE	0						Minutes
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 22-25. RTCAMIN Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6	0	R	0h	Always 0
5-0	Minutes	RW	undefined	Minutes (0 to 59)

### **22.3.25 RTCAMIN Register – Calendar Mode With BCD Format**

Real-Time Clock Minutes Alarm Register – Calendar Mode With BCD Format

**Figure 22-26. RTCAMIN Register**

7	6	5	4	3	2	1	0
AE		Minutes – high digit			Minutes – low digit		
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-26. RTCAMIN Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	0h	Alarm enable
6-4	Minutes – high digit	RW	undefined	Minutes – high digit (0 to 5)
3-0	Minutes – low digit	RW	undefined	Minutes – low digit (0 to 9)

**22.3.26 RTCAHOUR Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Hours Alarm Register – Calendar Mode With Hexadecimal Format

**Figure 22-27. RTCAHOUR Register**

7	6	5	4	3	2	1	0
AE	0				Hours		
rw	r-0	r-0	rw	rw	rw	rw	rw

**Table 22-27. RTCAHOUR Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6-5	0	R	0h	Always 0
4-0	Hours	RW	undefined	Hours (0 to 23)

**22.3.27 RTCAHOUR Register – Calendar Mode With BCD Format**

Real-Time Clock Hours Alarm Register – Calendar Mode With BCD Format

**Figure 22-28. RTCAHOUR Register**

7	6	5	4	3	2	1	0
AE	0		Hours – high digit			Hours – low digit	
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 22-28. RTCAHOUR Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6	0	R	0h	Always 0
5-4	Hours – high digit	RW	undefined	Hours – high digit (0 to 2)
3-0	Hours – low digit	RW	undefined	Hours – low digit (0 to 9)

### 22.3.28 RTCADOW Register

Real-Time Clock Day of Week Alarm Register – Calendar Mode

**Figure 22-29. RTCADOW Register**

7	6	5	4	3	2	1	0
AE		0				Day of week	
rw	r-0	r-0	r-0	r-0	rw	rw	rw

**Table 22-29. RTCADOW Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6-3	0	R	0h	Always 0
2-0	Day of week	RW	undefined	Day of week (0 to 6)

### 22.3.29 RTCADAY Register – Calendar Mode With Hexadecimal Format

Real-Time Clock Day of Month Alarm Register – Calendar Mode With Hexadecimal Format

**Figure 22-30. RTCADAY Register**

7	6	5	4	3	2	1	0
AE		0			Day of month		
rw	r-0	r-0	rw	rw	rw	rw	rw

**Table 22-30. RTCADAY Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6-5	0	R	0h	Always 0
4-0	Day of month	RW	undefined	Day of month (1 to 28, 29, 30, 31)

### 22.3.30 RTCADAY Register – Calendar Mode With BCD Format

Real-Time Clock Day of Month Alarm Register – Calendar Mode With BCD Format

**Figure 22-31. RTCADAY Register**

7	6	5	4	3	2	1	0
AE	0	Day of month – high digit			Day of month – low digit		
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 22-31. RTCADAY Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6	0	R	0h	Always 0
5-4	Day of month – high digit	RW	undefined	Day of month – high digit (0 to 3)
3-0	Day of month – low digit	RW	undefined	Day of month – low digit (0 to 9)

**22.3.31 RTCPS0CTL Register**

Real-Time Clock Prescale Timer 0 Control Register

**Figure 22-32. RTCPS0CTL Register**

15	14	13	12	11	10	9	8
Reserved	RT0SSEL	RT0PSDIV			Reserved	RT0PSHOLD	
rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0	rw-1
7	6	5	4	3	2	1	0
Reserved			RT0IP			RT0PSIE	RT0PSIFG
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-(0)

**Table 22-32. RTCPS0CTL Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14	RT0SSEL	RW	0h	Prescale timer 0 clock source select. Selects clock input source to the RT0PS counter. In real-time clock calendar mode, these bits are do not care. RT0PS clock input is automatically set to the output of RT0PS. 0b = ACLK 1b = SMCLK
13-11	RT0PSDIV	RW	0h	Prescale timer 0 clock divide. These bits control the divide ratio of the RT0PS counter. In real-time clock calendar mode, these bits are don't care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128. 00b = Divide by 2 01b = Divide by 4 10b = Divide by 8 11b = Divide by 16 00b = Divide by 32 01b = Divide by 64 10b = Divide by 128 11b = Divide by 256
10-9	Reserved	R	0h	Reserved. Always reads as 0.
8	RT0PSHOLD	RW	1h	Prescale timer 0 hold. In real-time clock calendar mode, this bit is don't care. RT0PS is stopped via the RTCHOLD bit. 0b = RT0PS operational 1b = RT0PS held
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT0IP	RW	0h	Prescale timer 0 interrupt interval 00b = Divide by 2 01b = Divide by 4 10b = Divide by 8 11b = Divide by 16 00b = Divide by 32 01b = Divide by 64 10b = Divide by 128 11b = Divide by 256
1	RT0PSIE	RW	0h	Prescale timer 0 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
0	RT0PSIFG	RW	0h	Prescale timer 0 interrupt flag 0b = No time event occurred 1b = Time event occurred

### **22.3.32 RTCPS1CTL Register**

Real-Time Clock Prescale Timer 1 Control Register

**Figure 22-33. RTCPS1CTL Register**

15	14	13	12	11	10	9	8
RT1SSEL		RT1PSDIV			Reserved		RT1PSHOLD
rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0	rw-1
7	6	5	4	3	2	1	0
Reserved			RT1IP			RT1PSIE	RT1PSIFG
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-(0)

**Table 22-33. RTCPS1CTL Register Description**

Bit	Field	Type	Reset	Description
15-14	RT1SSEL	RW	0h	Prescale timer 1 clock source select. Selects clock input source to the RT1PS counter. In real-time clock calendar mode, these bits are do not care. RT1PS clock input is automatically set to the output of RT0PS. 00b = ACLK 01b = SMCLK 10b = Output from RT0PS 11b = Output from RT0PS
13-11	RT1PSDIV	RW	0h	Prescale timer 1 clock divide. These bits control the divide ratio of the RT0PS counter. In real-time clock calendar mode, these bits are don't care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128. 00b = Divide by 2 01b = Divide by 4 10b = Divide by 8 11b = Divide by 16 00b = Divide by 32 01b = Divide by 64 10b = Divide by 128 11b = Divide by 256
10-9	Reserved	R	0h	Reserved. Always reads as 0.
8	RT1PSHOLD	RW	1h	Prescale timer 1 hold. In real-time clock calendar mode, this bit is don't care. RT1PS is stopped via the RTCHOLD bit. 0b = RT1PS operational 1b = RT1PS held
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT1IP	RW	0h	Prescale timer 1 interrupt interval 00b = Divide by 2 01b = Divide by 4 10b = Divide by 8 11b = Divide by 16 00b = Divide by 32 01b = Divide by 64 10b = Divide by 128 11b = Divide by 256
1	RT1PSIE	RW	0h	Prescale timer 1 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
0	RT1PSIFG	RW	0h	Prescale timer 1 interrupt flag 0b = No time event occurred 1b = Time event occurred

**22.3.33 RT0PS Register**

Real-Time Clock Prescale Timer 0 Counter Register

**Figure 22-34. RT0PS Register**

7	6	5	4	3	2	1	0
RT0PS							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-34. RT0PS Register Description**

Bit	Field	Type	Reset	Description
7-0	RT0PS	RW	Undefined	Prescale timer 0 counter value

**22.3.34 RT1PS Register**

Real-Time Clock Prescale Timer 1 Counter Register

**Figure 22-35. RTPS1 Register**

7	6	5	4	3	2	1	0
RT1PS							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 22-35. RT1PS Register Description**

Bit	Field	Type	Reset	Description
7-0	RT1PS	RW	Undefined	Prescale timer 1 counter value

**22.3.35 RTCIV Register**

Real-Time Clock Interrupt Vector Register

**Figure 22-36. RTCIV Register**

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

**Table 22-36. RTCIV Register Description**

Bit	Field	Type	Reset	Description
15-0	RTCIV	R	0h	Real-time clock interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: RTC ready; Interrupt Flag: RTCRDYIFG 04h = Interrupt Source: RTC interval timer; Interrupt Flag: RTCTEVIFG 06h = Interrupt Source: RTC user alarm; Interrupt Flag: RTCAIFG 08h = Interrupt Source: RTC prescaler 0; Interrupt Flag: RT0PSIFG 0Ah = Interrupt Source: RTC prescaler 1; Interrupt Flag: RT1PSIFG 0Ch = Reserved 0Eh = Reserved 10h = Reserved ; Interrupt Priority: Lowest

## **Real-Time Clock B (RTC\_B)**

The real-time clock RTC\_B module provides clock counters with calendar mode, a flexible programmable alarm, and calibration. Note that the RTC\_B supports only calendar mode and not counter mode. The RTC\_B also support operation in LPMx.5 and device-dependent operation from a backup supply. See the device-specific data sheet for the supported features. This chapter describes the RTC\_B module.

Topic	Page
23.1 Real-Time Clock RTC_B Introduction .....	594
23.2 RTC_B Operation .....	596
23.3 RTC_B Registers .....	601

### 23.1 Real-Time Clock RTC\_B Introduction

The RTC\_B module provides configurable clock counters.

RTC\_B features include:

- Real-time clock and calendar mode providing seconds, minutes, hours, day of week, day of month, month, and year (including leap year correction)

Note that only the calendar mode is supported by RTC\_B; the counter mode that is available in some other RTC modules is not supported.

- Interrupt capability
- Selectable BCD or binary format
- Programmable alarms
- Calibration logic for time offset correction
- Operation in LPMx.5
- Operation from backup supply with programmable charger for backup supply (device-dependent) (see the [Battery Backup chapter](#)).

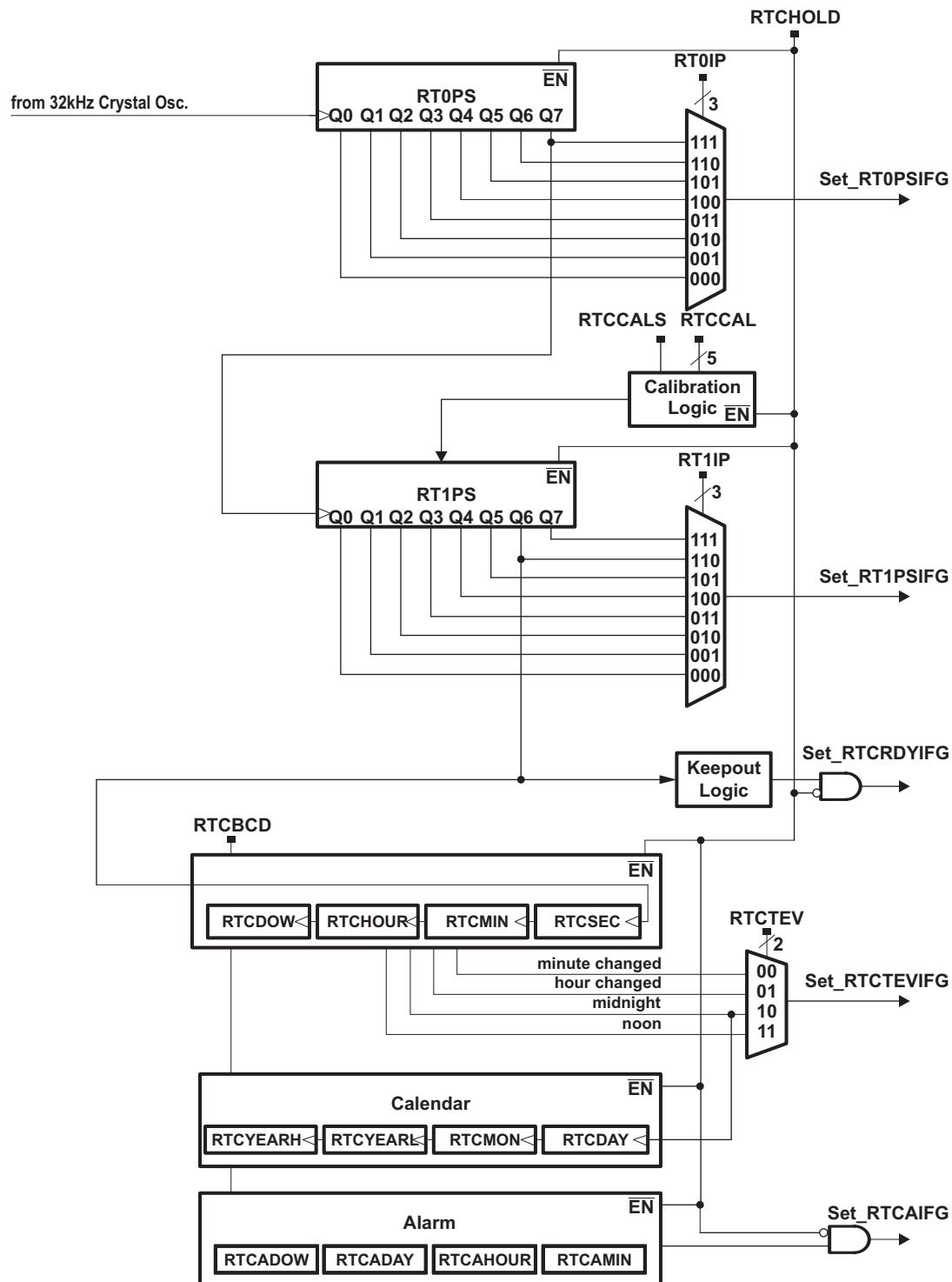
The RTC\_B block diagram for devices supporting LPMx.5 is shown in [Figure 23-1](#).

---

**NOTE: Real-time clock initialization**

Most RTC\_B module registers have no initial condition. These registers must be configured by user software before use.

---


**Figure 23-1. RTC\_B Block Diagram**

## 23.2 RTC\_B Operation

The RTC\_B module provides seconds, minutes, hours, day of week, day of month, month, and year in selectable BCD or hexadecimal format. The calendar includes a leap-year algorithm that considers all years evenly divisible by four as leap years. This algorithm is accurate from the year 1901 through 2099.

### 23.2.1 Real-Time Clock and Prescale Dividers

The prescale dividers, RT0PS and RT1PS, are automatically configured to provide a 1-s clock interval for the RTC\_B. The low-frequency oscillator must be operated at 32768 Hz (nominal) for proper RTC\_B operation. RT0PS is sourced from the low-frequency oscillator XT1. The output of RT0PS / 256 (Q7) is used to source RT1PS. RT1PS is further divider and the /128 output sources the real-time clock counter registers providing the required 1-second time interval.

When RTCBCD = 1, BCD format is selected for the calendar registers. It is possible to switch between BCD and hexadecimal format while the RTC is counting.

Setting RTCHOLD halts the real-time counters and prescale counters, RT0PS, and RT1PS.

### 23.2.2 Real-Time Clock Alarm Function

The RTC\_B module provides for a flexible alarm system. There is a single user-programmable alarm that can be programmed based on the settings contained in the alarm registers for minutes, hours, day of week, and day of month.

Each alarm register contains an alarm enable (AE) bit that can be used to enable the respective alarm register. By setting AE bits of the various alarm registers, a variety of alarm events can be generated.

- Example 1: A user wishes to set an alarm every hour at 15 minutes past the hour (that is, at 00:15:00, 01:15:00, 02:15:00, etc). This is possible by setting RTCAMIN to 15. By setting the AE bit of the RTCAMIN and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the RTCAIFG is set when the count transitions from 00:14:59 to 00:15:00, 01:14:59 to 01:15:00, 02:14:59 to 02:15:00, and so on.
- Example 2: A user wishes to set an alarm every day at 04:00:00. This is possible by setting RTCAHOUR to 4. By setting the AE bit of the RTCHOUR and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the RTCAIFG is set when the count transitions from 03:59:59 to 04:00:00.
- Example 3: A user wishes to set an alarm for 06:30:00. RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00. In this case, the alarm event occurs every day at 06:30:00.
- Example 4: A user wishes to set an alarm every Tuesday at 06:30:00. RTCADOW would be set to 2, RTCAHOUR would be set to 6, and RTCAMIN would be set to 30. By setting the AE bits of RTCADOW, RTCAHOUR, and RTCAMIN, the alarm is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCDOW transitions from 1 to 2.
- Example 5: A user wishes to set an alarm the fifth day of each month at 06:30:00. RTCADAY would be set to 5, RTCAHOUR would be set to 6, and RTCAMIN would be set to 30. By setting the AE bits of RTCADAY, RTCAHOUR, and RTCAMIN, the alarm is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCDAY equals 5.

---

#### NOTE: Setting the alarm

Prior to setting an initial alarm, all alarm registers including the AE bits should be cleared.

To prevent potential erroneous alarm conditions from occurring, the alarms should be disabled by clearing the RTCAIE, RTCAIFG, and AE bits prior to writing initial or new time values to the RTC time registers.

---

**NOTE: Invalid alarm settings**

Invalid alarm settings are not checked via hardware. It is the user's responsibility that valid alarm settings are entered.

**NOTE: Invalid time and date values**

Writing of invalid date or time information or data values outside the legal ranges specified in the RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCDOW, RTCYEAR, RTCAMIN, RTCAHOUR, RTCADAY, and RTCADOW registers can result in unpredictable behavior.

### 23.2.3 Reading or Writing Real-Time Clock Registers

Because the system clock may in fact be asynchronous to the RTC\_B clock source, special care must be used when accessing the real-time clock registers.

The real-time clock registers are updated once per second. To prevent reading any real-time clock register at the time of an update, which could result in an invalid time being read, a keep-out window is provided. The keep-out window is centered approximately 128/32768 seconds around the update transition. The read-only RTCRDY bit is reset during the keep-out window period and set outside the keep-out the window period. Any read of the clock registers while RTCRDY is reset is considered to be potentially invalid, and the time read should be ignored.

An easy way to safely read the real-time clock registers is to utilize the RTCRDYIFG interrupt flag. Setting RTCRDYIE enables the RTCRDYIFG interrupt. Once enabled, an interrupt is generated based on the rising edge of the RTCRDY bit, causing the RTCRDYIFG to be set. At this point, the application has nearly a complete second to safely read any or all of the real-time clock registers. This synchronization process prevents reading the time value during transition. The RTCRDYIFG flag is reset automatically when the interrupt is serviced, or it can be reset with software.

**NOTE: Reading or writing real-time clock registers**

When the counter clock is asynchronous to the CPU clock, any read from any RTCSEC, RTCMIN, RTCHOUR, RTCDOW, RTCDAY, RTCMON, or RTCYEAR register while the RTCRDY is reset may result in invalid data being read. To safely read the counting registers, either polling of the RTCRDY bit or the synchronization procedure previously described can be used. Alternatively, the counter register can be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Reading the RT0PS and RT1PS can only be handled by reading the registers multiple times and a majority vote taken in software to determine the correct reading or by halting the counters.

Any write to any counting register takes effect immediately. However, the clock is stopped during the write. In addition, RT0PS and RT1PS registers are reset. This could result in losing up to 1 second during a write. Writing of data outside the legal ranges or invalid time stamp combinations results in unpredictable behavior.

### 23.2.4 Real-Time Clock Interrupts

Six sources for interrupts are available, namely RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, RTCAIFG, and RTCOFIFG. These flags are prioritized and combined to source a single interrupt vector. The interrupt vector register (RTCIV) is used to determine which flag requested an interrupt.

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

Any access, read or write, of the RTCIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. In addition, all flags can be cleared via software.

The user-programmable alarm event sources the real-time clock interrupt, RTCAIFG. Setting RTCAIE enables the interrupt. In addition to the user-programmable alarm, the RTC\_B module provides for an interval alarm that sources real-time clock interrupt, RTCTEVIFG. The interval alarm can be selected to cause an alarm event when RTCMIN changed or RTCHOUR changed, every day at midnight (00:00:00) or every day at noon (12:00:00). The event is selectable with the RTCTEV bits. Setting the RTCTEVIE bit enables the interrupt.

The RTCRDY bit sources the real-time clock interrupt, RTCRDYIFG, and is useful in synchronizing the read of time registers with the system clock. Setting the RTCRDYIE bit enables the interrupt.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. RT0PS is sourced with low-frequency oscillator clock at 32768 Hz, so intervals of 16384 Hz, 8192 Hz, 4096 Hz, 2048 Hz, 1024 Hz, 512 Hz, 256 Hz, or 128 Hz are possible. Setting the RT0PSIE bit enables the interrupt.

RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. RT1PS is sourced with the output of RT0PS, which is 128 Hz (32768/256 Hz). Therefore, intervals of 64 Hz, 32 Hz, 16 Hz, 8 Hz, 4 Hz, 2 Hz, 1 Hz, or 0.5 Hz are possible. Setting the RT1PSIE bit enables the interrupt.

---

**NOTE: Changing RT0IP or RT1IP**

Changing the settings of the interrupt interval bits RT0IP or RT1IP while the corresponding pre-scaler is running or is stopped in a non-zero state can result in setting the corresponding interrupt flags.

---

The RTCOFIFG bit flags a failure of the 32-kHz crystal oscillator. Its main purpose is to wake up the CPU from LPM3.5 if an oscillator failure occurs. On devices with a backup-supply subsystem, it also stores a failure event that occurred while the RTC was operating on the backup supply.

#### 23.2.4.1 RTCIV Software Example

The following software example shows the recommended use of RTCIV and the handling overhead. The RTCIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

; Interrupt handler for RTC interrupt flags.

```

RTC_HND          ; Interrupt latency      6
    ADD &RTCIV,PC   ; Add offset to Jump table 3
    RETI           ; Vector 0: No interrupt  5
    JMP RTCRDYIFG_HND ; Vector 2: RTCRDYIFG  2
    JMP RTCTEVIFG_HND ; Vector 4: RTCTEVIFG  2
    JMP RTCAIFG_HND  ; Vector 6: RTCAIFG   5
    JMP RT0PSIFG_HND ; Vector 8: RT0PSIFG   5
    JMP RT1PSIFG_HND ; Vector A: RT1PSIFG   5
    JMP RTCOFIFG_HND ; Vector C: RTCOFIFG   5
    RETI           ; Vector E: Reserved   5

RTCRDYIFG_HND   ; Vector 2: RTCRDYIFG Flag
...
RETI           ; Task starts here
                ; Back to main program  5

RTCTEVIFG_HND   ; Vector 4: RTCTEVIFG Flag
...
RETI           ; Task starts here
                ; Back to main program  5

RTCAIFG_HND     ; Vector 6: RTCAIFG Flag
...
RETI           ; Task starts here
                ; Back to main program  5

```

RT0PSIFG_HND	; Vector 8: RT0PSIFG Flag	
...	; Task starts here	
RETI	; Back to main program	5
RT1PSIFG_HND	; Vector A: RT1PSIFG Flag	
...	; Task starts here	
RETI	; Back to main program	5
RTCOFIFG_HND	; Vector C: RTCOFIFG Flag	
...	; Task starts here	
RETI	; Back to main program	5

### **23.2.5 Real-Time Clock Calibration**

The RTC\_B module has calibration logic that allows for adjusting the crystal frequency in approximately +4-ppm or -2-ppm steps, allowing for higher time keeping accuracy from standard crystals. The RTCCALx bits are used to adjust the frequency. When RTCCALS is set, each RTCCALx LSB causes a  $\approx$  +4-ppm adjustment. When RTCCALS is cleared, each RTCCALx LSB causes a  $\approx$  -2-ppm adjustment.

Calibration is accomplished by periodically adjusting the RT1PS counter based on the RTCCALS and RTCCALx settings. The RT0PS divides the nominal 37268-Hz low-frequency (LF) crystal clock input by 256. A 60-minute period has  $32768 \text{ cycles/sec} \times 60 \text{ sec/min} \times 60 \text{ min} = 117964800$  cycles. Therefore, a -2-ppm reduction in frequency (down calibration) approximately equates to adding an additional 256 cycles every 117964800 cycles ( $256/117964800 = 2.17 \text{ ppm}$ ). This is accomplished by holding the RT1PS counter for one additional clock of the RT0PS output within a 60-minute period. Similarly, a +4-ppm increase in frequency (up calibration) approximately equates to removing 512 cycles every 117964800 cycle ( $512/117964800 = 4.34 \text{ ppm}$ ). This is accomplished by incrementing the RT1PS counter for two additional clocks of the RT0PS output within a 60-minute period. Each RTCCALx calibration bit causes either 256 LF crystal clock cycles to be added every 60 minutes or 512 LF crystal clock cycles to be subtracted every 60 minutes, giving a frequency adjustment of approximately -2 ppm or +4 ppm, respectively.

To calibrate the frequency, the RTCCLK output signal is available at a pin. RTCCALF bits can be used to select the frequency rate of the output signal, either no signal, 512 Hz, 256 Hz, or 1 Hz.

The basic flow to calibrate the frequency is as follows:

1. Configure the RTCCLK pin.
2. Measure the RTCCLK output signal with an appropriate resolution frequency counter ; that is, within the resolution required.
3. Compute the absolute error in ppm: Absolute error (ppm) =  $|10^6 (f_{\text{MEASURED}} - f_{\text{RTCCLK}})/f_{\text{RTCCLK}}|$ , where  $f_{\text{RTCCLK}}$  is the expected frequency of 512 Hz, 256 Hz, or 1 Hz.
4. Adjust the frequency by performing the following:
  - (a) If the frequency is too low, set RTCCALS = 1 and apply the appropriate RTCCALx bits, where  $\text{RTCCALx} = (\text{Absolute Error}) / 4.34$  rounded to the nearest integer
  - (b) If the frequency is too high, clear RTCCALS = 0 and apply the appropriate RTCCALx bits, where  $\text{RTCCALx} = (\text{Absolute Error}) / 2.17$  rounded to the nearest integer

For example, assume that RTCCLK is configured to output at a frequency of 512 Hz. The measured RTCCLK is 511.9658 Hz. This frequency error is approximately 66.8 ppm too low. To increase the frequency by 66.8 ppm, RTCCALS would be set, and RTCCALx would be set to 15 ( $66.8 / 4.34$ ). Similarly, assume that the measured RTCCLK is 512.0125 Hz. The frequency error is approximately 24.4 ppm too high. To decrease the frequency by 24.4 ppm, RTCCALS would be cleared, and RTCCAL would be set to 11 ( $24.4 / 2.17$ ).

The calibration corrects only initial offsets and does not adjust for temperature and aging effects. These effects can be handled by periodically measuring temperature and using the crystal's characteristic curve to adjust the ppm based on temperature, as required.

**NOTE: Minimum Possible Calibration**

The minimal calibration possible is -4 ppm or +8 ppm. For example, setting RTCCALS = 0 and RTCCAL = 0h would result in a -4 ppm decrease in frequency. Similarly, setting RTCCALS = 1 and RTCCAL = 0h would result in a +8 ppm increase in frequency.

**NOTE: Calibration output frequency**

The 512-Hz and 256-Hz output frequencies observed at the RTCCLK pin are not affected by changes in the calibration settings, because these output frequencies are generated prior to the calibration logic. The 1-Hz output frequency is affected by changes in the calibration settings. Because the frequency change is small and infrequent over a very long time interval, it can be difficult to observe.

**23.2.6 Real-Time Clock Operation in LPMx.5 Low-Power Mode**

The regulator of the Power Management Module (PMM) is disabled upon entering LPMx.5, which causes most of the RTC\_B configuration registers to be lost; only the counters and the backup RAM are retained. Table 23-1 lists the retained registers in LPMx.5. Also the configuration of the interrupts is stored so that the configured interrupts can cause a wakeup upon exit from LPMx.5. Interrupt flags that are set prior to entering LPM3.5 are cleared upon entering LPM3.5 (Note: this can only happen if the corresponding interrupt is not enabled). The interrupt flags RTCTEVIFG, RTCAIFG, RT1PSIFG, and RTCOFIFG can be used as RTC\_B wake-up interrupt sources. After restoring the configuration registers (and clearing LOCKLPM5) the interrupts can be serviced as usual. The detailed flow is as follows:

1. Set all I/Os to general purpose I/Os and configure as needed. Optionally configure input interrupt pins for wake-up. Configure RTC\_B interrupts for wake-up (set RTCTEVIE, RTCAIE, RT1PSIE, or RTCOFIE. If the alarm interrupt is also used as wake-up event, the alarm registers must be configured as needed).
2. Enter LPMx.5 with LPMx.5 entry sequence.

```
MOV #PMMKEY + PMMREGOFF, &PMMCTL0 ; Open PMM registers for write and set PMMREGOFF
;
BIS #LPM4,SR                      ; Enter LPMx.5 when PMMREGOFF is set
```

3. LOCKLPM5 is automatically set by hardware upon entering LPMx.5, the core voltage regulator is disabled, and all clocks are disabled except for the 32-kHz crystal oscillator clock if the RTC is enabled with RTCHOLD = 0.
4. An LPMx.5 wake-up event, such as an edge on a wake-up input pin, are an RTC\_B interrupt event and start the BOR entry sequence together with the core voltage regulator. All peripheral registers are set to their default conditions. The I/O pin state remains locked as well as the interrupt configuration for the RTC\_B.
5. The device can be configured. The I/O configuration and the RTC\_B interrupt configuration that was not retained during LPMx.5 should be restored to the values prior to entering LPMx.5. Then the LOCKLPM5 bit can be cleared, this releases the I/O pin conditions as well as the RTC\_B interrupt configuration.
6. After enabling I/O and RTC\_B interrupts, the interrupt that caused the wake-up can be serviced.
7. To re-enter LPMx.5, the LOCKLPM5 bit must be cleared prior to re-entry, otherwise LPMx.5 is not entered.

If the RTC is enabled (RTCHOLD = 0), the 32-kHz oscillator remains active during LPMx.5. The fault detection also remains functional. If a fault occurs during LPMx.5 and the RTCOFIE was set before entering LPMx.5, a wake-up event is issued.

### 23.3 RTC\_B Registers

The RTC\_B module registers are listed in [Table 23-1](#). This table also lists the retention during LPMx.5. Registers that are not retained during LPMx.5 must be restored after exit from LPMx.5. The base address for the RTC\_B module registers can be found in the device-specific data sheet. The address offsets are given in [Table 23-1](#).

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

**Table 23-1. RTC\_B Registers**

Offset	Acronym	Register Name	Type	Access	Reset	LPMx.5 Operation
00h	RTCCTL01	Real-Time Clock Control 0, 1	Read/write	Word	4000h	not retained
00h	RTCCTL0	Real-Time Clock Control 0 or RTCCTL01_L	Read/write	Byte	00h	not retained
01h	RTCCTL1	Real-Time Clock Control 1 or RTCCTL01_H	Read/write	Byte	40h	not retained
02h	RTCCTL23	Real-Time Clock Control 2, 3	Read/write	Word	0000h	retained
02h	RTCCTL2	Real-Time Clock Control 2 or RTCCTL23_L	Read/write	Byte	00h	retained
03h	RTCCTL3	Real-Time Clock Control 3 or RTCCTL23_H	Read/write	Byte	00h	retained
08h	RTCP0CTL	Real-Time Prescale Timer 0 Control	Read/write	Word	0000h	not retained
08h	RTCP0CTLL	Real-Time Prescale Timer 0 Control or RTCP0CTL_L	Read/write	Byte	00h	not retained
09h	RTCP0CTLH	Real-Time Prescale Timer 0 Control or RTCP0CTL_H	Read/write	Byte	00h	not retained
0Ah	RTCP1CTL	Real-Time Prescale Timer 1 Control	Read/write	Word	0000h	not retained
0Ah	RTCP1CTLL	Real-Time Prescale Timer 1 Control or RTCP1CTL_L	Read/write	Byte	00h	not retained
0Bh	RTCP0CTLH	Real-Time Prescale Timer 1 Control or RTCP0CTL_H	Read/write	Byte	00h	not retained
0Ch	RTCP0	Real-Time Prescale Timer 0, 1 Counter	Read/write	Word	none	retained
0Ch	RT0PS	Real-Time Prescale Timer 0 Counter or RTCP0_L	Read/write	Byte	none	retained
0Dh	RT1PS	Real-Time Prescale Timer 1 Counter or RTCP0_H	Read/write	Byte	none	retained
0Eh	RTCIV	Real Time Clock Interrupt Vector	Read	Word	0000h	not retained
10h	RTCTIM0	Real-Time Clock Seconds, Minutes	Read/write	Word	undefined	retained
10h	RTCSEC	Real-Time Clock Seconds or RTCTIM0_L	Read/write	Byte	undefined	retained
11h	RTCMIN	Real-Time Clock Minutes or RTCTIM0_H	Read/write	Byte	undefined	retained
12h	RTCTIM1	Real-Time Clock Hour, Day of Week	Read/write	Word	undefined	retained
12h	RTCHOUR	Real-Time Clock Hour or RTCTIM1_L	Read/write	Byte	undefined	retained
13h	RTCDOW	Real-Time Clock Day of Week or RTCTIM1_H	Read/write	Byte	undefined	retained

**Table 23-1. RTC\_B Registers (continued)**

Offset	Acronym	Register Name	Type	Access	Reset	LPMx.5 Operation
14h	RTCDATE	Real-Time Clock Date	Read/write	Word	undefined	retained
14h	RTCDAY or RTCDATE_L	Real-Time Clock Day of Month	Read/write	Byte	undefined	retained
15h	RTCMON or RTCDATE_H	Real-Time Clock Month	Read/write	Byte	undefined	retained
16h	RTCYEAR	Real-Time Clock Year <sup>(1)</sup>	Read/write	Word	undefined	retained
18h	RTCAMINHR	Real-Time Clock Minutes, Hour Alarm	Read/write	Word	undefined	retained
18h	RTCAMIN or RTCAMINHR_L	Real-Time Clock Minutes Alarm	Read/write	Byte	undefined	retained
19h	RTCAHOUR or RTCAMINHR_H	Real-Time Clock Hours Alarm	Read/write	Byte	undefined	retained
1Ah	RTCADOWDAY	Real-Time Clock Day of Week, Day of Month Alarm	Read/write	Word	undefined	retained
1Ah	RTCADOW or RTCADOWDAY_L	Real-Time Clock Day of Week Alarm	Read/write	Byte	undefined	retained
1Bh	RTCADAY or RTCADOWDAY_H	Real-Time Clock Day of Month Alarm	Read/write	Byte	undefined	retained
1Ch	BIN2BCD	Binary-to-BCD Conversion Register	Read/write	Word	00h	not retained
1Eh	BCD2BIN	BCD-to-Binary Conversion Register	Read/write	Word	00h	not retained

<sup>(1)</sup> Do not access the RTCYEAR register in byte mode.

### 23.3.1 RTCCTL0 Register

Real-Time Clock Control 0 Register

**Figure 23-2. RTCCTL0 Register**

7	6	5	4	3	2	1	0
RTCOFIE <sup>(1)</sup>	RTCTEVIE <sup>(1)</sup>	RTCAIE <sup>(1)</sup>	RTCRDYIE	RTCOFIG	RTCTEVIFG	RTCAIFG	RTCRDYIFG
rw-0	rw-0	rw-0	rw-0	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

**Table 23-2. RTCCTL0 Register Description**

Bit	Field	Type	Reset	Description
7	RTCOFIE	RW	0h	32-kHz crystal oscillator fault interrupt enable. This interrupt can be used as LPMx.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
6	RTCTEVIE	RW	0h	Real-time clock time event interrupt enable. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
5	RTCAIE	RW	0h	Real-time clock alarm interrupt enable. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
4	RTCRDYIE	RW	0h	Real-time clock ready interrupt enable. 0b = Interrupt not enabled 1b = Interrupt enabled
3	RTCOFIG	RW	0h	32-kHz crystal oscillator fault interrupt flag. This interrupt can be used as LPMx.5 wake-up event. 0b = No interrupt pending 1b = Interrupt pending. A 32-kHz crystal oscillator fault occurred after last reset.
2	RTCTEVIFG	RW	0h	Real-time clock time event interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = No time event occurred 1b = Time event occurred
1	RTCAIFG	RW	0h	Real-time clock alarm interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = No time event occurred 1b = Time event occurred
0	RTCRDYIFG	RW	0h	Real-time clock ready interrupt flag 0b = RTC cannot be read safely 1b = RTC can be read safely

**23.3.2 RTCCTL1 Register**

Real-Time Clock Control Register 1

**Figure 23-3. RTCCTL1 Register**

7	6	5	4	3	2	1	0
RTCBBCD	RTCHOLD <sup>(1)</sup>	Reserved	RTCRDY	Reserved	Reserved	RTCTEVx <sup>(1)</sup>	
rw-(0)	rw-(1)	r1	r-(1)	r0	r0	rw-(0)	rw-(0)

<sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

**Table 23-3. RTCCTL1 Register Description**

Bit	Field	Type	Reset	Description
7	RTCBBCD	RW	0h	Real-time clock BCD select. Selects BCD counting for real-time clock. 0b = Binary-hexadecimal code selected 1b = BCD Binary coded decimal (BCD) code selected
6	RTCHOLD	RW	1h	Real-time clock hold 0b = Real-time clock is operational. 1b = The calendar is stopped as well as the prescale counters, RT0PS, and RT1PS.
5	Reserved	R	1h	Reserved. Always read as 1.
4	RTCRDY	RW	1h	Real-time clock ready 0b = RTC time values in transition 1b = RTC time values safe for reading. This bit indicates when the real-time clock time values are safe for reading.
3-2	Reserved	R	0h	Reserved. Always read as 0.
1-0	RTCTEVx	RW	0h	Real-time clock time interrupt event 00b = Minute changed 01b = Hour changed 10b = Every day at midnight (00:00) 11b = Every day at noon (12:00)

### 23.3.3 RTCCTL2 Register

Real-Time Clock Control 2 Register

**Figure 23-4. RTCCTL2 Register**

7	6	5	4	3	2	1	0
RTCCALS	Reserved			RTCCALX			
rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 23-4. RTCCTL2 Register Description**

Bit	Field	Type	Reset	Description
7	RTCCALS	RW	0h	Real-time clock calibration sign 0b = Frequency adjusted down 1b = Frequency adjusted up
6	Reserved	R	0h	Reserved. Always read as 0.
5-0	RTCCALX	RW	0h	Real-time clock calibration. Each LSB represents approximately +4-ppm (RTCCALS = 1) or a -2-ppm (RTCCALS = 0) adjustment in frequency.

### 23.3.4 RTCCTL3 Register

Real-Time Clock Control 3 Register

**Figure 23-5. RTCCTL3 Register**

7	6	5	4	3	2	1	0
		Reserved			RTCCALFx		
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)

**Table 23-5. RTCCTL3 Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always read as 0.
1-0	RTCCALFx	RW	0h	Real-time clock calibration frequency. Selects frequency output to RTCCLK pin for calibration measurement. The corresponding port must be configured for the peripheral module function. 00b = No frequency output to RTCCLK pin 01b = 512 Hz 10b = 256 Hz 11b = 1 Hz

**23.3.5 RTCSEC Register – Hexadecimal Format**

Real-Time Clock Seconds Register – Hexadecimal Format

**Figure 23-6. RTCSEC Register**

7	6	5	4	3	2	1	0
0	0						Seconds
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 23-6. RTCSEC Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-0	Seconds	RW	undefined	Seconds. Valid values are 0 to 59.

**23.3.6 RTCSEC Register – BCD Format**

Real-Time Clock Seconds Register – BCD Format

**Figure 23-7. RTCSEC Register**

7	6	5	4	3	2	1	0
0		Seconds – high digit			Seconds – low digit		
r-0	rw	rw	rw	rw	rw	rw	rw

**Table 23-7. RTCSEC Register Description**

Bit	Field	Type	Reset	Description
7	0	R	0h	Always reads as 0.
6-4	Seconds – high digit	RW	undefined	Seconds – high digit. Valid values are 0 to 5.
3-0	Seconds – low digit	RW	undefined	Seconds – low digit. Valid values are 0 to 9.

### **23.3.7 RTCMIN Register – Hexadecimal Format**

Real-Time Clock Minutes Register – Hexadecimal Format

**Figure 23-8. RTCMIN Register**

7	6	5	4	3	2	1	0
0	0						Minutes
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 23-8. RTCMIN Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-0	Minutes	RW	undefined	Minutes. Valid values are 0 to 59.

### **23.3.8 RTCMIN Register – BCD Format**

Real-Time Clock Minutes Register – BCD Format

**Figure 23-9. RTCMIN Register**

7	6	5	4	3	2	1	0
0		Minutes – high digit			Minutes – low digit		
r-0	rw	rw	rw	rw	rw	rw	rw

**Table 23-9. RTCMIN Register Description**

Bit	Field	Type	Reset	Description
7	0	R	0h	Always reads as 0.
6-4	Minutes – high digit	RW	undefined	Minutes – high digit. Valid values are 0 to 5.
3-0	Minutes – low digit	RW	undefined	Minutes – low digit. Valid values are 0 to 9.

**23.3.9 RTCHOUR Register – Hexadecimal Format**

Real-Time Clock Hours Register – Hexadecimal Format

**Figure 23-10. RTCHOUR Register**

7	6	5	4	3	2	1	0
Hours							
r-0	r-0	r-0	rw	rw	rw	rw	rw

**Table 23-10. RTCHOUR Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always reads as 0.
4-0	Hours	RW	undefined	Hours. Valid values are 0 to 23.

**23.3.10 RTCHOUR Register – BCD Format**

Real-Time Clock Hours Register – BCD Format

**Figure 23-11. RTCHOUR Register**

7	6	5	4	3	2	1	0
Hours – high digit				Hours – low digit			
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 23-11. RTCHOUR Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-4	Hours – high digit	RW	undefined	Hours – high digit. Valid values are 0 to 2.
3-0	Hours – low digit	RW	undefined	Hours – low digit. Valid values are 0 to 9.

### 23.3.11 RTCDOW Register

Real-Time Clock Day of Week Register

**Figure 23-12. RTCDOW Register**

7	6	5	4	3	2	1	0
0	0	0	0	0			Day of week
r-0	r-0	r-0	r-0	r-0	rw	rw	rw

**Table 23-12. RTCDOW Register Description**

Bit	Field	Type	Reset	Description
7-3	0	R	0h	Always reads as 0.
2-0	Day of week	RW	undefined	Day of week. Valid values are 0 to 6.

### 23.3.12 RTCDAY Register – Hexadecimal Format

Real-Time Clock Day of Month Register – Hexadecimal Format

**Figure 23-13. RTCDAY Register**

7	6	5	4	3	2	1	0
0	0	0					Day of month
r-0	r-0	r-0	rw	rw	rw	rw	rw

**Table 23-13. RTCDAY Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always reads as 0.
4-0	Day of month	RW	undefined	Day of month. Valid values are 1 to 31.

### 23.3.13 RTCDAY Register – BCD Format

Real-Time Clock Day of Month Register – BCD Format

**Figure 23-14. RTCDAY Register**

7	6	5	4	3	2	1	0
0	0		Day of month – high digit			Day of month – low digit	
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 23-14. RTCDAY Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-4	Day of month – high digit	RW	undefined	Day of month – high digit. Valid values are 0 to 3.
3-0	Day of month – low digit	RW	undefined	Day of month – low digit. Valid values are 0 to 9.

**23.3.14 RTCMON Register – Hexadecimal Format**

Real-Time Clock Month Register – Hexadecimal Format

**Figure 23-15. RTCMON Register**

7	6	5	4	3	2	1	0
0	0	0	0			Month	
r-0	r-0	r-0	r-0	rw	rw	rw	rw

**Table 23-15. RTCMON Register Description**

Bit	Field	Type	Reset	Description
7-4	0	R	0h	Always reads as 0.
3-0	Month	RW	undefined	Month. Valid values are 1 to 12.

**23.3.15 RTCMON Register – BCD Format**

Real-Time Clock Month Register

**Figure 23-16. RTCMON Register**

7	6	5	4	3	2	1	0
0	0	0	Month – high digit		Month – low digit		
r-0	r-0	r-0	rw	rw	rw	rw	rw

**Table 23-16. RTCMON Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always reads as 0.
4	Month – high digit	RW	undefined	Month – high digit. Valid values are 0 or 1.
3-0	Month – low digit	RW	undefined	Month – low digit. Valid values are 0 to 9.

### 23.3.16 RTCYEAR Register – Hexadecimal Format

Real-Time Clock Year Register – Hexadecimal Format

**Figure 23-17. RTCYEAR Register**

15	14	13	12	11	10	9	8			
0	0	0	0	Year – high byte						
r-0	r-0	r-0	r-0	rw	rw	rw	rw			
7	6	5	4	3	2	1	0			
Year – low byte										
rw	rw	rw	rw	rw	rw	rw	rw			

**Table 23-17. RTCYEAR Register Description**

Bit	Field	Type	Reset	Description
15-12	0	R	0h	Always reads as 0.
11-8	Year – high byte	RW	undefined	Year – high byte. Valid values of Year are 0 to 4095.
7-0	Year – low byte	RW	undefined	Year – low byte. Valid values of Year are 0 to 4095.

### 23.3.17 RTCYEAR Register – BCD Format

Real-Time Clock Year Register – BCD Format

**Figure 23-18. RTCYEAR Register**

15	14	13	12	11	10	9	8
0	Century – high digit				Century – low digit		
r-0	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Decade				Year – lowest digit			
rw	rw	rw	rw	rw	rw	rw	rw

**Table 23-18. RTCYEAR Register Description**

Bit	Field	Type	Reset	Description
15	0	R	0h	Always reads as 0.
14-12	Century – high digit	RW	undefined	Century – high digit . Valid values are 0 to 4.
11-8	Century – low digit	RW	undefined	Century – low digit. Valid values are 0 to 9.
7-4	Decade	RW	undefined	Decade. Valid values are 0 to 9.
3-0	Year – lowest digit	RW	undefined	Year – lowest digit. Valid values are 0 to 9.

**23.3.18 RTCAMIN Register – Hexadecimal Format**

Real-Time Clock Minutes Alarm Register – Hexadecimal Format

**Figure 23-19. RTCAMIN Register**

7	6	5	4	3	2	1	0
AE	0						Minutes
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 23-19. RTCAMIN Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6	0	R	0h	Always reads as 0.
5-0	Minutes	RW	undefined	Minutes. Valid values are 0 to 59.

**23.3.19 RTCAMIN Register – BCD Format**

Real-Time Clock Minutes Alarm Register – BCD Format

**Figure 23-20. RTCAMIN Register**

7	6	5	4	3	2	1	0
AE		Minutes – high digit			Minutes – low digit		
rw	rw	rw	rw	rw	rw	rw	rw

**Table 23-20. RTCAMIN Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-4	Minutes – high digit	RW	undefined	Minutes – high digit. Valid values are 0 to 5.
3-0	Minutes – low digit	RW	undefined	Minutes – low digit. Valid values are 0 to 9.

### 23.3.20 RTCAHOUR Register – Hexadecimal Format

Real-Time Clock Hours Alarm Register – Hexadecimal Format

**Figure 23-21. RTCAHOUR Register**

7	6	5	4	3	2	1	0
AE	0	0				Hours	
rw	r-0	r-0	rw	rw	rw	rw	rw

**Table 23-21. RTCAHOUR Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-5	0	R	0h	Always reads as 0.
4-0	Hours	RW	undefined	Hours. Valid values are 0 to 23.

### 23.3.21 RTCAHOUR Register – BCD Format

Real-Time Clock Hours Alarm Register – BCD Format

**Figure 23-22. RTCAHOUR Register**

7	6	5	4	3	2	1	0
AE	0		Hours – high digit			Hours – low digit	
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 23-22. RTCAHOUR Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6	0	R	0h	Always reads as 0.
5-4	Hours – high digit	RW	undefined	Hours – high digit. Valid values are 0 to 2.
3-0	Hours – low digit	RW	undefined	Hours – low digit. Valid values are 0 to 9.

**23.3.22 RTCADOW Register**

Real-Time Clock Day of Week Alarm Register

**Figure 23-23. RTCADOW Register**

7	6	5	4	3	2	1	0
AE	0	0	0	0		Day of week	
rw	r-0	r-0	r-0	r-0	rw	rw	rw

**Table 23-23. RTCADOW Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-3	0	R	0h	Always reads as 0.
2-0	Day of week	RW	undefined	Day of week. Valid values are 0 to 6.

### 23.3.23 RTCADAY Register – Hexadecimal Format

Real-Time Clock Day of Month Alarm Register – Hexadecimal Format

**Figure 23-24. RTCADAY Register**

7	6	5	4	3	2	1	0
AE	0	0				Day of month	
rw	r-0	r-0	rw	rw	rw	rw	rw

**Table 23-24. RTCADAY Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-5	0	R	0h	Always reads as 0.
4-0	Day of month	RW	undefined	Day of month. Valid values are 1 to 31.

### 23.3.24 RTCADAY Register – BCD Format

Real-Time Clock Day of Month Alarm Register – BCD Format

**Figure 23-25. RTCADAY Register**

7	6	5	4	3	2	1	0
AE	0	Day of month – high digit			Day of month – low digit		
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 23-25. RTCADAY Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6	0	R	0h	Always reads as 0.
5-4	Day of month – high digit	RW	undefined	Day of month – high digit. Valid values are 0 to 3.
3-0	Day of month – low digit	RW	undefined	Day of month – low digit. Valid values are 0 to 9.

**23.3.25 RTCPS0CTL Register**

Real-Time Clock Prescale Timer 0 Control Register

**Figure 23-26. RTCPS0CTL Register**

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved			RT0IPx <sup>(1)</sup>			RT0PSIE	RT0PSIFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

- <sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

**Table 23-26. RTCPS0CTL Register Description**

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT0IPx	RW	0h	Prescale timer 0 interrupt interval 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
1	RT0PSIE	RW	0h	Prescale timer 0 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
0	RT0PSIFG	RW	0h	Prescale timer 0 interrupt flag 0b = No time event occurred 1b = Time event occurred

### 23.3.26 RTCPS1CTL Register

Real-Time Clock Prescale Timer 1 Control Register

**Figure 23-27. RTCPS1CTL Register**

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved			RT1IPx <sup>(1)</sup>			RT1PSIE <sup>(1)</sup>	RT1PSIFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

- <sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

**Table 23-27. RTCPS1CTL Register Description**

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT1IPx	RW	0h	Prescale timer 1 interrupt interval 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
1	RT1PSIE	RW	0h	Prescale timer 1 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
0	RT1PSIFG	RW	0h	Prescale timer 1 interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = No time event occurred 1b = Time event occurred

**23.3.27 RTCPS0 Register**

Real-Time Clock Prescale Timer 0 Counter Register

**Figure 23-28. RTCPS0 Register**

7	6	5	4	3	2	1	0
RT0PS							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 23-28. RTCPS0 Register Description**

Bit	Field	Type	Reset	Description
7-0	RT0PS	RW	undefined	Prescale timer 0 counter value

**23.3.28 RTCPS1 Register**

Real-Time Clock Prescale Timer 1 Counter Register

**Figure 23-29. RTCPS1 Register**

7	6	5	4	3	2	1	0
RT1PS							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 23-29. RTCPS1 Register Description**

Bit	Field	Type	Reset	Description
7-0	RT1PS	RW	undefined	Prescale timer 1 counter value

### 23.3.29 RTCIV Register

Real-Time Clock Interrupt Vector Register

**Figure 23-30. RTCIV Register**

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

**Table 23-30. RTCIV Register Description**

Bit	Field	Type	Reset	Description
15-0	RTClVx	R	0h	<p>Real-time clock interrupt vector value</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: RTC ready; Interrupt Flag: RTCRDYIFG; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: RTC interval timer; Interrupt Flag: RTCTEVIFG</p> <p>06h = Interrupt Source: RTC user alarm; Interrupt Flag: RTCAIFG</p> <p>08h = Interrupt Source: RTC prescaler 0; Interrupt Flag: RT0PSIFG</p> <p>0Ah = Interrupt Source: RTC prescaler 1; Interrupt Flag: RT1PSIFG</p> <p>0Ch = Interrupt Source: RTC oscillator failure; Interrupt Flag: RTCOFIFG</p> <p>0Eh = Reserved; Interrupt Priority: Lowest</p>

**23.3.30 BIN2BCD Register**

Binary-to-BCD Conversion Register

**Figure 23-31. BIN2BCD Register**

15	14	13	12	11	10	9	8
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 23-31. BIN2BCD Register Description**

Bit	Field	Type	Reset	Description
15-0	BIN2BCDx	RW	0h	Read: 16-bit BCD conversion of previously written 12-bit binary number Write: 12-bit binary number to be converted

**23.3.31 BCD2BIN Register**

BCD-to-Binary Conversion Register

**Figure 23-32. BCD2BIN Register**

15	14	13	12	11	10	9	8
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 23-32. BCD2BIN Register Description**

Bit	Field	Type	Reset	Description
15-0	BCD2BINx	RW	0h	Read: 12-bit binary conversion of previously written 16-bit BCD number Write: 16-bit BCD number to be converted

## **Real-Time Clock C (RTC\_C)**

The Real-Time Clock C (RTC\_C) module provides clock counters with calendar mode, a flexible programmable alarm, offset calibration, and a provision for temperature compensation. The RTC\_C also supports operation in LPM3.5. This chapter describes the RTC\_C module.

Topic	Page
<b>24.1 Real-Time Clock (RTC_C) Introduction .....</b>	<b>622</b>
<b>24.2 RTC_C Operation .....</b>	<b>624</b>
<b>24.3 RTC_C Operation - Device-Dependent Features .....</b>	<b>632</b>
<b>24.4 RTC_C Registers .....</b>	<b>635</b>

## 24.1 Real-Time Clock (RTC\_C) Introduction

The RTC\_C module provides configurable clock counters.

RTC\_C features include:

- Real-time clock and calendar mode providing seconds, minutes, hours, day of week, day of month, month, and year (including leap year correction)
- Protection for real-time clock registers
- Interrupt capability
- Selectable BCD or binary format
- Programmable alarms
- Real-time clock calibration for crystal offset error
- Real-time clock compensation for crystal temperature drift
- Operation in LPM3.5

The RTC\_C provides the following device-dependent features. Refer to the device-specific datasheet for availability information.

- General-purpose counter mode ([Section 24.3.1](#))
- Event/Tamper Detection with Time Stamp ([Section 24.3.2](#))
- Operation from a separate voltage supply.

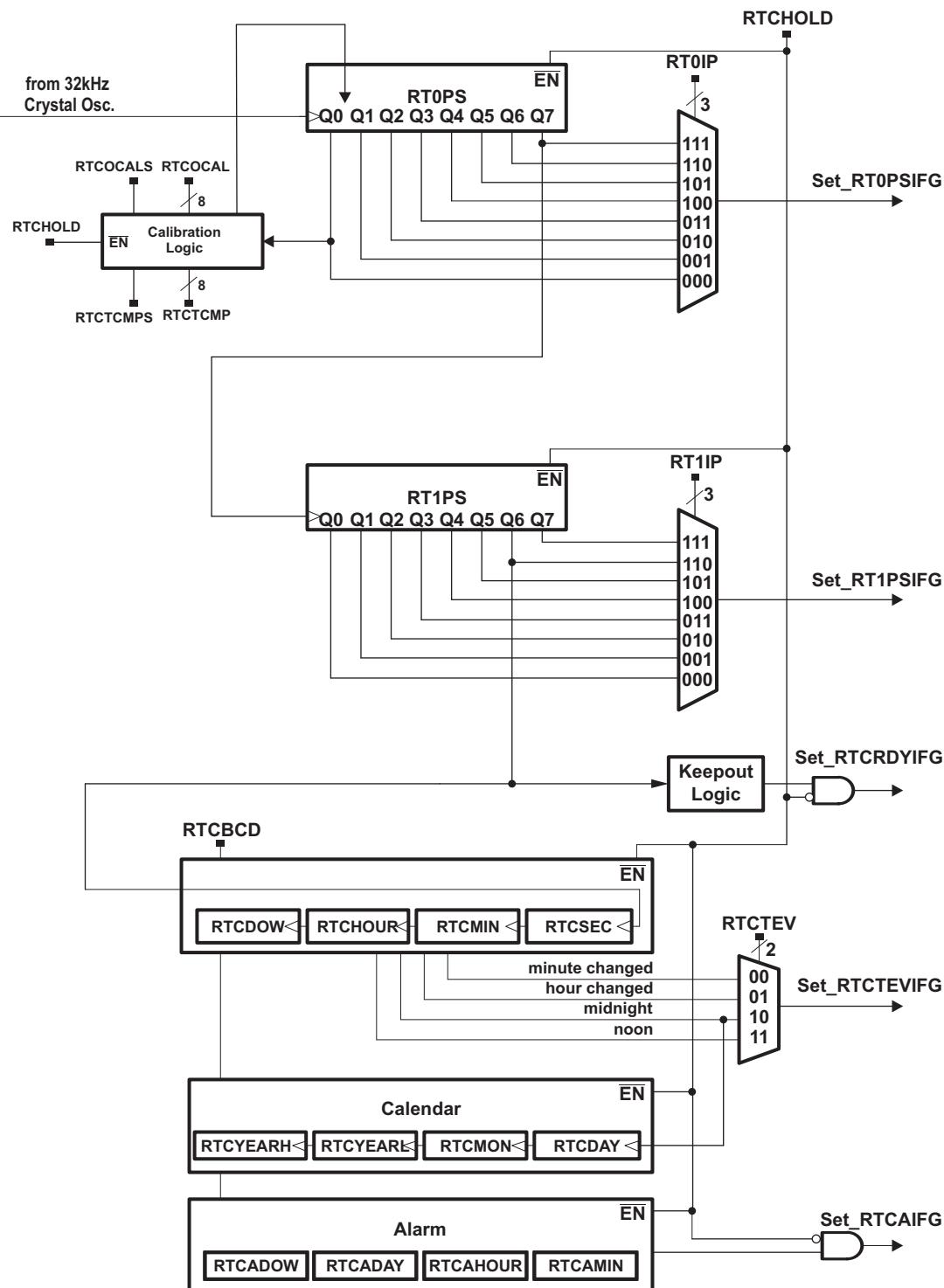
---

**NOTE: Real-time clock initialization**

Most RTC\_C module registers have no initial condition. These registers must be configured by user software before use.

---

The RTC\_C block diagram is shown in [Figure 24-1](#).


**Figure 24-1. RTC\_C Block Diagram**

## 24.2 RTC\_C Operation

### 24.2.1 Calendar Mode

Calendar mode is selected when RTCMODE is set. In calendar mode, the RTC\_C module provides seconds, minutes, hours, day of week, day of month, month, and year in selectable BCD or hexadecimal format. The calendar includes a leap-year algorithm that considers all years evenly divisible by four as leap years. This algorithm is accurate from the year 1901 through 2099. Switching from counter mode (if available) to calendar mode **does not** reset the calendar registers (RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCDOW, and RTCYEAR) nor the prescale counters (RT0PS, RT1PS). These registers must be configured by user software before use.

### 24.2.2 Real-Time Clock and Prescale Dividers

The prescale dividers, RT0PS and RT1PS, are automatically configured to provide a 1-second clock interval for the RTC\_C. The low-frequency oscillator must be operated at 32768 Hz, nominal for proper RTC\_C operation. RT0PS is sourced from the low-frequency oscillator XT1. The output of RT0PS /256 (Q7) is used to source RT1PS. RT1PS is further divider and the /128 output sources the real-time clock counter registers providing the required 1-second time interval.

When RTCBCD = 1, BCD format is selected for the calendar registers. It is possible to switch between BCD and hexadecimal format while the RTC is counting.

Setting RTCHOLD halts the real-time counters and prescale counters, RT0PS and RT1PS.

---

**NOTE: For reliable update to all Calendar Mode registers**

Keep RTCHOLD = '1' before writing into ANY of the calendar/prescalar registers (RTCPS0/1, RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCMON, RTCYEAR).

---

### 24.2.3 Real-Time Clock Alarm Function

The RTC\_C module provides for a flexible alarm system. There is a single user-programmable alarm that can be programmed based on the settings contained in the alarm registers for minutes, hours, day of week, and day of month.

Each alarm register contains an alarm enable (AE) bit that can be used to enable the respective alarm register. By setting AE bits of the various alarm registers, a variety of alarm events can be generated.

- Example 1: A user wishes to set an alarm every hour at 15 minutes past the hour, i.e., 00:15:00, 01:15:00, 02:15:00, etc. This is possible by setting RTCAMIN to 15. By setting the AE bit of the RTCAMIN and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the RTCAIFG is set when the count transitions from 00:14:59 to 00:15:00, 01:14:59 to 01:15:00, 02:14:59 to 02:15:00, etc.
- Example 2: A user wishes to set an alarm every day at 04:00:00. This is possible by setting RTCAHOUR to 4. By setting the AE bit of the RTCHOUR and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the RTCAIFG is set when the count transitions from 03:59:59 to 04:00:00.
- Example 3: A user wishes to set an alarm for 06:30:00. RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00. In this case, the alarm event occurs every day at 06:30:00.
- Example 4: A user wishes to set an alarm every Tuesday at 06:30:00. RTCADOW would be set to 2, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADOW, RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCADOW transitions from 1 to 2.
- Example 5: A user wishes to set an alarm the fifth day of each month at 06:30:00. RTCADAY would be set to 5, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADAY, RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the RTCAIFG is set

when the time count transitions from 06:29:59 to 06:30:00 and the RTCDAY equals 5.

---

**NOTE: Invalid alarm settings**

Invalid alarm settings are not checked via hardware. It is the user's responsibility that valid alarm settings are entered.

---

**NOTE: Invalid time and date values**

Writing of invalid date and/or time information or data values outside the legal ranges specified in the RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCDOW, RTCYEARH, RTCYEARL, RTCAMIN, RTCAHOUR, RTCADAY, and RTCADOW registers can result in unpredictable behavior.

---

**NOTE: Setting the alarm**

Prior to setting an initial alarm, all alarm registers including the AE bits should be cleared.

To prevent potential erroneous alarm conditions from occurring, the alarms should be disabled by clearing the RTCAIE, RTCAIFG, and AE bits prior to writing initial or new time values to the RTC time registers.

---

#### 24.2.4 Real-Time Clock Protection

RTC\_C registers are key protected to ensure clock integrity and module configuration against software crash or from runaway code. Key protection does not apply for read from RTC\_C registers. That is, any RTC\_C register can be read at any time without having to unlock the module. But some predefined registers of RTC\_C are key protected for write access. The control registers, clock registers, calendar register, prescale timer registers, and offset error calibration registers are protected. RTC\_C alarm function registers, prescale timer control registers, interrupt vector register, and temperature compensation registers are not protected. RTC\_C registers that are not protected can be written at any time without unlocking the module. See tables in [Section 24.4](#) for details on registers covered under the protection scheme.

RTCCTL0\_H register implements key protection and controls lock or unlock state of the module. When this register is written with correct key, 0A5h, the module is unlocked and unlimited write access possible to RTC\_C registers. Once the module is unlocked, it remains unlocked until user writes any incorrect key or until the module is reset. A read from RTCCTL0\_H register returns value 96h. Write access to any protected registers of RTC\_C is ignored when the module is locked.

```
RTC_C Key Protection Software Example
; Unlock/lock sequence for RTC_C
MOV.B #RTCKEY, &RTCCTL0_H                                ; Write correct key to unlock RTC_C
MOV.B #8bit_value, &RTCSEC                               ; Write 8 bit value into RTCSEC
MOV.B #8bit_value, &RTCMIN                               ; Write 8 bit value into RTCMIN
MOV.W #16bit_value, &RTCTIM1                            ; Write 16bit value into RTCTIM1
MOV.W #RTCKEY+8bit_value, &RTCCTL0                         ; Write into RTCCTL0 with correct key in
word mode
...
MOV.B #00h, &RTCCTL0_H                                    ; Write incorrect key to lock
RTC_C
```

#### 24.2.5 Reading or Writing Real-Time Clock Registers

Because the system clock may in fact be asynchronous to the RTC\_C clock source, special care must be used when accessing the real-time clock registers.

The real-time clock registers are updated once per second. To prevent reading any real-time clock register at the time of an update that could result in an invalid time being read, a keep-out window is provided. The keep-out window is centered approximately 128/32768 seconds around the update transition. The read only RTCRDY bit is reset during the keep-out window period and set outside the keep-out the window period. Any read of the clock registers while RTCRDY is reset is considered to be potentially invalid, and the time read should be ignored.

An easy way to safely read the real-time clock registers is to utilize the RTCRDYIFG interrupt flag. Setting RTCRDYIE enables the RTCRDYIFG interrupt. Once enabled, an interrupt is generated based on the rising edge of the RTCRDY bit, causing the RTCRDYIFG to be set. At this point, the application has nearly a complete second to safely read any or all of the real-time clock registers. This synchronization process prevents reading the time value during transition. The RTCRDYIFG flag is reset automatically when the interrupt is serviced or can be reset with software.

---

**NOTE: Reading or writing real-time clock registers**

When the counter clock is asynchronous to the CPU clock, any read from any RTCSEC, RTCMIN, RTCHOUR, RTCDOW, RTCDAY, RTCMON, RTCYEARL, or RTCYEARH register while the RTCRDY is reset may result in invalid data being read. To safely read the counting registers, either polling of the RTCRDY bit or the synchronization procedure previously described can be used. Alternatively, the counter register can be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Reading the RT0PS and RT1PS can only be handled by reading the registers multiple times and a majority vote taken in software to determine the correct reading or by halting the counters.

Any write to any counting register takes effect immediately. However, the clock is stopped during the write. In addition, RT0PS and RT1PS registers are reset. This could result in losing up to 1 second during a write. Writing of data outside the legal ranges or invalid time stamp combinations results in unpredictable behavior.

---

#### 24.2.6 Real-Time Clock Interrupts

At least six sources for interrupts are available, namely RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, RTCAIFG, and RTCOFIFG. These flags are prioritized and combined to source a single interrupt vector. The interrupt vector register (RTCIV) is used to determine which flag requested an interrupt.

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

Writes into RTCIV register clear all pending interrupt conditions. Reads from RTCIV register clear the highest priority pending interrupt condition. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. In addition, all flags can be cleared via software.

The user-programmable alarm event sources the real-time clock interrupt, RTCAIFG. Setting RTCAIE enables the interrupt. In addition to the user-programmable alarm, the RTC\_C module provides for an interval alarm that sources real-time clock interrupt, RTCTEVIFG. The interval alarm can be selected to cause an alarm event when RTCMIN changed or RTCHOUR changed, every day at midnight (00:00:00) or every day at noon (12:00:00). The event is selectable with the RTCTEV bits. Setting the RTCTEVIE bit enables the interrupt.

The RTCRDY bit sources the real-time clock interrupt, RTCRDYIFG, and is useful in synchronizing the read of time registers with the system clock. Setting the RTCRDYIE bit enables the interrupt.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. RT0PS is sourced with low-frequency oscillator clock at 32768 Hz, so intervals of 16384 Hz, 8192 Hz, 4096 Hz, 2048 Hz, 1024 Hz, 512 Hz, 256 Hz, or 128 Hz are possible. Setting the RT0PSIE bit enables the interrupt.

RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. RT1PS is sourced with the output of RT0PS, which is 128 Hz (32768/256 Hz). Therefore, intervals of 64 Hz, 32 Hz, 16 Hz, 8 Hz, 4 Hz, 2 Hz, 1 Hz, or 0.5 Hz are possible. Setting the RT1PSIE bit enables the interrupt.

**NOTE: Changing RT0IP or RT1IP**

Changing the settings of the interrupt interval bits RT0IP or RT1IP while the corresponding pre-scaler is running or is stopped in a non-zero state can result in setting the corresponding interrupt flags.

The RTCOFIFG bit flags the failure of the 32-kHz crystal oscillator. Its main purpose is to wake-up the CPU from LPM3.5 if an oscillator failure occurs. On devices with separate supply for RTC, this flag also stores a failure event that occurs when the core supply is not available.

**24.2.6.1 RTCIV Software Example**

The following software example shows the recommended use of RTCIV and the handling overhead. The RTCIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

; Interrupt handler for RTC interrupt flags.

RTC_HND				
ADD &RTCIV, PC		; Interrupt latency	6	
RETI		; Add offset to Jump table	3	
JMP RTCOFIFG_HND		; Vector 0: No interrupt	5	
JMP RTCRDYIFG_HND		; Vector 2: RTCOFIFG	2	
JMP RTCTEVIFG_HND		; Vector 4: RTCRDYIFG	2	
JMP RTCAIFG		; Vector 6: RTCTEVIFG	2	
JMP RT0PSIFG		; Vector 8: RTCAIFG	5	
JMP RT1PSIFG		; Vector A: RT0PSIFG	5	
RETI		; Vector C: RT1PSIFG	5	
		; Vector E: Reserved	5	
RTCOFIFG_HND				
...		; Vector 2: RTCOFIFG Flag		
RETI		; Task starts here		
		; Back to main program	5	
RTCRDYIFG_HND				
...		; Vector 4: RTCRDYIFG Flag		
RETI		; Task starts here		
		; Back to main program	5	
RTCTEVIFG_HND				
...		; Vector 6: RTCTEVIFG		
RETI		; Task starts here		
		; Back to main program	5	
RTCAIFG_HND				
...		; Vector 8: RTCAIFG		
RETI		; Task starts here		
		; Back to main program	5	
RT0PSIFG_HND				
...		; Vector A: RT0PSIFG		
RETI		; Task starts here		
		; Back to main program	5	
RT1PSIFG_HND				
...		; Vector C: RT1PSIFG		
RETI		; Task starts here		
		; Back to main program	5	

## 24.2.7 Real-Time Clock Calibration for Crystal Offset Error

The RTC\_C module can be calibrated for crystal manufacturing tolerance or offset error for higher time keeping accuracy. The crystal frequency error of up to +/- 240ppm can be calibrated smoothly over a period of 60 seconds. RTCOCAL\_L register is used to adjust the frequency. The calibration value is written into RTCOCAL\_L register and each LSB in this register represent approximately +/- 1ppm correction based on RTCOCALS bit in RTCOCAL\_H register. When RTCOCALS bit is set (up calibration), each LSB in RTCOCAL\_L represent +1ppm adjustment and when RTCOCALS is cleared (down calibration), each LSB in RTCOCAL\_L represent -1ppm adjustment to frequency. Both RTCOCAL\_L and RTCOCAL\_H registers are protected and requires RTC\_C to be unlocked prior to writing into these registers.

### 24.2.7.1 Calibration Frequency

To calibrate the frequency, the RTCCLK output signal is available at a pin. RTCCALFx bits in RTCCTL3 register can be used to select the frequency rate of the output signal. When RTCCALFx = 00, no signal is output on RTCCLK pin. The other settings of RTCCALFx select one of the three frequencies 512Hz, 256Hz or 1Hz. RTCCLK can be measured and the result of this measurement can be applied to the RTCOCALS and RTCOCALx bits to effectively reduce the initial offset of the clock.

#### Calibration Mechanism

RTCOCAL\_L is an 8 bit register. Software can write up to a value of 256ppm into this register but the maximum frequency error that can be corrected is only 240ppm. Software should take care of writing legal values into this register. Read from RTCOCAL always returns the value that was written by software. Real-time clock offset error calibration is inactive when RTC\_C is not enabled (RTCHOLD = 0) or when RTCOCALx bits are zero. RTCOCAL should only be written when RTCHOLD=1. Writing RTCOCAL resets temperature compensation to zero.

In RTC\_C, the offset error calibration takes place over a period of 60 seconds. To achieve approximately +/- 1ppm correction, the 16kHz clock (Q0 output of RT0PS) is adjusted to add or subtract 1 clock pulse. For +1ppm correction, 1 clock pulse is added to 16kHz clock and for -1ppm correction, 1 clock pulse is subtracted from 16kHz clock. This correction happens once every quarter second until the programmed ppm error is compensated.

$$f_{\text{ACLK,meas}} < 32768\text{Hz} \Rightarrow \text{RTCOCALS} = 1, \text{RTCOCALx} = \text{Round}(60 \times 16384 \times (1-f_{\text{ACLK,meas}}/32768))$$

$$f_{\text{ACLK,meas}} \geq 32768\text{Hz} \Rightarrow \text{RTCOCALS} = 0, \text{RTCOCALx} = \text{Round}(60 \times 16384 \times (1-f_{\text{ACLK,meas}}/32768))$$

As an example for up calibration, when the measured frequency is 511.9658Hz against the reference frequency of 512Hz, the frequency error is approximately 67ppm low. In order to increase the frequency by 67ppm, RTCOCALS should be set, and RTCOCALx should be set to Round(60 x 16384 x (1- 511.9658x64/32768)) = 66.

As an example for down calibration, when the measured frequency is 512.0241Hz against the reference frequency of 512Hz, the frequency error is approximately 47ppm high. In order to decrease the frequency by 47ppm, RTCOCALS should be cleared, and RTCOCALx should be set to Round(60 x 16384 x (1- 512.0241x64/32768)) = 46.

All three possible output frequencies 512Hz, 256Hz, and 1Hz at RTCCLK pin are affected by calibration settings. RT0PS interrupt triggered by RT0PS-Q0 (RT0IPx = 000) is based on un-calibrated clock while RT0PS interrupt triggered by RT0PS – Q1 to Q7 (RT0IPx ≠ 000) is based on calibrated clock. RT1PS interrupt (RT1PSIFG) and RTC counter interrupt (RTCTEVIFG) are also based on calibrated clock.

## 24.2.8 Real-Time Clock Compensation for Crystal Temperature Drift

The frequency output of the crystal varies considerably due to drift in temperature. It would be necessary to compensate the real-time clock for this temperature drift for higher time keeping accuracy from standard crystals. A hybrid software and hardware approach can be followed to achieve temperature compensation for RTC\_C.

The software can make use of an (on-chip) temperature sensor to measure the temperature at desired intervals (for example, once every few seconds or minutes). The temperature sensor parameters are calibrated at production and stored in the flash memory. Using the temperature sensor parameters and the measured temperature, software can do parabolic calculations to find out the corresponding frequency error in ppm.

This frequency error can be written into RTCTCMP\_L register for temperature compensation. RTCTCMP\_L is an 8 bit register that can cover the frequency error up to +/- 240ppm. Each LSB in this register represent +/- 1ppm based on RTCTCMPS bit in RTCTCMP\_H register. When RTCTCMPS bit is set, each LSB in RTCTCMP represent +1ppm adjustment (up calibration) and when RTCTCMPS is cleared, each LSB in RTCTCMP represent -1ppm adjustment (down calibration). RTCTCMP register is not protected and can be written any time without having to unlock RTC\_C.

#### 24.2.8.1 Temperature Compensation Scheme

RTCTCMP\_L is an 8 bit register. Software can write up to value of 256ppm into this register but the maximum frequency error that can be corrected including the crystal offset error is only 240ppm. Real-time clock temperature compensation is inactive when RTC\_C is not enabled (RTCHOLD = 0) or when RTCTCMPx bits are zero

When the temperature compensation value is written into RTCTCMP\_L, it is added with offset error calibration value and the resulting value is taken into account from next calibration cycle onwards. The ongoing calibration cycle will not be affected by writes into RTCTCMP register. The maximum frequency error that can be corrected to account for both offset error and temperature variation is +/-240ppm. This means the sign addition of offset error value and temperature compensation value should not exceed maximum of +/- 240ppm otherwise the excess value above +/- 240ppm will be ignored by hardware. Reading from RTCTCMP register at any time returns the cumulative value which is the signed addition of RTCOCALx and RTCTCMPx values. (Note that writing RTCOCAL reset the temperature compensation value to zero.)

For example, when RTCOCAL value is +150ppm and the value written into RTCTCMP is +200ppm the effective value taken in for next calibration cycle is +240ppm. Software is expected to do temperature measurement at certain regularity, calculate the frequency error and write into RTCTCMP register to not to exceed the max limit of +/- 240ppm.

Changing the sign-bit by writing to RTCTCMP\_H will only become effective after having written RTCTCMP\_L as well. Thus it is recommended to write the sign-bit together with compensation value as a 16-bit value into RTCTCMP.

#### 24.2.8.2 Writing to RTCTCMP register

As the system clock could be asynchronous to the RTC\_C clock source, RTCTCRDY bit in RTCTCMP\_H register should be considered for reliable writing into RTCTCMP register. RTCTCRDY is a read only bit that gets set when the hardware is ready to take in the new temperature compensation value. Write to RTCTCMP should be avoided when RTCTCRDY bit is reset. Writes into RTCTCMP register when RTCTCRDY is reset will be ignored.

RTCTCOK is a status bit that indicates if the write to RTCTCMP register is successful or not. RTCTCOK will be set if the write to RTCTCMP is successful and reset if the write is unsuccessful. The status remains same until impacted by the next write to RTCTCMP register. If the write to RTCTCMP is unsuccessful then user needs to reattempt writing into RTCTCMP when RTCTCRDY is set.

Figure 24-2. shows the scheme for real-time clock offset error calibration and temperature compensation.

## RTC\_C Operation

www.ti.com

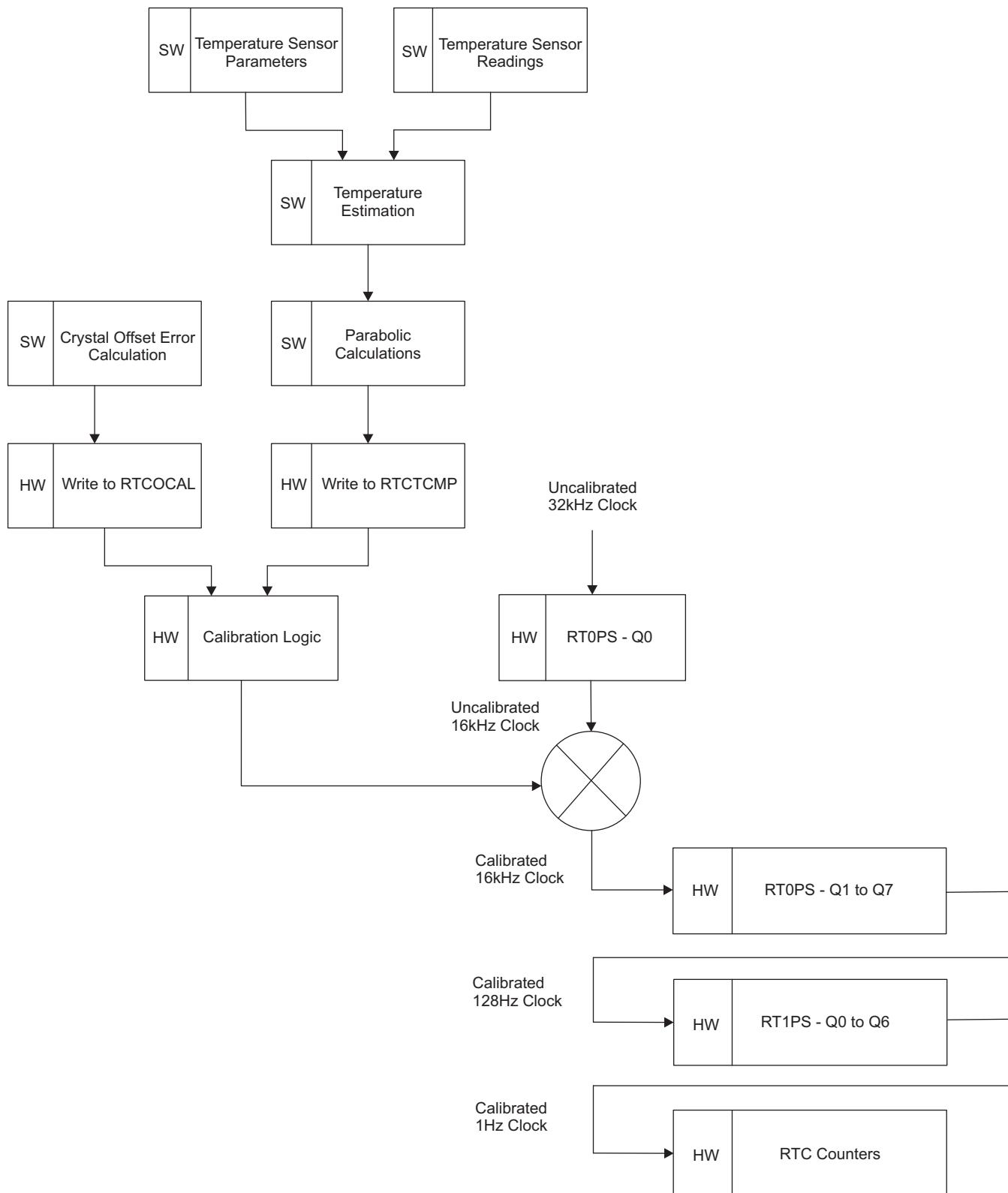


Figure 24-2. RTC\_C Offset Error Calibration and Temperature Compensation Scheme

#### 24.2.8.3 Temperature Measurement and Updates to RTC\_C

The user may wish to perform temperature measurement once every few seconds or once every minute or once in several minutes. Writing to RTCTCMP register for temperature compensation is effective always once in one minute. This means that if the user performs temperature measurement every minute and updates RTCTCMP register with the frequency error, compensation would immediately work fine. But if software performs temperature measurement more frequently than once per minute (for example once every 5 seconds) then it needs to average the error over one minute and update RTCTCMP register once per minute. If the software performs temperature measurement less frequent than a minute (for example, once every 5 minutes) then it needs to calculate the frequency error for the measured temperature and write into RTCTCMP register. The value written into RTCTCMP in this case would be effective until it is updated again by software.

#### 24.2.9 Real-Time Clock Operation in LPM3.5 Low-Power Mode

The regulator of the Power Management Module (PMM) is disabled upon entering LPM3.5, which causes most of the RTC\_C configuration registers to be lost; only the counters and calibration registers are retained. The tables in [Section 24.4](#) list the retained registers in LPM3.5. Also the configuration of the interrupt enables is stored so that the configured interrupts can cause a wakeup upon exit from LPM3.5. Interrupt flags that are set prior to entering LPM3.5 will be cleared upon entering LPM3.5 (Note: this can only happen if the corresponding interrupt is not enabled). The interrupt flags RTCTEVIFG, RTCALIFG, RT1PSIFG, and RTCOFIFG can be used as RTC\_C wake-up interrupt sources. Any interrupt event that occurs during LPM3.5 is stored in the corresponding flags, but only enabled interrupts cause a wake-up. After restoring the configuration registers (and clearing LOCKLPM5), the interrupts can be serviced as usual.

The detailed flow is as follows:

1. Set all I/Os to general-purpose I/Os and configure as needed. Optionally, configure input interrupt pins for wake-up. Configure RTC\_C interrupts for wake-up (set RTCTEVIE, RTCALIE, RT1PSIE, or RTCOFIE. If the alarm interrupt is used as wake-up event also, the alarm registers must be configured as needed).
2. Enter LPM3.5 with LPM3.5 entry sequence.

```
bic #RTCHOLD, &RTCCTL13
```

```
bis #PMMKEY + REGOFF, &PMMCTL0
```

```
bis #LPM4, SR
```

3. LOCKLPM5 is automatically set by hardware upon entering LPM3.5, the core voltage regulator is disabled, and all clocks are disabled except for the 32kHz crystal oscillator clock as the RTC\_C is enabled with RTCHOLD=0.
4. An LPM3.5 wake-up event like an edge on a wake-up input pin or an RTC\_C interrupt event starts the BOR entry sequence together with the core voltage regulator. All peripheral registers are set to their default conditions. The I/O pin state and the interrupt configuration for the RTC\_C remain locked.
5. The device can be configured. The I/O configuration and the RTC\_C interrupt configuration that was not retained during LPM3.5 should be restored to the values prior to entering LPM3.5. Then the LOCKLPM5 bit can be cleared, this releases the I/O pin conditions as well as the RTC\_C interrupt configuration. Registers that are retained during LPM3.5 should not be altered before LOCKLPM5 is cleared.
6. After enabling I/O and RTC\_C interrupts the interrupt that caused the wake-up can be serviced.

If the RTC\_C is enabled (RTCHOLD=0) the 32-kHz oscillator remains active during LPM3.5. Also the fault detection remains functional. If during LPM3.5 a fault occurs and the RTCOFIE was set before entering LPM3.5 a wake-up event will be issued.

## 24.3 RTC\_C Operation - Device-Dependent Features

### 24.3.1 Counter Mode

---

**NOTE:** This feature is available only on selected devices. See the device specific datasheet for more information on the availability of this feature.

---

The RTC\_C module can be configured as a real-time clock with calendar function (calendar mode) or as a 32-bit general purpose counter (counter mode) with the RTCMODE bit.

Counter mode is selected when RTCMODE is reset. In this mode, a 32-bit counter is provided that is directly accessible by software. Switching from calendar mode to counter mode **does not** reset the count value (RTCNT1, RTCNT2, RTCNT3, RTCNT4) nor the prescale counters (RT0PS, RT1PS). These registers must be configured by user software before use.

The clock to increment the counter can be sourced from the 32kHz crystal oscillator, or prescaled versions of the 32kHz crystal oscillator clock. Prescaled versions are sourced from the prescale dividers (RT0PS and RT1PS). RT0PS and RT1PS can output /2, /4, /8, 16, /32, /64, /128, and /256 versions of the 32kHz clock. The output of RT0PS can be cascaded with RT1PS. The cascaded output can also be used as a clock source input to the 32-bit counter.

Four individual 8-bit counters are cascaded to provide the 32-bit counter. This provides 8-bit, 16-bit, 24-bit, or 32-bit overflow intervals of the counter clock. The RTCTEV bits select the respective trigger event. An RTCTEV event can trigger an interrupt by setting the RTCTEVIE bit. Each counter, RTCNT1 through RTCNT4, is individually accessible and may be written to.

RT0PS and RT1PS can be configured as two 8-bit counters or cascaded into a single 16-bit counter. RT0PS and RT1PS can be halted on an individual basis by setting their respective RT0PSHOLD and RT1PSHOLD bits. When RT0PS is cascaded with RT1PS, setting RT0PSHOLD causes both RT0PS and RT1PS to be halted. The 32-bit counter can be halted several ways depending on the configuration. If the 32-bit counter is sourced directly by the 32kHz crystal clock, it can be halted by setting RTCHOLD. If it is sourced from the output of RT1PS, it can be halted by setting RT1PSHOLD or RTCHOLD. Finally, if it is sourced from the cascaded outputs of RT0PS and RT1PS, it can be halted by setting RT0PSHOLD, RT1PSHOLD, or RTCHOLD.

---

**NOTE: Accessing the RTCNT1, RTCNT2, RTCNT3, RTCNT4, RT0PS, RT1PS registers**

When the counter clock is asynchronous to the CPU clock, any read from any RTCNT1, RTCNT2, RTCNT3, RTCNT4, RT0PS, or RT1PS register should occur while the counter is not operating. Otherwise, the results may be unpredictable. Alternatively, the counter may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to these registers takes effect immediately.

---

**NOTE: For reliable update to all Counter Mode registers**

Depending on the cascading of counters, when a write occurs, hold all subsequent counters. Eg., if RTPS0 is being updated, RTCPS1HOLD = 1, If RTPS1 is being updated, RTCHOLD = 1

#### 24.3.1.1 Real-Time Clock Interrupts in Counter Mode

In counter mode, four interrupt sources are available: RT0PSIFG, RT1PSIFG, RTCTEVIFG, and RTCOFIFG. RTCAIFG and RTCRDYIFG are cleared. RTCRDYIE and RTCAIE are don't care.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. In counter mode, divide ratios of /2, /4, /8, /16, /32, /64, /128, and /256 of the clock source are possible. Setting the RT0PSIE bit enables the interrupt.

RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. In counter mode, RT1PS is sourced with low-frequency oscillator clock, or the output of RT0PS, so divide ratios of /2, /4, /8, /16, /32, /64, /128, and /256 of the respective clock source are possible. Setting the RT1PSIE bit enables the interrupt.

In Counter Mode, the RTC\_C module provides for an interval timer that sources real-time clock interrupt, RTCTEVIFG. The interval timer can be selected to cause an interrupt event when an 8-bit, 16-bit, 24-bit, or 32-bit overflow occurs within the 32-bit counter. The event is selectable with the RTCTEV bits. Setting the RTCTEVIE bit enables the interrupt.

The RTCOFIG bit flags a failure of the 32-kHz crystal oscillator. Its main purpose is to wake-up the CPU from LPM3.5 in case an oscillator failure occurred.

### **24.3.2 Real-Time Clock Event/Tamper Detection With Time Stamp**

---

**NOTE:** This feature is available only on selected devices. Please refer to the device specific datasheet for more information on the availability of this feature.

---

The RTC\_C module provides an external event/tamper detection and time stamp for up to two external events. The pins, RTCCAP0 and RTCCAP1 (these pins are present only on devices that support this feature of RTC\_C. Please refer to the device specific datasheet for the availability of this feature), can be used as an event or tamper detection input of an external switch (mechanical or electronic). After device power-up, this feature needs to be enabled by the user by setting the bit TCEN in RTCTCCTL0 register. Event/Tamper Detection with time stamp is supported in all MSP430 operating modes, as long as there is a valid RTC power supply.

- When there is an event on RTCCAPx pin and the time capture feature is enabled (TCEN=1), the corresponding CAPEV bit in RTCCAPxCTL register is set and the corresponding time stamp information (seconds, minutes, hours, day of month, month and year) is stored in the respective back-up registers (RTCSECBAKx, RTCMINBAKx, RTCHOURBAKx, RTCDAYBAKx, RTCMONBAKx and RTCYEARBAKx).
- In case of multiple events, **ONLY** the time stamp of the event that occurred the first will be stored in the respective back-up registers. Once CAPEV is set by the first event on RTCCAPx, all subsequent events on RTCCAPx will be ignored until the respective CAPEV bit is cleared by the User.
- The CAPES bit in the RTCCAPxCTL register sets the event edge for the corresponding RTCCAPx pin.
  - Bit = 0: CAPEV flag is set with a low-to-high transition
  - Bit = 1: CAPEV flag is set with a high-to-low transition

---

**NOTE: Writing to CAPESx**

Writing to CAPES can result in setting the corresponding interrupt flags.

<b>CAPESx</b>	<b>RTCCAPx</b>	<b>RTCCAPIFG</b>
0 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 0	1	May be set

- 
- The interrupt flag RTCCAPIFG is set when any of the individual CAPEV bits are set. If the RTCIV is read RTCCAPIFG is cleared but not the status flags (CAPEV bits). They are then read by CPU and need to be cleared by SW only.
  - By setting the RTCCAPIE bit, an event on RTCCAPx generates an interrupt and this interrupt can be used as LPM3.5/LPM4.5 wake-up event in modules supporting LPM3.5/LPM4.5.
  - When the time capture feature is enabled (TCEN=1), all back-up registers (RTCSECBAKx, RTCMINBAKx, RTCHOURBAKx, RTCDAYBAKx, RTCMONBAKx and RTCYEARBAKx) are read-only to user and can only be written by the RTC hardware. When RTCBCD = 1 and TCEN = 1, BCD format is selected for the backup registers. If the back-up registers were written to by the hardware before

setting TCEN, then the previous values are retained until there is a time capture event that overrides the values with the time-stamp

- When the time capture feature is disabled (TCEN=0), all the backup registers (RTCSECBAKx, RTCMINBAKx, RTCHOURBAKx, RTCDAYBAKx, RTCMONBAKx and RTCYEARBAKx) can be written to, only by the CPU. When TCEN = 0, RTCBCD bit setting is ignored for the backup registers. The data in the back-up registers when TCEN=1 will be retained until the user writes new values after TCEN is cleared.
- When TCEN is cleared, all CAPEV bits and RTCCAPIFG will be cleared.
- [Table 24-1](#) shows how to use the DIR, REN, and OUT bits in RTCCAPxCTL for proper configuration of RTCCAPx pins.

**Table 24-1. RTCCAPx Pin Configuration**

DIR	REN	OUT	RTCCAPx Configuration
0	0	x	Input
0	1	0	Input with pull-down resistor
0	1	1	Input with pull-up resistor
1	x	x	Output

#### 24.3.2.1 Real-Time Clock Event/Tamper Detection Interrupts

With the event/tamper detection feature one addition interrupt sources is available, namely RTCCAPIFG. This flag are prioritized and combined with the other interrupt flags to source a single interrupt vector. The interrupt vector register (RTCIV) is used to determine which flag requested an interrupt.

The RTCCAPIFG bit flags the occurrence of a tamper event. The exact source of the interrupt among multiple tamper events can be found out by reading the CAPEV bit in respective RTCCAPxCTL registers (one per temper source). If RTCIV is read the RTCCAPIFG is cleared but not the status flags (CAPEV bits).

## 24.4 RTC\_C Registers

The RTC\_C module registers are shown in [Table 24-2](#). This table also shows which registers are key protected and which are retained during LPM3.5. The registers that are retained during LPM3.5 and given with a reset value are not reset on POR; they are reset based on a signal derived from the RTC supply. Registers that are not retained during LPM3.5 must be restored after exit from LPM3.5.

The high-side SVS must not be disabled by software if the real-time clock feature is needed. When the high-side SVS is disabled, the RTC\_C registers with LPM3.5 retention are not accessible by the CPU.

The base address for the RTC\_C module registers can be found in the device-specific data sheet. The address offsets are shown in [Table 24-2](#).

The additional registers that are available if Event/Tamper Detection is implemented are shown in [Table 24-3](#) together with the corresponding address offsets.

If the counter mode is supported, the register aliases shown in [Table 24-4](#) can be used to access the counter registers.

---

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

---

**Table 24-2. RTC\_C Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Key Protected	LPM3.5 Retention
00h	RTCCTL0	Real-Time Clock Control 0	Read/write	Word	9600h	yes	not retained
00h	RTCCTL0_L	Real-Time Clock Control 0 Low	Read/write	Byte	00h	yes	not retained
01h	RTCCTL0_H	Real-Time Clock Control 0 High	Read/write	Byte	96h	n/a	not retained
02h	RTCCTL13	Real-Time Clock Control 1, 3	Read/write	Word	0070h	yes	high byte retained
02h	RTCCTL1 or RTCCTL13_L	Real-Time Clock Control 1	Read/write	Byte	70h	yes	not retained
03h	RTCCTL3 or RTCCTL13_H	Real-Time Clock Control 3	Read/write	Byte	00h	yes	retained
04h	RTCOCAL	Real-Time Clock Offset Calibration	Read/write	Word	0000h	yes	retained
04h	RTCOCAL_L		Read/write	Byte	00h	yes	retained
05h	RTCOCAL_H		Read/write	Byte	00h	yes	retained
06h	RTCTCMP	Real-Time Clock Temperature Compensation	Read/write	Word	4000h	no	retained
06h	RTCTCMP_L		Read/write	Byte	00h	no	retained
07h	RTCTCMP_H		Read/write	Byte	40h	no	retained
08h	RTCPS0CTL	Real-Time Prescale Timer 0 Control	Read/write	Word	0100h	no	not retained
08h	RTCPS0CTL_L		Read/write	Byte	00h	no	not retained
09h	RTCPS0CTL_H		Read/write	Byte	01h	no	not retained
0Ah	RTCPS1CTL	Real-Time Prescale Timer 1 Control	Read/write	Word	0100h	no	not retained
0Ah	RTCPS1CTL_L		Read/write	Byte	00h	no	not retained
0Bh	RTCPS1CTL_H		Read/write	Byte	01h	no	not retained
0Ch	RTCPS	Real-Time Prescale Timer 0, 1 Counter	Read/write	Word	none	yes	retained
0Ch	RT0PS	Real-Time Prescale Timer 0 Counter	Read/write	Byte	none	yes	retained
		or RTCPS_L					

**Table 24-2. RTC\_C Registers (continued)**

Offset	Acronym	Register Name	Type	Access	Reset	Key Protected	LPM3.5 Retention
0Dh	RT1PS	Real-Time Prescale Timer 1 Counter	Read/write	Byte	none	yes	retained
	or RTCPS_H						
0Eh	RTClV	Real Time Clock Interrupt Vector	Read	Word	0000h	no	not retained
10h	RTCTIM0	Real-Time Clock Seconds, Minutes	Read/write	Word	undefined	yes	retained
10h	RTCSEC	Real-Time Clock Seconds	Read/write	Byte	undefined	yes	retained
	or RTCTIM0_L						
11h	RTCMIN	Real-Time Clock Minutes	Read/write	Byte	undefined	yes	retained
	or RTCTIM0_H						
12h	RTCTIM1	Real-Time Clock Hour, Day of Week	Read/write	Word	undefined	yes	retained
12h	RTCHOUR	Real-Time Clock Hour	Read/write	Byte	undefined	yes	retained
	or RTCTIM1_L						
13h	RTCDOW	Real-Time Clock Day of Week	Read/write	Byte	undefined	yes	retained
	or RTCTIM1_H						
14h	RTCDATE	Real-Time Clock Date	Read/write	Word	undefined	yes	retained
14h	RTCDAY	Real-Time Clock Day of Month	Read/write	Byte	undefined	yes	retained
	or RTCDATE_L						
15h	RTCMON	Real-Time Clock Month	Read/write	Byte	undefined	yes	retained
	or RTCDATE_H						
16h	RTCYEAR	Real-Time Clock Year <sup>(1)</sup>	Read/write	Word	undefined	yes	retained
18h	RTCAMINHR	Real-Time Clock Minutes, Hour Alarm	Read/write	Word	undefined	no	retained
18h	RTCAMIN	Real-Time Clock Minutes Alarm	Read/write	Byte	undefined	no	retained
	or RTCAMINHR_L						
19h	RTCAHOUR	Real-Time Clock Hours Alarm	Read/write	Byte	undefined	no	retained
	or RTCAMINHR_H						
1Ah	RTCADOWDAY	Real-Time Clock Day of Week, Day of Month Alarm	Read/write	Word	undefined	no	retained
1Ah	RTCADOW	Real-Time Clock Day of Week Alarm	Read/write	Byte	undefined	no	retained
	or RTCADOWDAY_L						
1Bh	RTCADAY	Real-Time Clock Day of Month Alarm	Read/write	Byte	undefined	no	retained
	or RTCADOWDAY_H						
1Ch	BIN2BCD	Binary-to-BCD conversion register	Read/write	Word	0000h	no	not retained
1Eh	BCD2BIN	BCD-to-binary conversion register	Read/write	Word	0000h	no	not retained

<sup>(1)</sup> The year register RTCYEAR must not be accessed in byte mode.

**Table 24-3. RTC\_C Event/Tamper Detection Registers**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Access</b>	<b>Reset</b>	<b>Key Protected</b>	<b>LPM3.5 Retention</b>
20h	RTCTCCTL0	Real-Time Clock Time Capture Control Register 0	Read/write	Byte	02h	yes	retained
21h	RTCTCCTL1	Real-Time Clock Time Capture Control Register 1	Read/write	Byte	00h	yes	not retained
22h	RTCCAP0CTL	Tamper Detect Pin 0 Control Register	Read/write	Byte	00h	yes	not retained
23h	RTCCAP1CTL	Tamper Detect Pin 1 Control Register	Read/write	Byte	00h	yes	not retained
30h	RTCSECBAK0	Real-Time Clock Seconds Backup Register 0	Read/write	Byte	00h	yes	retained
31h	RTCMINBAK0	Real-Time Clock Minutes Backup Register 0	Read/write	Byte	00h	yes	retained
32h	RTCHOURBAK0	Real-Time Clock Hours Backup Register 0	Read/write	Byte	00h	yes	retained
33h	RTCDAYBAK0	Real-Time Clock Days Backup Register 0	Read/write	Byte	00h	yes	retained
34h	RTCMONBAK0	Real-Time Clock Months Backup Register 0	Read/write	Byte	00h	yes	retained
36h	RTCYEARBAK0	Real-Time Clock year Backup Register 0	Read/write	Word	00h	yes	retained
38h	RTCSECBAK1	Real-Time Clock Seconds Backup Register 1	Read/write	Byte	00h	yes	retained
39h	RTCMINBAK1	Real-Time Clock Minutes Backup Register 1	Read/write	Byte	00h	yes	retained
3Ah	RTCHOURBAK1	Real-Time Clock Hours Backup Register 1	Read/write	Byte	00h	yes	retained
3Bh	RTCDAYBAK1	Real-Time Clock Days Backup Register 1	Read/write	Byte	00h	yes	retained
3Ch	RTCMONBAK1	Real-Time Clock Months Backup Register 1	Read/write	Byte	00h	yes	retained
3Eh	RTCYEARBAK1	Real-Time Clock Year Backup Register 1	Read/write	Word	00h	yes	retained

**Table 24-4. RTC\_C Real-Time Clock Counter Mode Aliases**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Access</b>	<b>Reset</b>	<b>Key Protected</b>	<b>LPM3.5 Retention</b>
10h	RTCCNT12	Real-Time Counter 1, 2	Read/write	Word	undefined	yes	retained
10h	RTCCNT1	Real-Time Counter 1	Read/write	Byte	undefined	yes	retained
11h	RTCCNT2	Real-Time Counter 2	Read/write	Byte	undefined	yes	retained
12h	RTCCNT34	Real-Time Counter 3, 4	Read/write	Word	undefined	yes	retained
12h	RTCCNT3	Real-Time Counter 3	Read/write	Byte	undefined	yes	retained
13h	RTCCNT4	Real-Time Counter 4	Read/write	Byte	undefined	yes	retained

**24.4.1 RTCCTL0\_L Register**

Real-Time Clock Control 0 Low Register

**Figure 24-3. RTCCTL0\_L Register**

7	6	5	4	3	2	1	0
RTCOFIE <sup>(1)</sup>	RTCTEVIE <sup>(1)</sup>	RTCAIE <sup>(1)</sup>	RTCRDYIE	RTCOFIG	RTCTEVIFG	RTCAIFG	RTCRDYIFG
rw-0	rw-0	rw-0	rw-0	rw-(0)	rw-(0)	rw-(0)	rw-(0)

- <sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration is required after wake-up from LPMx.5 before clearing LOCKLPM5.

**Table 24-5. RTCCTL0\_L Register Description**

Bit	Field	Type	Reset	Description
7	RTCOFIE	RW	0h	32-kHz crystal oscillator fault interrupt enable. This interrupt can be used as LPM3.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPM3.5 wake-up enabled)
6	RTCTEVIE	RW	0h	Real-time clock time event interrupt enable. In modules supporting LPM3.5 this interrupt can be used as LPM3.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPM3.5 wake-up enabled)
5	RTCAIE	RW	0h	Real-time clock alarm interrupt enable. In modules supporting LPM3.5 this interrupt can be used as LPM3.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPM3.5 wake-up enabled)
4	RTCRDYIE	RW	0h	Real-time clock ready interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
3	RTCOFIG	RW	0h	32-kHz crystal oscillator fault interrupt flag. This interrupt can be used as LPM3.5 wake-up event. It also indicates a clock failure during backup operation. 0b = No interrupt pending 1b = Interrupt pending. A 32-kHz crystal oscillator fault occurred after last reset.
2	RTCTEVIFG	RW	0h	Real-time clock time event interrupt flag. In modules supporting LPM3.5 this interrupt can be used as LPM3.5 wake-up event. 0b = No time event occurred 1b = Time event occurred
1	RTCAIFG	RW	0h	Real-time clock alarm interrupt flag. In modules supporting LPM3.5 this interrupt can be used as LPM3.5 wake-up event. 0b = No time event occurred 1b = Time event occurred
0	RTCRDYIFG	RW	0h	Real-time clock ready interrupt flag 0b = RTC cannot be read safely 1b = RTC can be read safely

#### **24.4.2 RTCCTL0\_H Register**

Real-Time Clock Control 0 High Register

**Figure 24-4. RTCCTL0\_H Register**

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

**Table 24-6. RTCCTL0\_H Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCKEY	RW	96h	Real-time clock key. This register should be written with A5h to unlock RTC_C. Any write with value other than A5h will lock the module. Read from this register always returns 96h.

**24.4.3 RTCCTL1 Register**

Real-Time Clock Control Register 1

**Figure 24-5. RTCCTL1 Register**

7	6	5	4	3	2	1	0
RTBCD	RTCHOLD <sup>(1)</sup>	RTCMODE <sup>(1)</sup>	RTCRDY	RTCSSELx <sup>(1)</sup>	RTCTEVx <sup>(1)</sup>		
rw-(0)	rw-(1)	rw-(1)	r-(1)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration is required after wake-up from LPMx.5 before clearing LOCKLPM5.

**Table 24-7. RTCCTL1 Register Description**

Bit	Field	Type	Reset	Description
7	RTBCD	RW	0h	Real-time clock BCD select. Selects BCD counting for real-time clock. Applies to calendar mode (RTCMODE = 1) only; setting is ignored in counter mode. 0b = Binary (hexadecimal) code selected 1b = Binary coded decimal (BCD) code selected
6	RTCHOLD	RW	1h	Real-time clock hold 0b = Real-time clock (32-bit counter or calendar mode) is operational. 1b = In counter mode (RTCMODE = 0), only the 32-bit counter is stopped. In calendar mode (RTCMODE = 1), the calendar is stopped as well as the prescale counters, RT0PS and RT1PS. RT0PSHOLD and RT1PSHOLD are don't care.
5	RTCMODE	RW	1h	Real-time clock mode. In RTC_C modules without counter mode support this bit is read-only and always reads 1. 0b = 32-bit counter mode 1b = Calendar mode. Switching between counter and calendar mode does not reset the real-time clock counter registers. These registers must be configured by user software before use.
4	RTCRDY	R	1h	Real-time clock ready 0b = RTC time values in transition (calendar mode only) 1b = RTC time values safe for reading (calendar mode only). This bit indicates when the real-time clock time values are safe for reading (calendar mode only). In counter mode, RTCRDY remains cleared.
3-2	RTCSSELx	RW	0h	Real-time clock source select. In counter mode, selects clock input source to the 32-bit counter. In calendar mode, these bits are don't care. The clock input is automatically set to the output of RT1PS. 00b = 32-kHz crystal oscillator clock 01b = 32-kHz crystal oscillator clock 10b = Output from RT1PS 11b = Output from RT1PS
1-0	RTCTEVx	RW	0h	Real-time clock time event Calendar Mode (RTCMODE = 1) 00b = Minute changed 01b = Hour changed 10b = Every day at midnight (00:00) 11b = Every day at noon (12:00) Counter Mode (RTCMODE = 0) 00b = 8-bit overflow 01b = 16-bit overflow 10b = 24-bit overflow 11b = 32-bit overflow

#### 24.4.4 RTCCTL3 Register

Real-Time Clock Control 3 Register

**Figure 24-6. RTCCTL3 Register**

7	6	5	4	3	2	1	0
Reserved						RTCCALFx <sup>(1)</sup>	
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from AUXVCC3 supply voltage level.

**Table 24-8. RTCCTL3 Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1-0	RTCCALFx	RW	0h	Real-time clock calibration frequency. Selects frequency output to RTCCLK pin for calibration measurement. The corresponding port must be configured for the peripheral module function. The RTCCLK is not available in counter mode and remains low, and the RTCCALF bits are don't care. 00b = No frequency output to RTCCLK pin 01b = 512 Hz 10b = 256 Hz 11b = 1 Hz

#### 24.4.5 RTCOCAL Register

Real-Time Clock Offset Calibration Register

**Figure 24-7. RTCOCAL Register**

15	14	13	12	11	10	9	8
RTCOALS <sup>(1)</sup>		Reserved					
rw-(0)	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
RTCOCALx <sup>(1)</sup>							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from AUXVCC3 supply voltage level.

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from AUXVCC3 supply voltage level.

**Table 24-9. RTCOCAL Register Description**

Bit	Field	Type	Reset	Description
15	RTCOALS	RW	0h	Real-time clock offset error calibration sign. This bit decides the sign of offset error calibration. 0b = Down calibration. Frequency adjusted down. 1b = Up calibration. Frequency adjusted up.
14-8	Reserved	R	0h	Reserved. Always reads as 0.
7-0	RTCOCALx	RW	0h	Real-time clock offset error calibration. Each LSB represents approximately +1ppm (RTCOALS = 1) or -1ppm (RTCOALS = 0) adjustment in frequency. Maximum effective calibration value is +/-240ppm. Excess values written above +/-240ppm will be ignored by hardware.

**24.4.6 RTCTCMP Register**

Real-Time Clock Temperature Compensation Register

**Figure 24-8. RTCTCMP Register**

15	14	13	12	11	10	9	8
RTCTCMPS <sup>(1)</sup>	RTCTCRDY <sup>(1)</sup>	RTCTCOK <sup>(1)</sup>			Reserved		
rw-(0)	r-(1)	r-(0)	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
RTCTCMPx <sup>(1)</sup>							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from AUXVCC3 supply voltage level.<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from AUXVCC3 supply voltage level.**Table 24-10. RTCTCMP Register Description**

Bit	Field	Type	Reset	Description
15	RTCTCMPS	RW	0h	Real-time clock temperature compensation sign. This bit decides the sign of temperature compensation. <sup>(1)</sup> 0b = Down calibration. Frequency adjusted down. 1b = Up calibration. Frequency adjusted up.
14	RTCTCRDY	R	1h	Real-time clock temperature compensation ready. This is a read only bit that indicates when the RTCTCMPx can be written. Write to RTCTCMPx should be avoided when RTCTCRDY is reset.
13	RTCTCOK	R	0h	Real-time clock temperature compensation write OK. This is a read-only bit that indicates if the write to RTCTCMP is successful or not. 0b = Write to RTCTCMPx is unsuccessful 1b = Write to RTCTCMPx is successful
12-8	Reserved	R	0h	Reserved. Always reads as 0.
7-0	RTCTCMPx	RW	0h	Real-time clock temperature compensation. Value written into this register is used for temperature compensation of RTC_C. Each LSB represents approximately +1ppm (RTCTCMPS = 1) or -1ppm (RTCTCMPS = 0) adjustment in frequency. Maximum effective calibration value is +/-240ppm. Excess values written above +/-240ppm are ignored by hardware.

<sup>(1)</sup> Changing the sign-bit by writing to RTCTCMP\_H becomes effective only after also writing RTCTCMP\_L.

#### 24.4.7 RTCNT1 Register

Real-Time Clock Counter 1 Register – Counter Mode

**Figure 24-9. RTCNT1 Register**

7	6	5	4	3	2	1	0
RTCNT1							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-11. RTCNT1 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT1	RW	undefined	The RTCNT1 register is the count of RTCNT1

#### 24.4.8 RTCNT2 Register

Real-Time Clock Counter 2 Register – Counter Mode

**Figure 24-10. RTCNT2 Register**

7	6	5	4	3	2	1	0
RTCNT2							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-12. RTCNT2 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT2	RW	undefined	The RTCNT2 register is the count of RTCNT2

#### 24.4.9 RTCNT3 Register

Real-Time Clock Counter 3 Register – Counter Mode

**Figure 24-11. RTCNT3 Register**

7	6	5	4	3	2	1	0
RTCNT3							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-13. RTCNT3 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT3	RW	undefined	The RTCNT3 register is the count of RTCNT3

#### 24.4.10 RTCNT4 Register

Real-Time Clock Counter 4 Register – Counter Mode

**Figure 24-12. RTCNT4 Register**

7	6	5	4	3	2	1	0
RTCNT4							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-14. RTCNT4 Register Description**

Bit	Field	Type	Reset	Description
7-0	RTCNT4	RW	undefined	The RTCNT4 register is the count of RTCNT4.

**24.4.11 RTCSEC Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Seconds Register – Calendar Mode With Hexadecimal Format

**Figure 24-13. RTCSEC Register**

7	6	5	4	3	2	1	0
Seconds							
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 24-15. RTCSEC Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always 0
5-0	Seconds	RW	undefined	Seconds (0 to 59)

**24.4.12 RTCSEC Register – Calendar Mode With BCD Format**

Real-Time Clock Seconds Register – Calendar Mode With BCD Format

**Figure 24-14. RTCSEC Register**

7	6	5	4	3	2	1	0
Seconds – high digit				Seconds – low digit			
r-0	rw	rw	rw	rw	rw	rw	rw

**Table 24-16. RTCSEC Register Description**

Bit	Field	Type	Reset	Description
7	0	R	0h	Always 0
6-4	Seconds – high digit	RW	undefined	Seconds – high digit (0 to 5)
3-0	Seconds – low digit	RW	undefined	Seconds – low digit (0 to 9)

#### **24.4.13 RTCMIN Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Minutes Register – Calendar Mode With Hexadecimal Format

**Figure 24-15. RTCMIN Register**

7	6	5	4	3	2	1	0
0	Minutes						
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 24-17. RTCMIN Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always 0
5-0	Minutes	RW	undefined	Minutes (0 to 59)

#### **24.4.14 RTCMIN Register – Calendar Mode With BCD Format**

Real-Time Clock Minutes Register – Calendar Mode With BCD Format

**Figure 24-16. RTCMIN Register**

7	6	5	4	3	2	1	0
0	Minutes – high digit						
r-0	rw	rw	rw	rw	rw	rw	rw

**Table 24-18. RTCMIN Register Description**

Bit	Field	Type	Reset	Description
7	0	R	0h	Always 0
6-4	Minutes – high digit	RW	undefined	Minutes – high digit (0 to 5)
3-0	Minutes – low digit	RW	undefined	Minutes – low digit (0 to 9)

**24.4.15 RTCHOUR Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Hours Register – Calendar Mode With Hexadecimal Format

**Figure 24-17. RTCHOUR Register**

7	6	5	4	3	2	1	0
				Hours			
r-0	r-0	r-0	rw	rw	rw	rw	rw

**Table 24-19. RTCHOUR Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always 0
4-0	Hours	RW	undefined	Hours (0 to 23)

**24.4.16 RTCHOUR Register – Calendar Mode With BCD Format**

Real-Time Clock Hours Register – Calendar Mode With BCD Format

**Figure 24-18. RTCHOUR Register**

7	6	5	4	3	2	1	0
			Hours – high digit		Hours – low digit		
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 24-20. RTCHOUR Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always 0
5-4	Hours – high digit	RW	undefined	Hours – high digit (0 to 2)
3-0	Hours – low digit	RW	undefined	Hours – low digit (0 to 9)

#### 24.4.17 RTCDOW Register – Calendar Mode

Real-Time Clock Day of Week Register – Calendar Mode

**Figure 24-19. RTCDOW Register**

7	6	5	4	3	2	1	0		
			0	Day of week					
r-0	r-0	r-0	r-0	r-0	rw	rw	rw		

**Table 24-21. RTCDOW Register Description**

Bit	Field	Type	Reset	Description
7-3	0	R	0h	Always 0
2-0	Day of week	RW	undefined	Day of week (0 to 6)

#### 24.4.18 RTCDAY Register – Calendar Mode With Hexadecimal Format

Real-Time Clock Day of Month Register – Calendar Mode With Hexadecimal Format

**Figure 24-20. RTCDAY Register**

7	6	5	4	3	2	1	0		
		0		Day of month					
r-0	r-0	r-0	rw	rw	rw	rw	rw		

**Table 24-22. RTCDAY Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always 0
4-0	Day of month	RW	undefined	Day of month (1 to 28, 29, 30, 31)

#### 24.4.19 RTCDAY Register – Calendar Mode With BCD Format

Real-Time Clock Day of Month Register – Calendar Mode With BCD Format

**Figure 24-21. RTCDAY Register**

7	6	5	4	3	2	1	0
	0	Day of month – high digit				Day of month – low digit	
r-0	r-0	rw	rw	rw	rw	rw	rw

**Table 24-23. RTCDAY Register Description**

Bit	Field	Type	Reset	Description
7-6	0	R	0h	
5-4	Day of month – high digit	RW	undefined	Day of month – high digit (0 to 3)
3-0	Day of month – low digit	RW	undefined	Day of month – low digit (0 to 9)

**24.4.20 RTCMON Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Month Register – Calendar Mode With Hexadecimal Format

**Figure 24-22. RTCMON Register**

7	6	5	4	3	2	1	0
0				Month			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

**Table 24-24. RTCMON Register Description**

Bit	Field	Type	Reset	Description
7-4	0	R	0h	Always 0
3-0	Month	RW	undefined	Month (1 to 12)

**24.4.21 RTCMON Register – Calendar Mode With BCD Format**

Real-Time Clock Month Register – Calendar Mode With BCD Format

**Figure 24-23. RTCMON Register**

7	6	5	4	3	2	1	0
0				Month – high digit	Month – low digit		
r-0	r-0	r-0	rw	rw	rw	rw	rw

**Table 24-25. RTCMON Register Description**

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always 0
4	Month – high digit	RW	undefined	Month – high digit (0 or 1)
3-0	Month – low digit	RW	undefined	Month – low digit (0 to 9)

#### 24.4.22 RTCYEAR Register – Calendar Mode With Hexadecimal Format

Real-Time Clock Year Low-Byte Register – Calendar Mode With Hexadecimal Format

**Figure 24-24. RTCYEAR Register**

15	14	13	12	11	10	9	8		
		0		Year – high byte					
r-0	r-0	r-0	r-0	rw	rw	rw	rw		
7	6	5	4	3	2	1	0		
Year – low byte									
rw	rw	rw	rw	rw	rw	rw	rw		

**Table 24-26. RTCYEAR Register Description**

Bit	Field	Type	Reset	Description
15-12	0	R	0h	Always 0
11-8	Year – high byte	RW	undefined	Year – high byte. Valid values for Year are 0 to 4095.
7-0	Year – low byte	RW	undefined	Year – low byte. Valid values for Year are 0 to 4095.

#### 24.4.23 RTCYEAR Register – Calendar Mode With BCD Format

Real-Time Clock Year Low-Byte Register – Calendar Mode With BCD Format

**Figure 24-25. RTCYEAR Register**

15	14	13	12	11	10	9	8
0		Century – high digit			Century – low digit		
r-0	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Decade				Year – lowest digit			
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-27. RTCYEAR Register Description**

Bit	Field	Type	Reset	Description
15	0	R	0h	Always 0
14-10	Century – high digit	RW	undefined	Century – high digit (0 to 4)
11-8	Century – low digit	RW	undefined	Century – low digit (0 to 9)
7-4	Decade	RW	undefined	Decade (0 to 9)
3-0	Year – lowest digit	RW	undefined	Year – lowest digit (0 to 9)

**24.4.24 RTCAMIN Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Minutes Alarm Register – Calendar Mode With Hexadecimal Format

**Figure 24-26. RTCAMIN Register**

7	6	5	4	3	2	1	0
AE	0						Minutes
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 24-28. RTCAMIN Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6	0	R	0h	Always 0.
5-0	Minutes	RW	undefined	Minutes (0 to 59)

**24.4.25 RTCAMIN Register – Calendar Mode With BCD Format**

Real-Time Clock Minutes Alarm Register – Calendar Mode With BCD Format

**Figure 24-27. RTCAMIN Register**

7	6	5	4	3	2	1	0
AE		Minutes – high digit			Minutes – low digit		
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-29. RTCAMIN Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	0h	Alarm enable
6-4	Minutes – high digit	RW	undefined	Minutes – high digit (0 to 5)
3-0	Minutes – low digit	RW	undefined	Minutes – low digit (0 to 9)

#### **24.4.26 RTCAHOUR Register**

Real-Time Clock Hours Alarm Register – Calendar Mode With Hexadecimal Format

**Figure 24-28. RTCAHOUR Register**

7	6	5	4	3	2	1	0
AE	0					Hours	
rw	r-0	r-0	rw	rw	rw	rw	rw

**Table 24-30. RTCAHOUR Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6-5	0	R	0h	Always 0
4-0	Hours	RW	undefined	Hours (0 to 23)

#### **24.4.27 RTCAHOUR Register – Calendar Mode With BCD Format**

Real-Time Clock Hours Alarm Register – Calendar Mode With BCD Format

**Figure 24-29. RTCAHOUR Register**

7	6	5	4	3	2	1	0
AE	0		Hours – high digit			Hours – low digit	
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 24-31. RTCAHOUR Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6	0	R	0h	Always 0
5-4	Hours – high digit	RW	undefined	Hours – high digit (0 to 2)
3-0	Hours – low digit	RW	undefined	Hours – low digit (0 to 9)

**24.4.28 RTCADOW Register – Calendar Mode**

Real-Time Clock Day of Week Alarm Register – Calendar Mode

**Figure 24-30. RTCADOW Register**

7	6	5	4	3	2	1	0
AE		0				Day of week	
rw	r-0	r-0	r-0	r-0	rw	rw	rw

**Table 24-32. RTCADOW Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6-3	0	R	0h	Always 0
2-0	Day of week	RW	undefined	Day of week (0 to 6)

**24.4.29 RTCADAY Register – Calendar Mode With Hexadecimal Format**

Real-Time Clock Day of Month Alarm Register – Calendar Mode With Hexadecimal Format

**Figure 24-31. RTCADAY Register**

7	6	5	4	3	2	1	0
AE		0			Day of month		
rw	r-0	r-0	rw	rw	rw	rw	rw

**Table 24-33. RTCADAY Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6-5	0	R	0h	Always 0
4-0	Day of month	RW	undefined	Day of month (1 to 28, 29, 30, 31)

**24.4.30 RTCADAY Register – Calendar Mode With BCD Format**

Real-Time Clock Day of Month Alarm Register – Calendar Mode With BCD Format

**Figure 24-32. RTCADAY Register**

7	6	5	4	3	2	1	0
AE	0	Day of month – high digit			Day of month – low digit		
rw	r-0	rw	rw	rw	rw	rw	rw

**Table 24-34. RTCADAY Register Description**

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable
6	0	R	0h	Always 0
5-4	Day of month – high digit	RW	undefined	Day of month – high digit (0 to 3)
3-0	Day of month – low digit	RW	undefined	Day of month – low digit (0 to 9)

### **24.4.31 RTCPS0CTL Register**

Real-Time Clock Prescale Timer 0 Control Register

**Figure 24-33. RTCPS0CTL Register**

15	14	13	12	11	10	9	8
Reserved		RT0PSDIV <sup>(1)</sup>			Reserved		RT0PSHOLD <sup>(1)</sup>
r0	r0	rw-(0)	rw-(0)	rw-(0)	r0	r0	rw-(1)
7	6	5	4	3	2	1	0
Reserved		RT0IP <sup>(1)</sup>			RT0PSIE	RT0PSIFG	
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

<sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration is required after wake-up from LPMx.5 before clearing LOCKLPM5.

<sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration is required after wake-up from LPMx.5 before clearing LOCKLPM5.

**Table 24-35. RTCPS0CTL Register Description**

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0.
13-11	RT0PSDIV	RW	0h	Prescale timer 0 clock divide. These bits control the divide ratio of the RT0PS counter. In real-time clock calendar mode, these bits are don't care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128. 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
10-9	Reserved	R	0h	Reserved. Always reads as 0.
8	RT0PSHOLD	RW	1h	Prescale timer 0 hold. In real-time clock calendar mode, this bit is don't care. RT0PS is stopped via the RTCHOLD bit. 0b = RT0PS is operational 1b = RT0PS is held
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT0IP	RW	0h	Prescale timer 0 interrupt interval 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
1	RT0PSIE	RW	0h	Prescale timer 0 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
0	RT0PSIFG	RW	0h	Prescale timer 0 interrupt flag 0b = No time event occurred 1b = Time event occurred

**24.4.32 RTCPS1CTL Register**

Real-Time Clock Prescale Timer 1 Control Register

**Figure 24-34. RTCPS1CTL Register**

15	14	13	12	11	10	9	8
RT1SSELx <sup>(1)</sup>		RT1PSDIVx <sup>(1)</sup>			Reserved		RT1PSHOLD <sup>(1)</sup>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0	r0	rw-(1)
7	6	5	4	3	2	1	0
Reserved		RT1IPx <sup>(1)</sup>			RT1PSIE	RT1PSIFG	
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

<sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration is required after wake-up from LPMx.5 before clearing LOCKLPM5.

<sup>(1)</sup> The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration is required after wake-up from LPMx.5 before clearing LOCKLPM5.

**Table 24-36. RTCPS1CTL Register Description**

Bit	Field	Type	Reset	Description
15-14	RT1SSELx	RW	0h	Prescale timer 1 clock source select. Selects clock input source to the RT1PS counter. In real-time clock calendar mode, these bits are do not care. RT1PS clock input is automatically set to the output of RT0PS. 00b = 32-kHz crystal oscillator clock 01b = 32-kHz crystal oscillator clock 10b = Output from RT0PS 11b = Output from RT0PS
13-11	RT1PSDIVx	RW	0h	Prescale timer 1 clock divide. These bits control the divide ratio of the RT0PS counter. In real-time clock calendar mode, these bits are don't care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128. 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
10-9	Reserved	R	0h	Reserved. Always reads as 0.
8	RT1PSHOLD	RW	1h	Prescale timer 1 hold. In real-time clock calendar mode, this bit is don't care. RT1PS is stopped via the RTCHOLD bit. 0b = RT1PS is operational 1b = RT1PS is held
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT1IPx	RW	0h	Prescale timer 1 interrupt interval 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
1	RT1PSIE	RW	0h	Prescale timer 1 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)

**Table 24-36. RTCPS1CTL Register Description (continued)**

Bit	Field	Type	Reset	Description
0	RT1PSIFG	RW	0h	Prescale timer 1 interrupt flag. This interrupt can be used as LPMx.5 wake-up event. 0b = No time event occurred 1b = Time event occurred

**24.4.33 RTCPS0 Register**

Real-Time Clock Prescale Timer 0 Counter Register

**Figure 24-35. RTCPS0 Register**

7	6	5	4	3	2	1	0
RT0PS							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-37. RTCPS0 Register Description**

Bit	Field	Type	Reset	Description
7-0	RT0PS	RW	undefined	Prescale timer 0 counter value

**24.4.34 RTCPS1 Register**

Real-Time Clock Prescale Timer 1 Counter Register

**Figure 24-36. RTCPS1 Register**

7	6	5	4	3	2	1	0
RT1PS							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 24-38. RTCPS1 Register Description**

Bit	Field	Type	Reset	Description
7-0	RT1PS	RW	undefined	Prescale timer 1 counter value

### **24.4.35 RTCIV Register**

Real-Time Clock Interrupt Vector Register

**Figure 24-37. RTCIV Register**

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

**Table 24-39. RTCIV Register Description**

Bit	Field	Type	Reset	Description
15-0	RTCIVx	R	0h	<p>Real-time clock interrupt vector value</p> <p>Without Event/Tamper Detection implemented:</p> <ul style="list-style-type: none"> <li>00h = No interrupt pending</li> <li>02h = Interrupt Source: RTC oscillator failure; Interrupt Flag: RTCOFIFG; Interrupt Priority: Highest</li> <li>04h = Interrupt Source: RTC ready; Interrupt Flag: RTCRDYIFG</li> <li>06h = Interrupt Source: RTC interval timer; Interrupt Flag: RTCTEVIFG</li> <li>08h = Interrupt Source: RTC user alarm; Interrupt Flag: RTCAIFG</li> <li>0Ah = Interrupt Source: RTC prescaler 0; Interrupt Flag: RT0PSIFG</li> <li>0Ch = Interrupt Source: RTC prescaler 1; Interrupt Flag: RT1PSIFG</li> <li>0Eh = Reserved</li> <li>10h = Reserved ; Interrupt Priority: Lowest</li> </ul> <p>Without Event/Tamper Detection implemented:</p> <ul style="list-style-type: none"> <li>00h = No interrupt pending</li> <li>02h = Interrupt Source: RTC oscillator failure; Interrupt Flag: RTCOFIFG; Interrupt Priority: Highest</li> <li>04h = Interrupt Source: RTC Tamper Event; Interrupt Flag: RTCCAPIFG</li> <li>06h = Interrupt Source: RTC ready; Interrupt Flag: RTCRDYIFG</li> <li>08h = Interrupt Source: RTC interval timer; Interrupt Flag: RTCTEVIFG</li> <li>0Ah = Interrupt Source: RTC user alarm; Interrupt Flag: RTCAIFG</li> <li>0Ch = Interrupt Source: RTC prescaler 0; Interrupt Flag: RT0PSIFG</li> <li>0Eh = Interrupt Source: RTC prescaler 1; Interrupt Flag: RT1PSIFG</li> <li>10h = Reserved ; Interrupt Priority: Lowest</li> </ul>

**24.4.36 BIN2BCD Register**

Binary-to-BCD Conversion Register

**Figure 24-38. BIN2BCD Register**

15	14	13	12	11	10	9	8
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 24-40. BIN2BCD Register Description**

Bit	Field	Type	Reset	Description
15-0	BIN2BCDx	RW	0h	Read: 16-bit BCD conversion of previously written 12-bit binary number. Write: 12-bit binary number to be converted.

**24.4.37 BCD2BIN Register**

BCD-to-Binary Conversion Register

**Figure 24-39. BCD2BIN Register**

15	14	13	12	11	10	9	8
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 24-41. BCD2BIN Register Description**

Bit	Field	Type	Reset	Description
15-0	BCD2BINx	RW	0h	Read: 12-bit binary conversion of previously written 16-bit BCD number. Write: 16-bit BCD number to be converted.

#### 24.4.38 RTCSECBAKx Register – Hexadecimal Format

Real-Time Clock Seconds Backup Register – Hexadecimal Format

**Figure 24-40. RTCSECBAKx Register**

7	6	5	4	3	2	1	0
0	0						Seconds <sup>(1)</sup>
rw-(0)							

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-42. RTCSECBAKx Register Description**

Bit	Field	Type	Reset	Description
7-6	0	RW	0h	Always 0.
5-0	Seconds	RW	0h	Seconds. Valid values are 0 to 59.

#### 24.4.39 RTCSECBAKx Register – BCD Format

Real-Time Clock Seconds Backup Register – BCD Format

**Figure 24-41. RTCSECBAKx Register**

7	6	5	4	3	2	1	0
0		Seconds – high digit <sup>(1)</sup>			Seconds – low digit <sup>(1)</sup>		
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-43. RTCSECBAKx Register Description**

Bit	Field	Type	Reset	Description
7	0	RW	0h	Always 0.
6-4	Seconds – high digit	RW	0h	Seconds – high digit. Valid values are 0 to 5.
3-0	Seconds – low digit	RW	0h	Seconds – low digit. Valid values are 0 to 9.

**24.4.40 RTCMINBAKx Register – Hexadecimal Format**

Real-Time Clock Minutes Backup Register – Hexadecimal Format

**Figure 24-42. RTCMINBAKx Register**

7	6	5	4	3	2	1	0
0	0						Minutes <sup>(1)</sup>
rw-(0)							

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-44. RTCMINBAKx Register Description**

Bit	Field	Type	Reset	Description
7-6	0	RW	0h	Always 0.
5-0	Minutes	RW	0h	Minutes. Valid values are 0 to 59.

**24.4.41 RTCMINBAKx Register – BCD Format**

Real-Time Clock Minutes Backup Register – BCD Format

**Figure 24-43. RTCMINBAKx Register**

7	6	5	4	3	2	1	0
0		Minutes – high digit <sup>(1)</sup>			Minutes – low digit <sup>(1)</sup>		
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-45. RTCMINBAKx Register Description**

Bit	Field	Type	Reset	Description
7	0	RW	0h	Always 0.
6-4	Minutes – high digit	RW	0h	Minutes – high digit. Valid values are 0 to 5.
3-0	Minutes – low digit	RW	0h	Minutes – low digit. Valid values are 0 to 9.

#### 24.4.42 RTCHOURBAKx Register – Hexadecimal Format

Real-Time Clock Hours Backup Register – Hexadecimal Format

**Figure 24-44. RTCHOURBAKx Register**

7	6	5	4	3	2	1	0
0	0	0					Hours <sup>(1)</sup>
rw-(0)							

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-46. RTCHOURBAKx Register Description**

Bit	Field	Type	Reset	Description
7-5	0	RW	0h	Always 0.
4-0	Hours	RW	0h	Hours. Valid values are 0 to 23.

#### 24.4.43 RTCHOURBAKx Register – BCD Format

Real-Time Clock Hours Backup Register – BCD Format

**Figure 24-45. RTCHOURBAKx Register**

7	6	5	4	3	2	1	0
0	0		Hours – high digit <sup>(1)</sup>			Hours – low digit <sup>(1)</sup>	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-47. RTCHOURBAKx Register Description**

Bit	Field	Type	Reset	Description
7-6	0	RW	0h	Always 0.
5-4	Hours – high digit	RW	0h	Hours – high digit. Valid values are 0 to 2.
3-0	Hours – low digit	RW	0h	Hours – low digit. Valid values are 0 to 9.

**24.4.44 RTCDAYBAKx Register – Hexadecimal Format**

Real-Time Clock Day of Month Backup Register – Hexadecimal Format

**Figure 24-46. RTCDAYBAKx Register**

7	6	5	4	3	2	1	0			
0	0	0		Day of month <sup>(1)</sup>						
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)			

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-48. RTCDAYBAKx Register Description**

Bit	Field	Type	Reset	Description
7-5	0	RW	0h	Always 0.
4-0	Day of month	RW	0h	Day of month. Valid values are 1 to 31.

**24.4.45 RTCDAYBAKx Register – BCD Format**

Real-Time Clock Day of Month Backup Register – BCD Format

**Figure 24-47. RTCDAYBAKx Register**

7	6	5	4	3	2	1	0			
0	0		Day of month – high digit <sup>(1)</sup>	Day of month – low digit <sup>(1)</sup>						
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)			

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-49. RTCDAYBAKx Register Description**

Bit	Field	Type	Reset	Description
7-6	0	RW	0h	Always 0.
5-4	Day of month – high digit	RW	0h	Day of month – high digit. Valid values are 0 to 3.
3-0	Day of month – low digit	RW	0h	Day of month – low digit. Valid values are 0 to 9.

#### 24.4.46 RTCMONBAKx Register – Hexadecimal Format

Real-Time Clock Month Backup Register – Hexadecimal Format

**Figure 24-48. RTCMONBAKx Register**

7	6	5	4	3	2	1	0
0	0	0	0				Month <sup>(1)</sup>
rw-(0)							

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-50. RTCMONBAKx Register Description**

Bit	Field	Type	Reset	Description
7-4	0	RW	0h	Always 0.
3-0	Month	RW	0h	Month. Valid values are 1 to 12.

#### 24.4.47 RTCMONBAKx Register – BCD Format

Real-Time Clock Month Backup Register – BCD Format

**Figure 24-49. RTCMONBAKx Register**

7	6	5	4	3	2	1	0
0	0		Month – high digit <sup>(1)</sup>			Month – low digit <sup>(1)</sup>	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-51. RTCMONBAKx Register Description**

Bit	Field	Type	Reset	Description
7-6	0	RW	0h	Always 0.
5-4	Month – high digit	RW	0h	Month – high digit. Valid values are 0 to 3.
3-0	Month – low digit	RW	0h	Month – low digit. Valid values are 0 to 9.

**24.4.48 RTCYEARBAKx Register – Hexadecimal Format**

Real-Time Clock Year Low-Byte Backup Register – Hexadecimal Format

**Figure 24-50. RTCYEARBAKx Register**

15	14	13	12	11	10	9	8			
0	0	0	0	Year – high byte <sup>(1)</sup>						
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)			
7	6	5	4	3	2	1	0			
Year – low byte <sup>(1)</sup>										
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)			

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.**Table 24-52. RTCYEARBAKx Register Description**

Bit	Field	Type	Reset	Description
15-12	0	RW	0h	Always 0.
11-8	Year – high byte	RW	0h	Year – high byte. Valid values of Year are 0 to 4095.
7-0	Year – low byte	RW	0h	Year – low byte. Valid values of Year are 0 to 4095.

**24.4.49 RTCYEARBAKx Register – BCD Format**

Real-Time Clock Year Low-Byte Backup Register – BCD Format

**Figure 24-51. RTCYEARBAKx Register**

15	14	13	12	11	10	9	8
0	Century – high digit <sup>(1)</sup>						
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Decade <sup>(1)</sup>							Year – lowest digit <sup>(1)</sup>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.**Table 24-53. RTCYEARBAKx Register Description**

Bit	Field	Type	Reset	Description
15	0	RW	0h	Always 0.
14-12	Century – high digit	RW	0h	Century – high digit. Valid values are 0 to 4.
11-8	Century – low digit	RW	0h	Century – low digit. Valid values are 0 to 9.
7-4	Decade	RW	0h	Decade. Valid values are 0 to 9.
3-0	Year – lowest digit	RW	0h	Year – lowest digit. Valid values are 0 to 9.

#### 24.4.50 RTCTCCTL0 Register

Real-Time Clock Time Capture Control Register 0

**Figure 24-52. RTCTCCTL0 Register**

7	6	5	4	3	2	1	0	
			Reserved					
r-0	r-0	r-0	r-0	r-0	r-0	rw-(1)	rw-(0)	

<sup>(1)</sup> These bits are not reset on POR; they are reset based on a signal derived from the RTC supply.

**Table 24-54. RTCTCCTL0 Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1	AUX3RST	RW	1h	Indication of power cycle on AUXVCC3 0b = No power cycle on AUXVCC3 since the last clear by the User 1b = Indication of AUXVCC3 power cycle. Needs to be cleared by User to observe the next power cycle on AUXVCC3
0	TCEN	RW	0h	Enable for RTC tamper detection with time stamp 0b = Tamper detection with time stamp disabled 1b = Tamper detection with time stamp enabled

#### 24.4.51 RTCTCCTL1 Register

Real-Time Clock Time Capture Control Register 1

**Figure 24-53. RTCTCCTL1 Register**

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 24-55. RTCTCCTL1 Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1	RTCCAPIE	RW	0h	Tamper event interrupt enable. In modules that support LPM3.5 or LPM4.5, this interrupt can be used as LPM3.5 or LPM4.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPM3.5 and LPM4.5 wake-up enabled)
0	RTCCAPIFG	RW	0h	Common interrupt flag for all tamper events. In modules that support LPM3.5 or LPM4.5, this interrupt can be used as LPM3.5 or LPM4.5 wake-up event. 0b = Tamper event did not occur 1b = At least one tamper event occurred. Status of individual tamper events can be found from the CAPEV bit in RTCCAPxCTL.

**24.4.52 RTCCAPxCTL Register**

Tamper Detect Pin Control Register

**Figure 24-54. RTCCAPxCTL Register**

7	6	5	4	3	2	1	0
Reserved	OUT <sup>(1)</sup>	DIR <sup>(1)</sup>	IN <sup>(1)</sup>	REN <sup>(1)</sup>	CAPES <sup>(1)</sup>	Reserved	CAPEV <sup>(1)</sup>
r-0	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	r-0	r/w0

<sup>(1)</sup> The configuration of these bits is retained during LPM5 until LOCKLPM5 is cleared, but not the bits themselves; therefore, reconfiguration is required after wake-up from LPM5 before clearing LOCKLPM5.

**Table 24-56. RTCCAPxCTL Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	R	0h	Reserved. Always reads as 0.
6	OUT	RW	0h	RTCCAPx output 0b = Output low 1b = Output high
5	DIR	RW	0h	RTCCAPx pin direction 0b = RTCCAPx pin configured as input 1b = RTCCAPx pin configured as output
4	IN	R	0h	RTCCAPx input. The external input on RTCCAPx pin can be read by this bit. 0b = Input is low 1b = Input is high
3	REN	RW	0h	RTCCAPx pin pullup or pulldown resistor enable. When respective pin is configured as input, setting this bit enables the pullup or pulldown (see <a href="#">Table 24-1</a> ). 0b = Pullup or pulldown disabled 1b = Pullup or pulldown enabled
2	CAPES	RW	0h	Event edge selection 0b = Event on a low-to-high transition 1b = Event on a high-to-low transition
1	Reserved	R	0h	Reserved. Always reads as 0.
0	CAPEV	RW	0h	Tamper event status flag. All subsequent events on RTCCAPx after CAPEV is set are ignored until CAPEV is cleared by the user. Can only be written as 0. 0b = Tamper event did not occur 1b = Tamper event occurred

## **32-Bit Hardware Multiplier (MPY32)**

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

Topic	Page
<b>25.1 32-Bit Hardware Multiplier (MPY32) Introduction .....</b>	<b>668</b>
<b>25.2 MPY32 Operation .....</b>	<b>670</b>
<b>25.3 MPY32 Registers .....</b>	<b>682</b>

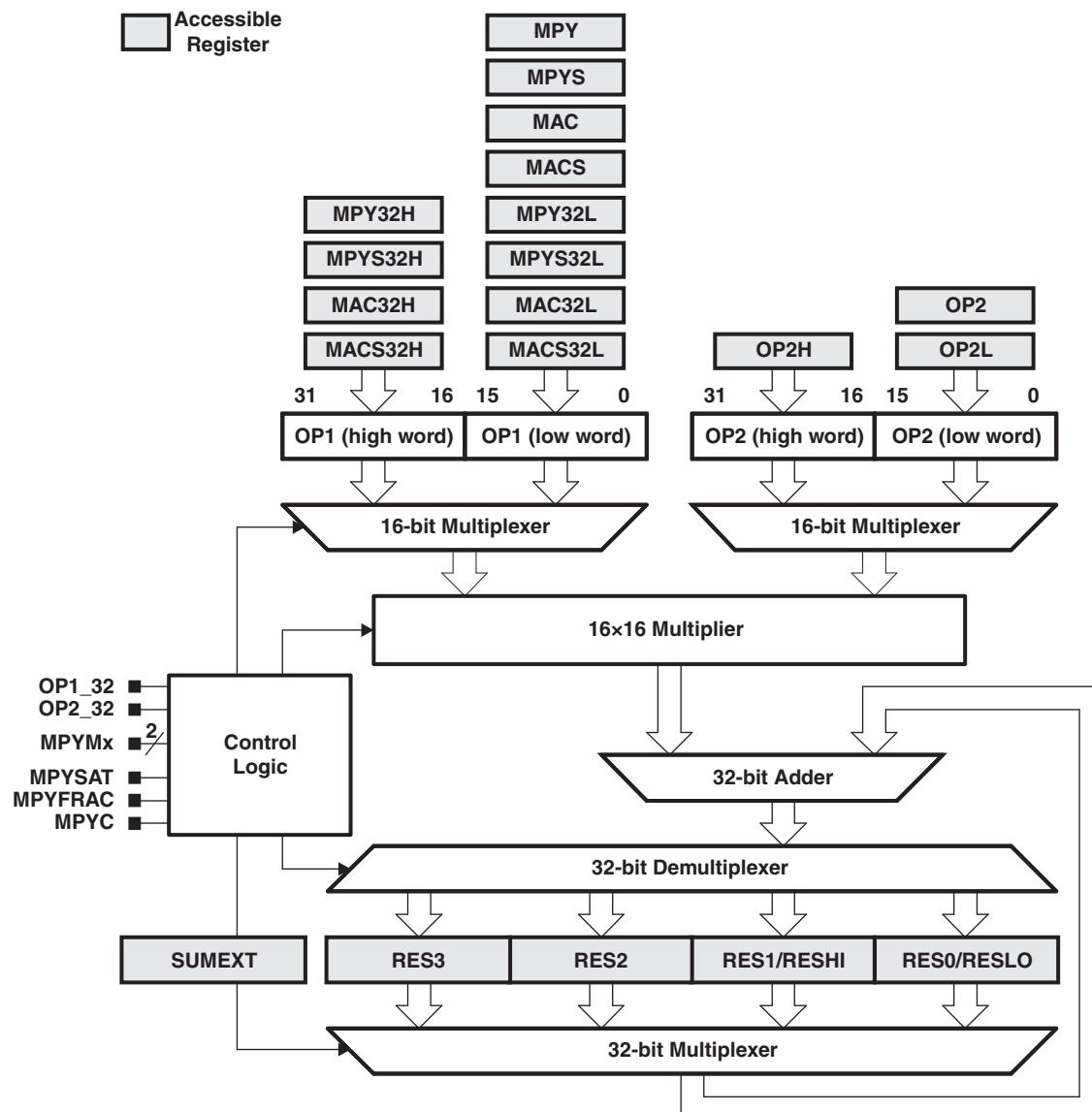
## 25.1 32-Bit Hardware Multiplier (MPY32) Introduction

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 25-1](#).


**Figure 25-1. MPY32 Block Diagram**

## 25.2 MPY32 Operation

The MPY32 supports 8-bit, 16-bit, 24-bit, and 32-bit operands with unsigned multiply, signed multiply, unsigned multiply-accumulate, and signed multiply-accumulate operations. The size of the operands are defined by the address the operand is written to and if it is written as word or byte. The type of operation is selected by the address the first operand is written to.

The hardware multiplier has two 32-bit operand registers – operand one (OP1) and operand two (OP2), and a 64-bit result register accessible via registers RES0 to RES3. For compatibility with the 16x16 hardware multiplier, the result of a 8-bit or 16-bit operation is accessible via RESLO, RESHI, and SUMEXT, as well. RESLO stores the low word of the 16x16-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 25-1** 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 25-1. Result Availability (MPYFRAC = 0, MPYSAT = 0)**

Operation (OP1 x OP2)	Result Ready in MCLK Cycles					After
	RES0	RES1	RES2	RES3	MPYC Bit	
8/16 x 8/16	3	3	4	4	3	OP2 written
24/32 x 8/16	3	5	6	7	7	OP2 written
8/16 x 24/32	3	5	6	7	7	OP2L written
	N/A	3	4	4	4	OP2H written
24/32 x 24/32	3	8	10	11	11	OP2L written
	N/A	3	5	6	6	OP2H written

### 25.2.1 Operand Registers

Operand one (OP1) has 12 registers (see [Table 25-2](#)) used to load data into the multiplier and also select the multiply mode. Writing the low word of the first operand to a given address selects the type of multiply operation to be performed, but does not start any operation. When writing a second word to a high-word register with suffix 32H, the multiplier assumes a 32-bit-wide OP1, otherwise, 16 bits are assumed. The last address written prior to writing OP2 defines the width of the first operand. For example, if MPY32L is written first followed by MPY32H, all 32 bits are used and the data width of OP1 is set to 32 bits. If MPY32H is written first followed by MPY32L, the multiplication ignores MPY32H and assumes a 16-bit-wide OP1 using the data written into MPY32L.

Repeated multiply operations may be performed without reloading OP1 if the OP1 value is used for successive operations. It is not necessary to rewrite the OP1 value to perform the operations.

**Table 25-2. 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 25-3. 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 MPYDLYWRDEN 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 25-1](#) for how many CPU cycles are needed until a certain result register is ready and valid for each of the different modes.

**25.2.2 Result Registers**

The multiplication result is always 64 bits wide. It is accessible via registers RES0 to RES3. Used with a signed operation, MPYS or MACS, the results are appropriately sign extended. If the result registers are loaded with initial values before a MACS operation, the user software must take care that the written value is properly sign extended to 64 bits.

**NOTE: Changing of result registers during multiplication**

The result registers must not be modified by the user software after writing the second operand into OP2 or OP2L until the initiated operation is completed.

In addition to RES0 to RES3, for compatibility with the 16x16 hardware multiplier, the 32-bit result of a 8-bit or 16-bit operation is accessible via RESLO, RESHI, and SUMEXT. In this case, the result low register RESLO holds the lower 16 bits of the calculation result and the result high register RESHI holds the upper 16 bits. RES0 and RES1 are identical to RESLO and RESHI, respectively, in usage and access of calculated results.

The sum extension register SUMEXT contents depend on the multiply operation and are listed in [Table 25-4](#). 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 25-4](#) 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 25-4. 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. 0000h Result was positive or zero 0FFFh Result was negative	MPYC contains the sign of the result. 0 Result was positive or zero 1 Result was negative
MAC	SUMEXT contains the carry of the result. 0000h No carry for result 0001h Result has a carry	MPYC contains the carry of the result. 0 No carry for result 1 Result has a carry
MACS	SUMEXT contains the extended sign of the result. 0000h Result was positive or zero 0FFFh Result was negative	MPYC contains the carry of the result. 0 No carry for result 1 Result has a carry

### **25.2.2.1 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.

### **25.2.3 Software Examples**

Examples for all multiplier modes follow. All 8x8 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

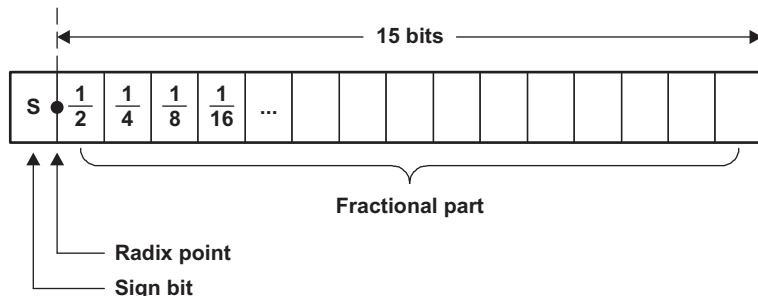
; 32x32 Signed Multiply
    MOV      #01234h,&MPYS32L    ; Load low word of 1st operand
    MOV      #01234h,&MPYS32H    ; Load high word of 1st operand
    MOV      #05678h,&OP2L       ; Load low word of 2nd operand
    MOV      #05678h,&OP2H       ; Load high word of 2nd operand
;   ...                           ; 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
```

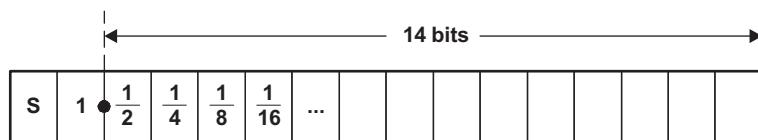
## 25.2.4 Fractional Numbers

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 25-2](#) 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  $0.999969482 \approx 1.0$  for the signed Q15 format with 16 bits.



**Figure 25-2. Q15 Format Representation**

The range can be increased by shifting the radix point to the right as shown in [Figure 25-3](#). The signed Q14 format with 16 bits gives a range from  $-2.0$  to  $1.999938965 \approx 2.0$ .



**Figure 25-3. 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 numbers in the range from  $-1.0$  to  $1.0$  is always in that same range.

### 25.2.4.1 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  $16 \times 16$ -bit operations and bits 64 and 65 for  $32 \times 32$ -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 25-5. 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

#### 25.2.4.2 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.

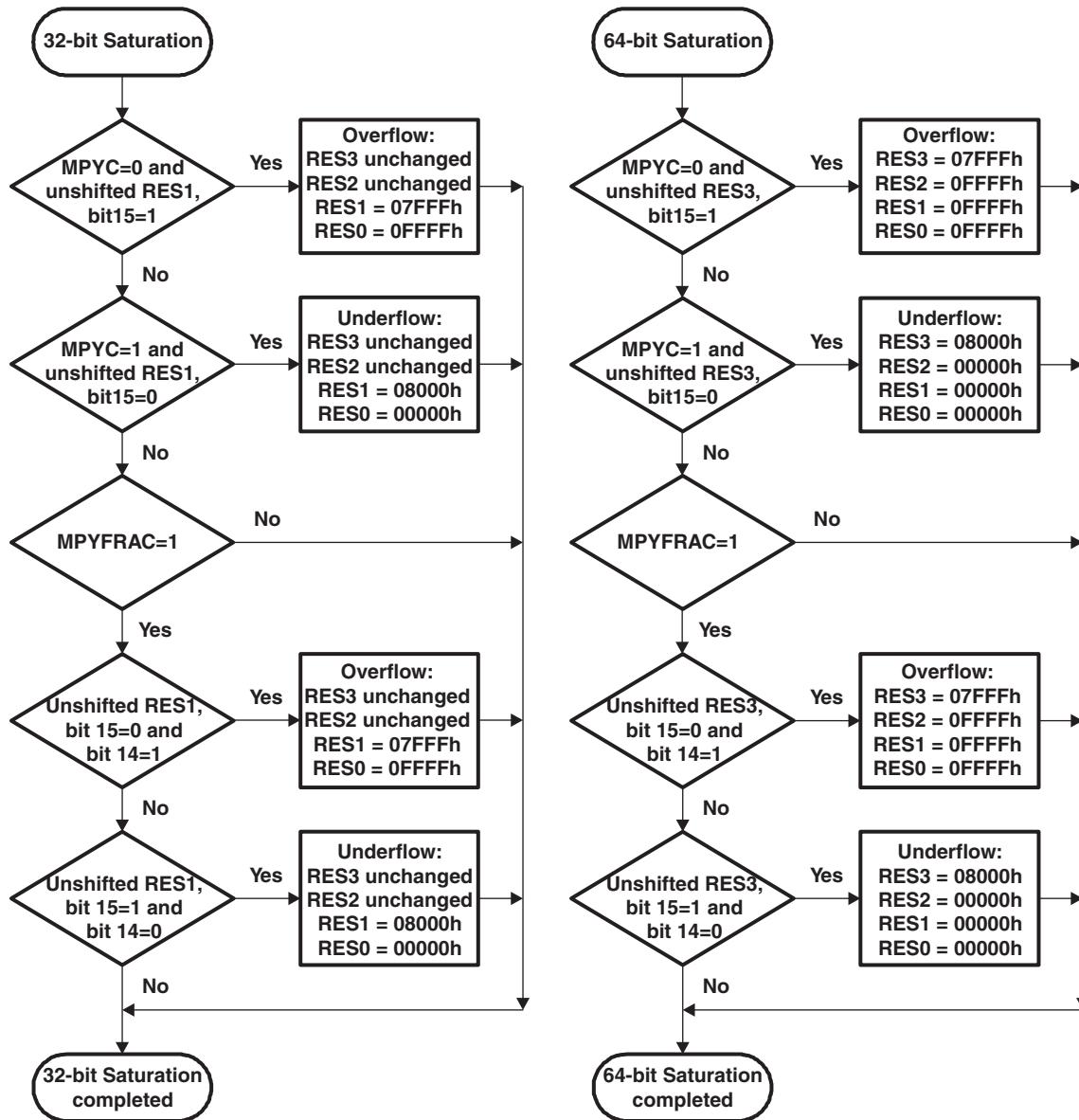
Enabling the saturation mode does not affect the content of the SUMEXT register nor the content of the MPYC bit.

```
; 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 25-6. 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 25-4 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.

**Figure 25-4. Saturation Flow Chart**

**NOTE: Saturation in fractional mode**

In case of multiplying  $-1.0 \times -1.0$  in fractional mode, the result of +1.0 is out of range, thus, the saturated result gives the most positive result.

When using multiply-and-accumulate operations, the accumulated values are saturated as if MPYFRAC = 0; only during read accesses to the result registers the values are saturated taking the fractional mode into account. This provides additional dynamic range during the calculation and only the end result is then saturated if needed.

The following example illustrates a special case showing the saturation function in fractional mode. It also uses the 8-bit functionality of the MPY32 module.

```
; Turn on fractional and saturation mode,  
; clear all other bits in MPY32CTL0:  
MOV      #MPYSAT+MPYFRAC ,&MPY32CTL0  
;Pre-load result registers to demonstrate overflow  
MOV      #0,&RES3          ;  
MOV      #0,&RES2          ;  
MOV      #07FFFh,&RES1        ;  
MOV      #0FA60h,&RES0        ;  
MOV.B    #050h,&MACS_B       ; 8-bit signed MAC operation  
MOV.B    #012h,&OP2_B        ; Start 16x16 bit operation  
MOV      &RES0,R6            ; R6 = 0FFFFh  
MOV      &RES1,R7            ; R7 = 07FFFh
```

The result is saturated because already the result not converted into a fractional number shows an overflow. The multiplication of the two positive numbers 00050h and 00012h gives 005A0h. 005A0h added to 07FFF FA60h results in 8000 059Fh, without MPYC being set. Because the MSB of the unmodified result RES1 is 1 and MPYC = 0, the result is saturated according [Figure 25-4](#).

**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.

### 25.2.5 Putting It All Together

[Figure 25-5](#) shows the complete multiplication flow, depending on the various selectable modes for the MPY32 module.

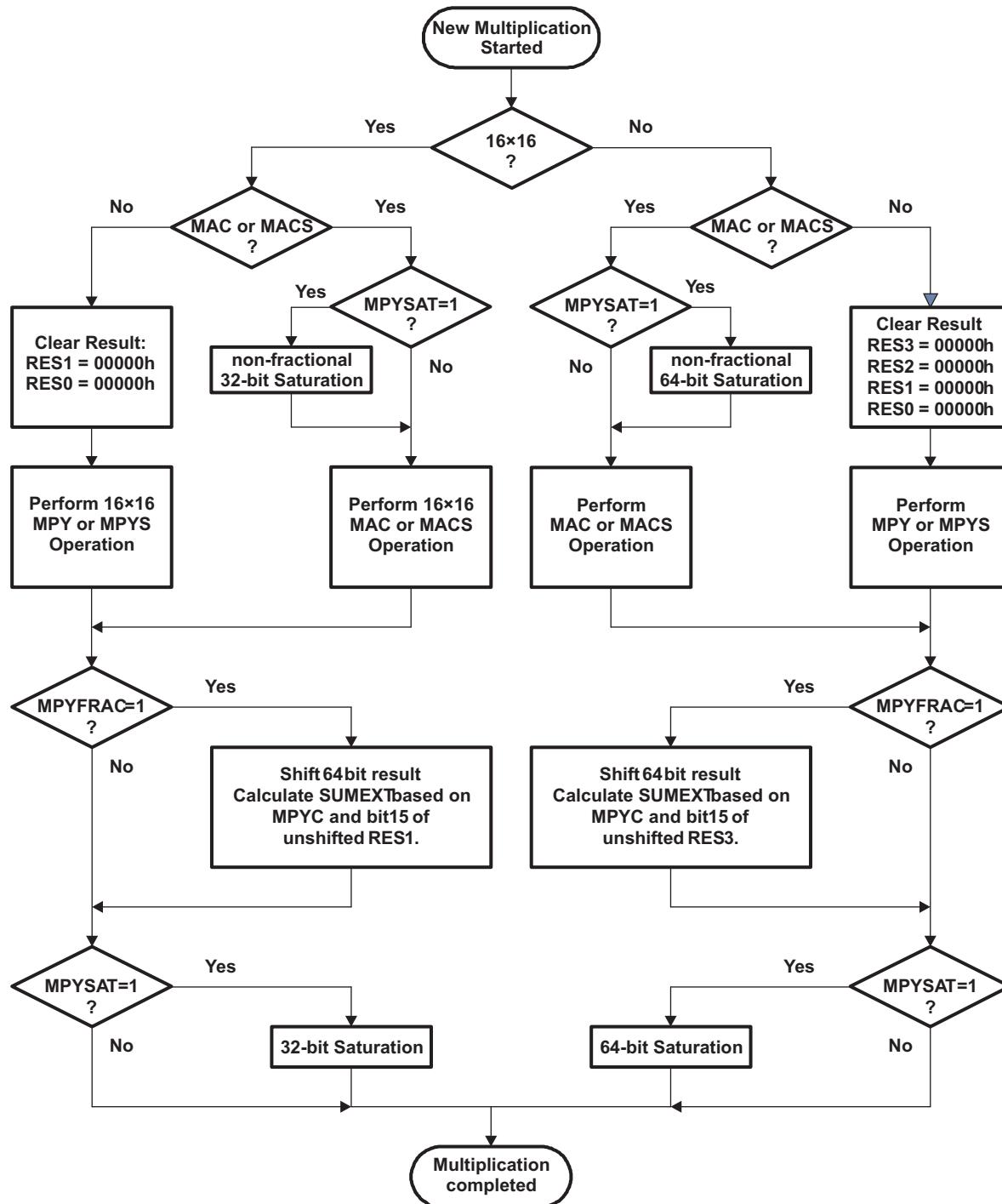


Figure 25-5. Multiplication Flow Chart

Given the separation in processing of 16-bit operations (32-bit results) and 32-bit operations (64-bit results) by the module, it is important to understand the implications when using MAC/MACS operations and mixing 16-bit operands or results with 32-bit operands or results. User software must address these points during use when mixing these operations. The following code illustrates the issue.

```
; Mixing 32x24 multiplication with 16x16 MACS operation
MOV      #MPYSAT,&MPY32CTL0    ; Saturation mode
MOV      #052C5h,&MPY32L      ; Load low word of 1st operand
MOV      #06153h,&MPY32H      ; Load high word of 1st operand
MOV      #001ABh,&OP2L       ; Load low word of 2nd operand
MOV.B   #023h,&OP2H_B       ; Load high word of 2nd operand
                  ;... 5 NOPs required
MOV      &RES0,R6           ; R6 = 00E97h
MOV      &RES1,R7           ; R7 = 0A6EAh
MOV      &RES2,R8           ; R8 = 04F06h
MOV      &RES3,R9           ; R9 = 0000Dh
                  ; Note that MPYC = 0!
MOV      #0CCC3h,&MACS       ; Signed MAC operation
MOV      #0FFB6h,&OP2        ; 16x16 bit operation
MOV      &RESLO,R6          ; R6 = 0FFFFh
MOV      &RESHI,R7          ; R7 = 07FFFh
```

The second operation gives a saturated result because the 32-bit value used for the 16×16-bit MACS operation was already saturated when the operation was started; the carry bit MPYC was 0 from the previous operation, but the MSB in result register RES1 is set. As one can see in the flow chart, the content of the result registers are saturated for multiply-and-accumulate operations after starting a new operation based on the previous results, but depending on the size of the result (32 bit or 64 bit) of the newly initiated operation.

The saturation before the multiplication can cause issues if the MPYC bit is not properly set as the following code shows.

```
;Pre-load result registers to demonstrate overflow
MOV      #0,&RES3          ;
MOV      #0,&RES2          ;
MOV      #0,&RES1          ;
MOV      #0,&RES0          ;
; Saturation mode and set MPYC:
MOV      #MPYSAT+MPYC,&MPY32CTL0
MOV.B   #082h,&MACS_B      ; 8-bit signed MAC operation
MOV.B   #04Fh,&OP2_B       ; Start 16x16 bit operation
MOV      &RES0,R6          ; R6 = 00000h
MOV      &RES1,R7          ; R7 = 08000h
```

Even though the result registers were loaded with all zeros, the final result is saturated. This is because the MPYC bit was set, causing the result used for the multiply-and-accumulate to be saturated to 08000 0000h. Adding a negative number to it would again cause an underflow, thus, the final result is also saturated to 08000 0000h.

### 25.2.6 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 25-1](#), 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 32x16 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
```

### 25.2.7 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
```

### 25.2.7.1 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
```

### 25.2.8 Using DMA

In devices with a DMA controller, the multiplier can trigger a transfer when the complete result is available. The DMA controller needs to start reading the result with MPY32RES0 successively up to MPY32RES3. Not all registers need to be read. The trigger timing is such that the DMA controller starts reading MPY32RES0 when its ready, and that the MPY32RES3 can be read exactly in the clock cycle when it is available to allow fastest access via DMA. The signal into the DMA controller is 'Multiplier ready' (see the DMA Controller chapter for details).

## 25.3 MPY32 Registers

MPY32 registers are listed in [Table 25-7](#). The base address can be found in the device-specific data sheet. The address offsets are listed in [Table 25-7](#).

---

**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 25-7. 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 25-7. MPY32 Registers (continued)**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Access</b>	<b>Reset</b>
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 25-8](#) are treated equally.

**Table 25-8. Alternative Registers**

<b>Register</b>	<b>Alternative 1</b>	<b>Alternative 2</b>
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

**25.3.1 MPY32CTL0 Register**

32-Bit Hardware Multiplier Control 0 Register

**Figure 25-6. MPY32CTL0 Register**

15	14	13	12	11	10	9	8
			Reserved				
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 25-9. 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	MPYDLYWRTE	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

The REF module is a general purpose reference system that is used to generate voltage references required for other subsystems available on a given device such as digital-to-analog converters, analog-to-digital converters, comparators, etc. This chapter describes the REF module.

Topic	Page
26.1 REF Introduction .....	686
26.2 Principle of Operation .....	688
26.3 REF Registers .....	692

## 26.1 REF Introduction

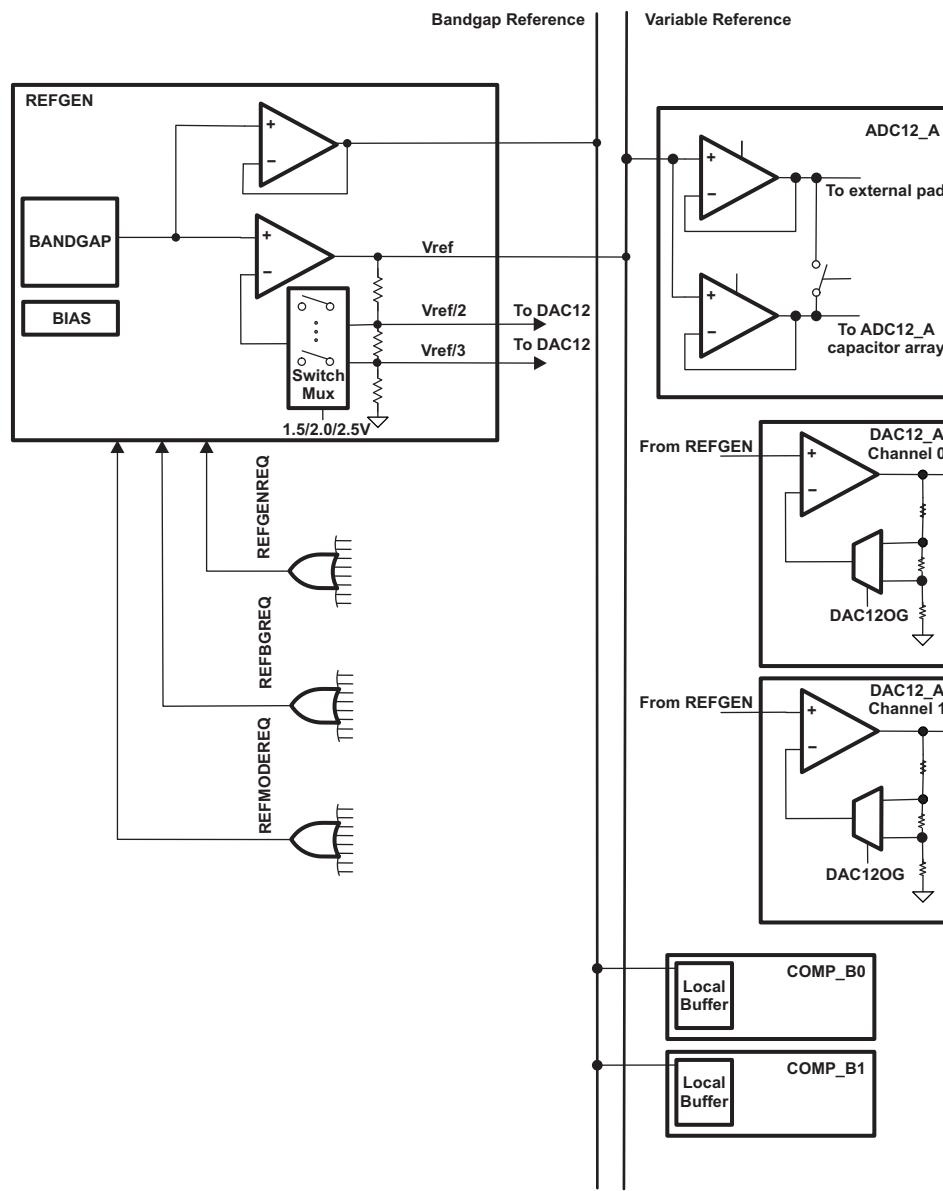
The reference module (REF) is responsible for generation of all critical reference voltages that can be used by various analog peripherals in a given device. These include, but are not necessarily limited to, the ADC10\_A, ADC12\_A, DAC12\_A, LCD\_B, and COMP\_B modules dependent upon the particular device. The heart of the reference system is the bandgap from which all other references are derived by unity or non-inverting gain stages. The REFGEN subsystem consists of the bandgap, the bandgap bias, and the non-inverting buffer stage which generates the three primary voltage reference available in the system, namely 1.5 V, 2.0 V, and 2.5 V. In addition, when enabled, a buffered bandgap voltage is also available.

Features of the REF include:

- Centralized, factory trimmed bandgap with excellent PSRR, temperature coefficient, and accuracy
- 1.5-V, 2.0-V, or 2.5-V user selectable internal references
- Buffered bandgap voltage available to rest of system
- Power saving features
- Backward compatibility to existing reference system

The block diagram of the REF module (example of a device with ADC12\_A) is shown in [Figure 26-1](#).

Devices with ADC10\_A might not include the reference voltage output to the external pad. Please refer to the device-specific data sheet.


**Figure 26-1. REF Block Diagram**

## 26.2 Principle of Operation

The REF module provides all the necessary voltage references to be used by various peripheral modules throughout the system. These may include, but are not limited to, devices that contain an ADC10\_A, ADC12\_A, DAC12\_A, LCD\_B, or COMP\_B.

The REFGEN subsystem contains a high-performance bandgap. This bandgap has very good accuracy (factory trimmed), low temperature coefficient, and high PSRR while operating at low power. The bandgap voltage is used to generate three voltages via a non-inverting amplifier stage, namely 1.5 V, 2.0 V, and 2.5 V. One voltage can be selected at a time. One output of the REFGEN subsystem is the variable reference line. The variable reference line provides either 1.5 V, 2.0 V, or 2.5 V to the rest of the system. A second output of the REFGEN subsystem provides a buffered bandgap reference line that can also be used by modules throughout the system. Additionally, the REFGEN supports voltage references required for the DAC12\_A module, when available. Lastly, the REFGEN subsystem also includes the temperature sensor circuitry, because it is derived from the bandgap. The temperature sensor is used by an ADC to measure a voltage proportional to temperature.

### 26.2.1 Low-Power Operation

The REF module is capable of supporting low-power applications such as LCD generation. Many of these applications do not require a very accurate reference, compared to data conversion, yet power is of prime concern. To support these kinds of applications, the bandgap is capable of being used in a sampled mode. In sampled mode, the bandgap circuitry is clocked via the VLO at an appropriate duty cycle. This reduces the average power of the bandgap circuitry significantly, at the cost of accuracy. When not in sampled mode, the bandgap is in static mode. Its power is at its highest, but so is its accuracy.

Modules automatically can request static mode or sampled mode via their own individual request lines. In this way, the particular module determines what mode is appropriate for its proper operation and performance. Any one active module that requests static mode causes all other modules to use static mode, even if another module is requesting sampled mode. In other words, static mode always has higher priority than sampled mode.

## 26.2.2 REFCTL

The REFCTL registers provide a way to control the reference system from one centralized set of registers. By default, REFCTL is used as the primary control of the reference system.

### 26.2.2.1 REFMSTR = 1

This mode is implemented in all devices with ADC10\_A. Also all ADC12\_A devices except for MSP430F5438 and MSP430F5438A support this mode.

Setting the reference master bit (REFMSTR = 1), allows the reference system to be controlled via the REFCTL register. This is the default setting.

Devices with ADC12\_A: In this mode (REFMSTR = 1), the legacy control bits inside the ADC register set (ADC12REFON, ADC12REF2\_5, ADC12TCOFF, and ADC12REFOUT) are do not care. The ADC12SR and ADC12REFBURST are still controlled via the ADC12\_A, because these are very specific to the ADC12\_A module. If REFMSTR is cleared, all settings in the REFCTL are do not care and the reference system is controlled completely by the legacy control bits inside the ADC12\_A module.

Devices with ADC10\_A: This is the only mode supported. REFMSTR must be set at all times. ADC10SR is controlled via the ADC10\_A, because these are very specific to the ADC10\_A module.

[Table 26-1](#) summarizes the REFCTL bits and their effect on the REF module.

**Table 26-1. REF Control of Reference System (REFMSTR = 1) (Default)**

REF Register Setting	Function
REFON	Setting this bit enables the REFGEN subsystem which includes the bandgap, the bandgap bias circuitry, and the 1.5-V, 2.0-V, 2.5-V buffer. Setting this bit causes the REFGEN subsystem to remain enabled regardless of whether or not any module has requested it. Clearing this bit disables the REFGEN subsystem only when there are no pending requests for REFGEN from any module. REFON must also be set to enable the temperature sensor when required.
REFVSEL	Selects 1.5 V, 2.0 V, or 2.5 V to be present on the variable reference line when REFON = 1 or REFGEN is requested by any module.
REFOUT	Setting this bit enables the variable reference line voltage to be present external to the device via a buffer (external reference buffer).
REFTCOFF	Setting this bit disables the temperature sensor (when available) to conserve power.

### 26.2.2.2 REFMSTR = 0

This setting is applicable to devices with ADC12\_A.

On legacy devices, the ADC12\_A provided the control bits necessary to configure the reference system, namely ADC12REFON, ADC12REF2\_5, ADC12TCOFF, ADC12REFOUT, ADC12SR, and ADC12REFBURST. The ADC12SR and ADC12REFBURST bits are very specific to the ADC12 operation and therefore are not included in REFCTL. All legacy control bits can still be used to configure the reference system allowing for backward compatibility by clearing REFMSTR. In this case, the REFCTL register bits are a 'do not care'.

Devices with ADC10\_A do not support this mode. REFMSTR bit must not be cleared.

[Table 26-2](#) summarizes the ADC12\_A control bits and their effect on the REF module. Please see the ADC12\_A module description for further details.

**NOTE:** Although the REF module supports using the ADC12\_A bits as control for the reference system, it is recommended that the use of the new REFCTL register be used and older code migrated to this methodology. This allows the logical partitioning of the reference system to be separate from the ADC12\_A system and forms a more natural partitioning for future products.

**Table 26-2. Control of Reference System (REFMSTR = 0, ADC12\_A only)**

ADC12_A Register Setting	Function
ADC12REFON	Setting this bit enables the REFGEN subsystem which includes the bandgap, the bandgap bias circuitry, and the 1.5-V, 2.0-V, 2.5-V buffer. Setting this bit causes the REFGEN subsystem to remain enabled regardless if any module has requested it. Clearing this bit disables the REFGEN subsystem only when there are no pending requests for REFGEN from all modules.
ADC12REF2_5	Setting this bits causes 2.5 V to be present on the variable reference line when ADC12REFON = 1. Clearing this bit causes 1.5 V to be present on the variable reference line when ADC12REFON = 1.
ADC12REFOUT	Setting this bits enables the variable reference line voltage to be present external to the device via a buffer (external reference buffer).
ADC12TCOFF	Setting this bit disables the temperature sensor to conserve power.

As stated previously, the ADC12REFBURST does have an effect on the reference system and can be controlled via the ADC12\_A. This bit is in effect regardless if REFCTL or the ADC12\_A is controlling the reference system. Setting ADC12REFBURST = 1 enables burst mode when REFON = 1 and REFMSTR = 1 or when ADC12REFON = 1 and REFMSTR = 0. In burst mode, the internal buffer (ADC12REFOUT = 0) or the external buffer (ADC12REFOUT = 1) is enabled only during a conversion and disabled automatically to conserve power.

---

**NOTE:** The legacy ADC12\_A bit ADC12REF2\_5 only allows for selecting either 1.5 V or 2.5 V. To select 2.0 V, the REFVSEL control bits must be used (REFMSTR = 1).

---

### 26.2.3 Reference System Requests

There are three basic reference system requests that are used by the reference system. Each module can utilize these requests to obtain the proper response from the reference system. The three basic requests are REFGENREQ, REFBGREQ, and REFMODEREQ. No interaction is required by the user code. The modules select the proper requests automatically.

A reference request signal, REFGENREQ, is available as an input into the REFGEN subsystem. This signal represents a logical OR of individual requests coming from the various modules in the system that require a voltage reference to be available on the variable reference line. When a module requires a voltage reference, it asserts its corresponding REFGENREQ signal. When the REFGENREQ is asserted, the REFGEN subsystem is enabled. After the specified settling time, the variable reference line voltage is stable and ready for use. The REFVSEL settings determine which voltage is generated on the variable reference line.

In addition to the REFGENREQ, a second reference request signal, REFBGREQ is available. The REFBGREQ signal represents a logical OR of requests coming from the various modules that require the bandgap reference line. When the REFBGREQ is asserted, the bandgap, along with its bias circuitry and local buffer, is enabled if it is not already enabled by a prior request.

The REFMODEREQ request signal is available that configures the bandgap and its bias circuitry to operate in a sampled or static mode of operation. The REFMODEREQ signal basically represents a logical AND of individual requests coming from the various analog modules. In reality, a REFMODEREQ occurs only if a module's REFGENREQ or REFBGQ is also asserted, otherwise it is a do not care. When REFMODEREQ = 1, the bandgap operates in sampled mode. When a module asserts its corresponding REFMODEREQ signal, it is requesting that the bandgap operate in sampled mode. Because REMODEREQ is a logical AND of all individual requests, any modules requesting static mode cause the bandgap to operate in static mode. The BGMODE bit can be used as an indicator of static or sampled mode of operation.

#### 26.2.3.1 REFBGACT, REFGENACT, REFGENBUSY

Any module that is using the variable reference line causes REFGENACT to be set inside the REFCTL register. This bit is read only and indicates to the user that the REFGEN is active or off. Similarly, the REFBGACT is active any time one or more modules is actively utilizing the bandgap reference line and indicates to the user that the REFBG is active or off.

The REFGENBUSY signal, when asserted, indicates that a module is using the reference and cannot have any of its settings changed. For example, during an active ADC12\_A conversion, the reference voltage level should not be changed. REFGENBUSY is asserted when there is an active ADC12\_A conversion ( $ENC = 1$ ) or when the DAC12\_A is actively converting ( $DAC12AMPx > 1$  and  $DAC12SREFx = 0$ ). REFGENBUSY when asserted, write protects the REFCTL register. This prevents the reference from being disabled or its level changed during any active conversion. Please note that there is no such protection for the DAC12\_A if the ADC12\_A legacy control bits are used for the reference control. If the user changes the ADC12\_A settings and the DAC12\_A is using the reference, the DAC12\_A conversion is affected.

#### 26.2.3.2 ADC10\_A

For devices that contain an ADC10\_A module, the ADC10\_A module contains only one local buffer. This buffer is required when using the internal reference voltage and must be enabled and stable prior to a conversion.

In devices without a reference output buffer REFOUT must be written 0. Please refer to the device-specific data sheet.

In devices with ADC10\_A the REFMSTR bit must be set at all times.

In devices with ADC10\_A the REFON bit must be set if the internal reference voltage is used.

#### 26.2.3.3 ADC12\_A

For devices that contain an ADC12\_A module, the ADC12\_A module contains two local buffers. The larger buffer can be used to drive the reference voltage, present on the variable reference line, external to the device. This buffer has larger power consumption due to a selectable burst mode, as well as, its need to drive larger DC loads that may be present outside the device. The large buffer is enabled continuously when  $REFON = 1$ ,  $REFOUT = 1$ , and  $ADC12REFBURST = 0$ . When  $ADC12REFBURST = 1$ , the buffer is enabled only during an ADC conversion, shutting down automatically upon completion of a conversion to save power. In addition, when  $REFON = 1$  and  $REFOUT = 1$ , the second smaller buffer is automatically disabled. In this case, the output of the large buffer is connected to the capacitor array via an internal analog switch. This ensures the same reference is used throughout the system. If  $REFON = 1$  and  $REFOUT = 0$ , the internal buffer is used for ADC conversion and the large buffer remains disabled. The small internal buffer can operate in burst mode as well by setting  $ADC12REFBURST = 1$ .

#### 26.2.3.4 DAC12\_A

Some devices may contain a DAC12\_A module. The DAC12\_A can use the 1.5 V, 2.0 V, or 2.5 V from the variable reference line for its reference. The DAC12\_A can request its reference directly by the settings within the DAC12\_A module itself. Therefore, if the DAC is enabled and the internal reference is selected, the DAC requests the reference voltage from the REF module. In addition, as before, setting  $REFON = 1$  ( $REFMSTR = 1$ ) or  $ADC12REFON = 1$  ( $REFMSTR = 0$ ) can enable the variable reference line independent of the DAC12\_A control bits.

The REGEN subsystem provides divided versions of the variable reference line for use in the DAC12\_A module. The DAC12\_A module requires either /2 or /3 of the variable reference. The selection of these depends on the control bits inside the DAC12\_A module (DAC12IR, DAC12OG) and is handled automatically by the REF module.

When the DAC12\_A selects AVcc or VeREF+ as its reference, the DAC12\_A has its own /2 and /3 resistor string available that scales the input reference appropriately based on the DAC12IR and DAC12OG settings.

#### 26.2.3.5 LCD\_B

Devices that contain an LCD use the LCD\_B module. The LCD\_B module requires a reference to generate the proper LCD voltages. The bandgap reference line from the REFGEN subsystem is used for this purpose. The LCD is enabled when  $LCDON = 1$  of the LCD\_B module. This causes a REFBGREQ from the LCD module to be asserted. The buffered bandgap is made available on the bandgap reference line for use inside the LCD\_B module.

## 26.3 REF Registers

The REF registers are listed in [Table 26-3](#). The base address can be found in the device specific datasheet. The address offset is listed in [Table 26-3](#).

---

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

---

**Table 26-3. REF Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	REFCTL0	REFCTL0	Read/write	Word	0080h	<a href="#">Section 26.3.1</a>
00h		REFCTL0_L	Read/write	Byte	80h	
01h		REFCTL0_H	Read/write	Byte	00h	

### 26.3.1 REFCTL0 Register (offset = 00h) [reset = 0080h]

REF Control Register 0

**Figure 26-2. REFCTL0 Register**

15	14	13	12	11	10	9	8
Reserved				BGMODE	REFGENBUSY	REFBGACT	REFGENACT
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r-(0)
7	6	5	4	3	2	1	0
REFMSTR	Reserved	REFVSEL		REFTCOFF	Reserved	REFOUT	REFON
rw-(1)	r0	rw-(0)	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)

Can be modified only when REFGENBUSY = 0.

**Table 26-4. REFCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11	BGMODE	R	0h	Bandgap mode. Read only. 0b = Static mode 1b = Sampled mode
10	REFGENBUSY	R	0h	Reference generator busy. Read only. 0b = Reference generator not busy 1b = Reference generator busy
9	REFBGACT	R	0h	Reference bandgap active. Read only. 0b = Reference bandgap buffer not active 1b = Reference bandgap buffer active
8	REFGENACT	R	0h	Reference generator active. Read only. 0b = Reference generator not active 1b = Reference generator active
7	REFMSTR	RW	0h	REF master control. ADC10_A devices: Must be written 1. 0b = Reference system controlled by legacy control bits inside the ADC12_A module when available. 1b = Reference system controlled by REFCTL register. Common settings inside the ADC12_A module (if exists) are do not care.
6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	REFVSEL	RW	0h	Reference voltage level select 00b = 1.5 V available when reference requested or REFON = 1 01b = 2.0 V available when reference requested or REFON = 1 10b = 2.5 V available when reference requested or REFON = 1 11b = 2.5 V available when reference requested or REFON = 1
3	REFTCOFF	RW	0h	Temperature sensor disabled 0b = Temperature sensor enabled 1b = Temperature sensor disabled to save power
2	Reserved	R	0h	Reserved. Always reads as 0.
1	REFOUT	RW	0h	Reference output buffer. ADC10_A devices without reference output buffer: Must be written 0. 0b = Reference output not available externally. 1b = Reference output available externally. If ADC12REFBURST = 0, or DAC12_A is enabled, output is available continuously. If ADC12REFBURST = 1, output is available only during an ADC12_A conversion.

**Table 26-4. REFCTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
0	REFON	RW	0h	<p>Reference enable.</p> <p>ADC10_A: The ADC10_A does not support the reference request. REFON must be set if the internal reference voltage is used.</p> <p>0b = Disables reference if no other reference requests are pending.</p> <p>1b = Enables reference.</p>

## **ADC10\_A**

The ADC10\_A module is a high-performance 10-bit analog-to-digital converter (ADC). This chapter describes the operation of the ADC10\_A module.

<b>Topic</b>	<b>Page</b>
<b>27.1 ADC10_A Introduction .....</b>	<b>696</b>
<b>27.2 ADC10_A Operation .....</b>	<b>698</b>
<b>27.3 ADC10_A Registers .....</b>	<b>710</b>

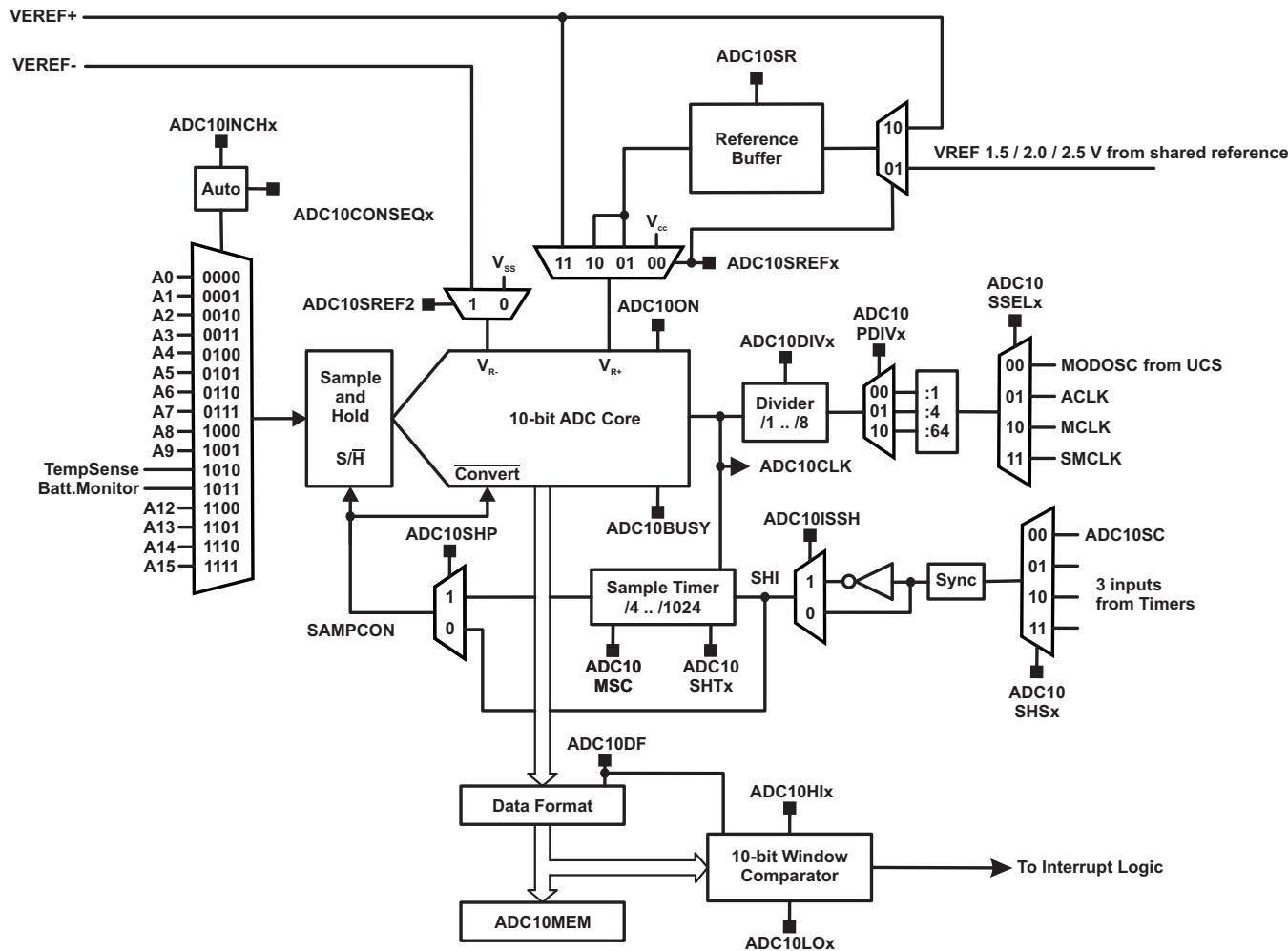
## 27.1 ADC10\_A Introduction

The ADC10\_A module supports fast 10-bit analog-to-digital conversions. The module implements a 10-bit SAR core with sample select control and a window comparator.

ADC10\_A features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 10-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 using the REF module or external reference
- 12 individually configurable external input channels
- Conversion channel for temperature sensor of the REF module
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence (autoscan), and repeat-sequence (repeated autoscan) conversion modes
- Window comparator for low power monitoring of input signals
- Interrupt vector register for fast decoding of six ADC Interrupts (ADC10IFG0, ADC10TOVIFG, ADC10OVIFG, ADC10LOIFG, ADC10INIFG, ADC10HIIFG)

The block diagram of ADC10\_A is shown in [Figure 27-1](#). The on-chip reference voltage generation is located in the reference module (see the device-specific data sheet).



A The MODOSC is part of the UCS. See the [UCS chapter](#) for more information.

B When using ADC10SHP = 0 no synchronisation of the trigger input is done.

**Figure 27-1. ADC10\_A Block Diagram**

## 27.2 ADC10\_A Operation

The ADC10\_A module is configured with user software. The setup and operation of the ADC10\_A is discussed in the following sections.

### 27.2.1 10-Bit ADC Core

The ADC core converts an analog input to its 10-bit digital representation and stores the result in the conversion register ADC10MEM0. The core uses two programmable/selectable voltage levels ( $V_{R+}$  and  $V_{R-}$ ) to define the upper and lower limits of the conversion. The digital output ( $N_{ADC}$ ) is full scale (03FFh) when the input signal is equal to or higher than  $V_{R+}$ , and zero when the input signal is equal to or lower than  $V_{R-}$ . The input channel and the reference voltage levels ( $V_{R+}$  and  $V_{R-}$ ) are defined in the conversion-control memory. The conversion formula for the ADC result  $N_{ADC}$  is shown in [Equation 11](#):

$$N_{ADC} = 1023 \times \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \quad (11)$$

The ADC10\_A core is configured by the control registers ADC10CTL0, ADC10CTL1 and ADC10CTL2. The core is enabled with the ADC10ON bit. The ADC10\_A can be turned off when not in use to save power. With few exceptions, the ADC10\_A control bits can only be modified when ADC10ENC = 0. ADC10ENC must be set to 1 before any conversion can take place.

#### 27.2.1.1 Conversion Clock Selection

The ADC10CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC10\_A source clock is selected using the ADC10SELx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK, and the MODOSC. The input clock can be divided from 1–512 using both the ADC10DIVx bits and the ADC10PDIVx bits.

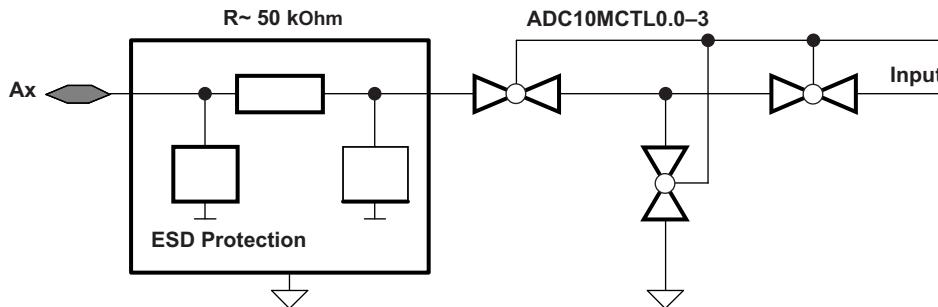
MODOSC, generated internally in the UCS, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific data sheet for the MODOSC specification.

The user must ensure that the clock chosen for ADC10CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation does not complete and any result is invalid.

### 27.2.2 ADC10\_A Inputs and Multiplexer

The 14 external and 2 internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching (see [Figure 27-2](#)). The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground (AV<sub>SS</sub>), so that the stray capacitance is grounded to eliminate crosstalk.

The ADC10\_A uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.



**Figure 27-2. Analog Multiplexer**

### 27.2.2.1 Analog Port Selection

The ADC10\_A inputs are multiplexed with digital port pins. When analog signals are applied to digital gates, parasitic current can flow from V<sub>CC</sub> 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 provide the ability to disable the port pin input and output buffers.

```
; Py.0 and Py.1 configured for analog input  
BIS.B #3h,&PySEL      ; Py.1 and Py.0 ADC10_A function
```

### 27.2.3 Voltage Reference Generator

The ADC10\_A module is designed to be used either with the on-chip reference supplied by the REF module or an externally reference voltage supplied on external pins.

The on-chip reference is capable of supplying 1.5 V, 2.0 V, and 2.5 V. The reference voltages are controlled by the control registers of the REF module (see the [REF chapter](#) for details). The internal V<sub>CC</sub> can also be used as the voltage reference.

External reference voltages may be supplied for V<sub>R+</sub> and V<sub>R-</sub> through pins VEREF+ and VEREF-, respectively.

#### 27.2.3.1 Internal Reference Low-Power Features

The on-chip reference is designed for low-power applications. This reference includes a band-gap voltage source and a separate reference buffer both located in the REF module. The current consumption of each is specified separately in the device-specific data sheet. The ADC10\_A also contains an internal buffer for reference voltages. This buffer is automatically enabled when the internal reference is selected for VREF+, but it is also optionally available for VEREF+. The on-chip reference from the REF module must be enabled by software. Its settling time is typical 25  $\mu$ s. See the device-specific data sheet and the [REF chapter](#) for further information on the on-chip reference.

The reference buffer of the ADC10\_A also has selectable speed versus power settings. When the maximum conversion rate is below 50 ksps, setting ADC10SR = 1 reduces the current consumption of the buffer approximately 50%.

### 27.2.4 Auto Power Down

The ADC10\_A is designed for low-power applications. When the ADC10\_A is not actively converting, the core is automatically disabled and automatically reenabled when needed. The MODOSC is also automatically enabled when needed and disabled when not needed.

### 27.2.5 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 ADC10SHSx bits and includes the following:

- ADC10SC bit and
- Three timer outputs

The polarity of the SHI signal source can be inverted with the ADC10ISSH 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 12 ADC10CLK cycles in 10-bit resolution mode. One additional ADC10CLK is used for the window comparator. Two different sample-timing methods are defined by control bit ADC10SHP, extended sample mode, and pulse mode.

### 27.2.5.1 Extended Sample Mode

The extended sample mode is selected when ADC10SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period  $t_{\text{sample}}$ . When SAMPCON is high, sampling is active. The high-to-low SAMPON transition starts the conversion after synchronization with ADC10CLK (see Figure 27-3).

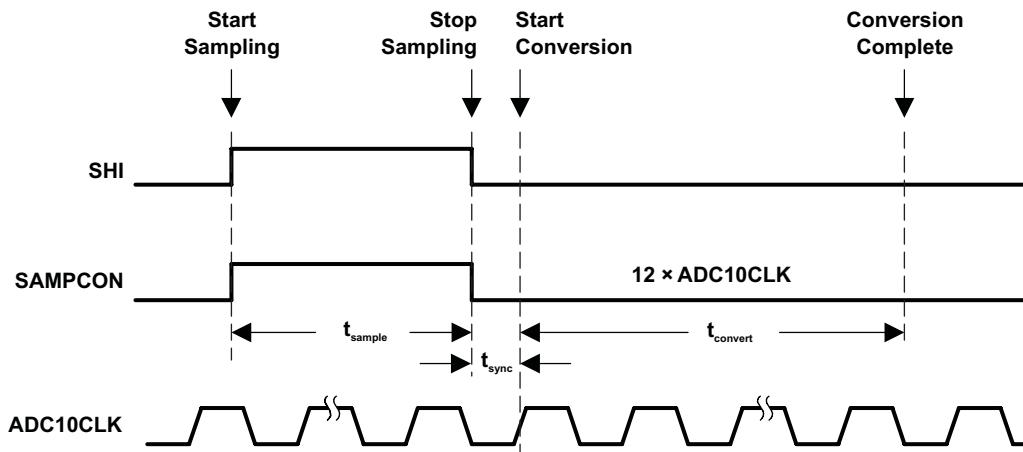


Figure 27-3. Extended Sample Mode

### 27.2.5.2 Pulse Sample Mode

The pulse sample mode is selected when ADC10SHP = 1. The SHI signal is used to trigger the sampling timer. The ADC10SHTx bits in ADC10CTL0 control the interval of the sampling timer that defines the SAMPON sample period  $t_{\text{sample}}$ . The sampling timer keeps SAMPON high after synchronization with AD10CLK for a programmed interval  $t_{\text{sample}}$ . The total sampling time is  $t_{\text{sample}}$  plus  $t_{\text{sync}}$  (see Figure 27-4).

The ADC10SHTx bits select the sampling time in 4x multiples of ADC10CLK.

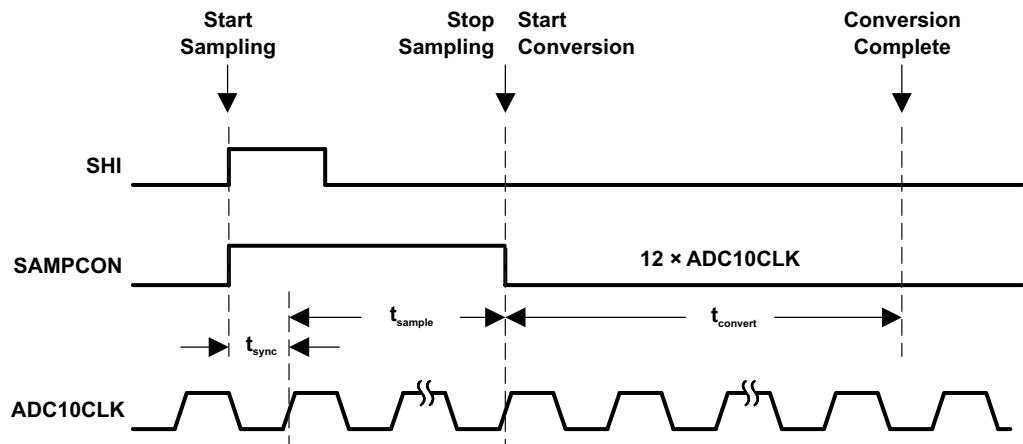
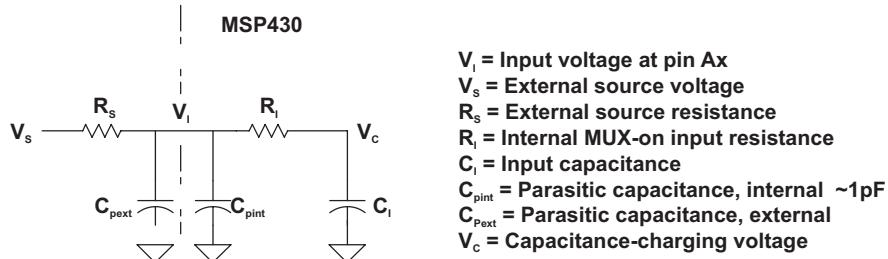


Figure 27-4. Pulse Sample Mode

### 27.2.5.3 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  $t_{\text{sample}}$  (see [Figure 27-5](#)). An internal MUX-on input resistance  $R_i$  (see device specific datasheet) in series with capacitor  $C_i$  (see device specific datasheet) 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 conversion.



**Figure 27-5. Analog Input Equivalent Circuit**

The resistance of the source  $R_s$  and  $R_i$  affect  $t_{\text{sample}}$ . The minimum sample time must not be violated. Violation of the minimum sample time may cause a conversion not to take place. See the device specific datasheet for the  $t_{\text{sample}}$  limits.

### 27.2.6 Conversion Result

The conversion result is accessible using the ADC10MEM0 register independently of the conversion mode selected by the user. When a conversion result is written to ADC10MEM0, the ADC10IFG0 is set.

### 27.2.7 ADC10\_A Conversion Modes

The ADC10\_A has four operating modes selected by the CONSEQx bits as listed in [Table 27-1](#).

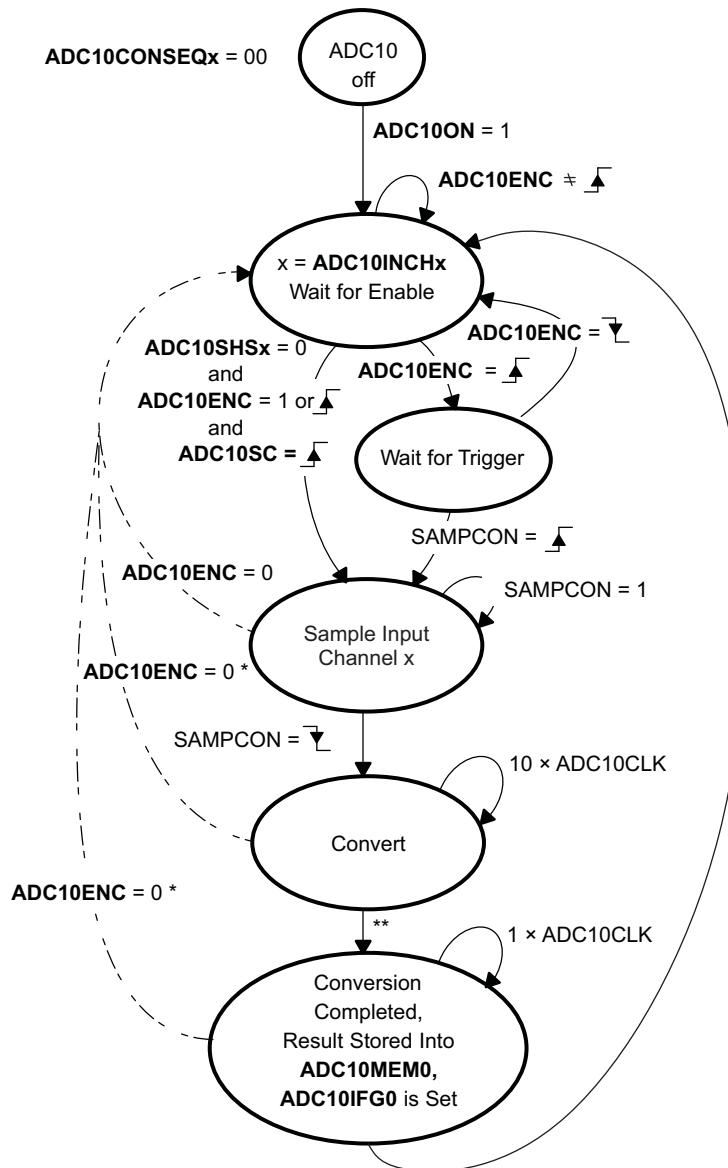
**Table 27-1. Conversion Mode Summary**

ADC10CONSEQx	Mode	Operation
00	Single-channel single-conversion	A single channel is converted once.
01	Sequence-of-channels (autoscan)	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels (repeated autoscan)	A sequence of channels is converted repeatedly.

### 27.2.7.1 Single-Channel Single-Conversion Mode

A single channel selected by ADC10INCH $x$  is sampled and converted once. The ADC result is written to ADC10MEM0. Figure 27-6 shows the flow of the single-channel single-conversion mode. When ADC10SC triggers a conversion, successive conversions can be triggered by the ADC10SC bit. When any other trigger source is used, ADC10ENC must be toggled between each conversion.

Resetting ADC10ON bit within a conversion causes the ADC10\_A to go back into "ADC10 off" state. In this case, the value of the conversion register and the value of the interrupt flags are unpredictable.



\* Conversion result is unpredictable

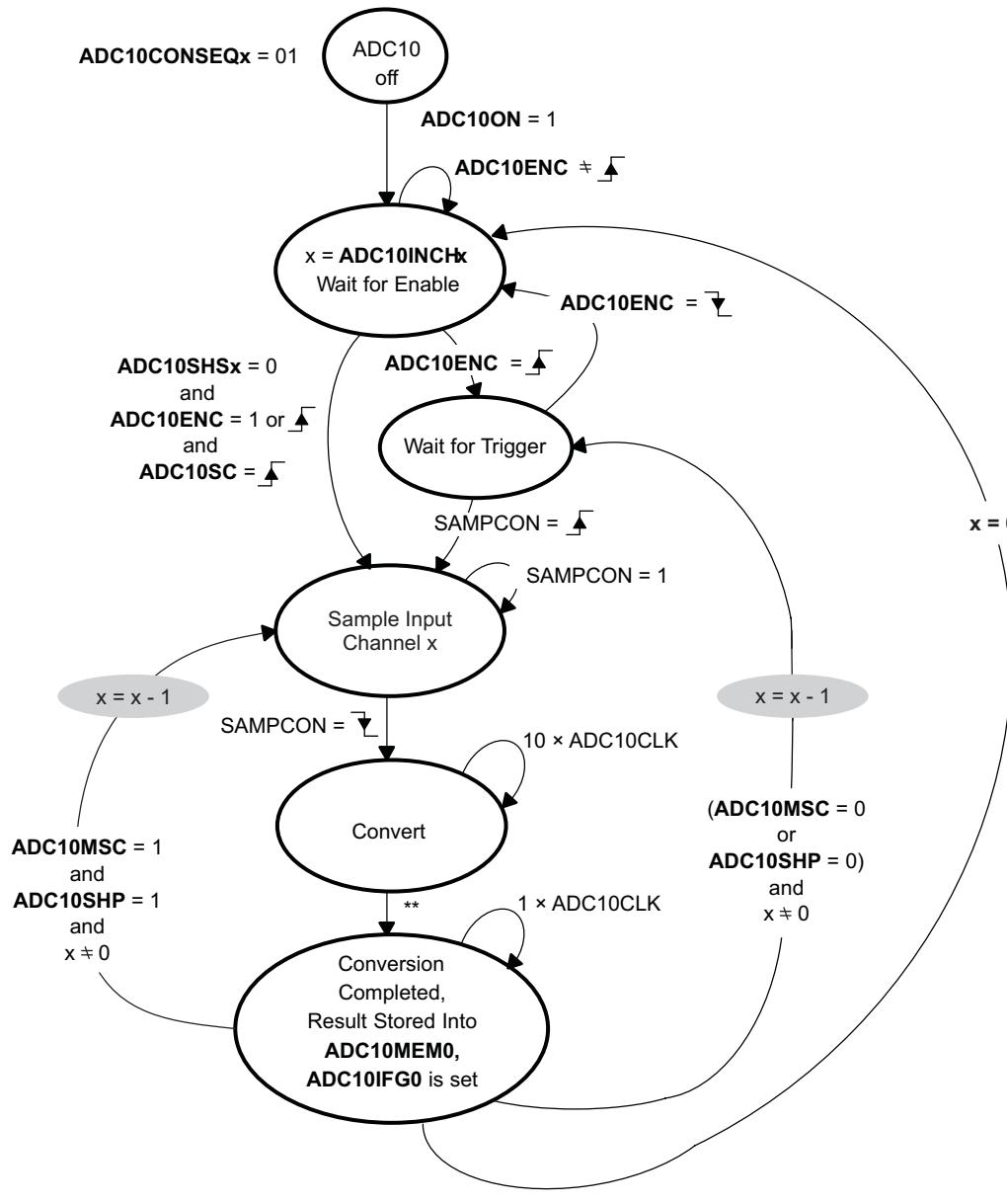
\*\* Two ADC10CLK cycles needed

All bit- or registernames are marked with bold font, signals are noted in normal font

Figure 27-6. Single-Channel Single-Conversion Mode

### 27.2.7.2 Sequence-of-Channels Mode (Autoscan Mode)

In sequence-of-channels mode, also referred to as autoscan mode, a sequence of channels is sampled and converted once. The sequence begins with the channel selected by the ADC10INCH $x$  bits and decrements to channel A0. Each ADC result is written to ADC10MEM0. The sequence stops after conversion of channel A0. Figure 27-7 shows the sequence-of-channels mode. When ADC10SC triggers a sequence, successive sequences can be triggered by the ADC10SC bit. When any other trigger source is used, ADC10ENC must be toggled between each sequence. As in all conversion modes resetting ADC10ON bit within a conversion causes the ADC10\_A to go back into "ADC10 off" state.



x - input channel Ax

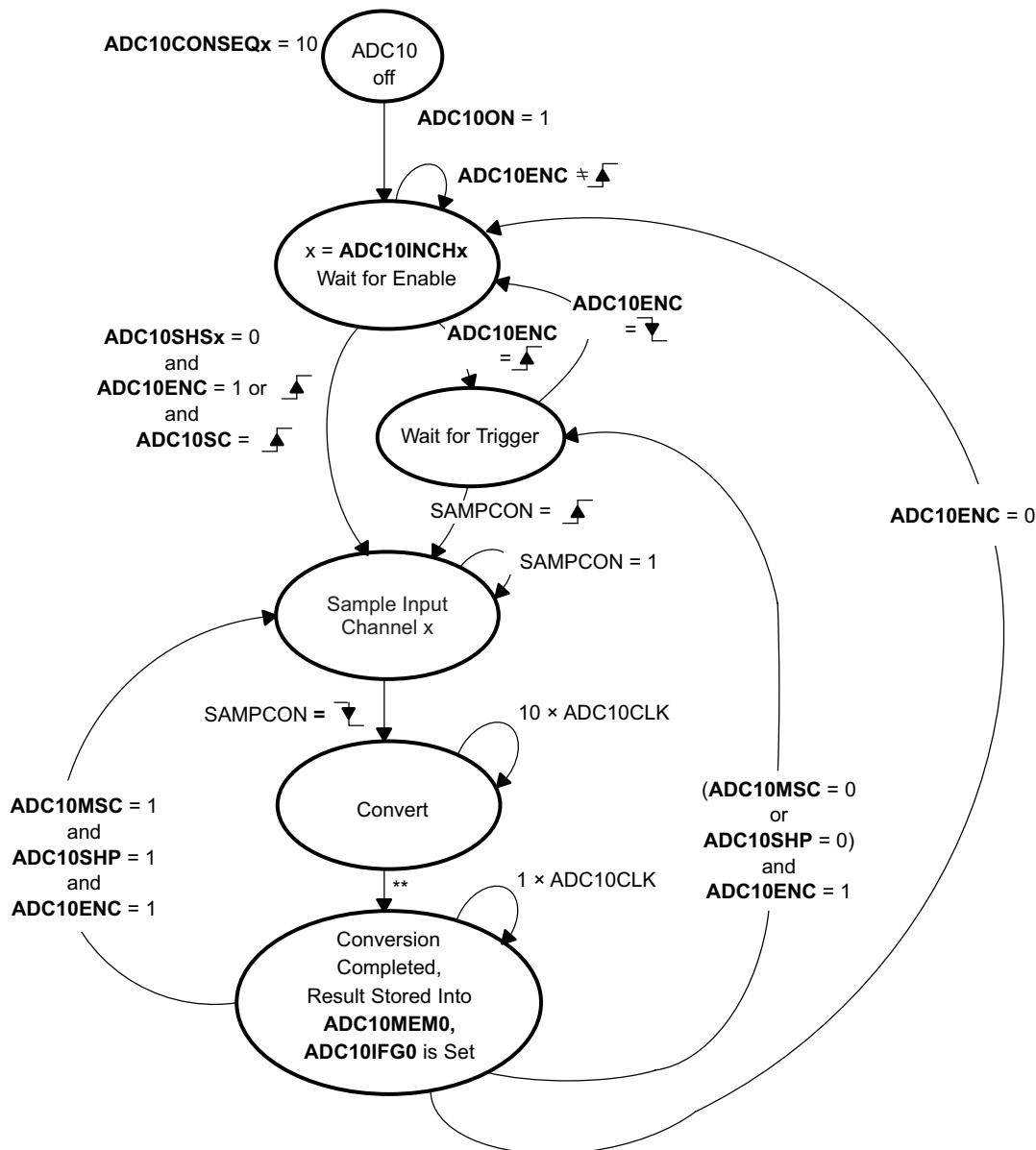
\*\* Two ADC10CLK cycles needed

All bit- or registernames are marked with bold font, signals are noted in normal font

**Figure 27-7. Sequence-of-Channels Mode**

### 27.2.7.3 Repeat-Single-Channel Mode

A single channel selected by ADC10INCH $x$  is sampled and converted continuously. Each ADC result is written to ADC10MEM0. Figure 27-8 shows the repeat-single-channel mode.



$x$  - pointer to the selected ADC10\_A channel defined by **ADC10INCH $x$**

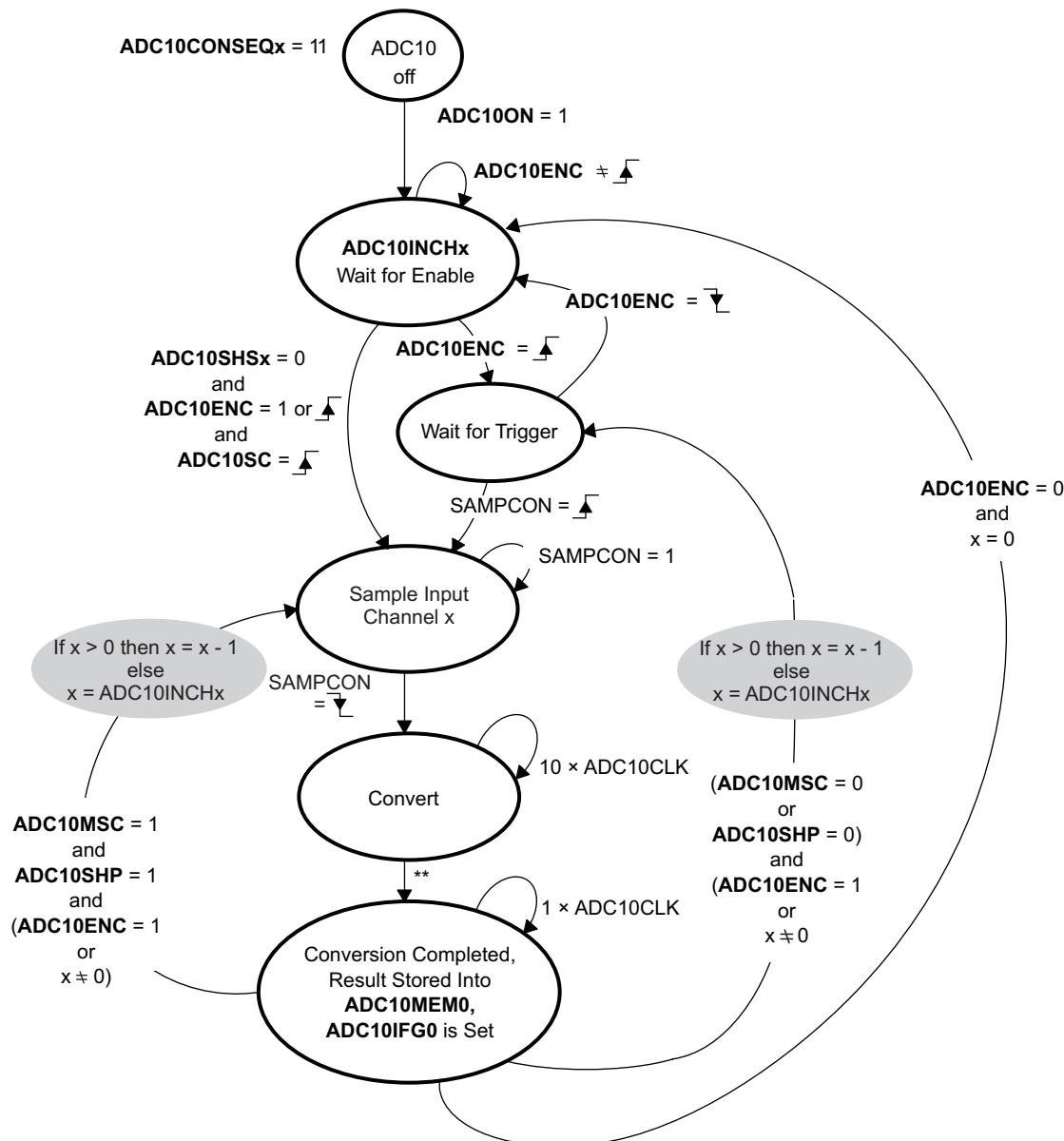
\*\* Two ADC10CLK cycles needed

All bit- or registernames are marked with bold font, signals are noted in normal font

Figure 27-8. Repeat-Single-Channel Mode

#### 27.2.7.4 Repeat-Sequence-of-Channels Mode (Repeated Autoscan Mode)

In this mode, a sequence of channels is sampled and converted repeatedly. This mode is also referred to as repeated autoscan mode. The sequence begins with the channel selected by ADC10INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM0. The sequence ends after conversion of channel A0, and the next trigger signal re-starts the sequence. [Figure 27-9](#) shows the repeat-sequence-of-channels mode.



**Figure 27-9. Repeat-Sequence-of-Channels Mode**

### 27.2.7.5 Using the Multiple Sample and Convert (ADC10MSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When ADC10MSC = 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 ADC10ENC bit is toggled in repeat-single-channel or repeated-sequence modes. The function of the ADC10ENC bit is unchanged when using the ADC10MSC bit.

### 27.2.7.6 Stopping Conversions

Stopping ADC10\_A activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ADC10ENC 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 ADC10ENC.
- Resetting ADC10ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ADC10ENC during a sequence or repeat-sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the CONSEQx = 0 and resetting the ADC10ENC bit. Conversion data are unreliable.

### 27.2.8 Window Comparator

The window comparator allows to monitor analog signals without any CPU interaction. In the following list one can find the available Interrupt flags and the conditions, when they are asserted:

- The ADC10LO-Interrupt flag (ADC10LOIFG) gets set if the current result of the ADC10\_A conversion is below the low threshold defined in register ADC10LO
- The ADC10HI-Interrupt flag (ADC10HIIFG) gets set if the current result of the ADC10\_A conversion is greater than the high threshold defined in register ADC10HI
- The ADC10IN-Interrupt flag (ADC10INIFG) gets set if the current result of the ADC10\_A conversion is greater than the low threshold defined in register ADC10LO and less than the high threshold defined in ADC10HI

These Interrupts are generated independently of the conversion mode selected by the user. The update of the window comparator interrupt-flags happens in parallel to the ADC10IFG0.

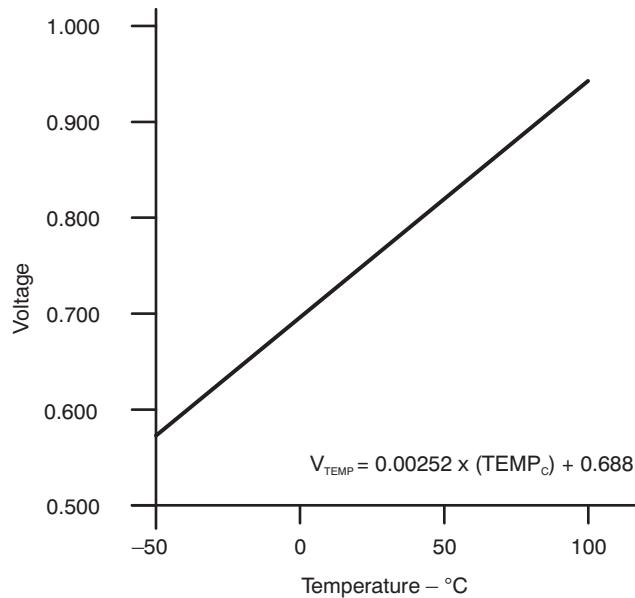
The user always needs to ensure, that the values in the ADC10HI and ADC10LO registers are in the correct data format. If for example the binary data format is selected (ADC10DF = 0), then the thresholds in the threshold registers ADC10HI and ADC10LO also need to be entered binary coded. Changing the ADC10DF or the ADC10RES resets the threshold registers.

The interrupt flags need to be reset by the user software. The ADC10\_A only updates the flags each time a new value is available in the ADC10MEM0. This update is only a set of the corresponding interrupt flag. When the user uses the window comparator flags, it must be ensured that they are reset by software according to the application needs.

### 27.2.9 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel ADC10INCHx = 1010. Any other configuration is done as if an external channel was selected, including reference selection, conversion-mode selection, etc. The temperature sensor is located in the REF module of the device is configured by using the REF module's control registers.

The typical temperature sensor transfer function is shown in [Figure 27-10](#). When using the temperature sensor, the sample period must be greater than 30  $\mu$ s. The temperature sensor offset error can be large and may need to be calibrated for most applications (see the device-specific data sheet for parameters).



**Figure 27-10. Typical Temperature Sensor Transfer Function**

### 27.2.10 ADC10\_A Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

Ground loops are formed when return current from the A/D flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small, unwanted offset voltages that can add to or subtract from the reference or input voltages of the ADC. The connections shown in Figure 27-11 prevent this.

In addition to grounding, ripple and noise spikes on the power-supply lines due to digital switching or switching power supplies can corrupt the conversion result. A noise-free design using separate analog and digital ground planes with a single-point connection is recommended to achieve high accuracy.

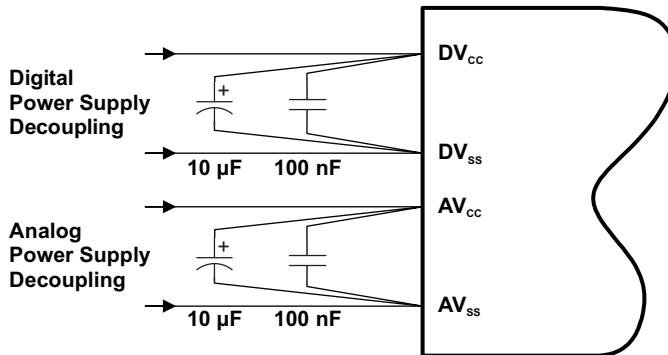


Figure 27-11. ADC10\_A Grounding and Noise Considerations

### 27.2.11 ADC10\_A Interrupts

The ADC10\_A has 6 Interrupt sources:

- ADC10IFG0 : conversion ready Interrupt
- ADC10OVIFG : ADC10MEM0 overflow
- ADC10TOVIFG : ADC10\_A conversion time overflow
- ADC10LOIFG, ADC10NIFG, ADC10HIIFG : window comparator Interrupt flags

The ADC10IFG0 bit is set when the ADC10MEM0 memory register is loaded with the conversion result. An Interrupt request is generated if ADC10IE0 bit and the GIE bit are set. The ADC10OV condition occurs when a conversion result is written to the ADC10MEM0 before its previous conversion result was read. The ADC10TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed. The DMA is triggered after each conversion.

The window comparator Interrupt flags are set corresponding to the description in the Window Comparator section (see [Section 27.2.8](#)).

#### 27.2.11.1 ADC10IV, Interrupt Vector Generator

All ADC10\_A Interrupt sources are prioritized and combined to source a single Interrupt vector. The Interrupt vector register ADC10IV is used to determine which enabled ADC10\_A Interrupt source requested an Interrupt.

The highest-priority enabled ADC10\_A Interrupt generates a number in the ADC10IV register (see register description). This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled ADC10\_A Interrupts do not affect the ADC10IV value.

Read access of the ADC10IV register automatically resets the highest-pending Interrupt condition and flag. Only the ADC10IFG0 is not reset by this ADC10IV read access. ADC10IFG0 is automatically reset by reading the ADC10MEM0 register or may be reset with software.

Write access of the ADC10IV 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 ADC10OV, ADC10HIIFG and ADC10IFG0 Interrupts are pending when the Interrupt service routine accesses the ADC10IV register, the highest priority interrupt (ADC10OV Interrupt condition) is reset automatically. After the RETI instruction of the Interrupt service routine is executed, the ADC10HIIFG generates another Interrupt.

### 27.2.11.2 ADC10\_A Interrupt Handling Software Example

The following software example shows the recommended use of the ADC10IV. The ADC10IV value is added to the PC to automatically jump to the appropriate routine.

- ADC10IFG0, ADC10TOV, and ADC10OV: 16 cycles

```
; Interrupt handler for ADC10_A.  
INT_ADC10_A                                ; Enter Interrupt Service Routine  
    ADD    &ADC10IV, PC                      ; Add offset to PC  
    RETI                                         ; Vector 0: No Interrupt  
    JMP    ADOV                                ; Vector 2: ADC10_A overflow  
    JMP    ADTOV                               ; Vector 4: ADC10_A timing overflow  
    JMP    ADHI                                ; Vector 6: ADC10_A window comparator high  
    Interrupt  
        JMP    ADLO                                ; Vector 8: ADC10_A window comparator low  
    Interrupt  
        JMP    ADIN                                ; Vector 10: ADC10_A window comparator in  
    Interrupt  
;  
; Handler for ADC10IFG0 starts here. No JMP required.  
;  
ADMEM  MOV  &ADC10MEM0,xxx                  ; Move result, flag is reset  
      ...  
      RETI                                         ; Other instruction needed?  
ADOV   ...  
      RETI                                         ; Return ;  
      ; Handle ADCMEM0 overflow  
ADTOV  ...  
      RETI                                         ; Return ;  
      ; Handle Conv. time overflow  
ADHI   ...  
      RETI                                         ; Return ;  
      ; Handle window comparator high Interrupt  
ADLO   ...  
      RETI                                         ; Return ;  
      ; Handle window comparator low Interrupt  
ADIN   ...  
      RETI                                         ; Return ;  
      ; Handle window comparator in window Interrupt  
      ; Return
```

## 27.3 ADC10\_A Registers

The ADC10\_A registers are listed in [Table 27-2](#). The base address of the ADC10\_A can be found in the device-specific data sheet. The address offset of each ADC10\_A register is given in [Table 27-2](#).

**Table 27-2. ADC10\_A Registers**

Offset	Acronym	Register Name	Type	Reset	Section
00h	ADC10CTL0	ADC10_A Control 0 register	Read/write	0000h	<a href="#">Section 27.3.1</a>
02h	ADC10CTL1	ADC10_A Control 1 register	Read/write	0000h	<a href="#">Section 27.3.2</a>
04h	ADC10CTL2	ADC10_A Control 2 register	Read/write	1000h	<a href="#">Section 27.3.3</a>
06h	ADC10LO	ADC10_A Window Comparator Low Threshold register	Read/write	0000h	<a href="#">Section 27.3.9</a>
08h	ADC10HI	ADC10_A Window Comparator High Threshold register	Read/write	FF03h	<a href="#">Section 27.3.7</a>
0Ah	ADC10MCTL0	ADC10_A Memory Control register	Read/write	00h	<a href="#">Section 27.3.6</a>
12h	ADC10MEM0	ADC10_A Conversion Memory register	Read/write	undefined	<a href="#">Section 27.3.4</a>
1Ah	ADC10IE	ADC10_A Interrupt Enable register	Read/write	0000h	<a href="#">Section 27.3.11</a>
1Ch	ADC10IFG	ADC10_A Interrupt Flag register	Read/write	0000h	<a href="#">Section 27.3.12</a>
1Eh	ADC10IV	ADC10_A Interrupt Vector register	Read/write	0000h	<a href="#">Section 27.3.13</a>

### 27.3.1 ADC10CTL0 Register

ADC10\_A Control Register 0

**Figure 27-12. ADC10CTL0 Register**

15	14	13	12	11	10	9	8
Reserved				ADC10SHTx			
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10MSC	Reserved	ADC10ON	ADC10ON	Reserved	ADC10ENC	ADC10SC	ADC10SC
rw-(0)	r0	r0	rw-(0)	r0	r0	rw-(0)	rw-(0)

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

**Table 27-3. ADC10CTL0 Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	ADC10SHTx	RW	0h	ADC10_A sample-and-hold time. These bits define the number of ADC10CLK cycles in the sampling period for the ADC10. 0000b = 4 ADC10CLK cycles 0001b = 8 ADC10CLK cycles 0010b = 16 ADC10CLK cycles 0011b = 32 ADC10CLK cycles 0100b = 64 ADC10CLK cycles 0101b = 96 ADC10CLK cycles 0110b = 128 ADC10CLK cycles 0111b = 192 ADC10CLK cycles 1000b = 256 ADC10CLK cycles 1001b = 384 ADC10CLK cycles 1010b = 512 ADC10CLK cycles 1011b = 768 ADC10CLK cycles 1100b = 1024 ADC10CLK cycles 1101b = 1024 ADC10CLK cycles 1110b = 1024 ADC10CLK cycles 1111b = 1024 ADC10CLK cycles
7	ADC10MSC	RW	0h	ADC10_A multiple sample and conversion. Valid only for sequence or repeated modes. 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	ADC10ON	RW	0h	ADC10_A on 0b = ADC10_A off 1b = ADC10_A on
3-2	Reserved	R	0h	Reserved. Always reads as 0.
1	ADC10ENC	RW	0h	ADC10_A enable conversion 0b = ADC10_A disabled 1b = ADC10_A enabled
0	ADC10SC	RW	0h	ADC10_A start conversion. Software-controlled sample-and-conversion start. ADC10SC and ADC10ENC may be set together with one instruction. ADC10SC is reset automatically. 0b = No sample-and-conversion-start 1b = Start sample-and-conversion

**27.3.2 ADC10CTL1 Register**

## ADC10\_A Control Register 1

**Figure 27-13. ADC10CTL1 Register**

15	14	13	12	11	10	9	8
Reserved				ADC10SHSx		ADC10SHP	ADC10ISSH
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx				ADC10SSELx		ADC10CONSEQx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

**Table 27-4. ADC10CTL1 Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-10	ADC10SHSx	RW	0h	ADC10_A sample-and-hold source select Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active. 00b = ADC10SC bit 01b = Timer trigger 0 - see device specific datasheet 10b = Timer trigger 1 - see device specific datasheet 11b = Timer trigger 2 - see device specific datasheet
9	ADC10SHP	RW	0h	ADC10_A sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also 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	ADC10ISSH	RW	0h	ADC10_A invert signal sample-and-hold Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active. 0b = The sample-input signal is not inverted. 1b = The sample-input signal is inverted.
7-5	ADC10DIVx	RW	0h	ADC10_A clock divider Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also 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 27-4. ADC10CTL1 Register Description (continued)**

Bit	Field	Type	Reset	Description
4-3	ADC10SSELx	RW	0h	ADC10_A clock source select Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active. 00b = MODCLK 01b = ACLK 10b = MCLK 11b = SMCLK
2-1	ADC10CONSEQx	RW	0h	ADC10_A conversion sequence mode select Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active. 00b = Single-channel, single-conversion 01b = Sequence-of-channels 10b = Repeat-single-channel 11b = Repeat-sequence-of-channels
0	ADC10BUSY	R	0h	ADC10_A busy. This bit indicates an active sample or conversion operation. 0b = No operation is active. 1b = A sequence, sample, or conversion is active.

**27.3.3 ADC10CTL2 Register**

## ADC10\_A Control Register 2

**Figure 27-14. ADC10CTL2 Register**

15	14	13	12	11	10	9	8
Reserved						ADC10PDIVx	
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			ADC10RES	ADC10DF	ADC10SR	Reserved	
r0	r0	r0	rw-(1)	rw-(0)	rw-(0)	r0	r0

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

**Table 27-5. ADC10CTL2 Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-8	ADC10PDIVx	RW	0h	ADC10_A predivider. This bit predvides the selected ADC10_A clock source before it gets divided again using ADC10DIVx. 00b = Predivide by 1 01b = Predivide by 4 10b = Predivide by 64 11b = Reserved
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4	ADC10RES	RW	1h	ADC10_A resolution. This bit defines the conversion result resolution. 0b = 8 bit (10 clock cycle conversion time) 1b = 10 bit (12 clock cycle conversion time)
3	ADC10DF	RW	0h	ADC10_A 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	ADC10SR	RW	0h	ADC10_A sampling rate. This bit selects drive capability of the ADC10_A reference buffer for the maximum sampling rate. Setting ADC10SR reduces the current consumption of this buffer. 0b = ADC10_A buffer supports up to approximately 200 ksps. 1b = ADC10_A buffer supports up to approximately 50 ksps.
1-0	Reserved	R	0h	Reserved. Always reads as 0.

### 27.3.4 ADC10MEM0 Register

ADC10\_A Conversion Memory Register

**Figure 27-15. ADC10MEM0 Register**

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

**Table 27-6. ADC10MEM0 Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	Conversion_Results	RW	undefined	The 10-bit conversion results are right justified. 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. This data format is used if ADC10DF = 0.

### 27.3.5 ADC10MEM0 Register, 2s-Complement Format

ADC10\_A Conversion Memory Register, 2s-Complement Format

**Figure 27-16. ADC10MEM0 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		Reserved					
rw	rw	r0	r0	r0	r0	r0	r0

**Table 27-7. ADC10MEM0 Register Description**

Bit	Field	Type	Reset	Description
15-6	Conversion_Results	RW	undefined	The 10-bit conversion results are left justified, 2s-complement format. Bit 15 is the MSB. Bits 5–0 are 0 in 10-bit mode, and bits 7–0 are 0 in 8-bit mode. This data format is used if ADC10DF = 1. 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.
5-0	Reserved	R	0h	Reserved. Always reads as 0.

### 27.3.6 ADC10MCTL0 Register

ADC10\_A Conversion Memory Control Register

**Figure 27-17. ADC10MCTL0 Register**

7	6	5	4	3	2	1	0
Reserved	ADC10SREFx						ADC10INCHx
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

**Table 27-8. ADC10MCTL0 Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	ADC10SREFx	RW	0h	Select reference. It is not recommended to change this setting while a conversion is ongoing. Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also 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	ADC10INCHx	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 ADC10CONSEQ = 01,11 returns the channel currently converted. ADC10INCHx is not synchronized, so a read while the state machine is not in "wait for enable" or "wait for trigger" could lead to a wrong result. Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active. 0000b = A0 0001b = A1 0010b = A2 0011b = A3 0100b = A4 0101b = A5 0110b = A6 0111b = A7 1000b = A8 1001b = A9 1010b = A10 1011b = A11 1100b = A12 1101b = A13 1110b = A14 1111b = A15

### 27.3.7 ADC10HI Register

ADC10\_A Window Comparator High Threshold Register

**Figure 27-18. ADC10HI Register**

15	14	13	12	11	10	9	8
Reserved						High_Threshold	
r0	r0	r0	r0	r0	r0	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)	rw-(1)

**Table 27-9. ADC10HI Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	High_Threshold	RW	3FFh	The 10-bit threshold value needs to be right justified. Bit 9 is the MSB. Bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. This data format is used if ADC10DF = 0.

### 27.3.8 ADC10HI Register, 2s-Complement Format

ADC10\_A Window Comparator High Threshold Register, 2s-Complement Format

**Figure 27-19. ADC10HI Register**

15	14	13	12	11	10	9	8
High_Threshold						rw-(1)	
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		Reserved					
rw-(1)	rw-(1)	r0	r0	r0	r0	r0	r0

**Table 27-10. ADC10HI Register Description**

Bit	Field	Type	Reset	Description
15-6	High_Threshold	RW	1FFh	The 10-bit threshold value needs to be left justified if 2s-complement format is chosen. Bit 15 is the MSB. Bits 5–0 are 0 in 10-bit mode, and bits 7–0 are 0 in 8-bit mode. This data format is used if ADC10DF = 1.
5-0	Reserved	R	0h	Reserved. Always reads as 0.

**27.3.9 ADC10LO Register**

ADC10\_A Window Comparator Low Threshold Register

**Figure 27-20. ADC10LO Register**

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

**Table 27-11. ADC10LO Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	Low_Threshold	RW	0h	The 10-bit threshold value needs to be right justified. Bit 9 is the MSB. Bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. This data format is used if ADC10DF = 0.

**27.3.10 ADC10LO Register, 2s-Complement Format**

ADC10\_A Window Comparator Low Threshold Register, 2s-Complement Format

**Figure 27-21. ADC10LO Register**

15	14	13	12	11	10	9	8
Low_Threshold						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		Reserved					
rw-(0)	rw-(0)	r0	r0	r0	r0	r0	r0

**Table 27-12. ADC10LO Register Description**

Bit	Field	Type	Reset	Description
15-6	Low_Threshold	RW	200h	The 10-bit threshold value needs to be left justified if 2s-complement format is chosen. Bit 15 is the MSB. Bits 5–0 are 0 in 10-bit mode, and bits 7–0 are 0 in 8-bit mode. This data format is used if ADC10DF = 1.
5-0	Reserved	R	0h	Reserved. Always reads as 0.

### 27.3.11 ADC10IE Register

ADC10\_A Interrupt Enable Register

**Figure 27-22. ADC10IE 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	ADC10TOVIE	ADC10OVIE	ADC10HIIIE	ADC10LOIE	ADC10INIE	ADC10IE0	ADC10IE0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 27-13. ADC10IE Register Description**

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5	ADC10TOVIE	RW	0h	ADC10_A conversion-time-overflow Interrupt enable. 0b = Conversion time overflow Interrupt disabled 1b = Conversion time overflow Interrupt enabled
4	ADC10OVIE	RW	0h	ADC10MEM0 overflow Interrupt enable. 0b = Overflow Interrupt disabled 1b = Overflow Interrupt enabled
3	ADC10HIIIE	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	ADC10LOIE	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	ADC10INIE	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	ADC10IE0	RW	0h	Interrupt enable. This bits enable or disable the Interrupt request for a completed ADC10_A conversion. 0b = Interrupt disabled 1b = Interrupt enabled

**27.3.12 ADC10IFG Register**

ADC10\_A Interrupt Flag Register

**Figure 27-23. ADC10IFG 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	ADC10TOVIFG	ADC10OVIFG	ADC10HIIIFG	ADC10LOIFG	ADC10INIFG	ADC10IFG0	ADC10IFG0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 27-14. ADC10IFG Register Description**

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5	ADC10TOVIFG	RW	0h	The ADC10TOVIFG is set when an ADC10_A conversion is triggered before the actual conversion has completed. 0b = No Interrupt pending 1b = Interrupt pending
4	ADC10OVIFG	RW	0h	The ADC10OVIFG is set when the ADC10MEM0 register is written before the last conversion result has been read. 0b = No Interrupt pending 1b = Interrupt pending
3	ADC10HIIIFG	RW	0h	The ADC10HIIIFG is set when the result of the current ADC10_A conversion is greater than the upper threshold defined by the Window Comparators upper threshold register. 0b = No Interrupt pending 1b = Interrupt pending
2	ADC10LOIFG	RW	0h	The ADC10LOIFG is set when the result of the current ADC10_A conversion is below the lower threshold defined by the Window Comparators lower threshold register. 0b = No Interrupt pending 1b = Interrupt pending
1	ADC10INIFG	RW	0h	The ADC10INIFG is set when the result of the current ADC10_A conversion is within the thresholds defined by the Window Comparators threshold registers. 0b = No Interrupt pending 1b = Interrupt pending
0	ADC10IFG0	RW	0h	The ADC10IFG0 is set when an ADC10_A conversion is completed. This bit gets reset, when the ADC10MEM0 get read, or may be reset by software. 0b = No Interrupt pending 1b = Interrupt pending

### 27.3.13 ADC10IV Register

ADC10\_A Interrupt Vector Register

**Figure 27-24. ADC10IV Register**

15	14	13	12	11	10	9	8
ADC10IVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
ADC10IVx							
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

**Table 27-15. ADC10IV Register Description**

Bit	Field	Type	Reset	Description
15-0	ADC10IVx	R	0h	<p>ADC10_A 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.</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: ADC10MEM0 overflow; Interrupt Flag: ADC10OVIFG; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: Conversion time overflow; Interrupt Flag: ADC10TOVIFG</p> <p>06h = Interrupt Source: ADC10HI Interrupt flag; Interrupt Flag: ADC10HIIFG</p> <p>08h = Interrupt Source: ADC10LO Interrupt flag; Interrupt Flag: ADC10LOIFG</p> <p>0Ah = Interrupt Source: ADC10IN Interrupt flag; Interrupt Flag: ADC10INIFG</p> <p>0Ch = Interrupt Source: ADC10_A memory Interrupt flag; Interrupt Flag: ADC10IFG0; Interrupt Priority: Lowest</p>

## **ADC12\_A**

The ADC12\_A module is a high-performance 12-bit analog-to-digital converter (ADC). This chapter describes the operation of the ADC12\_A module.

Topic	Page
<b>28.1 ADC12_A Introduction .....</b>	<b>723</b>
<b>28.2 ADC12_A Operation .....</b>	<b>726</b>
<b>28.3 ADC12_A Registers .....</b>	<b>740</b>

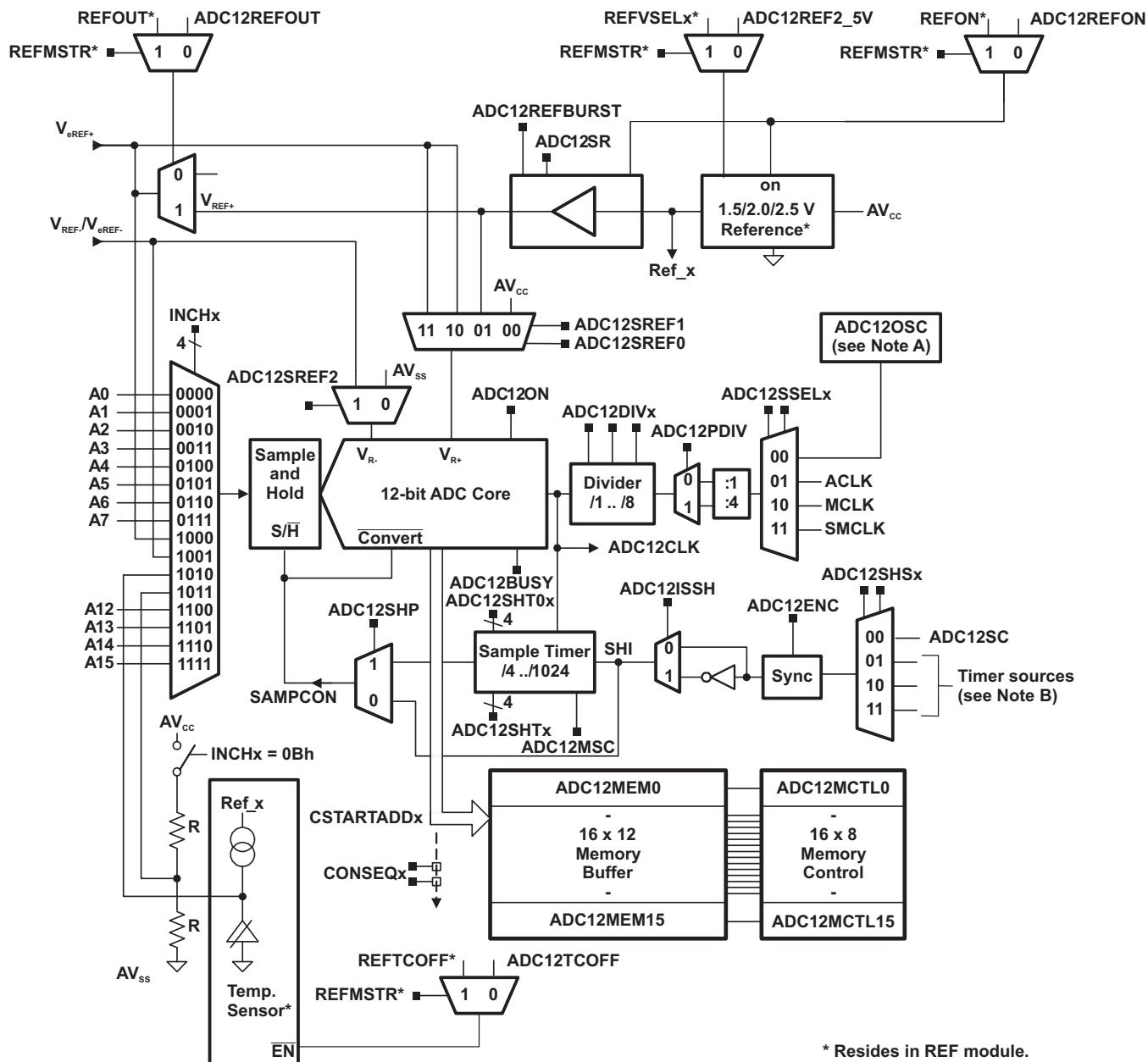
## 28.1 ADC12\_A Introduction

The ADC12\_A module supports fast 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, reference generator (MSP430F54xx (non-A only) – in other devices, separate REF module), and a 16-word conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent analog-to-digital converter (ADC) samples to be converted and stored without any CPU intervention.

ADC12\_A features include:

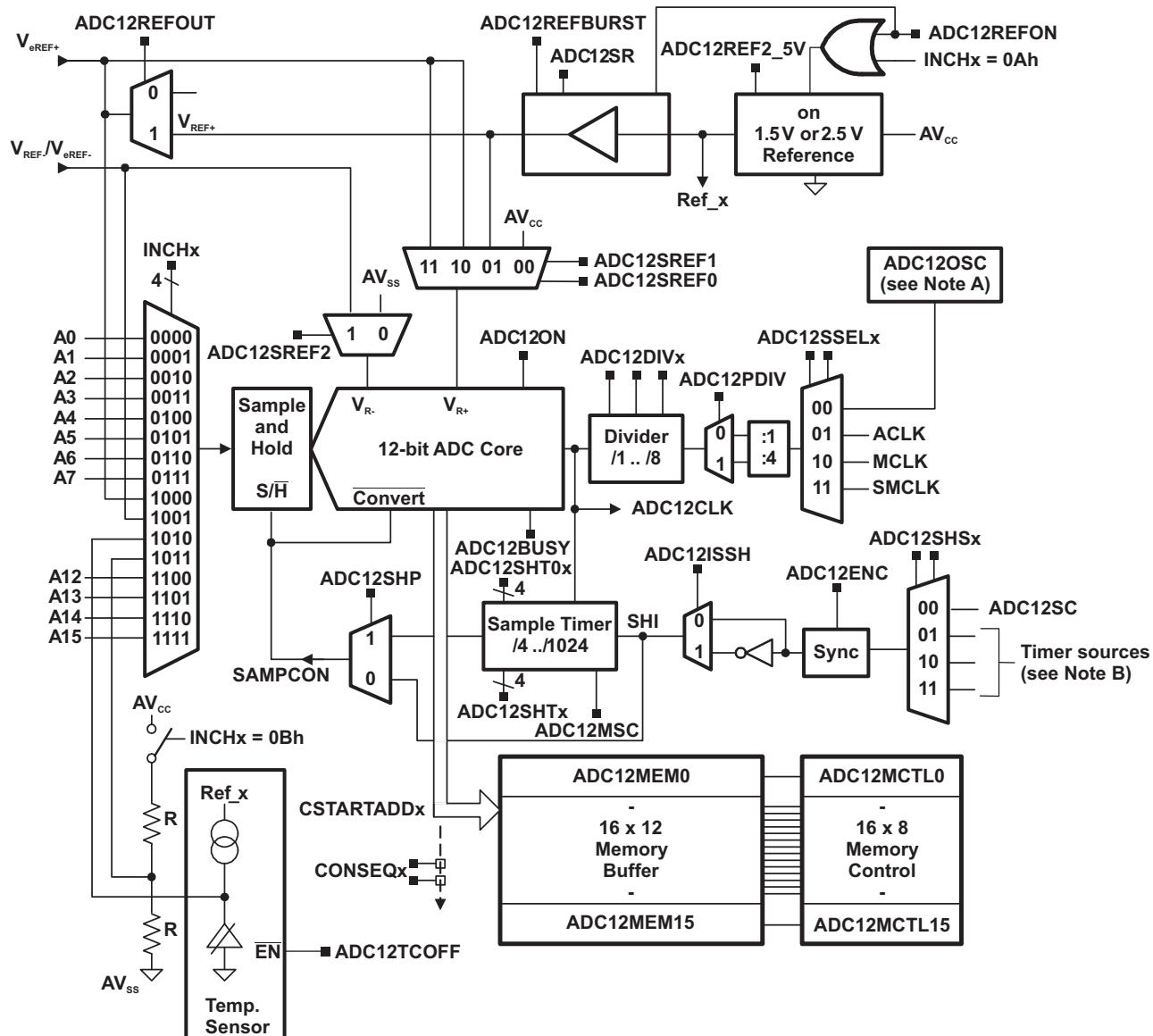
- Greater than 200-ksps maximum conversion rate
- Monotonic 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers
- Conversion initiation by software or timers
- Software-selectable on-chip reference voltage generation (MSP430F54xx (non-A only): 1.5 V or 2.5 V, all other devices: 1.5 V, 2.0 V, or 2.5 V)
- Software-selectable internal or external reference
- Up to 12 individually configurable external input channels
- Conversion channels for internal temperature sensor,  $AV_{CC}$ , and external references
- Independent channel-selectable reference sources for both positive and negative references
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence (autoscan), and repeat-sequence (repeated autoscan) conversion modes
- ADC core and reference voltage can be powered down separately
- Interrupt vector register for fast decoding of 18 ADC interrupts
- 16 conversion-result storage registers

The block diagram of ADC12\_A is shown in [Figure 28-1](#). In MSP430F54xx (non-A only), the reference generator is located in the ADC12\_A module itself. In other devices, the reference generator is located in the reference module, REF. See the REF module chapter and the device-specific data sheet for further details. [Figure 28-1](#) shows the block diagram for devices that have the REF module available. [Figure 28-2](#) shows the block diagram for the MSP430F54xx (non-A only) which does not incorporate the REF module.



- A The MODOSC is part of the UCS. See the [UCS chapter](#) for more information.  
 B See the device-specific data sheet for timer sources available.

Figure 28-1. ADC12\_A Block Diagram (Devices With REF Module)



A The MODOSC is part of the UCS. See the [UCS chapter](#) for more information.

B See the device-specific data sheet for timer sources available.

**Figure 28-2. ADC12\_A MSP430F54xx (non-A) Block Diagram**

## 28.2 ADC12\_A Operation

The ADC12\_A module is configured with user software. The setup and operation of the ADC12\_A is discussed in the following sections.

### 28.2.1 12-Bit ADC Core

The ADC core converts an analog input to its 12-bit digital representation and stores the result in conversion memory. The core uses two programmable and selectable voltage levels ( $V_{R+}$  and  $V_{R-}$ ) to define the upper and lower limits of the conversion. The digital output ( $N_{ADC}$ ) is full scale (0FFFh) when the input signal is equal to or higher than  $V_{R+}$ . The digital output ( $N_{ADC}$ ) 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:

$$N_{ADC} = 4095 \times \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC12\_A core is configured by two control registers, ADC12CTL0 and ADC12CTL1. The core is enabled with the ADC12ON bit. The ADC12\_A can be turned off when it is not in use to save power. With few exceptions, the ADC12\_A control bits can be modified only when ADC12ENC = 0. ADC12ENC must be set to 1 before any conversion can take place.

#### 28.2.1.1 Conversion Clock Selection

The ADC12CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC12\_A source clock is selected using the predivider controlled by the ADC12PDIV bit and the divider using the ADC12SSELx bits. The input clock can be divided from 1 to 32 using both the ADC12DIVx bits and the ADC12PDIV bit. Possible ADC12CLK sources are SMCLK, MCLK, ACLK, and the ADC12OSC.

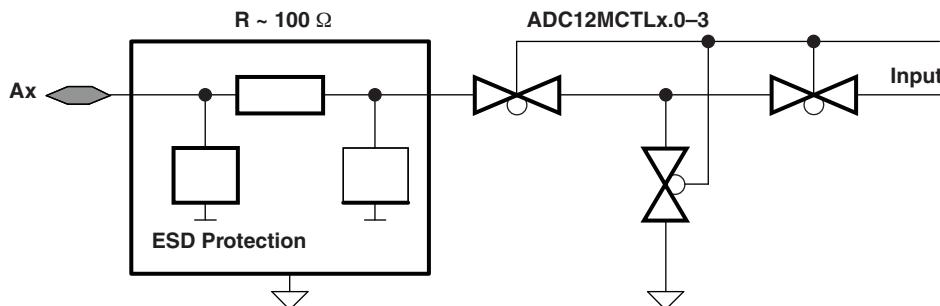
The ADC12OSC in the block diagram (see [Figure 28-1](#)) refers to the MODOSC 5-MHz oscillator from the UCS (see the UCS module for more information) which can vary with individual devices, supply voltage, and temperature. See the device-specific data sheet for the ADC12OSC specification.

The user must ensure that the clock chosen for ADC12CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation does not complete and the results are invalid.

### 28.2.2 ADC12\_A Inputs and Multiplexer

The 12 external and 4 internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching (see [Figure 28-3](#)). The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground (AV<sub>ss</sub>), so that the stray capacitance is grounded to eliminate crosstalk.

The ADC12\_A uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.



**Figure 28-3. Analog Multiplexer**

### 28.2.2.1 Analog Port Selection

The ADC12\_A inputs are multiplexed with digital port pins. When analog signals are applied to digital gates, parasitic current can flow from  $V_{CC}$  to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the digital part of the port pin eliminates the parasitic current flow and, therefore, reduces overall current consumption. The PySELx bits provide the ability to disable the port pin input and output buffers.

```
; Py.0 and Py.1 configured for analog input
BIS.B #3h,&PySEL    ; Py.1 and Py.0 ADC12_A function
```

### 28.2.3 Voltage Reference Generator

The ADC12\_A modules have a separate reference module (REF) that supplies three selectable voltage levels, 1.5 V, 2.0 V, and 2.5 V to the ADC12\_A. Any of these voltages may be used internally and externally on pin VREF+. The internal  $AV_{CC}$  can also be used as the reference.

The ADC12\_A module of the MSP430F54xx devices (non-A only) does not use the REF module and only has two selectable voltage levels, 1.5 V and 2.5 V. The internal  $AV_{CC}$  can also be used as the reference.

On devices with the REF module, the voltage reference settings can be controlled either by the REF module or by the ADC12\_A module. This is to allow for backward compatibility with older families. This is handled by the REFMSTR bit in the REF module. If REFMSTR = 1 (default), the REF module registers control the reference settings. If REFMSTR = 0, the ADC12\_A reference setting define the reference voltage of the ADC12\_A module. Four control settings that reside in the ADC12\_A can be controlled also by four corresponding settings in the REF module: ADC12REF2\_5V (REFVSEL), ADC12REFON (REFON), ADC12REFOUT (REFOUT), and ADC12TCOFF (REFTCOFF), respectively. When REFMSTR = 1, ADC12REF2\_5V, ADC12REFON, ADC12REFOUT, and ADC12TCOFF are do not care. Similarly, when REFMSTR = 0, REFVSEL, REFON, REFOUT, and REFTCOFF are do not care. See the REF module chapter for further details.

On devices with the REF module, to use the ADC12\_A reference control bits, set REFMSTR = 0. In this case, setting ADC12REFON = 1 enables the reference voltage of the ADC12\_A module. When ADC12REF2\_5V = 1, the internal reference is 2.5 V; when ADC12REF2\_5V = 0, the reference is 1.5 V. Similarly, on devices with the REF module, to use the REF module reference control bits, set REFMSTR = 1. In this case, setting REFON = 1 of the REF module enables the reference voltage. The REFVSEL bits of the REF module can be used to select either 1.5 V, 2.0 V, or 2.5 V. The reference can be turned off to save power when not in use. On the MSP430F54xx devices (non-A only), as stated previously, the REF module is not present and behaves the same as devices the REF module with REFMSTR = 0.

External references may be supplied for  $V_{R+}$  and  $V_{R-}$  through pins VREF+/VeREF+ and VREF-/VeREF-, respectively.

External storage capacitors are required only if ADC12REFOUT = 1 (REFOUT = 1 when using REF module) and the reference voltage is made available at the pins.

#### 28.2.3.1 Internal Reference Low-Power Features

The ADC12\_A internal reference generator is designed for low-power applications. The reference generator includes a bandgap voltage source and a separate buffer. The current consumption and settling time of each is specified separately in the device-specific data sheet. When ADC12REFON = 1 (REFON = 1 when using REF module), both are enabled; when ADC12REFON = 0 (REFON = 0 when using REF module), both are disabled.

When ADC12REFON = 1 (REFON = 1 when using REF module) and ADC12REFBURST = 1 but no conversion is active, the buffer is automatically disabled and automatically reenabled when needed. When the buffer is disabled, it consumes no current. In this case, the bandgap voltage source remains enabled.

The ADC12REFBURST bit controls the operation of the reference buffer. When ADC12REFBURST = 1, the buffer is automatically disabled when the ADC12\_A is not actively converting, and is automatically reenabled when needed. When ADC12REFBURST = 0, the buffer is on continuously. This allows the reference voltage to be present outside the device continuously if ADC12REFOUT = 1 (REFOUT = 1 when using REF module).

The internal reference buffer also has selectable speed versus power settings. When the maximum conversion rate is below 50 ksps, setting ADC12SR = 1 reduces the current consumption of the buffer by approximately 50%.

#### 28.2.4 Auto Power Down

The ADC12\_A is designed for low-power applications. When the ADC12\_A is not actively converting, the core is automatically disabled, and it is automatically reenabled when needed. The MODOSC is also automatically enabled when needed and disabled when not needed.

#### 28.2.5 Sample and Conversion Timing

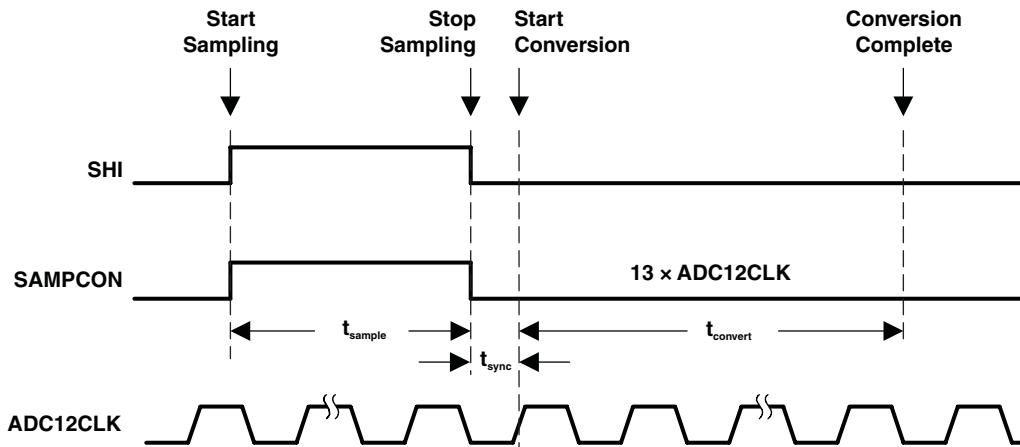
An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- ADC12SC bit
- Up to three timer outputs (see the device-specific data sheet for available timer sources)

The ADC12\_A supports 8-bit, 10-bit, and 12-bit resolution modes selectable by the ADC12RES bits. The analog-to-digital conversion requires 9, 11, and 13 ADC12CLK cycles, respectively. The polarity of the SHI signal source can be inverted with the ADC12ISSH bit. The SAMPSON signal controls the sample period and start of conversion. When SAMPSON is high, sampling is active. The high-to-low SAMPSON transition starts the analog-to-digital conversion. Two different sample-timing methods are defined by control bit ADC12SHP, extended sample mode, and pulse mode. See the device-specific data sheet for available timers for SHI sources.

##### 28.2.5.1 Extended Sample Mode

The extended sample mode is selected when ADC12SHP = 0. The SHI signal directly controls SAMPSON and defines the length of the sample period  $t_{sample}$ . When SAMPSON is high, sampling is active. The high-to-low SAMPSON transition starts the conversion after synchronization with ADC12CLK (see [Figure 28-4](#)).

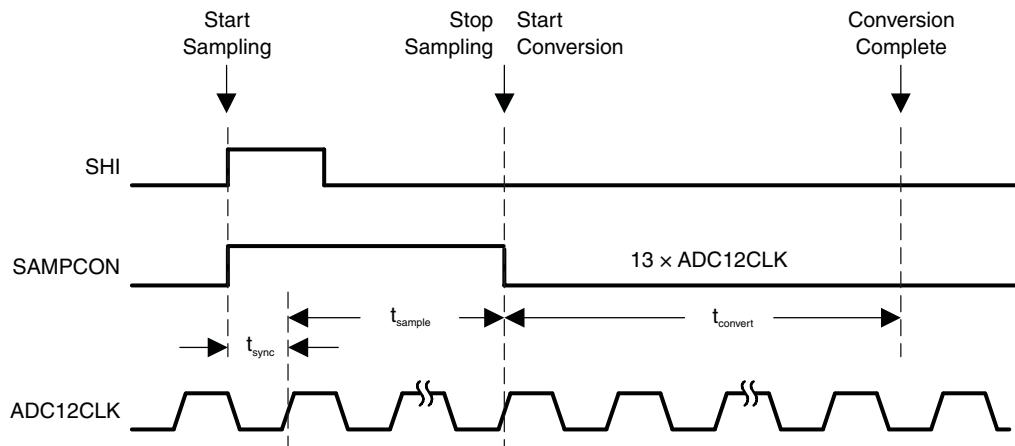


**Figure 28-4. Extended Sample Mode**

##### 28.2.5.2 Pulse Sample Mode

Set ADC12SHP = 1 to select the pulse sample mode. The SHI signal is used to trigger the sampling timer. The ADC12SHT0x and ADC12SHT1x bits in ADC12CTL0 control the interval of the sampling timer that defines the SAMPSON sample period  $t_{sample}$ . The sampling timer keeps SAMPSON high after synchronization with ADC12CLK for a programmed interval  $t_{sample}$ . The total sampling time is  $t_{sample}$  plus  $t_{sync}$  (see [Figure 28-5](#)).

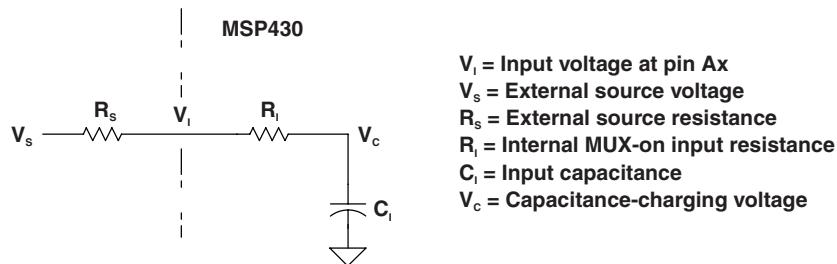
The ADC12SHTx bits select the sampling time in 4x multiples of ADC12CLK. ADC12SHT0x selects the sampling time for ADC12MCTL0 to ADC12MCTL7. ADC12SHT1x selects the sampling time for ADC12MCTL8 to ADC12MCTL15.



**Figure 28-5. Pulse Sample Mode**

#### 28.2.5.3 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 28-6). An internal MUX-on input resistance  $R_i$  (maximum 1.8 kΩ) in series with capacitor  $C_i$  (25 pF maximum) is seen by the source. The capacitor  $C_i$  voltage  $V_c$  must be charged to within one-half LSB of the source voltage  $V_s$  for an accurate n-bit conversion, where n is the bits of resolution required.



**Figure 28-6. Analog Input Equivalent Circuit**

The resistance of the source  $R_s$  and  $R_i$  affect  $t_{sample}$ . The following equation can be used to calculate the minimum sampling time  $t_{sample}$  for a n-bit conversion, where n equals the bits of resolution:

$$t_{sample} > (R_s + R_i) \times \ln(2^{n+1}) \times C_i + 800 \text{ ns}$$

Substituting the values for  $R_i$  and  $C_i$  given above, the equation becomes:

$$t_{sample} > (R_s + 1.8 \text{ k}\Omega) \times \ln(2^{n+1}) \times 25 \text{ pF} + 800 \text{ ns}$$

For example, for 12-bit resolution, if  $R_s$  is 10 kΩ,  $t_{sample}$  must be greater than 3.46 μs.

## 28.2.6 Conversion Memory

There are 16 ADC12MEMx conversion memory registers to store conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The SREFx bits define the voltage reference and the INCHx bits select the input channel. The ADC12EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM15 to ADC12MEM0 when the ADC12EOS bit in ADC12MCTL15 is not set.

The CSTARTADDx bits define the first ADC12MCTLx used for any conversion. If the conversion mode is single-channel or repeat-single-channel, the CSTARTADDx points to the single ADC12MCTLx to be used.

If the conversion mode selected is either sequence-of-channels or repeat-sequence-of-channels, CSTARTADDx points to the first ADC12MCTLx location to be used in a sequence. A pointer, not visible to software, is incremented automatically to the next ADC12MCTLx in a sequence when each conversion completes. The sequence continues until an ADC12EOS bit in ADC12MCTLx is processed; this is the last control byte processed.

When conversion results are written to a selected ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.

There are two formats available to store the conversion result, ADC12MEMx. When ADC12DF = 0, the conversion is right justified, unsigned. For 8-bit, 10-bit, and 12-bit resolutions, the upper 8, 6, and 4 bits of ADC12MEMx are always zeros, respectively. When ADC12DF = 1, the conversion result is left justified, two's complement. For 8-bit, 10-bit, and 12-bit resolutions, the lower 8, 6, and 4 bits of ADC12MEMx are always zeros, respectively. This is summarized in [Table 28-1](#).

**Table 28-1. ADC12\_A Conversion Result Formats**

Analog Input Voltage	ADC12DF	ADC12RES	Ideal Conversion Results	ADC12MEMx
$-V_{REF}$ to $+V_{REF}$	0	00	0 to 255	0000h - 00FFh
	0	01	0 to 1023	0000h - 03FFh
	0	10	0 to 4095	0000h - 0FFFh
	1	00	-128 to 127	8000h - 7F00h
	1	01	-512 to 511	8000h - 7FC0h
	1	10	-2048 to 2047	8000h - 7FF0h

## 28.2.7 ADC12\_A Conversion Modes

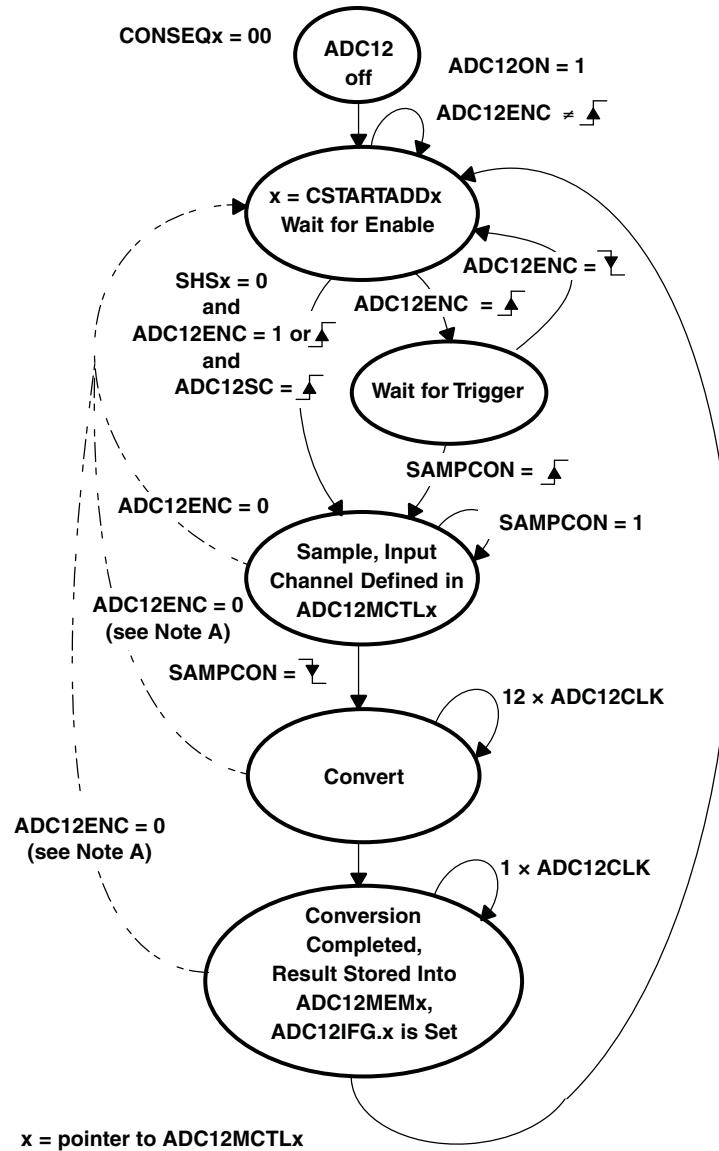
The ADC12\_A has four operating modes selected by the CONSEQx bits as listed in [Table 28-2](#). All state diagrams assume a 12-bit resolution setting.

**Table 28-2. Conversion Mode Summary**

ADC12CONSEQx	Mode	Operation
00	Single-channel single-conversion	A single channel is converted once.
01	Sequence-of-channels (autoscan)	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels (repeated autoscan)	A sequence of channels is converted repeatedly.

### 28.2.7.1 Single-Channel Single-Conversion Mode

A single channel is sampled and converted once. The ADC result is written to the ADC12MEMx defined by the CSTARTADDx bits. Figure 28-7 shows the flow of the single-channel single-conversion mode. When ADC12SC triggers a conversion, successive conversions can be triggered by the ADC12SC bit. When any other trigger source is used, ADC12ENC must be toggled between each conversion.



A Conversion result is unpredictable.

**Figure 28-7. Single-Channel Single-Conversion Mode**

### 28.2.7.2 Sequence-of-Channels Mode (Autoscan Mode)

In sequence-of-channels mode, also referred to as autoscan mode, a sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the ADCMEMx defined by the CSTARTADDx bits. The sequence stops after the measurement of the channel with a set ADC12EOS bit. Figure 28-8 shows the sequence-of-channels mode. When ADC12SC triggers a sequence, successive sequences can be triggered by the ADC12SC bit. The ADC12SC must be cleared by software after each sequence to trigger another sequence. When any other trigger source is used, ADC12ENC must be toggled between each sequence.

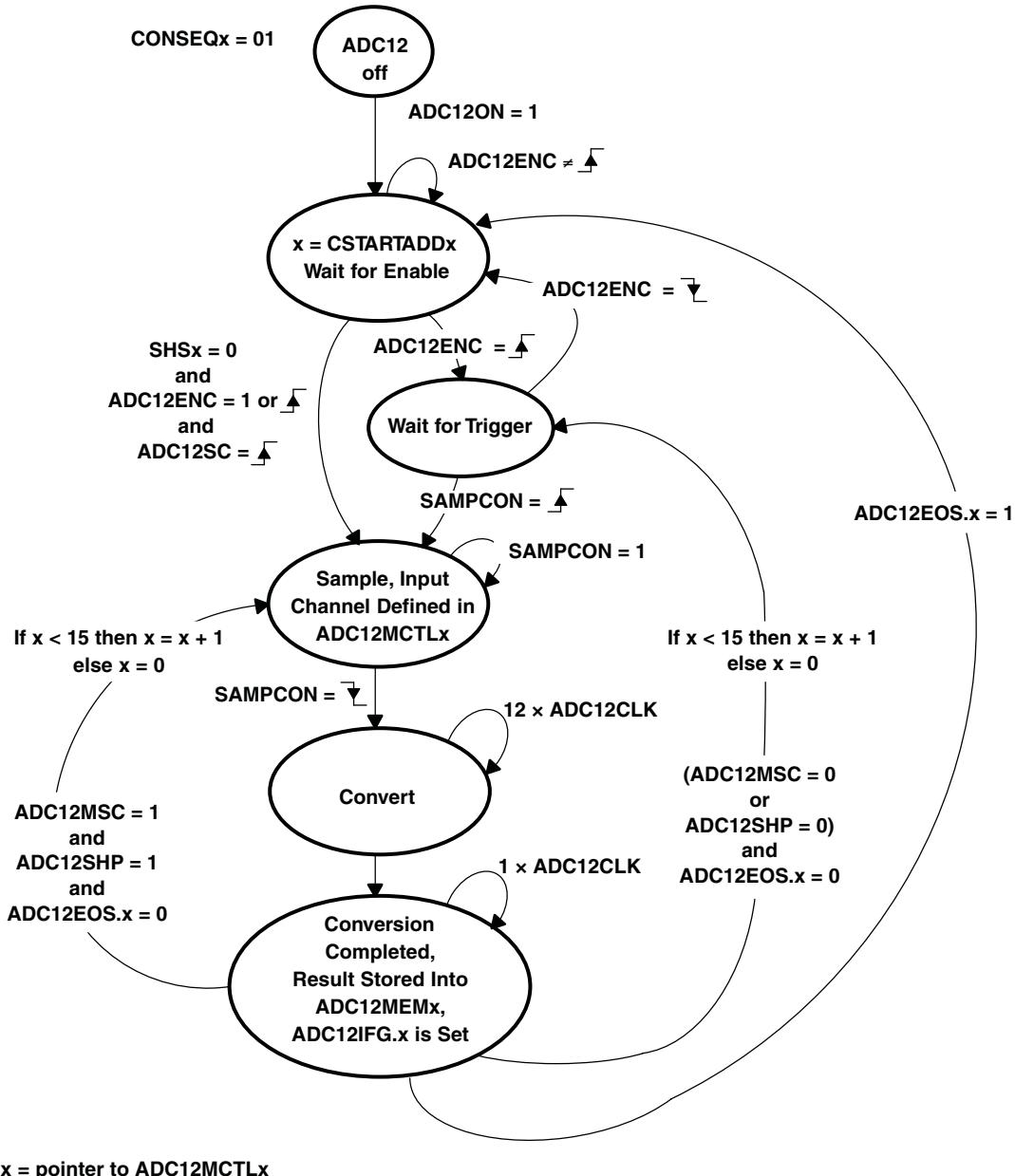
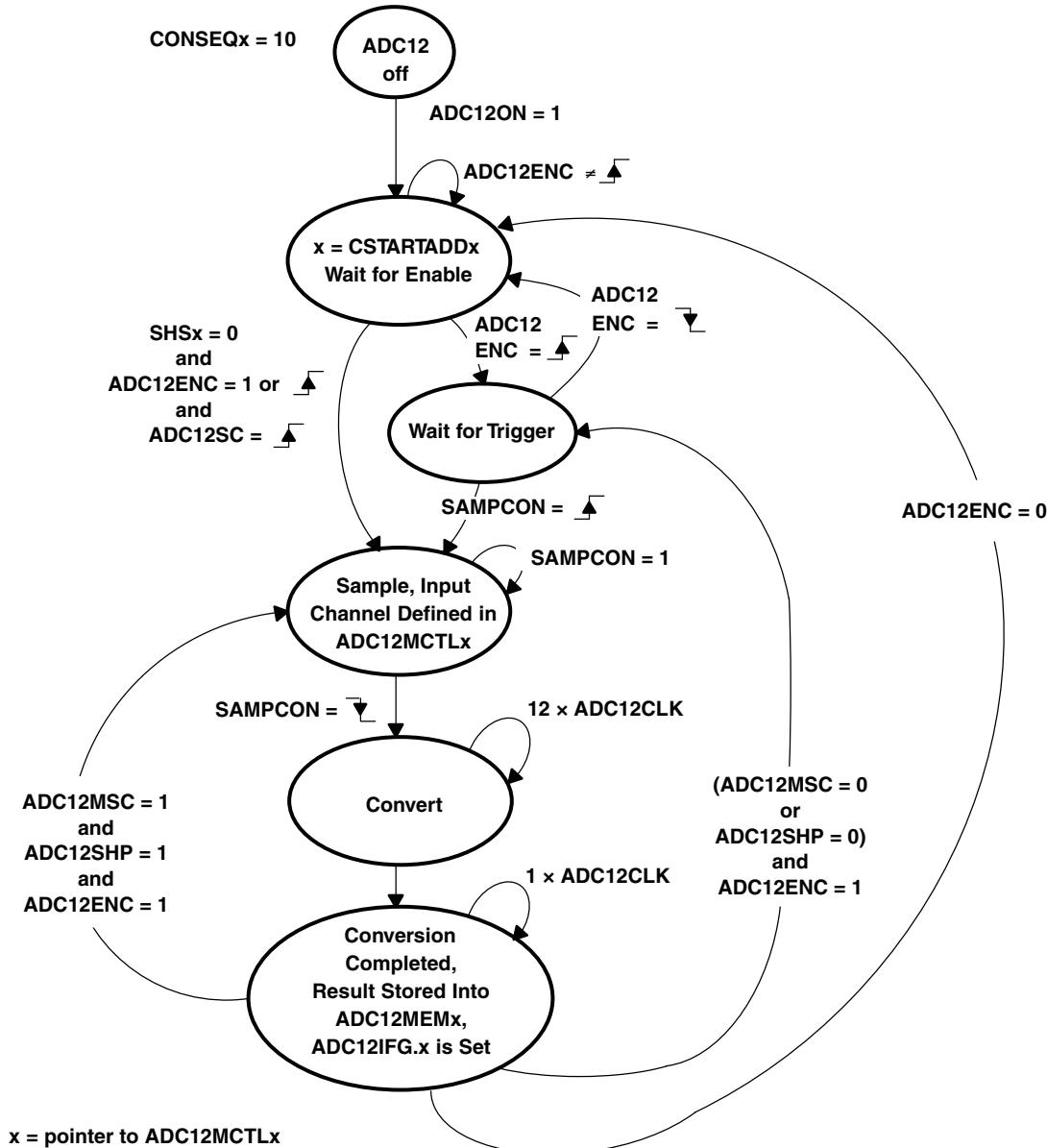


Figure 28-8. Sequence-of-Channels Mode

### 28.2.7.3 Repeat-Single-Channel Mode

A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the CSTARTADDx bits. It is necessary to read the result after the completed conversion because only one ADC12MEMx memory is used and is overwritten by the next conversion. [Figure 28-9](#) shows the repeat-single-channel mode.



**Figure 28-9. Repeat-Single-Channel Mode**

#### 28.2.7.4 Repeat-Sequence-of-Channels Mode (Repeated Autoscan Mode)

In this mode, a sequence of channels is sampled and converted repeatedly. This mode is also referred to as repeated autoscan mode. The ADC results are written to the conversion memories starting with the ADC12MEMx defined by the CSTARTADDx bits. The sequence ends after the measurement of the channel with a set ADC12EOS bit and the next trigger signal restarts the sequence. Figure 28-10 shows the repeat-sequence-of-channels mode.

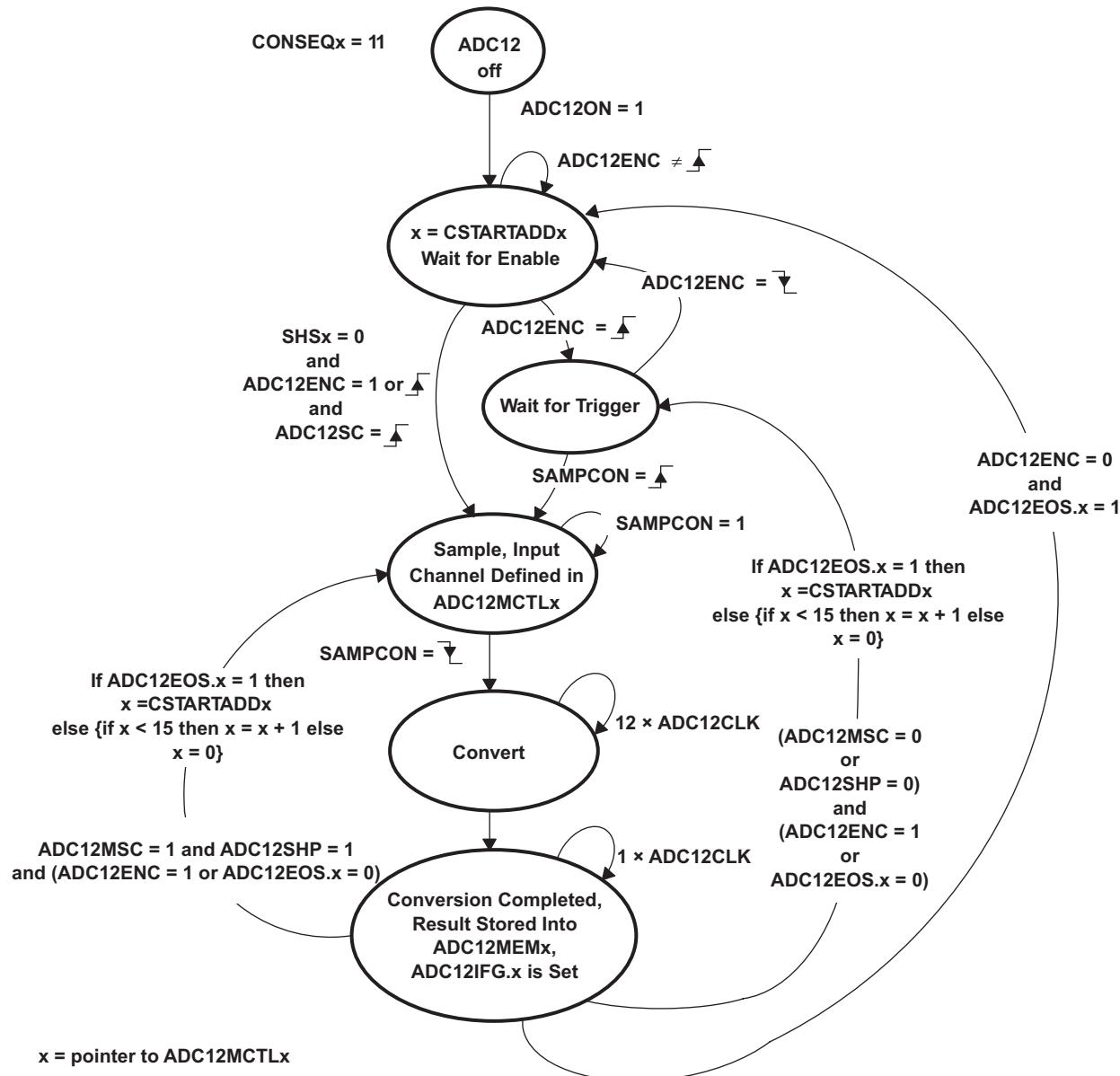


Figure 28-10. Repeat-Sequence-of-Channels Mode

### 28.2.7.5 Using the Multiple Sample and Convert (ADC12MSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When ADC12MSC = 1, CONSEQx > 0, and the sample timer is used, the first rising edge of the SHI signal triggers the first conversion. Successive conversions are triggered automatically as soon as the prior conversion is completed. Additional rising edges on SHI are ignored until the sequence is completed in the single-sequence mode, or until the ADC12ENC bit is toggled in repeat-single-channel or repeated-sequence modes. The function of the ADC12ENC bit is unchanged when using the ADC12MSC bit.

### 28.2.7.6 Stopping Conversions

Stopping ADC12\_A activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ADC12ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the busy bit until reset before clearing ADC12ENC.
- Resetting ADC12ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ADC12ENC during a sequence or repeat-sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the CONSEQx = 0 and resetting the ADC12ENC bit. Conversion data are unreliable.

---

**NOTE: No ADC12EOS bit set for sequence**

If no ADC12EOS bit is set and a sequence mode is selected, resetting the ADC12ENC bit does not stop the sequence. To stop the sequence, first select a single-channel mode and then reset ADC12ENC.

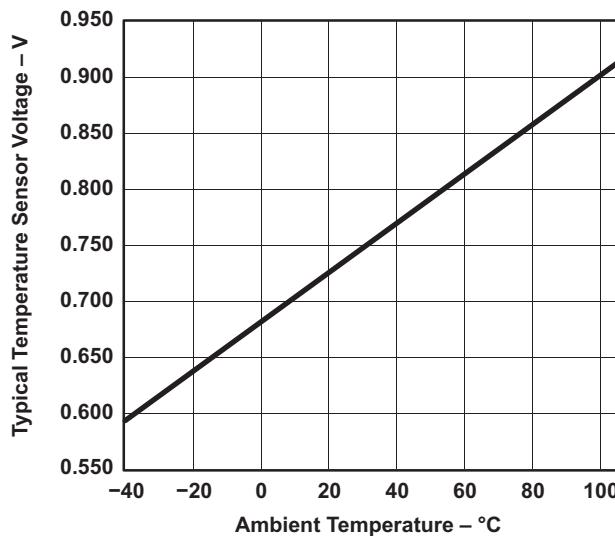
---

### 28.2.8 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel INCH<sub>x</sub> = 1010. Any other configuration is done as if an external channel were selected, including reference selection, conversion-memory selection, etc. The temperature sensor is part of the reference. Therefore, for devices with the REF module, in addition to the input channels selection INCH<sub>x</sub> = 1010, configuring ADC12REFON = 1 (for REFMSTR = 0) or REFON = 1 (for REFMSTR = 1) is required to enable the temperature sensor.

For the MSP430F54xx (non-A) devices, which do not include the REF module, selecting the temperature sensor by configuring INCH<sub>x</sub> = 1010 automatically enables the reference generator required for the temperature sensor. Any other configuration is done as if an external channel were selected, included reference selection, conversion-memory selection, etc.

A typical temperature sensor transfer function is shown in [Figure 28-11](#). The transfer function shown in [Figure 28-11](#) is only an example—the device-specific data sheet contains the actual parameters for a given device. When using the temperature sensor, the sample period must be greater than 30  $\mu$ s. The temperature sensor offset error can be large and may need to be calibrated for most applications. Temperature calibration values are available for use in the TLV descriptors (see the device-specific data sheet for locations).



**Figure 28-11. Typical Temperature Sensor Transfer Function**

### 28.2.9 ADC12\_A Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed circuit board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

Ground loops are formed when return current from the ADC flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small unwanted offset voltages that can add to or subtract from the reference or input voltages of the ADC. The connections shown in Figure 28-12 prevent this.

In addition to grounding, ripple and noise spikes on the power-supply lines due to digital switching or switching power supplies can corrupt the conversion result. A noise-free design using separate analog and digital ground planes with a single-point connection is recommended to achieve high accuracy.

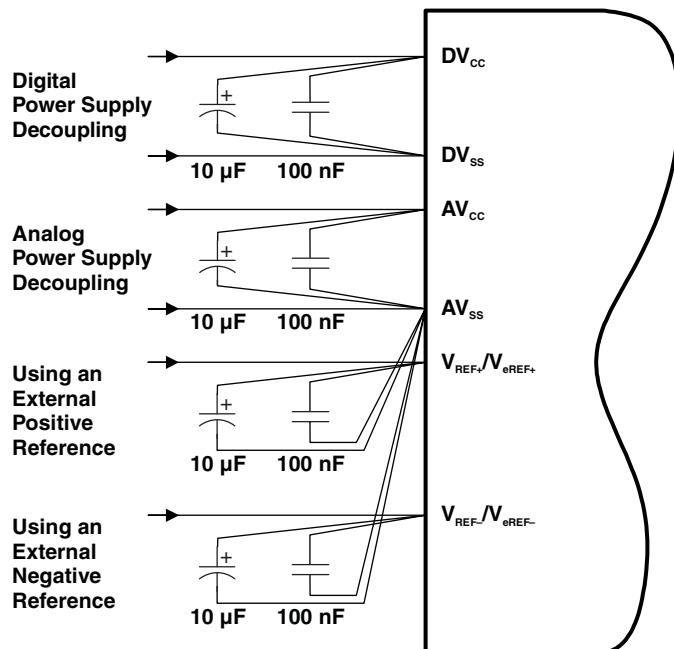


Figure 28-12. ADC12\_A Grounding and Noise Considerations

## 28.2.10 ADC12\_A Interrupts

The ADC12\_A has 18 interrupt sources:

- ADC12IFG0 to ADC12IFG15
- ADC12OV, ADC12MEMx overflow
- ADC12TOV, ADC12\_A conversion time overflow

The ADC12IFGx bits are set when their corresponding ADC12MEMx memory register is loaded with a conversion result. An interrupt request is generated if the corresponding ADC12IEx bit and the GIE bit are set. The ADC12OV condition occurs when a conversion result is written to any ADC12MEMx before its previous conversion result was read. The ADC12TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed. The DMA is triggered after the conversion in single-channel conversion mode or after the completion of a sequence of channel conversions in sequence-of-channels conversion mode.

### 28.2.10.1 ADC12IV, Interrupt Vector Generator

All ADC12\_A interrupt sources are prioritized and combined to source a single interrupt vector. The interrupt vector register ADC12IV is used to determine which enabled ADC12\_A interrupt source requested an interrupt.

The highest-priority enabled ADC12\_A interrupt generates a number in the ADC12IV register (see register description). This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled ADC12\_A interrupts do not affect the ADC12IV value.

Any access, read or write, of the ADC12IV register automatically resets the ADC12OV condition or the ADC12TOV condition, if either was the highest-pending interrupt. Neither interrupt condition has an accessible interrupt flag. The ADC12IFGx flags are not reset by an ADC12IV access. ADC12IFGx bits are reset automatically by accessing their associated ADC12MEMx register or may be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the ADC12OV and ADC12IFG3 interrupts are pending when the interrupt service routine accesses the ADC12IV register, the ADC12OV interrupt condition is reset automatically. After the RETI instruction of the interrupt service routine is executed, the ADC12IFG3 generates another interrupt.

### 28.2.10.2 ADC12\_A Interrupt Handling Software Example

The following software example shows the recommended use of the ADC12IV and handling overhead. The ADC12IV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- ADC12IFG0 to ADC12IFG14, ADC12TOV, and ADC12OV: 16 cycles
- ADC12IFG15: 14 cycles

The interrupt handler for ADC12IFG15 shows a way to check immediately if a higher-prioritized interrupt occurred during the processing of ADC12IFG15. This saves nine cycles if another ADC12\_A interrupt is pending.

```
; Interrupt handler for ADC12.
INT_ADC12          ; Enter Interrupt Service Routine
    ADD    &ADC12IV,PC   ; Add offset to PC
    RETI               ; Vector 0: No interrupt
    JMP    ADOV         ; Vector 2: ADC overflow
    JMP    ADTOV        ; Vector 4: ADC timing overflow
    JMP    ADM0         ; Vector 6: ADC12IFG0
    ...                ; Vectors 8-32
    JMP    ADM14        ; Vector 34: ADC12IFG14
;
; Handler for ADC12IFG15 starts here. No JMP required.
;
ADM15    MOV    &ADC12MEM15,xxx      ; Move result, flag is reset
        ...
        JMP    INT_ADC12       ; Check other int pending
;
; ADC12IFG14-ADC12IFG1 handlers go here
;
ADM0    MOV    &ADC12MEM0,xxx      ; Move result, flag is reset
        ...
        RETI               ; Other instruction needed?
;
RETI               ; Return
;
ADTOV   ...                  ; Handle Conv. time overflow
        RETI               ; Return
;
ADOV    ...                  ; Handle ADCMEMx overflow
        RETI               ; Return
```

## 28.3 ADC12\_A Registers

The ADC12\_A registers are listed in [Table 28-3](#). The base address of the ADC12\_A can be found in the device-specific data sheet. The address offset of each ADC12\_A register is given in [Table 28-3](#).

---

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

---

**Table 28-3. ADC12\_A Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	ADC12CTL0	ADC12_A Control 0	Read/write	Word	0000h	<a href="#">Section 28.3.1</a>
00h	ADC12CTL0_L		Read/write	Byte	00h	
01h	ADC12CTL0_H		Read/write	Byte	00h	
02h	ADC12CTL1	ADC12_A Control 1	Read/write	Word	0000h	<a href="#">Section 28.3.2</a>
02h	ADC12CTL1_L		Read/write	Byte	00h	
03h	ADC12CTL1_H		Read/write	Byte	00h	
04h	ADC12CTL2	ADC12_A Control 2	Read/write	Word	0020h	<a href="#">Section 28.3.3</a>
04h	ADC12CTL2_L		Read/write	Byte	20h	
05h	ADC12CTL2_H		Read/write	Byte	00h	
0Ah	ADC12IFG	ADC12_A Interrupt Flag	Read/write	Word	0000h	<a href="#">Section 28.3.7</a>
0Ah	ADC12IFG_L		Read/write	Byte	00h	
0Bh	ADC12IFG_H		Read/write	Byte	00h	
0Ch	ADC12IE	ADC12_A Interrupt Enable	Read/write	Word	0000h	<a href="#">Section 28.3.6</a>
0Ch	ADC12IE_L		Read/write	Byte	00h	
0Dh	ADC12IE_H		Read/write	Byte	00h	
0Eh	ADC12IV	ADC12_A Interrupt Vector	Read	Word	0000h	<a href="#">Section 28.3.8</a>
0Eh	ADC12IV_L		Read	Byte	00h	
0Fh	ADC12IV_H		Read	Byte	00h	
20h	ADC12MEM0	ADC12_A Memory 0	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
20h	ADC12MEM0_L		Read/write	Byte	undefined	
21h	ADC12MEM0_H		Read/write	Byte	undefined	
22h	ADC12MEM1	ADC12_A Memory 1	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
22h	ADC12MEM1_L		Read/write	Byte	undefined	
23h	ADC12MEM1_H		Read/write	Byte	undefined	
24h	ADC12MEM2	ADC12_A Memory 2	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
24h	ADC12MEM2_L		Read/write	Byte	undefined	
25h	ADC12MEM2_H		Read/write	Byte	undefined	
26h	ADC12MEM3	ADC12_A Memory 3	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
26h	ADC12MEM3_L		Read/write	Byte	undefined	
27h	ADC12MEM3_H		Read/write	Byte	undefined	
28h	ADC12MEM4	ADC12_A Memory 4	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
28h	ADC12MEM4_L		Read/write	Byte	undefined	
29h	ADC12MEM4_H		Read/write	Byte	undefined	
2Ah	ADC12MEM5	ADC12_A Memory 5	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
2Ah	ADC12MEM5_L		Read/write	Byte	undefined	
2Bh	ADC12MEM5_H		Read/write	Byte	undefined	
2Ch	ADC12MEM6	ADC12_A Memory 6	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
2Ch	ADC12MEM6_L		Read/write	Byte	undefined	
2Dh	ADC12MEM6_H		Read/write	Byte	undefined	

**Table 28-3. ADC12\_A Registers (continued)**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Access</b>	<b>Reset</b>	<b>Section</b>
2Eh	ADC12MEM7	ADC12_A Memory 7	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
2Eh	ADC12MEM7_L		Read/write	Byte	undefined	
2Fh	ADC12MEM7_H		Read/write	Byte	undefined	
30h	ADC12MEM8	ADC12_A Memory 8	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
30h	ADC12MEM8_L		Read/write	Byte	undefined	
31h	ADC12MEM8_H		Read/write	Byte	undefined	
32h	ADC12MEM9	ADC12_A Memory 9	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
32h	ADC12MEM9_L		Read/write	Byte	undefined	
33h	ADC12MEM9_H		Read/write	Byte	undefined	
34h	ADC12MEM10	ADC12_A Memory 10	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
34h	ADC12MEM10_L		Read/write	Byte	undefined	
35h	ADC12MEM10_H		Read/write	Byte	undefined	
36h	ADC12MEM11	ADC12_A Memory 11	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
36h	ADC12MEM11_L		Read/write	Byte	undefined	
37h	ADC12MEM11_H		Read/write	Byte	undefined	
38h	ADC12MEM12	ADC12_A Memory 12	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
38h	ADC12MEM12_L		Read/write	Byte	undefined	
39h	ADC12MEM12_H		Read/write	Byte	undefined	
3Ah	ADC12MEM13	ADC12_A Memory 13	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
3Ah	ADC12MEM13_L		Read/write	Byte	undefined	
3Bh	ADC12MEM13_H		Read/write	Byte	undefined	
3Ch	ADC12MEM14	ADC12_A Memory 14	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
3Ch	ADC12MEM14_L		Read/write	Byte	undefined	
3Dh	ADC12MEM14_H		Read/write	Byte	undefined	
3Dh	ADC12MEM15	ADC12_A Memory 15	Read/write	Word	undefined	<a href="#">Section 28.3.4</a>
3Dh	ADC12MEM15_L		Read/write	Byte	undefined	
3Eh	ADC12MEM15_H		Read/write	Byte	undefined	
10h	ADC12MCTL0	ADC12_A Memory Control 0	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
11h	ADC12MCTL1	ADC12_A Memory Control 1	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
12h	ADC12MCTL2	ADC12_A Memory Control 2	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
13h	ADC12MCTL3	ADC12_A Memory Control 3	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
14h	ADC12MCTL4	ADC12_A Memory Control 4	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
15h	ADC12MCTL5	ADC12_A Memory Control 5	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
16h	ADC12MCTL6	ADC12_A Memory Control 6	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
17h	ADC12MCTL7	ADC12_A Memory Control 7	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
18h	ADC12MCTL8	ADC12_A Memory Control 8	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
19h	ADC12MCTL9	ADC12_A Memory Control 9	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
1Ah	ADC12MCTL10	ADC12_A Memory Control 10	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
1Bh	ADC12MCTL11	ADC12_A Memory Control 11	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
1Ch	ADC12MCTL12	ADC12_A Memory Control 12	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
1Dh	ADC12MCTL13	ADC12_A Memory Control 13	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
1Eh	ADC12MCTL14	ADC12_A Memory Control 14	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>
1Fh	ADC12MCTL15	ADC12_A Memory Control 15	Read/write	Byte	undefined	<a href="#">Section 28.3.5</a>

**28.3.1 ADC12CTL0 Register**

ADC12\_A Control Register 0

**Figure 28-13. ADC12CTL0 Register**

15	14	13	12	11	10	9	8
ADC12SHT1x				ADC12SHT0x			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12MSC	ADC12REF2_5V	ADC12REFON	ADC12ON	ADC12OVIE	ADC12TOVIE	ADC12ENC	ADC12SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
Can be modified only when ADC12ENC = 0							

**Table 28-4. ADC12CTL0 Register Description**

Bit	Field	Type	Reset	Description
15-12	ADC12SHT1x	RW	0h	ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM15.
11-8	ADC12SHT0x	RW	0h	ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7. 0000b = 4 ADC12CLK cycles 0001b = 8 ADC12CLK cycles 0010b = 16 ADC12CLK cycles 0011b = 32 ADC12CLK cycles 0100b = 64 ADC12CLK cycles 0101b = 96 ADC12CLK cycles 0110b = 128 ADC12CLK cycles 0111b = 192 ADC12CLK cycles 1000b = 256 ADC12CLK cycles 1001b = 384 ADC12CLK cycles 1010b = 512 ADC12CLK cycles 1011b = 768 ADC12CLK cycles 1100b = 1024 ADC12CLK cycles 1101b = 1024 ADC12CLK cycles 1110b = 1024 ADC12CLK cycles 1111b = 1024 ADC12CLK cycles
7	ADC12MSC	RW	0h	ADC12_A multiple sample and conversion. Valid only for sequence or repeated modes. 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	ADC12REF2_5V	RW	0h	ADC12_A reference generator voltage. ADC12REFON must also be set. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = 1.5 V 1b = 2.5 V
5	ADC12REFON	RW	0h	ADC12_A reference generator on. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = Reference off 1b = Reference on

**Table 28-4. ADC12CTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
4	ADC12ON	RW	0h	ADC12_A on 0b = ADC12_A off 1b = ADC12_A on
3	ADC12OVIE	RW	0h	ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt. 0b = Overflow interrupt disabled 1b = Overflow interrupt enabled
2	ADC12TOVIE	RW	0h	ADC12_A conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0b = Conversion time overflow interrupt disabled 1b = Conversion time overflow interrupt enabled
1	ADC12ENC	RW	0h	ADC12_A enable conversion 0b = ADC12_A disabled 1b = ADC12_A enabled
0	ADC12SC	RW	0h	ADC12_A start conversion. Software-controlled sample-and-conversion start. ADC12SC and ADC12ENC may be set together with one instruction. ADC12SC is reset automatically. 0b = No sample-and-conversion-start 1b = Start sample-and-conversion

**28.3.2 ADC12CTL1 Register**

ADC12\_A Control Register 1

**Figure 28-14. ADC12CTL1 Register**

15	14	13	12	11	10	9	8
ADC12CSTARTADDx				ADC12SHSx		ADC12SHP	ADC12ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12DIVx				ADC12SSELx		ADC12CONSEQx	ADC12BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

Can be modified only when ADC12ENC = 0

**Table 28-5. ADC12CTL1 Register Description**

Bit	Field	Type	Reset	Description
15-12	ADC12CSTARTADDx	RW	0h	ADC12_A conversion start address. These bits select which ADC12_A conversion-memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15.
11-10	ADC12SHSx	RW	0h	ADC12_A sample-and-hold source select 00b = ADC12SC bit 01b = Timer source (see device-specific data sheet for exact timer and locations) 10b = Timer source (see device-specific data sheet for exact timer and locations) 11b = Timer source (see device-specific data sheet for exact timer and locations)
9	ADC12SHP	RW	0h	ADC12_A sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. 0b = SAMPCON signal is sourced from the sample-input signal. 1b = SAMPCON signal is sourced from the sampling timer.
8	ADC12ISSH	RW	0h	ADC12_A invert signal sample-and-hold 0b = The sample-input signal is not inverted. 1b = The sample-input signal is inverted.
7-5	ADC12DIVx	RW	0h	ADC12_A clock divider 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
4-3	ADC12SSELx	RW	0h	ADC12_A clock source select 00b = ADC12OSC (MODOSC) 01b = ACLK 10b = MCLK 11b = SMCLK
2-1	ADC12CONSEQx	RW	0h	ADC12_A conversion sequence mode select 00b = Single-channel, single-conversion 01b = Sequence-of-channels 10b = Repeat-single-channel 11b = Repeat-sequence-of-channels
0	ADC12BUSY	R	0h	ADC12_A busy. This bit indicates an active sample or conversion operation. 0b = No operation is active. 1b = A sequence, sample, or conversion is active.

### 28.3.3 ADC12CTL2 Register

ADC12\_A Control Register 2

**Figure 28-15. ADC12CTL2 Register**

15	14	13	12	11	10	9	8
Reserved							ADC12PDIV
r-0	r-0	r-0	r-0	r-0	r-0	r-0	rw-0
7	6	5	4	3	2	1	0
ADC12TCOFF	Reserved	ADC12RES		ADC12DF	ADC12SR	ADC12REFOUT	ADC12REFBURST
rw-(0)	r-0	rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ADC12ENC = 0

**Table 28-6. ADC12CTL2 Register Description**

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	ADC12PDIV	RW	0h	ADC12_A predivider. This bit predivides the selected ADC12_A clock source. 0b = Predivide by 1 1b = Predivide by 4
7	ADC12TCOFF	RW	0h	ADC12_A temperature sensor off. If the bit is set, the temperature sensor turned off. This is used to save power. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = Temperature sensor on 1b = Temperature sensor off
6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	ADC12RES	RW	2h	ADC12_A resolution. This bit defines the conversion result resolution. 00b = 8 bit (9 clock cycle conversion time) 01b = 10 bit (11 clock cycle conversion time) 10b = 12 bit (13 clock cycle conversion time) 11b = Reserved
3	ADC12DF	RW	0h	ADC12_A data read-back format. Data is always stored in the binary unsigned format. 0b = Binary unsigned. Theoretically, the analog input voltage -VREF results in 0000h, the analog input voltage +VREF results in 0FFFh. 1b = Signed binary (2s complement), left aligned. Theoretically, the analog input voltage -VREF results in 8000h, the analog input voltage +VREF results in 7FF0h.
2	ADC12SR	RW	0h	ADC12_A sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC12SR reduces the current consumption of the reference buffer. 0b = Reference buffer supports up to approximately 200 kspis. 1b = Reference buffer supports up to approximately 50 kspis.
1	ADC12REFOUT	RW	0h	Reference output. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = Reference output off 1b = Reference output on
0	ADC12REFBURST	RW	0h	Reference burst 0b = Reference buffer on continuously 1b = Reference buffer on only during sample-and-conversion

**28.3.4 ADC12MEMx Register**

ADC12\_A Conversion Memory Register

**Figure 28-16. ADC12MEMx 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	rw	rw	rw	rw

**Table 28-7. ADC12MEMx Register Description**

Bit	Field	Type	Reset	Description
15-0	Conversion Results	RW	undefined	Binary unsigned format: This data format is used if ADC12DF = 0. The 12-bit conversion results are right justified. Bit 11 is the MSB. Bits 15–12 are 0 in 12-bit mode, bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. Writing to the conversion memory registers corrupts the results.  2s-complement format: This data format is used if ADC12DF = 1. The 12-bit conversion results are left justified, 2s-complement format. Bit 15 is the MSB. Bits 3–0 are 0 in 12-bit mode, bits 5–0 are 0 in 10-bit mode, and bits 7–0 are 0 in 8-bit mode. The data is stored in the right-justified format and is converted to the left-justified 2s-complement format during read back.

### 28.3.5 ADC12MCTLx Register

ADC12\_A Conversion Memory Control Register

**Figure 28-17. ADC12MCTLx Register**

7	6	5	4	3	2	1	0
ADC12EOS	ADC12SREFx				ADC12INCHx		
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when ADC12ENC = 0

**Table 28-8. ADC12MCTLx Register Description**

Bit	Field	Type	Reset	Description
7	ADC12EOS	RW	0h	End of sequence. Indicates the last conversion in a sequence. 0b = Not end of sequence 1b = End of sequence
6-4	ADC12SREFx	RW	0h	Select reference 000b = V(R+) = AVCC and V(R-) = AVSS 001b = V(R+) = VREF+ and V(R-) = AVSS 010b = V(R+) = VeREF+ and V(R-) = AVSS 011b = V(R+) = VeREF+ and V(R-) = AVSS 100b = V(R+) = AVCC and V(R-) = VREF-/VeREF- 101b = V(R+) = VREF+ and V(R-) = VREF-/VeREF- 110b = V(R+) = VeREF+ and V(R-) = VREF-/VeREF- 111b = V(R+) = VeREF+ and V(R-) = VREF-/VeREF-
3-0	ADC12INCHx	RW	0h	Input channel select 0000b = A0 0001b = A1 0010b = A2 0011b = A3 0100b = A4 0101b = A5 0110b = A6 0111b = A7 1000b = VeREF+ 1001b = VREF-/VeREF- 1010b = Temperature diode 1011b = (AVCC – AVSS) / 2 1100b = A12. On devices with the Battery Backup System, VBAT can be measured internally by the ADC. 1101b = A13 1110b = A14 1111b = A15

**28.3.6 ADC12IE Register**

ADC12\_A Interrupt Enable Register

**Figure 28-18. ADC12IE Register**

15	14	13	12	11	10	9	8
ADC12IE15	ADC12IE14	ADC12IE13	ADC12IE12	ADC12IE11	ADC12IE10	ADC12IE9	ADC12IE8
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IE7	ADC12IE6	ADC12IE5	ADC12IE4	ADC12IE3	ADC12IE2	ADC12IE1	ADC12IE0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 28-9. ADC12IE Register Description**

Bit	Field	Type	Reset	Description
15	ADC12IE15	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG15 bit. 0b = Interrupt disabled 1b = Interrupt enabled
14	ADC12IE14	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG14 bit. 0b = Interrupt disabled 1b = Interrupt enabled
13	ADC12IE13	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG13 bit. 0b = Interrupt disabled 1b = Interrupt enabled
12	ADC12IE12	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG12 bit. 0b = Interrupt disabled 1b = Interrupt enabled
11	ADC12IE11	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG11 bit. 0b = Interrupt disabled 1b = Interrupt enabled
10	ADC12IE10	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG10 bit. 0b = Interrupt disabled 1b = Interrupt enabled
9	ADC12IE9	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG9 bit. 0b = Interrupt disabled 1b = Interrupt enabled
8	ADC12IE8	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG8 bit. 0b = Interrupt disabled 1b = Interrupt enabled
7	ADC12IE7	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG7 bit. 0b = Interrupt disabled 1b = Interrupt enabled
6	ADC12IE6	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG6 bit. 0b = Interrupt disabled 1b = Interrupt enabled

**Table 28-9. ADC12IE Register Description (continued)**

Bit	Field	Type	Reset	Description
5	ADC12IE5	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG5 bit. 0b = Interrupt disabled 1b = Interrupt enabled
4	ADC12IE4	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG4 bit. 0b = Interrupt disabled 1b = Interrupt enabled
3	ADC12IE3	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG3 bit. 0b = Interrupt disabled 1b = Interrupt enabled
2	ADC12IE2	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG2 bit. 0b = Interrupt disabled 1b = Interrupt enabled
1	ADC12IE1	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG1 bit. 0b = Interrupt disabled 1b = Interrupt enabled
0	ADC12IE0	RW	0h	Interrupt enable. This bit enables or disables the interrupt request for the ADC12IFG0 bit. 0b = Interrupt disabled 1b = Interrupt enabled

**28.3.7 ADC12IFG Register**

ADC12\_A Interrupt Flag Register

**Figure 28-19. ADC12IFG Register**

15	14	13	12	11	10	9	8
ADC12IFG15	ADC12IFG14	ADC12IFG13	ADC12IFG12	ADC12IFG11	ADC12IFG10	ADC12IFG9	ADC12IFG8
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IFG7	ADC12IFG6	ADC12IFG5	ADC12IFG4	ADC12IFG3	ADC12IFG2	ADC12IFG1	ADC12IFG0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 28-10. ADC12IFG Register Description**

Bit	Field	Type	Reset	Description
15	ADC12IFG15	RW	0h	ADC12MEM15 interrupt flag. This bit is set when ADC12MEM15 is loaded with a conversion result. This bit is reset if the ADC12MEM15 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
14	ADC12IFG14	RW	0h	ADC12MEM14 interrupt flag. This bit is set when ADC12MEM14 is loaded with a conversion result. This bit is reset if the ADC12MEM14 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
13	ADC12IFG13	RW	0h	ADC12MEM13 interrupt flag. This bit is set when ADC12MEM13 is loaded with a conversion result. This bit is reset if the ADC12MEM13 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
12	ADC12IFG12	RW	0h	ADC12MEM12 interrupt flag. This bit is set when ADC12MEM12 is loaded with a conversion result. This bit is reset if the ADC12MEM12 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
11	ADC12IFG11	RW	0h	ADC12MEM11 interrupt flag. This bit is set when ADC12MEM11 is loaded with a conversion result. This bit is reset if the ADC12MEM11 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
10	ADC12IFG10	RW	0h	ADC12MEM10 interrupt flag. This bit is set when ADC12MEM10 is loaded with a conversion result. This bit is reset if the ADC12MEM10 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
9	ADC12IFG9	RW	0h	ADC12MEM9 interrupt flag. This bit is set when ADC12MEM9 is loaded with a conversion result. This bit is reset if the ADC12MEM9 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
8	ADC12IFG8	RW	0h	ADC12MEM8 interrupt flag. This bit is set when ADC12MEM8 is loaded with a conversion result. This bit is reset if the ADC12MEM8 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending

**Table 28-10. ADC12IFG Register Description (continued)**

<b>Bit</b>	<b>Field</b>	<b>Type</b>	<b>Reset</b>	<b>Description</b>
7	ADC12IFG7	RW	0h	ADC12MEM7 interrupt flag. This bit is set when ADC12MEM7 is loaded with a conversion result. This bit is reset if the ADC12MEM7 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
6	ADC12IFG6	RW	0h	ADC12MEM6 interrupt flag. This bit is set when ADC12MEM6 is loaded with a conversion result. This bit is reset if the ADC12MEM6 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
5	ADC12IFG5	RW	0h	ADC12MEM5 interrupt flag. This bit is set when ADC12MEM5 is loaded with a conversion result. This bit is reset if the ADC12MEM5 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
4	ADC12IFG4	RW	0h	ADC12MEM4 interrupt flag. This bit is set when ADC12MEM4 is loaded with a conversion result. This bit is reset if the ADC12MEM4 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
3	ADC12IFG3	RW	0h	ADC12MEM3 interrupt flag. This bit is set when ADC12MEM3 is loaded with a conversion result. This bit is reset if the ADC12MEM3 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
2	ADC12IFG2	RW	0h	ADC12MEM2 interrupt flag. This bit is set when ADC12MEM2 is loaded with a conversion result. This bit is reset if the ADC12MEM2 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
1	ADC12IFG1	RW	0h	ADC12MEM1 interrupt flag. This bit is set when ADC12MEM1 is loaded with a conversion result. This bit is reset if the ADC12MEM1 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending
0	ADC12IFG0	RW	0h	ADC12MEM0 interrupt flag. This bit is set when ADC12MEM0 is loaded with a conversion result. This bit is reset if the ADC12MEM0 is accessed, or it may be reset with software. 0b = No interrupt pending 1b = Interrupt pending

**28.3.8 ADC12IV Register**

ADC12\_A Interrupt Vector Register

**Figure 28-20. ADC12IV Register**

15	14	13	12	11	10	9	8
ADC12IVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
ADC12IVx							
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

**Table 28-11. ADC12IV Register Description**

Bit	Field	Type	Reset	Description
15-0	ADC12IVx	R	0h	ADC12_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: ADC12MEMx overflow; Interrupt Flag: –; Interrupt Priority: Highest 04h = Interrupt Source: Conversion time overflow; Interrupt Flag: – 06h = Interrupt Source: ADC12MEM0 interrupt flag; Interrupt Flag: ADC12IFG0 08h = Interrupt Source: ADC12MEM1 interrupt flag; Interrupt Flag: ADC12IFG1 0Ah = Interrupt Source: ADC12MEM2 interrupt flag; Interrupt Flag: ADC12IFG2 0Ch = Interrupt Source: ADC12MEM3 interrupt flag; Interrupt Flag: ADC12IFG3 0Eh = Interrupt Source: ADC12MEM4 interrupt flag; Interrupt Flag: ADC12IFG4 10h = Interrupt Source: ADC12MEM5 interrupt flag; Interrupt Flag: ADC12IFG5 12h = Interrupt Source: ADC12MEM6 interrupt flag; Interrupt Flag: ADC12IFG6 14h = Interrupt Source: ADC12MEM7 interrupt flag; Interrupt Flag: ADC12IFG7 16h = Interrupt Source: ADC12MEM8 interrupt flag; Interrupt Flag: ADC12IFG8 18h = Interrupt Source: ADC12MEM9 interrupt flag; Interrupt Flag: ADC12IFG9 1Ah = Interrupt Source: ADC12MEM10 interrupt flag; Interrupt Flag: ADC12IFG10 1Ch = Interrupt Source: ADC12MEM11 interrupt flag; Interrupt Flag: ADC12IFG11 1Eh = Interrupt Source: ADC12MEM12 interrupt flag; Interrupt Flag: ADC12IFG12 20h = Interrupt Source: ADC12MEM13 interrupt flag; Interrupt Flag: ADC12IFG13 22h = Interrupt Source: ADC12MEM14 interrupt flag; Interrupt Flag: ADC12IFG14 24h = Interrupt Source: ADC12MEM15 interrupt flag; Interrupt Flag: ADC12IFG15; Interrupt Priority: Lowest

The SD24\_B is a multi-input, multi-converter sigma-delta analog-to-digital conversion module. This chapter describes the operation of the SD24\_B module.

Topic	Page
29.1 SD24_B Introduction .....	754
29.2 SD24_B Operation .....	758
29.3 SD24_B Registers .....	770

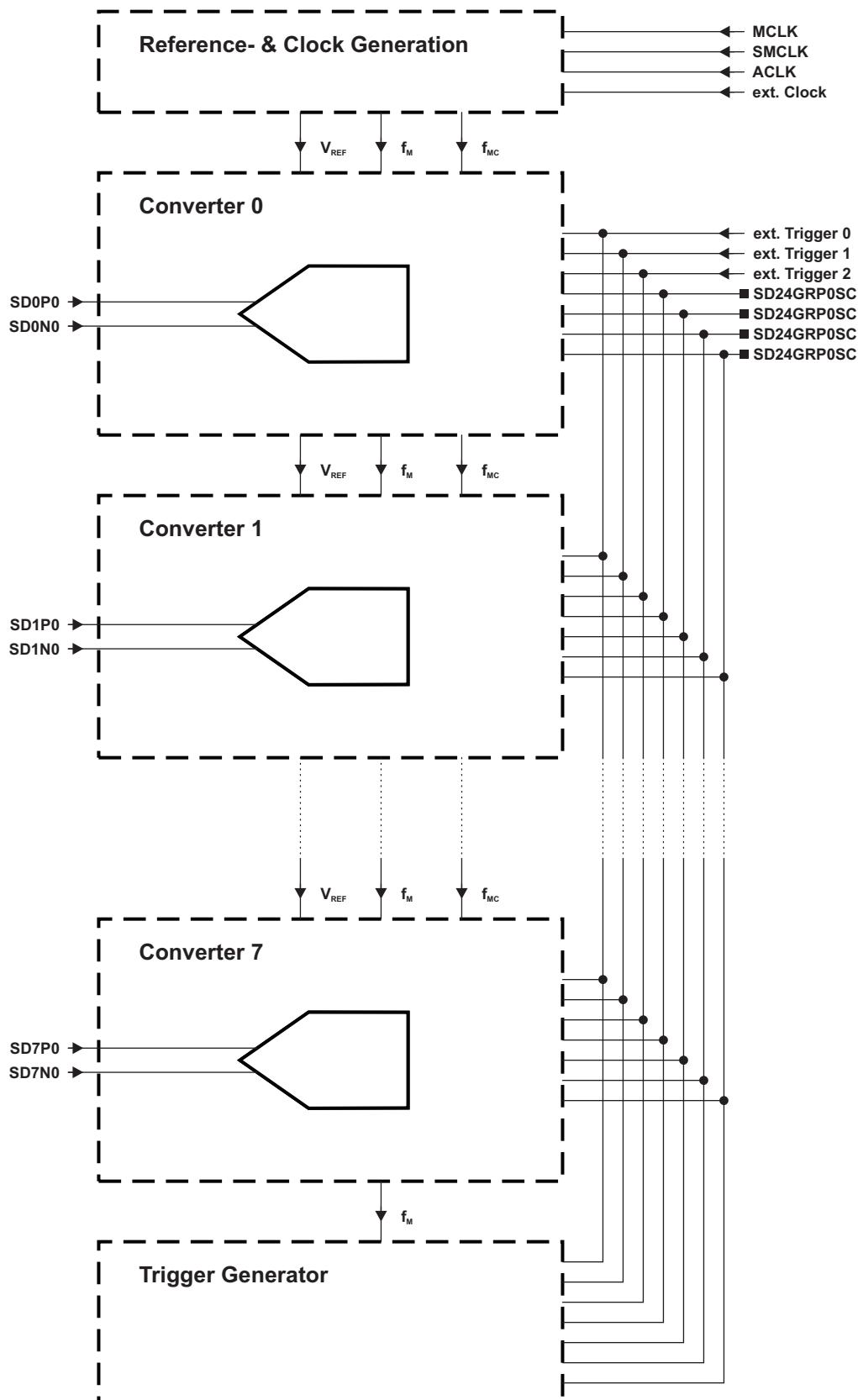
## 29.1 SD24\_B Introduction

The SD24\_B module consists of up to eight independent sigma-delta analog-to-digital converters. The converters are based on second-order oversampling sigma-delta modulators and digital decimation filters. The decimation filters are comb type filters with selectable oversampling ratios of up to 1024. Additional filtering can be done in software.

Features of the SD24\_B include:

- 2nd order sigma-delta architecture
- Up to eight independent, simultaneously-sampling ADCs. (The number of converters is device dependent, see the device-specific data sheet.)

An overview block diagram of the SD24\_B module is shown in [Figure 29-1](#). The block diagram of the voltage reference- and clock generation circuitry is shown in [Figure 29-2](#). The reference voltage generation is located in the shared reference module (see the device-specific data sheet). The converter specific block diagram is shown in [Figure 29-3](#).


**Figure 29-1. SD24\_B Overview Block Diagram**

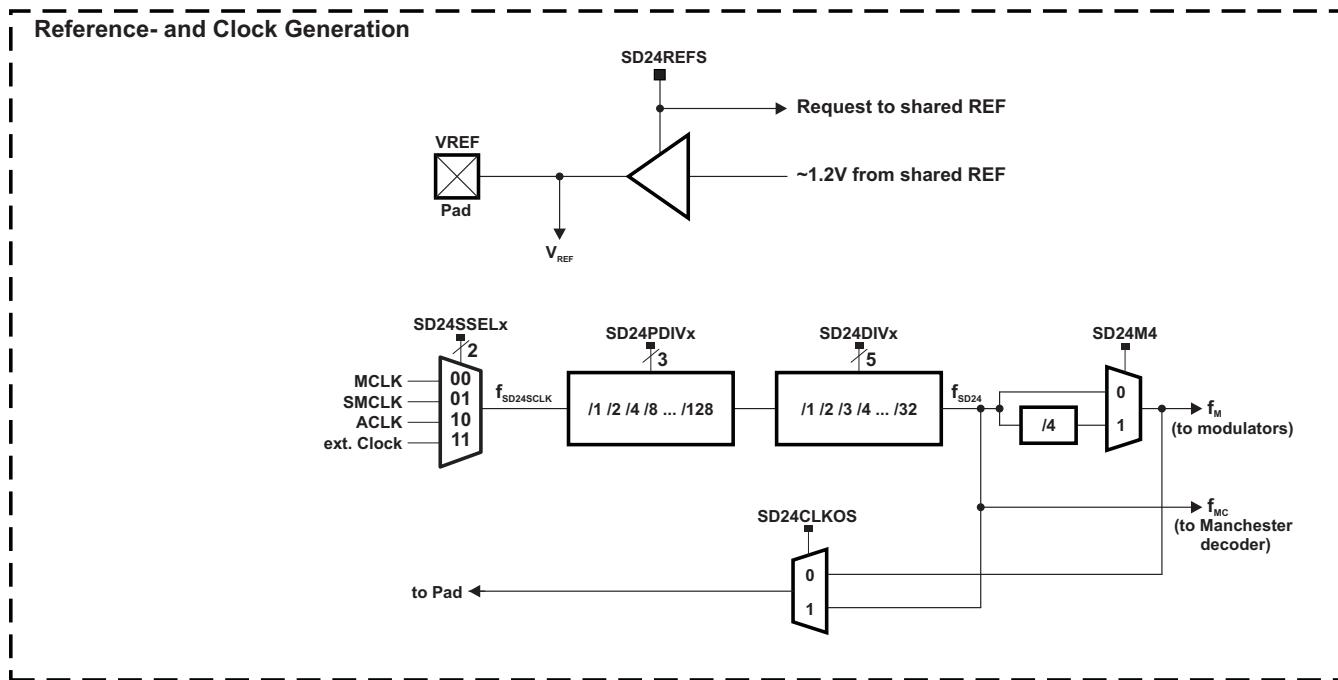
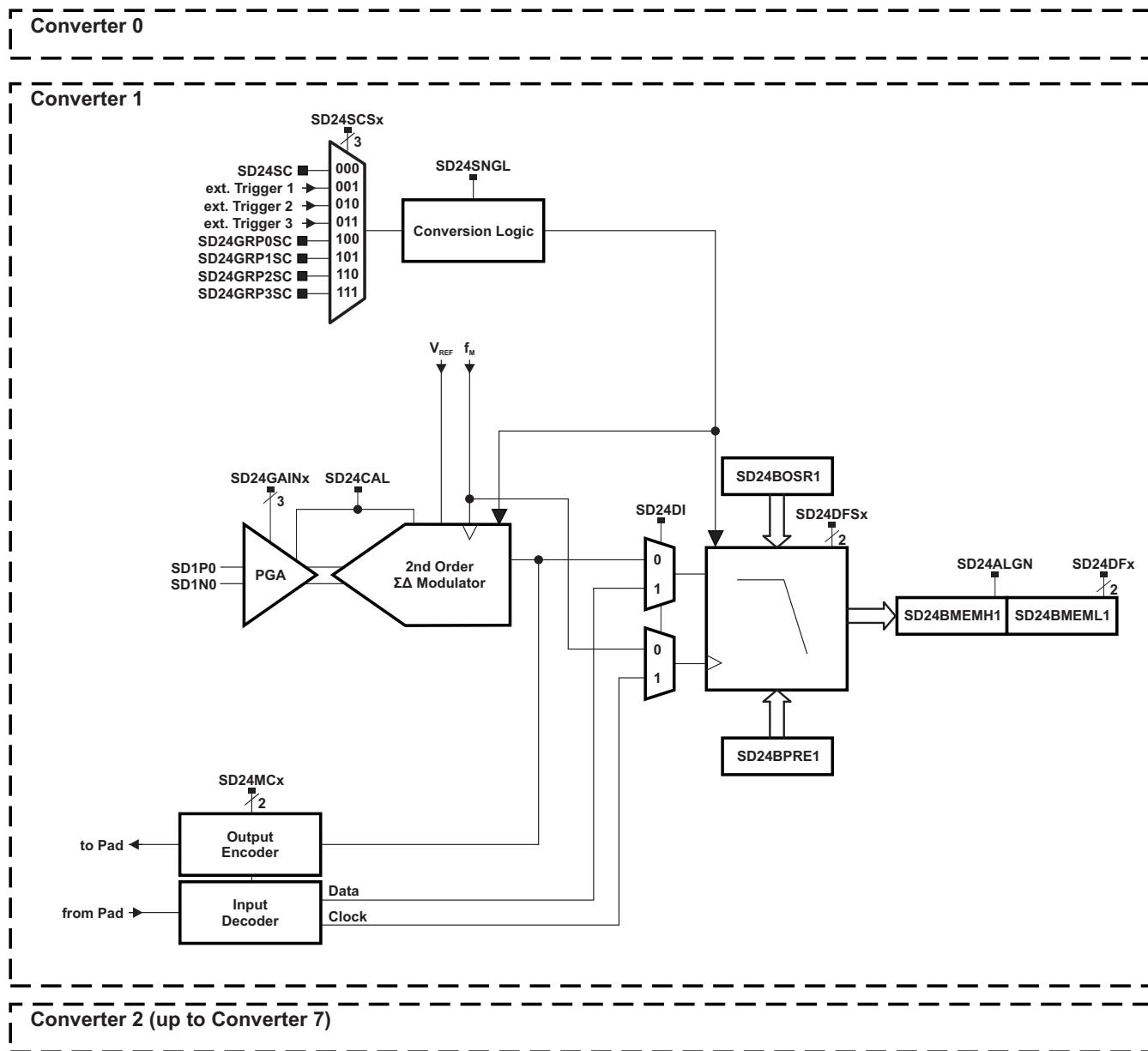


Figure 29-2. SD24\_B Reference- and Clock Generation Block Diagram



**Figure 29-3. SD24\_B Converter Block Diagram**

## 29.2 SD24\_B Operation

The SD24\_B module is configured with user software. The setup and operation of the SD24\_B is discussed in the following sections.

### 29.2.1 Principle of Operation

A sigma-delta analog-to-digital converter basically consists of two parts: the analog part (called the modulator) and the digital part (called the decimation filter). The modulator of the SD24\_B provides a bit stream of zeros and ones to the digital decimation filter. The digital filter averages the bitstream from the modulator over a given number of bits (specified by the oversampling rate) and provides samples at a reduced rate for further processing to the CPU.

Averaging can be used to increase the signal-to-noise performance of a conversion (see Figure 29-4 a and b). With a conventional ADC, each factor-of-4 oversampling can improve the SNR by approximately 6 dB or 1 bit. To achieve a 16-bit resolution out of a simple 1-bit ADC would require an impractical oversampling rate of  $4^{15} = 1.073.741.824$ . To overcome this limitation the sigma-delta modulator implements a technique called noise-shaping—due to an implemented feedback loop and integrators, the quantization noise is pushed to higher frequencies and, thus, much lower oversampling rates are sufficient to achieve high resolutions (see Figure 29-4 c).

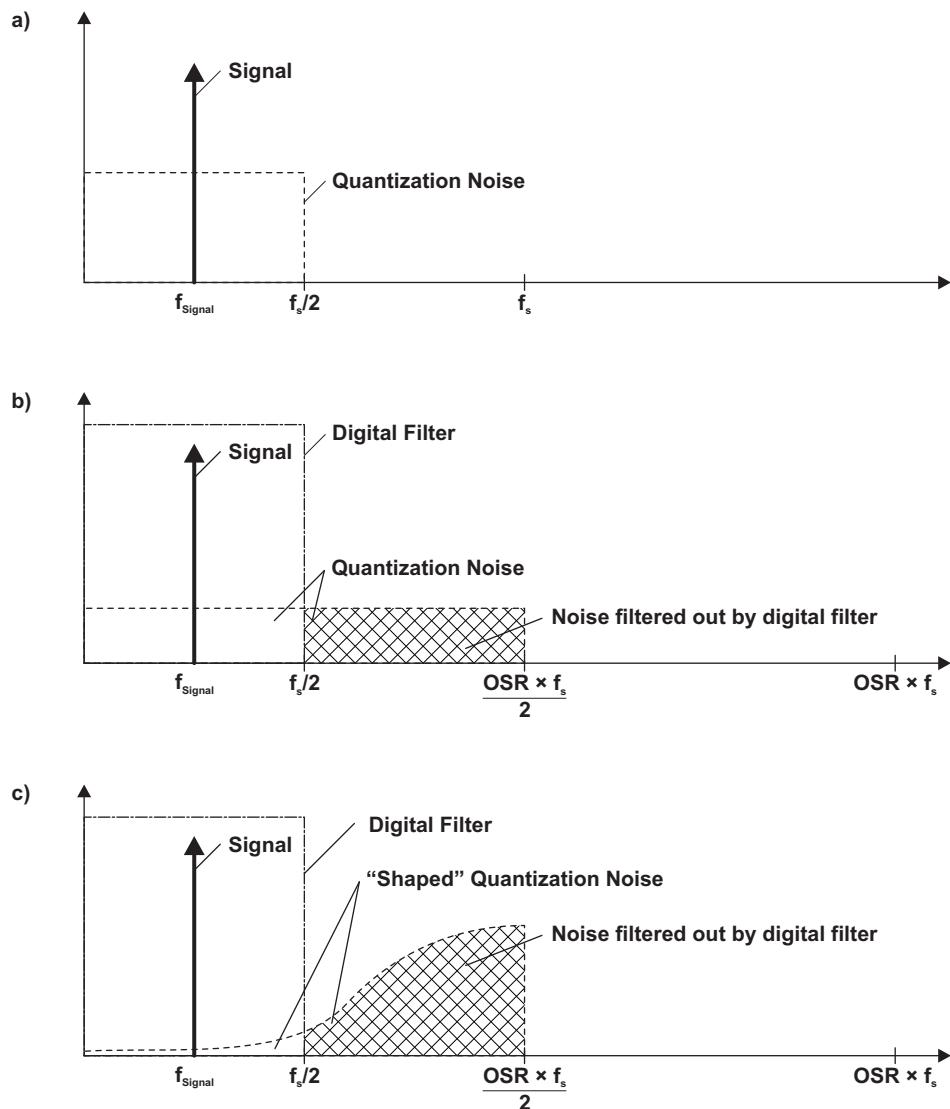


Figure 29-4. Sigma-Delta Principle

## 29.2.2 ADC Core

The analog-to-digital conversion is performed by a 1-bit second-order sigma-delta modulator. A single-bit comparator within the modulator quantizes the input signal with the modulator frequency  $f_M$ . The resulting 1-bit data stream is averaged by the digital filter for the conversion result.

## 29.2.3 Voltage Reference

The SD24\_B module can use an external reference voltage (SD24REFS = 0) or an internal reference (SD24REFS = 1). It uses the shared reference (REF) module as its internal voltage reference source. REF is used for each SD24\_B converter when requested with SD24REFS = 1. When using the internal reference, an external 100-nF capacitor connected from VREF to AVSS is recommended to reduce noise. See the device-specific data sheet for parameters.

An external voltage reference must be applied to the VREF input when SD24REFS = 0.

## 29.2.4 Modulator Clock

The modulator clock is generated globally for all modulators so that all modulators convert synchronously.

---

### NOTE: Using an FLL-based Clock

When an FLL-based clock with enabled clock modulation (DISMOD = 0 in UCSCTL1) is selected to clock the sigma-delta modulator, the clock that is provided to the modulator must be DCOCLK divided at least by 2. The division can be done in the FLL itself by selecting DCOCLKDIV instead of DCOCLK as the clock source and setting the FLLD bits to a non-zero value, by using the clock system clock dividers DIVM, DIVS, or DIVA corresponding to the sigma-delta input clock MCLK, SMCLK, or ACLK, respectively, or by using the clock divider build into the sigma-delta module.

---

## 29.2.5 Auto Power-Down

The SD24\_B is designed for low-power applications. When a SD24\_B converter is not actively converting, it is automatically disabled; it is automatically re-enabled when a conversion is started. When a converter is disabled, it consumes no current.

## 29.2.6 Analog Inputs

### 29.2.6.1 Analog Input Range and PGA

The full-scale input voltage range for each analog input pair is dependent on the gain setting of each converter. See the device-specific data sheet for full-scale input specifications.

### 29.2.6.2 Analog Input Setup

The analog input of each converter is configured using the SD24BINCTLx register. These settings can be independently configured for each SD24\_B converter.

The gain for each PGA is selected by the SD24GAINx bits. A total of eight gain settings are available.

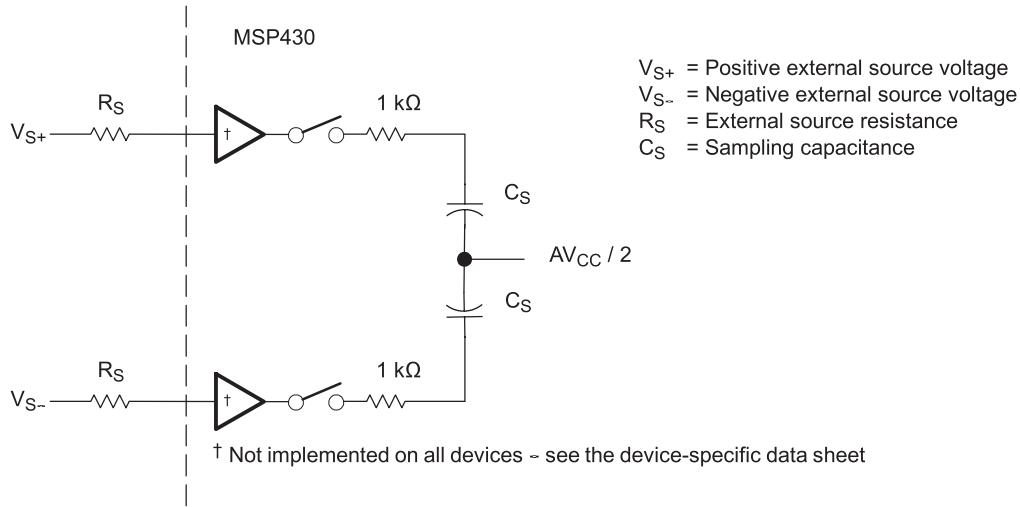
During conversion, any modification of the SD24BINCTLx register becomes effective with the next decimation step of the digital filter. After these bits are modified, the next three conversions may be invalid due to the settling time of the digital filter. This can be handled automatically with the SD24INTDLYx bits. When SD24INTDLYx = 00b, conversion interrupt requests do not begin until the fourth conversion after a start condition.

An external RC anti-aliasing filter is recommended for the SD24\_B to prevent aliasing of the input signal. The cutoff frequency should be less than 10 kHz for a 1 MHz modulator clock and OSR = 256. The cutoff frequency may be set to a lower frequency for applications that have lower bandwidth requirements.

With SD24CAL = 1, the offset of the converter can be measured and calibrated in software afterwards.

### 29.2.6.3 Analog Input Characteristics

The SD24\_B uses a switched-capacitor input stage that appears as an impedance to external circuitry (see Figure 29-5).



**Figure 29-5. Analog Input Equivalent Circuit**

The maximum modulator frequency  $f_M$  may be calculated from the minimum settling time  $t_{Settling}$  of the sampling circuit.

$$t_{Settling} \geq (R_S + 1\text{k}\Omega) \times C_S \times \ln \left( \frac{\text{GAIN} \times 2^{17} \times V_{Ax}}{V_{REF}} \right) \quad (12)$$

Where

$$f_M = \left( \frac{1}{2 \times t_{settling}} \right) \text{ and } V_{Ax} = \max \left( \left| \frac{AV_{CC}}{2} - V_{S+} \right|, \left| \frac{AV_{CC}}{2} - V_{S-} \right| \right) \quad (13)$$

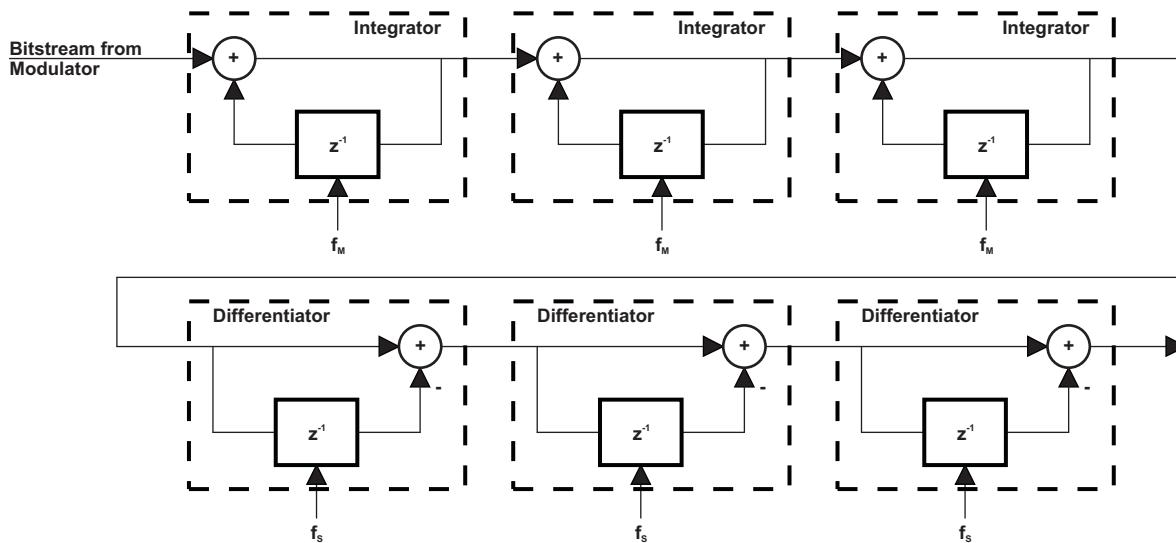
The sampling capacitor  $C_S$  varies with the gain setting. See the device-specific data sheet for parameters.

### 29.2.7 Digital Filter

The digital filter processes the 1-bit data stream from the modulator.

#### 29.2.7.1 SINC<sup>3</sup> Filter

The structure of a SINC<sup>3</sup> filter is shown in Figure 29-6.



**Figure 29-6. SINC<sup>3</sup> Filter Structure**

The transfer function is described in the z-domain by:

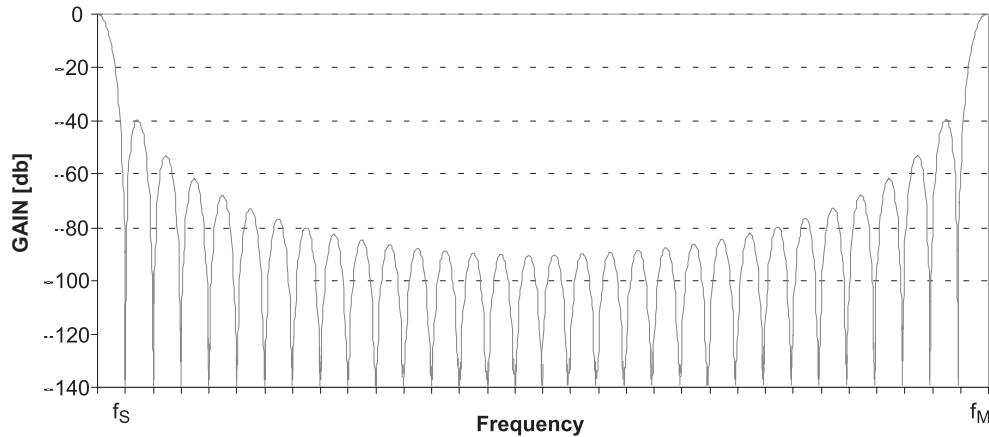
$$H(z) = \left( \frac{1}{OSR} \times \frac{1 - z^{-OSR}}{1 - z^{-1}} \right)^3 \quad (14)$$

The transfer function is described in the frequency domain by:

$$H(f) = \left( \frac{\text{sinc}\left(\text{OSR}\pi \frac{f}{f_M}\right)}{\text{sinc}\left(\pi \frac{f}{f_M}\right)} \right)^3 = \left( \frac{1}{OSR} \times \frac{\sin\left(\text{OSR} \times \pi \times \frac{f}{f_M}\right)}{\sin\left(\pi \times \frac{f}{f_M}\right)} \right)^3 \quad (15)$$

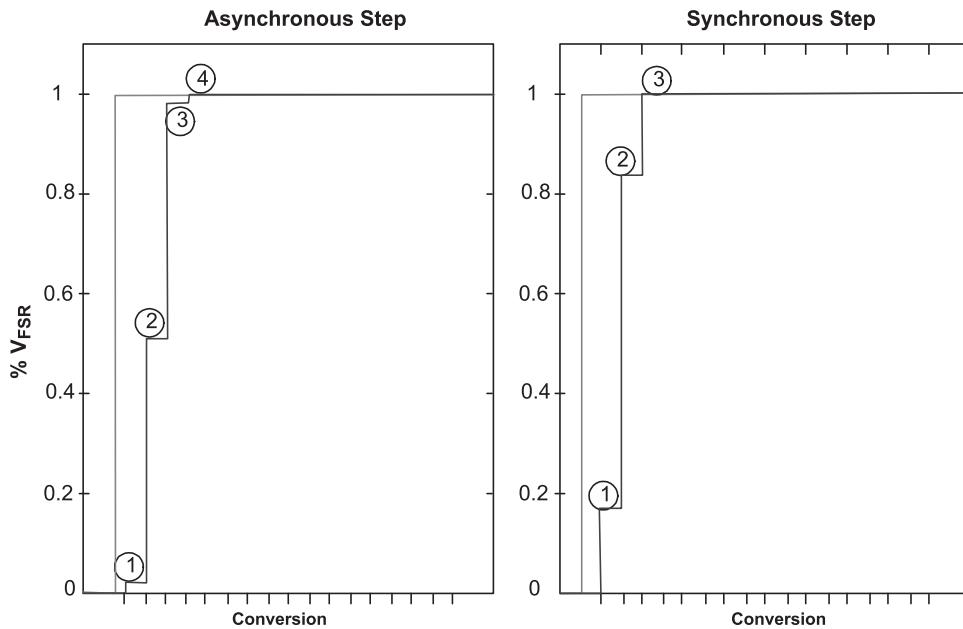
where the oversampling rate, OSR, is the ratio of the modulator frequency  $f_M$  to the sample frequency  $f_s$ . Figure 29-7 shows the filter's frequency response for an OSR of 32. The first filter notch is always at  $f_s = f_M/\text{OSR}$ . The notch's frequency can be adjusted by changing the modulator's frequency,  $f_M$ , using SD24SSELx, SD24PDIVx, and SD24DIVx or by adjusting the oversampling rate using the SD24BOSRx registers.

The digital filter for each enabled ADC converter completes the decimation of the digital bit stream and outputs new conversion results to the corresponding SD24BMEMx register at the sample frequency  $f_s$ .



**Figure 29-7. Comb Filter's Frequency Response With OSR = 32**

Figure 29-8 shows the digital filter step response and conversion points. For step changes at the input after start of conversion, a settling time must be allowed before a valid conversion result is available. The SD24INTDLYx bits can provide sufficient filter settling time for a full-scale change at the ADC input. If the step occurs synchronously to the decimation of the digital filter, the valid data is available on the third conversion. An asynchronous step requires one additional conversion before valid data is available.



**Figure 29-8. Digital Filter Step Response and Conversion Points Digital Filter Output**

#### 29.2.7.2 Cascade of Integrators

With each start of conversion or after change of the SD24INCTLx register for a given converter, the digital filter of this converter is reset. This allows the integrators implemented into the SINC filter to be used as a "Cascade of Integrators" by setting the interrupt delay to SD24INTDLYx = 11b and starting a new conversion (for example by writing SD24INCTLx) after each decimation step.

The full-scale value output by the "Cascade of Integrators" is given by  $FS = 1/6 \times (OSR^3 + 3 \times OSR^2 + 2 \times OSR)$ . In offset binary mode, the full-scale range is from 0 to FS. In 2s complement mode, the full-scale range is from -FS to +FS.

#### 29.2.7.3 Digital Filter Output

The full-scale value output by the SINC<sup>3</sup> digital filter is dependent on the oversampling ratio OSR and is given by  $FS = 2^{(3 \times \log_2(OSR))}$ . In offset binary mode, the full-scale range is from 0 to FS. In 2s complement mode, the full-scale range is from -FS to +FS.

For example, the maximum OSR of 1024 results in a full-scale value of 1073741824 or 4000 000h; that is, a value that can be represented by 31 bits. All 31 bits can be accessed using the SD24BMEMHx and SD24BMEMLx registers. Depending on the selected data format and alignment, the representation within the SD24BMEMx registers differs.

In offset binary mode (SD24DFx = 00b), the bit stream coming from the modulator is interpreted as a stream of ones (1) and zeros (0). In 2s complement mode (SD24DFx = 01b), the bit stream is interpreted as ones (1) and minus ones (-1). In offset binary mode (SD24DFx = 00b) the output values range from 0 to FS. In 2s complement mode (SD24DFx = 01b), the output values range from -FS to +FS.

If right alignment is selected with SD24ALGN = 0, the digital filter's output value is directly mapped to the SD24\_B conversion result registers SD24BMEMHx and SD24BMEMLx with the LSB of the filter's output being mapped to the LSB of SD24BMEMLx.

If left alignment is selected with SD24ALGN = 1, the mapping of the filter's output value to the conversion registers is dependent on the selected oversampling rate. In left aligned offset binary mode, the 32-bit value is left-shifted according to . In left aligned 2s complement mode, the 32-bit 2s complement of the digital filter's output is left-shifted according to [Table 29-2](#).

[Table 29-3](#) gives an example for an oversampling rate of 256.

**Table 29-1. Offset Binary Left Aligned Mapping**

OSR Range	SD24BOSRx Register	Filter Output Left Shifted by	Filter's LSB Mapped to
1 to 32	0000h to 001Fh	17 bits	Bit 0 of SD24BMEMHx
33 to 64	0020h to 003Fh	14 bits	Bit 13 of SD24BMEMLx
65 to 128	0040h to 007Fh	11 bits	Bit 10 of SD24BMEMLx
129 to 256	0080h to 00FFh	8 bits	Bit 7 of SD24BMEMLx
257 to 512	0100h to 01FFh	5 bits	Bit 4 of SD24BMEMLx
513 to 1024	0200h to 03FFh	2 bits	Bit 1 of SD24BMEMLx

**Table 29-2. 2s Complement Left Aligned Mapping**

OSR Range	SD24BOSRx Register	Filter Output Left Shifted by	Filter's LSB Mapped to
1 to 32	0000h to 001Fh	16 bits	Bit 15 of SD24BMEMLx
33 to 64	0020h to 003Fh	13 bits	Bit 12 of SD24BMEMLx
65 to 128	0040h to 007Fh	10 bits	Bit 9 of SD24BMEMLx
129 to 256	0080h to 00FFh	7 bits	Bit 6 of SD24BMEMLx
257 to 512	0100h to 01FFh	4 bits	Bit 3 of SD24BMEMLx
513 to 1024	0200h to 03FFh	1 bit	Bit 0 of SD24BMEMLx

**Table 29-3. Data Format Example for OSR = 256**

SD24DFx	SD24ALGN	Format	Analog Input	Filter Output (hex)	SD24BMEMHx SD24BMEMLx (hex)
00b	0	Bipolar offset binary, right aligned	+V <sub>FSR</sub>	0FF FFFF	00FF FFFF
			0V	080 0000	0080 0000
			-V <sub>FSR</sub>	000 0000	0000 0000
00b	1	Bipolar offset binary, left aligned	+V <sub>FSR</sub>	0FF FFFF	FFFF FF00
			0V	080 0000	8000 0000
			-V <sub>FSR</sub>	000 0000	0000 0000
01b	0	Bipolar 2s complement, right aligned	+V <sub>FSR</sub>	00FF FFFF	00FF FFFF
			0V	0000 0000	0000 0000
			-V <sub>FSR</sub>	FF00 0000	FF00 0000
01b	1	Bipolar 2s complement, left aligned	+V <sub>FSR</sub>	00FF FFFF	7FFF FF80
			0V	0000 0000	0000 0000
			-V <sub>FSR</sub>	FF00 0000	8000 0000

### 29.2.8 Bit Stream Input and Output

The bit stream of each modulator can be fed to pins. The SD24MCx selects the encoding of the output bit stream.

The SD24\_B module also allows to feed in a modulator bit stream into the digital filter with SD24DI = 1. Setting SD24DI = 1 for a converter disables the associated modulator. The incoming bit stream can be either synchronous to the modulator frequency  $f_M$  or it can be a Manchester decoded bit stream. The format is selected with the SD24MCx bits. The Manchester decoder requires a zero-to-one or a one-to-zero transition in the incoming bit stream to be able to synchronize correctly to it. Up to the first transition, the decoded data can be incorrect.

Figure 29-9 shows the block diagram of the output encoder and the input decoder.

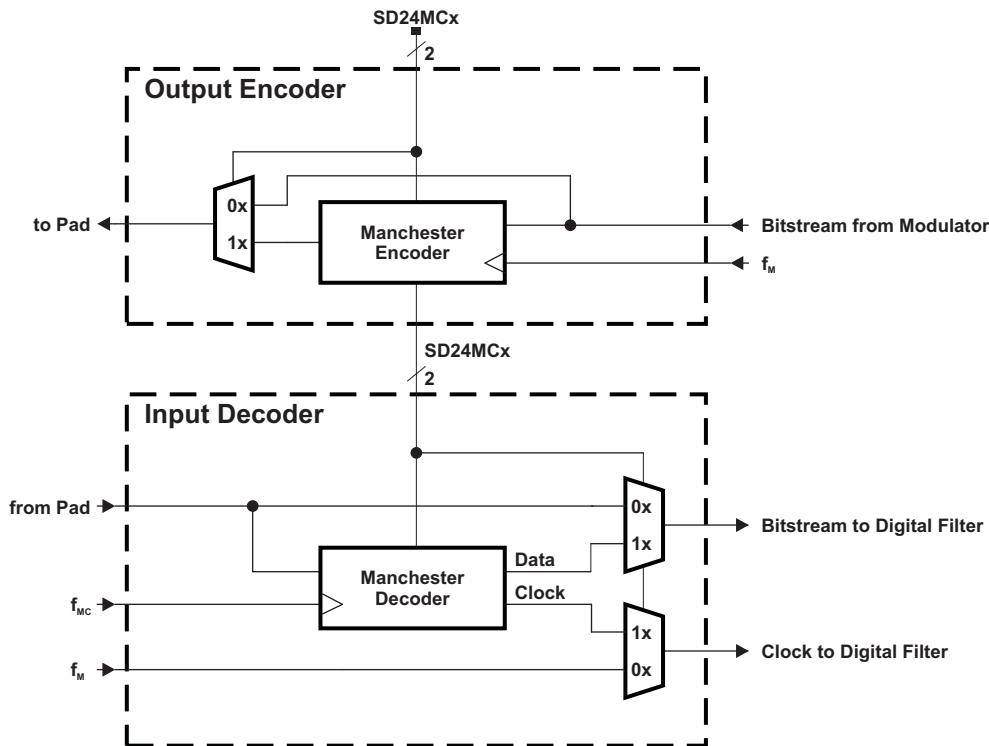


Figure 29-9. SD24\_B Output Encoder and Input Decoder Block Diagram

### 29.2.9 Conversion Modes

Each SD24\_B converter can be configured for two modes of operation, single conversion and continuous conversion, as listed in Table 29-4. The SD24SNGL bit selects the converter's conversion mode.

Table 29-4. Conversion Mode Summary

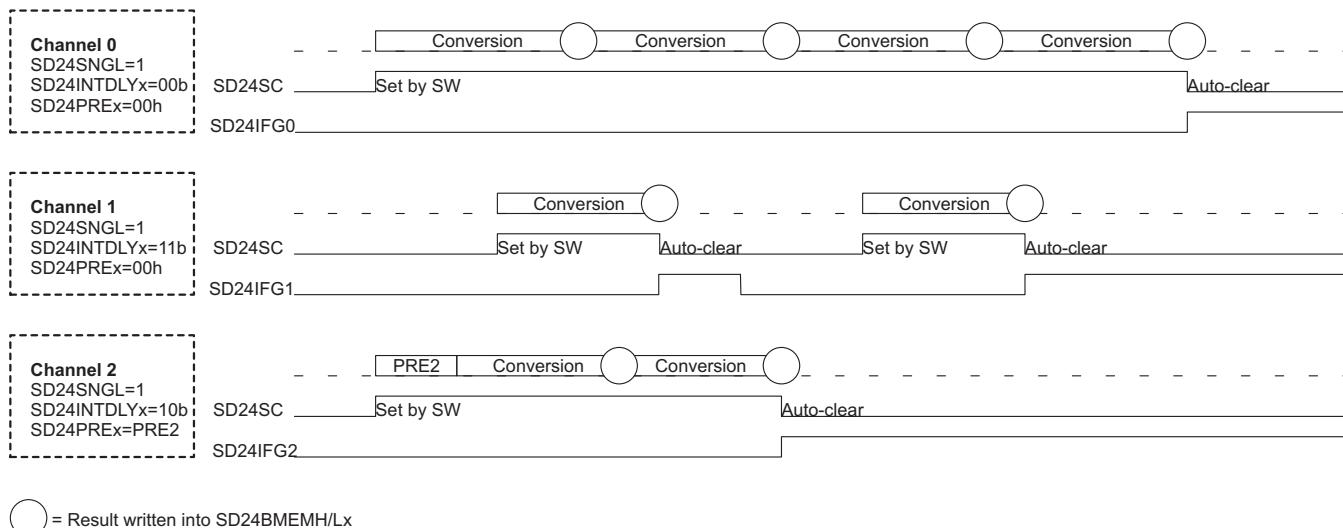
SD24SNGL	Mode	Operation
1	Single conversion	The converter converts once, until the corresponding IFG is set.
0	Continuous conversion	The converter converts continuously.

In addition, the converters can be grouped together by selecting a common start-of-conversion trigger using the SD24SCSx bits. There are up to four software start-of-conversion triggers and three external (for example a timer output) start-of-conversion triggers available.

#### 29.2.9.1 Single Converter, Single Conversion

Setting the SD24SC bit of a converter initiates one conversion on that converter when SD24SNGL = 1 and SD24SCSx = 00b; that is, the converter is not grouped with any other converters. The SD24SC bit is automatically cleared after conversion completion; that is, when the converter's interrupt flag is set. Clearing SD24SC before the conversion is completed immediately stops conversion of the selected converter, the converter is powered down, and the corresponding digital filter is turned off.

Figure 29-10 shows examples of single conversions with different SD24INTDLYx and SD24PREx settings.



**Figure 29-10. Single Conversion Examples**

#### 29.2.9.2 Single Converter, Continuous Conversion

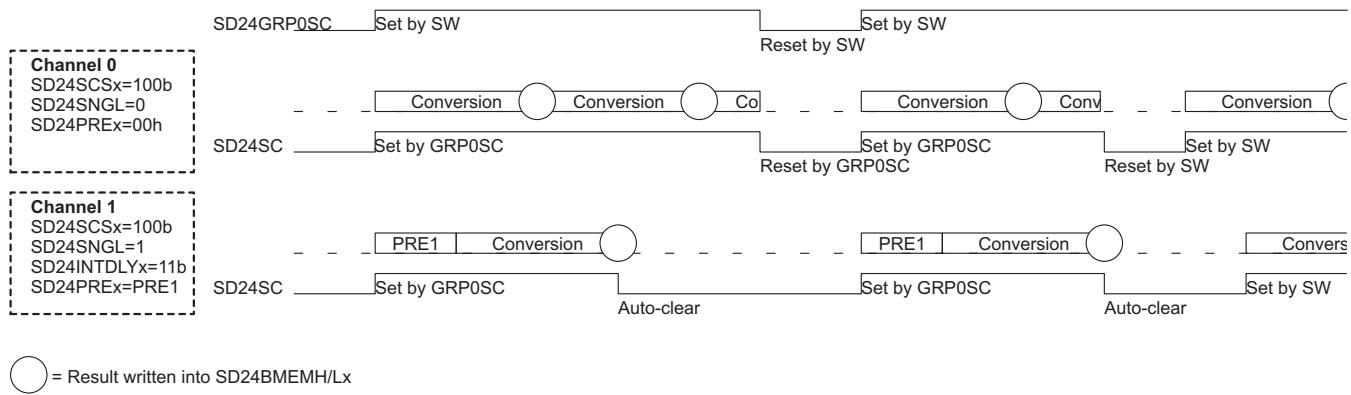
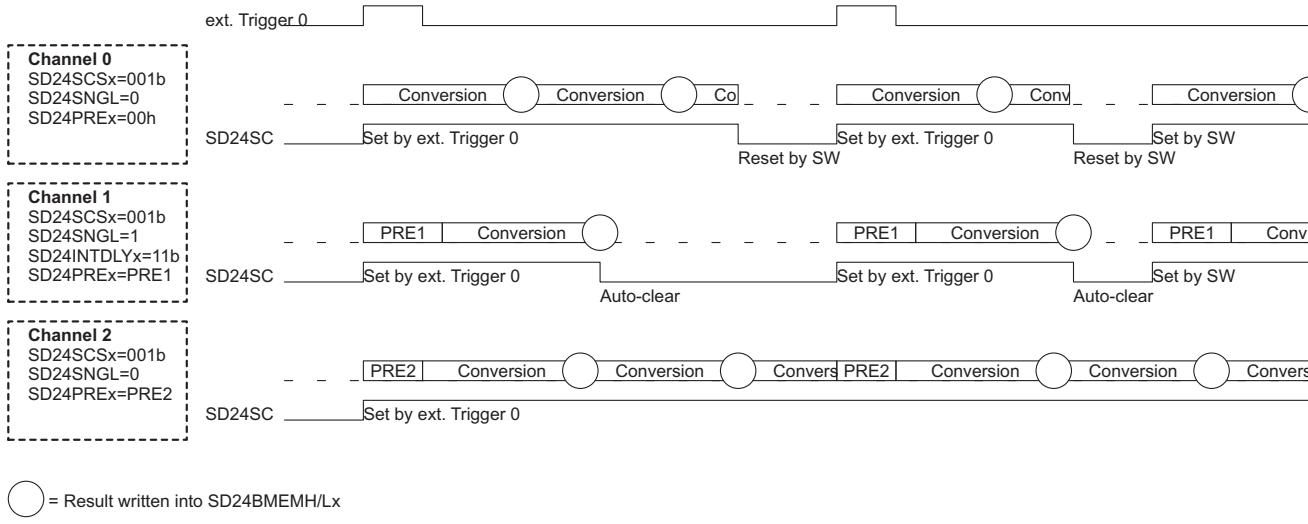
When SD24SNGL = 0, continuous conversion mode is selected. A converter configured with SD24SCSx = 00b begins converting when SD24SC is set and continues until the SD24SC bit is cleared by software. Clearing SD24SC immediately stops conversion of the selected converter, the converter is powered down, and the corresponding digital filter is turned off.

#### 29.2.9.3 Group of Converters

SD24\_B converters can be grouped together with a common start-of-conversion trigger by setting SD24SCSx to a non-zero value. All converters with the same SD24SCS settings form a group, and the conversion is started by the selected group for all attached converters synchronously. The conversion can be started either by software by setting the respective SD24GRPxSC bit (see Figure 29-11) or by a rising edge on an external trigger signal like a timer output (see Figure 29-12). The connected trigger sources are device dependent—see the device-specific data sheet for details.

When SD24SNGL = 1 for a converter in a group, single conversion mode is selected. A single conversion of this converter occurs synchronously when the group trigger starts conversion.

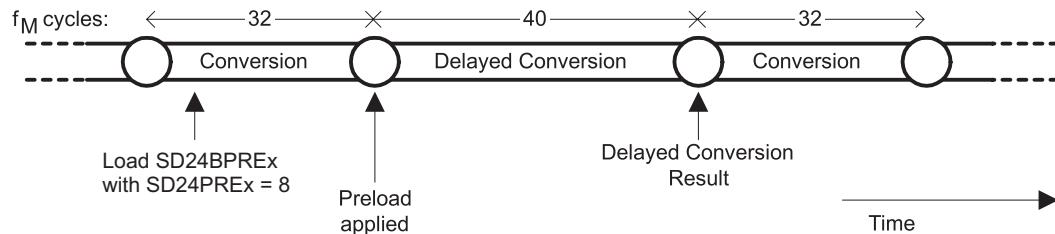
The start of conversion is also reflected in each converter's SD24SC bit. By clearing the SD24SC bit of one converter in a group, the conversion of this converter is immediately stopped. Setting the SD24SC bit of an individual converter in a group immediately starts conversion for this converter independent of the group's start of conversion trigger.

**Figure 29-11. Grouped Operation - Internal Start-of-Conversion Trigger****Figure 29-12. Grouped Operation - External Start-of-Conversion Trigger**

### 29.2.10 Conversion Operation Using Preload

When multiple converters are grouped, the SD24BPREx registers can be used to delay the conversion time frame for each converter. Using SD24BPREx, the decimation time of the digital filter is increased one time by the specified number of  $f_M$  clock cycles, which can range from 0 to 1023. [Figure 29-13](#) shows an example using SD24BPREx.

SD24OSRx = 32

**Figure 29-13. Conversion Delay Using Preload - Example**

The SD24BPReX delay is applied at the beginning of the next conversion cycle after it is written; that is, the specified number of delay cycles is applied to the conversion cycle following the write to SD24BPReX. Further conversions are not delayed. After modifying SD24BPReX, the next write to SD24BPReX should not occur until the current conversion cycle is completed and the written value is applied. For example, the preload value can be modified after the corresponding SD24IFGx is set. At start of conversion, the first conversion is delayed by the latest value written into SD24BPReX. If no delay at start of conversion is desired, a previously written non-zero value must be changed to zero before starting the conversion.

The accuracy of the result for the delayed conversion cycle using SD24BPReX is dependent on the length of the delay and the frequency of the analog signal being sampled. For example, when measuring a dc signal, SD24BPReX delay has no effect on the conversion result regardless of the duration. The user must determine when the delayed conversion result is useful in an application.

Figure 29-14 shows the operation of grouped converters 0 and 1. The preload register of converter 1 is loaded with zero, which results in immediate conversion, while the conversion cycle of converter 0 is delayed by setting SD24PRE0 = 8. The first converter 0 conversion uses SD24BPReX = 8, shifting all subsequent conversions by 8  $f_M$  clock cycles.

SD24OSRx = 32

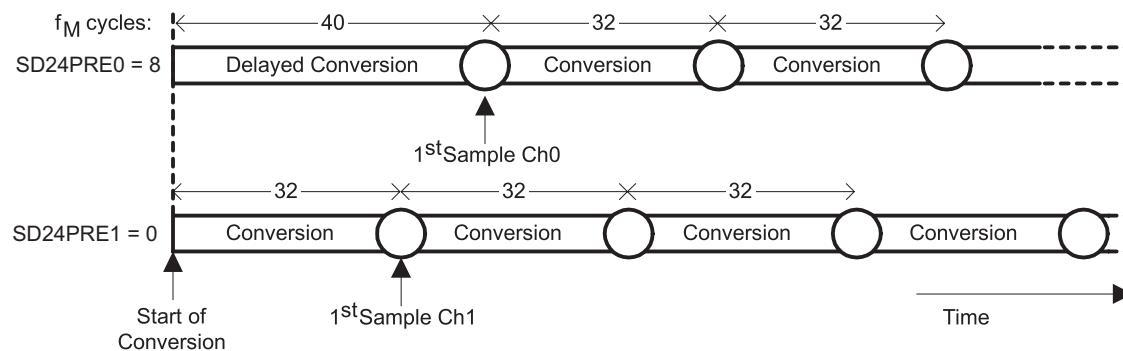


Figure 29-14. Start of Conversion Using Preload - Example

### 29.2.11 Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed circuit board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

Ground loops are formed when return current from the ADC flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small unwanted offset voltages that can add to or subtract from the reference or input voltages of the ADC. Therefore, solid decoupling on both the digital and analog supply is required (best with two capacitors, one 10  $\mu$ F and one 100 nF, per supply).

In addition to grounding, ripple and noise spikes on the power-supply lines due to digital switching or switching power supplies can corrupt the conversion result. A noise-free design using separate analog and digital ground planes with a single-point connection is recommended to achieve high accuracy.

If the internal reference is used, the reference voltage should be buffered externally by connecting a small (approximately 100 nF) capacitor to VREF to reduce the noise on the reference.

### 29.2.12 Trigger Generator

The SD24\_B module provides a trigger generator that allows synchronization of external modules (for example other ADCs) with the SD24\_B. Configuration of the trigger generator (see the block diagram in Figure 29-15) is similar to a converter (although it does not provide the analog part and thus all associated digital logic). It can be started the same as a converter, and the trigger pulse and the setting of the interrupt flag is generated as it would be generated by a converter with the same OSR and preload settings. This means that the trigger generator mimics the timing of a converter. If the trigger generator is configured the same as the converters and is started together with the converters, it can trigger external modules synchronous with the conversion (that is, decimation) of the SD24\_B converters.

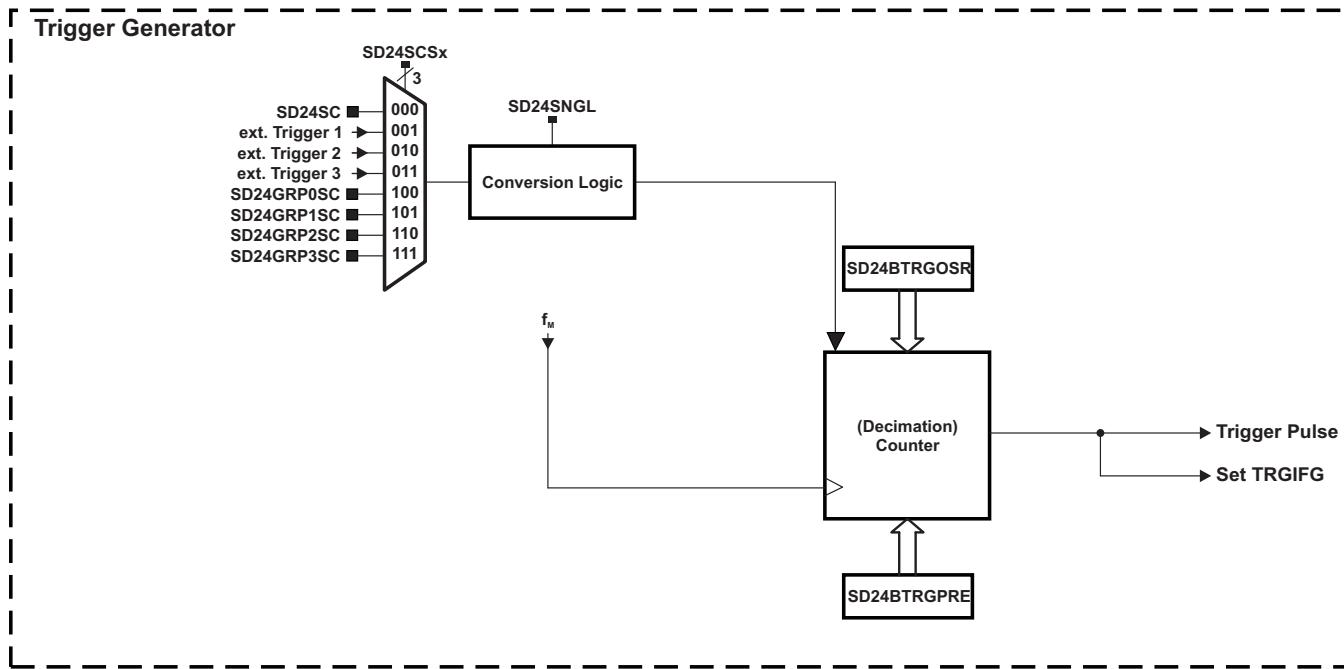


Figure 29-15. SD24\_B Trigger Generator Block Diagram

### 29.2.13 SD24\_B Interrupts

The SD24\_B has these interrupt sources for each ADC converter:

- SD24IFGx : conversion ready interrupt
- SD24OVIFGx : conversion memory overflow
- SD24TRGIFG: trigger generator interrupt

The SD24IFGx bits are set when their corresponding SD24BMEMHx and SD24BMEMLx memory registers are written with a conversion result. An interrupt request is generated if the corresponding SD24IE bit and the GIE bit are set. Whether or not an SD24\_B overflow condition sets a SD24OVIFG depends on the SD24OV32 setting: If SD24OV32 = 0, the overflow occurs when a conversion result is written to a SD24BMEMHx and SD24BMEMLx register pair before one of the registers was read. If SD24OV32 = 1, the overflow occurs when a new conversion result is written before the complete 32-bit of the previous conversion result was read.

#### 29.2.13.1 SD24BIV, Interrupt Vector Generator

All SD24\_B interrupt sources are prioritized and combined to source a single interrupt vector. SD24BIV is used to determine which enabled SD24\_B interrupt source requested an interrupt. The highest priority SD24\_B interrupt request that is enabled generates a number in the SD24BIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled SD24\_B interrupts do not affect the SD24BIV value.

Any read access of the SD24BIV register has no effect on the SD24OVIFG or SD24IFG flags. The SD24IFG flags are reset by reading the associated SD24BMEMx register or by clearing the flags in software. SD24OVIFG bits can only be reset with software. The SD24TRGIFG is reset by reading SD24BIV or by clearing the flag in software. A write access to SD24BIV clears all interrupt flags.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if a SD24OVIFGx and one or more SD24IFGx interrupts are pending when the interrupt service routine accesses the SD24BIV register, the SD24OVIFGx interrupt condition is serviced first and the corresponding flags must be cleared in software. After the RETI instruction of the interrupt service routine is executed, the highest priority SD24IFGx pending generates another interrupt request.

#### 29.2.13.2 Interrupt Delay Operation

The SD24INTDLYx bits control the timing for the first interrupt service request for the corresponding converter. This feature delays the interrupt request for a completed conversion by up to four conversion cycles allowing the digital filter to settle prior to generating an interrupt request. The delay is applied each time the conversion is started or when the SD24BINCTLx register is modified. SD24INTDLYx disables overflow interrupt generation for the converter for the selected number of delay cycles. Interrupt requests for the delayed conversions are not generated during the delay.

### 29.2.14 Using SD24\_B With DMA

Devices with an integrated DMA controller can automatically move data from any SD24BMEMHx and SD24BMEMLx register to another location. DMA transfers are done without CPU intervention and independent of any low-power modes.

The SD24DMAx bits in SD24BCTL1 register select the converter that triggers a DMA transfer. The transfer is triggered by the respective SD24IFGx flag. If the respective interrupt enable bit SD24IE is set, the selected SD24IFGx flag does not trigger a transfer. Any SD24IFGx is automatically cleared when the DMA controller accesses the corresponding SD24BMEMHx and SD24BMEMLx registers.

## 29.3 SD24\_B Registers

The SD24\_B registers are listed in [Table 29-5](#). The available converter registers are device-specific depending on the number of implemented converters.

The base address of the SD24\_B module can be found in the device-specific data sheet. The address offset of each SD24\_B register is given in [Table 29-5](#).

**Table 29-5. SD24\_B Registers**

Offset	Acronym	Register Name	Type	Reset	Section
00h	SD24BCTL0	SD24_B Control 0 Register	Read/write	0000h	<a href="#">Section 29.3.1</a>
02h	SD24BCTL1	SD24_B Control 1 Register	Read/write	0800h	<a href="#">Section 29.3.2</a>
04h	SD24BTRGCTL	SD24_B Trigger Control Register	Read/write	0000h	<a href="#">Section 29.3.3</a>
06h	SD24BTRGOSR	SD24_B Trigger OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.10</a>
08h	SD24BTRGPRE	SD24_B Trigger Preload Register	Read/write	0000h	<a href="#">Section 29.3.12</a>
0Ah	SD24BIFG	SD24_B Interrupt Flag Register	Read/write	0000h	<a href="#">Section 29.3.4</a>
0Ch	SD24BIE	SD24_B Interrupt Enable Register	Read/write	0000h	<a href="#">Section 29.3.5</a>
0Eh	SD24BIV	SD24_B Interrupt Vector Register	Read/write	0000h	<a href="#">Section 29.3.6</a>
10h	SD24BCCTL0	SD24_B Converter 0 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
12h	SD24BINCTL0	SD24_B Converter 0 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
14h	SD24BOSR0	SD24_B Converter 0 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
16h	SD24BPRE0	SD24_B Converter 0 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
18h	SD24BCCTL1	SD24_B Converter 1 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
1Ah	SD24BINCTL1	SD24_B Converter 1 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
1Ch	SD24BOSR1	SD24_B Converter 1 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
1Eh	SD24BPRE1	SD24_B Converter 1 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
20h	SD24BCCTL2	SD24_B Converter 2 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
22h	SD24BINCTL2	SD24_B Converter 2 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
24h	SD24BOSR2	SD24_B Converter 2 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
26h	SD24BPRE2	SD24_B Converter 2 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
28h	SD24BCCTL3	SD24_B Converter 3 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
2Ah	SD24BINCTL3	SD24_B Converter 3 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
2Ch	SD24BOSR3	SD24_B Converter 3 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
2Eh	SD24BPRE3	SD24_B Converter 3 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
30h	SD24BCCTL4	SD24_B Converter 4 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
32h	SD24BINCTL4	SD24_B Converter 4 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
34h	SD24BOSR4	SD24_B Converter 4 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
36h	SD24BPRE4	SD24_B Converter 4 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
38h	SD24BCCTL5	SD24_B Converter 5 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
3Ah	SD24BINCTL5	SD24_B Converter 5 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
3Ch	SD24BOSR5	SD24_B Converter 5 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
3Eh	SD24BPRE5	SD24_B Converter 5 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
40h	SD24BCCTL6	SD24_B Converter 6 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
42h	SD24BINCTL6	SD24_B Converter 6 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
44h	SD24BOSR6	SD24_B Converter 6 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
46h	SD24BPRE6	SD24_B Converter 6 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
48h	SD24BCCTL7	SD24_B Converter 7 Control Register	Read/write	0000h	<a href="#">Section 29.3.7</a>
4Ah	SD24BINCTL7	SD24_B Converter 7 Input Control Register	Read/write	0000h	<a href="#">Section 29.3.8</a>
4Ch	SD24BOSR7	SD24_B Converter 7 OSR Control Register	Read/write	00FFh	<a href="#">Section 29.3.9</a>
4Eh	SD24BPRE7	SD24_B Converter 7 Preload Register	Read/write	0000h	<a href="#">Section 29.3.11</a>
50h	SD24BMEML0	SD24_B Converter 0 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>
52h	SD24BMEMH0	SD24_B Converter 0 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>
54h	SD24BMEML1	SD24_B Converter 1 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>
56h	SD24BMEMH1	SD24_B Converter 1 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>
58h	SD24BMEML2	SD24_B Converter 2 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>

**Table 29-5. SD24\_B Registers (continued)**

Offset	Acronym	Register Name	Type	Reset	Section
5Ah	SD24BMEMH2	SD24_B Converter 2 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>
5Ch	SD24BMEML3	SD24_B Converter 3 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>
5Eh	SD24BMEMH3	SD24_B Converter 3 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>
60h	SD24BMEML4	SD24_B Converter 4 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>
62h	SD24BMEMH4	SD24_B Converter 4 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>
64h	SD24BMEML5	SD24_B Converter 5 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>
66h	SD24BMEMH5	SD24_B Converter 5 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>
68h	SD24BMEML6	SD24_B Converter 6 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>
6Ah	SD24BMEMH6	SD24_B Converter 6 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>
6Ch	SD24BMEML7	SD24_B Converter 7 Conversion Memory Low Word Register	Read/write	0000h	<a href="#">Section 29.3.13</a>
6Eh	SD24BMEMH7	SD24_B Converter 7 Conversion Memory High Word Register	Read/write	0000h	<a href="#">Section 29.3.14</a>

**29.3.1 SD24BCTL0 Register**

SD24\_B Control Register 0

**Figure 29-16. SD24BCTL0 Register**

15	14	13	12	11	10	9	8
SD24DIVx					SD24PDIVx		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD24CLKOS	SD24M4	SD24SSELx		Reserved	SD24REFS	SD24OV32	Reserved
rw-0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	r0

**Table 29-6. SD24BCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-11	SD24DIVx	RW	0h	SD24_B frequency divider. Together with SD24PDIVx the SD24_B frequency $f(SD24)$ is calculated as $f(SD24) = f(SD24SCLK) / [(SD24DIVx + 1) \times 2^{(SD24PDIVx)}]$ . 0000b = Divide by 1 0001b = Divide by 2 ⋮ 11110b = Divide by 31 11111b = Divide by 32
10-8	SD24PDIVx	RW	0h	SD24_B frequency pre-scaler. Together with SD24DIVx the SD24_B frequency $f(SD24)$ is calculated as $f(SD24) = f(SD24SCLK) / [(SD24DIVx + 1) \times 2^{(SD24PDIVx)}]$ . 000b = Divide by 1 001b = Divide by 2 010b = Divide by 4 011b = Divide by 8 100b = Divide by 16 101b = Divide by 32 110b = Divide by 64 111b = Divide by 128
7	SD24CLKOS	RW	0h	Clock output select 0b = Modulator clock, $f(M)$ 1b = Manchester decoder clock, $f(MC)$ . Depending on SD24M4, the Manchester decoder clock is equal to the modulator clock or four times the modulator clock.
6	SD24M4	RW	0h	Modulator clock to Manchester decoder clock ratio 0b = Modulator clock equals Manchester decoder clock, $f(M) = f(MC) = f(SD24)$ 1b = Modulator clock is 1/4 of the Manchester decoder clock, $f(M) = f(MC)/4 = f(SD24)/4$ , $f(MC) = f(SD24)$
5-4	SD24SSEL	RW	0h	SD24_B clock source select 00b = MCLK 01b = SMCLK 10b = ACLK 11b = External SD24_B clock (SD24CLK)
3	Reserved	R	0h	Reserved. Always reads as 0.
2	SD24REFS	RW	0h	Reference select 0b = External reference selected. Internal reference voltage buffer disabled. 1b = Internal reference from shared REF selected. This requests the reference voltage from the shared REF and buffers it internally to the SD24_B.

**Table 29-6. SD24BCTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
1	SD24OV32	RW	0h	SD24_B overflow control 0b = Overflow on 16-bit (1 word) only; that is, only one of the SD24BMEMLx and SD24BMEMHx registers must be read to prevent the overflow interrupt flag being set. 1b = Overflow on 32-bit (2 words); that is, both SD24BMEMx and SD24BMEMHx registers must be read to prevent the overflow interrupt flag being set.
0	Reserved	R	0h	Reserved. Always reads as 0.

**29.3.2 SD24BCTL1 Register**

SD24\_B Control Register 1

**Figure 29-17. SD24BCTL1 Register**

15	14	13	12	11	10	9	8
Reserved				SD24DMAx			
r0	r0	r0	r0	rw-1	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved				SD24GRP3SC	SD24GRP2SC	SD24GRP1SC	SD24GRP0SC
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

**Table 29-7. SD24BCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	SD24DMAx	RW	8h	DMA trigger select. Selects the converter that should trigger a DMA transfer. 0000b = SD24IFG0 triggers DMA (if SD24IE0 = 0) 0001b = SD24IFG1 triggers DMA (if SD24IE1 = 0) 0010b = SD24IFG2 triggers DMA (if SD24IE2 = 0) 0011b = SD24IFG3 triggers DMA (if SD24IE3 = 0) 0100b = SD24IFG4 triggers DMA (if SD24IE4 = 0) 0101b = SD24IFG5 triggers DMA (if SD24IE5 = 0) 0110b = SD24IFG6 triggers DMA (if SD24IE6 = 0) 0111b = SD24IFG7 triggers DMA (if SD24IE7 = 0) 1000b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0) 1001b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0) 1010b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0) 1011b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0) 1100b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0) 1101b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0) 1110b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0) 1111b = SD24TRGIFG triggers DMA (if SD24TRGIE = 0)
7-4	Reserved	R	0h	Reserved. Always reads as 0.
3	SD24GRP3SC	RW	0h	SD24_B Group 3 start of conversion. Software controlled start of conversion for all converters belonging to group 3. 0b = No conversion start 1b = Start conversion for group 3
2	SD24GRP2SC	RW	0h	SD24_B Group 2 start of conversion. Software controlled start of conversion for all converters belonging to group 2. 0b = No conversion start 1b = Start conversion for group 2
1	SD24GRP1SC	RW	0h	SD24_B Group 1 start of conversion. Software controlled start of conversion for all converters belonging to group 1. 0b = No conversion start 1b = Start conversion for group 1
0	SD24GRP0SC	RW	0h	SD24_B Group 0 start of conversion. Software controlled start of conversion for all converters belonging to group 0. 0b = No conversion start 1b = Start conversion for group 0

### 29.3.3 SD24BTRGCTL Register

SD24\_B Trigger Control Register

**Figure 29-18. SD24BTRGCTL Register**

15	14	13	12	11	10	9	8
		Reserved		SD24TRGIE	SD24TRGIFG	Reserved	SD24SNGL
r0	r0	r0	r0	rw-0	rw-0	r0	rw-0
7	6	5	4	3	2	1	0
		Reserved		SD24SCSx		SD24SC	
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

**Table 29-8. SD24BTRGCTL Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11	SD24TRGIE	RW	0h	SD24_B trigger interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
10	SD24TRGIFG	RW	0h	SD24_B trigger interrupt flag 0b = No interrupt pending 1b = Interrupt pending
9	Reserved	R	0h	Reserved. Always reads as 0.
8	SD24SNGL	RW	0h	Single trigger mode select 0b = Continuous triggers generated (like a converter in continuous conversion mode). 1b = Single trigger generated (like a converter in single conversion mode).
7-4	Reserved	R	0h	Reserved. Always reads as 0.
3-1	SD24SCSx	RW	0h	Start of conversion/trigger generation select 000b = SD24SC bit 001b = External trigger 1 (see the device-specific data sheet) 010b = External trigger 2 (see the device-specific data sheet) 011b = External trigger 3 (see the device-specific data sheet) 100b = Group 0 - Start of conversion defined by SD24GRP0SC bits in register SD24BCTL1 101b = Group 1 - Start of conversion defined by SD24GRP1SC bits in register SD24BCTL1 110b = Group 2 - Start of conversion defined by SD24GRP2SC bits in register SD24BCTL1 111b = Group 3 - Start of conversion defined by SD24GRP3SC bits in register SD24BCTL1
0	SD24SC	RW	0h	Start of conversion or trigger generation. Software controlled start of conversion if SD24SCS = 00b. Manual stop of conversion (independent of SD24SCS setting). If a conversion is ongoing (triggered by any source selected through SD24SCS) this is indicated by SD24SC = 1. 0b = No conversion start or stop conversion 1b = Start conversion or trigger generation

**29.3.4 SD24BIFG Register**

SD24\_B Interrupt Flag Register

**Figure 29-19. SD24BIFG Register**

15	14	13	12	11	10	9	8
SD24OVIFG7	SD24OVIFG6	SD24OVIFG5	SD24OVIFG4	SD24OVIFG3	SD24OVIFG2	SD24OVIFG1	SD24OVIFG0
rw-0							
7	6	5	4	3	2	1	0
SD24IFG7	SD24IFG6	SD24IFG5	SD24IFG4	SD24IFG3	SD24IFG2	SD24IFG1	SD24IFG0
rw-0							

**Table 29-9. SD24BIFG Register Description**

Bit	Field	Type	Reset	Description
15	SD24OVIFG7	RW	0h	SD24_B converter 7 overflow interrupt flag. These bits are set depending on the SD24OV32 bit settings: When none of the corresponding conversion registers SD24BMEML7 or SD24BMEMH7 are read before new values are loaded (SD24OV32 = 0). When only one of the corresponding conversion registers SD24BMEML7 or SD24BMEMH7 is read before new values are loaded (SD24OV32 = 1). 0b = No interrupt pending 1b = Interrupt pending
14	SD24OVIFG6	RW	0h	SD24_B converter 6 overflow interrupt flag. These bits are set depending on the SD24OV32 bit settings: When none of the corresponding conversion registers SD24BMEML6 or SD24BMEMH6 are read before new values are loaded (SD24OV32 = 0). When only one of the corresponding conversion registers SD24BMEML6 or SD24BMEMH6 is read before new values are loaded (SD24OV32 = 1). 0b = No interrupt pending 1b = Interrupt pending
13	SD24OVIFG5	RW	0h	SD24_B converter 5 overflow interrupt flag. These bits are set depending on the SD24OV32 bit settings: When none of the corresponding conversion registers SD24BMEML5 or SD24BMEMH5 are read before new values are loaded (SD24OV32 = 0). When only one of the corresponding conversion registers SD24BMEML5 or SD24BMEMH5 is read before new values are loaded (SD24OV32 = 1). 0b = No interrupt pending 1b = Interrupt pending
12	SD24OVIFG4	RW	0h	SD24_B converter 4 overflow interrupt flag. These bits are set depending on the SD24OV32 bit settings: When none of the corresponding conversion registers SD24BMEML4 or SD24BMEMH4 are read before new values are loaded (SD24OV32 = 0). When only one of the corresponding conversion registers SD24BMEML4 or SD24BMEMH4 is read before new values are loaded (SD24OV32 = 1). 0b = No interrupt pending 1b = Interrupt pending
11	SD24OVIFG3	RW	0h	SD24_B converter 3 overflow interrupt flag. These bits are set depending on the SD24OV32 bit settings: When none of the corresponding conversion registers SD24BMEML3 or SD24BMEMH3 are read before new values are loaded (SD24OV32 = 0). When only one of the corresponding conversion registers SD24BMEML3 or SD24BMEMH3 is read before new values are loaded (SD24OV32 = 1). 0b = No interrupt pending 1b = Interrupt pending

**Table 29-9. SD24BIFG Register Description (continued)**

Bit	Field	Type	Reset	Description
10	SD24OVIFG2	RW	0h	<p>SD24_B converter 2 overflow interrupt flag.</p> <p>These bits are set depending on the SD24OV32 bit settings:</p> <p>When none of the corresponding conversion registers SD24BMEML2 or SD24BMEMH2 are read before new values are loaded (SD24OV32 = 0).</p> <p>When only one of the corresponding conversion registers SD24BMEML2 or SD24BMEMH2 is read before new values are loaded (SD24OV32 = 1).</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>
9	SD24OVIFG1	RW	0h	<p>SD24_B converter 1 overflow interrupt flag.</p> <p>These bits are set depending on the SD24OV32 bit settings:</p> <p>When none of the corresponding conversion registers SD24BMEML1 or SD24BMEMH1 are read before new values are loaded (SD24OV32 = 0).</p> <p>When only one of the corresponding conversion registers SD24BMEML1 or SD24BMEMH1 is read before new values are loaded (SD24OV32 = 1).</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>
8	SD24OVIFG0	RW	0h	<p>SD24_B converter 0 overflow interrupt flag.</p> <p>These bits are set depending on the SD24OV32 bit settings:</p> <p>When none of the corresponding conversion registers SD24BMEML0 or SD24BMEMH0 are read before new values are loaded (SD24OV32 = 0).</p> <p>When only one of the corresponding conversion registers SD24BMEML0 or SD24BMEMH0 is read before new values are loaded (SD24OV32 = 1).</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>
7	SD24IFG7	RW	0h	<p>SD24_B converter 7 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML7 and SD24BMEMH7 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML7 or SD24BMEMH7 registers, or may be reset by software.</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>
6	SD24IFG6	RW	0h	<p>SD24_B converter 6 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML6 and SD24BMEMH6 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML6 or SD24BMEMH6 registers, or may be reset by software.</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>
5	SD24IFG5	RW	0h	<p>SD24_B converter 5 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML5 and SD24BMEMH5 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML5 or SD24BMEMH5 registers, or may be reset by software.</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>
4	SD24IFG4	RW	0h	<p>SD24_B converter 4 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML4 and SD24BMEMH4 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML4 or SD24BMEMH4 registers, or may be reset by software.</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>
3	SD24IFG3	RW	0h	<p>SD24_B converter 3 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML3 and SD24BMEMH3 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML3 or SD24BMEMH3 registers, or may be reset by software.</p> <p>0b = No interrupt pending 1b = Interrupt pending</p>

**Table 29-9. SD24BIFG Register Description (continued)**

Bit	Field	Type	Reset	Description
2	SD24IFG2	RW	0h	SD24_B converter 2 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML2 and SD24BMEMH2 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML2 or SD24BMEMH2 registers, or may be reset by software. 0b = No interrupt pending 1b = Interrupt pending
1	SD24IFG1	RW	0h	SD24_B converter 1 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML1 and SD24BMEMH1 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML1 or SD24BMEMH1 registers, or may be reset by software. 0b = No interrupt pending 1b = Interrupt pending
0	SD24IFG0	RW	0h	SD24_B converter 0 interrupt flag. These bits are set when the corresponding conversion registers SD24BMEML0 and SD24BMEMH0 are loaded with a new conversion result. The bits are reset by reading one of the SD24BMEML0 or SD24BMEMH0 registers, or may be reset by software. 0b = No interrupt pending 1b = Interrupt pending

### 29.3.5 SD24BIE Register

SD24\_B Interrupt Enable Register

**Figure 29-20. SD24BIE Register**

15	14	13	12	11	10	9	8
SD24OVIE7	SD24OVIE6	SD24OVIE5	SD24OVIE4	SD24OVIE3	SD24OVIE2	SD24OVIE1	SD24OVIE0
rw-0							
7	6	5	4	3	2	1	0
SD24IE7	SD24IE6	SD24IE5	SD24IE4	SD24IE3	SD24IE2	SD24IE1	SD24IE0
rw-0							

**Table 29-10. SD24BIE Register Description**

Bit	Field	Type	Reset	Description
15	SD24OVIE7	RW	0h	SD24_B converter 7 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
14	SD24OVIE6	RW	0h	SD24_B converter 6 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
13	SD24OVIE5	RW	0h	SD24_B converter 5 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
12	SD24OVIE4	RW	0h	SD24_B converter 4 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
11	SD24OVIE3	RW	0h	SD24_B converter 3 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
10	SD24OVIE2	RW	0h	SD24_B converter 2 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
9	SD24OVIE1	RW	0h	SD24_B converter 1 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
8	SD24OVIE0	RW	0h	SD24_B converter 0 overflow interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
7	SD24IE7	RW	0h	SD24_B converter 7 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
6	SD24IE6	RW	0h	SD24_B converter 6 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
5	SD24IE5	RW	0h	SD24_B converter 5 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
4	SD24IE4	RW	0h	SD24_B converter 4 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

**Table 29-10. SD24BIE Register Description (continued)**

Bit	Field	Type	Reset	Description
3	SD24IE3	RW	0h	SD24_B converter 3 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
2	SD24IE2	RW	0h	SD24_B converter 2 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
1	SD24IE1	RW	0h	SD24_B converter 1 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	SD24IE0	RW	0h	SD24_B converter 0 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

### 29.3.6 SD24BIV Register

SD24\_B Interrupt Vector Register

**Figure 29-21. SD24BIV Register**

15	14	13	12	11	10	9	8
SD24BIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
SD24BIVx							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

**Table 29-11. SD24BIV Register Description**

Bit	Field	Type	Reset	Description
15-0	SD24BIVx	R	0h	<p>SD24_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.</p> <p>When an overflow occurs (02h), the software must check all SD24OVIFGx flags to determine which converter caused the overflow.</p> <p>00h = No Interrupt pending</p> <p>02h = Interrupt Source: SD24_B memory overflow; Interrupt Flag: SD24OVIFGx; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: SD24_B trigger interrupt flag; Interrupt Flag: SD24TRGIFG</p> <p>06h = Interrupt Source: SD24_B converter 0 memory interrupt flag; Interrupt Flag: SD24IFG0</p> <p>08h = Interrupt Source: SD24_B converter 1 memory interrupt flag; Interrupt Flag: SD24IFG1</p> <p>0Ah = Interrupt Source: SD24_B converter 2 memory interrupt flag; Interrupt Flag: SD24IFG2</p> <p>0Ch = Interrupt Source: SD24_B converter 3 memory interrupt flag; Interrupt Flag: SD24IFG3</p> <p>0Eh = Interrupt Source: SD24_B converter 4 memory interrupt flag; Interrupt Flag: SD24IFG4</p> <p>10h = Interrupt Source: SD24_B converter 5 memory interrupt flag; Interrupt Flag: SD24IFG5</p> <p>12h = Interrupt Source: SD24_B converter 6 memory interrupt flag; Interrupt Flag: SD24IFG6</p> <p>14h = Interrupt Source: SD24_B converter 7 memory interrupt flag; Interrupt Flag: SD24IFG7; Interrupt Priority: Lowest</p>

**29.3.7 SD24BCCTLx Register**

SD24\_B Converter x Control Register

**Figure 29-22. SD24BCCTLx Register**

15	14	13	12	11	10	9	8
Reserved	SD24MCx	SD24DI	SD24DFSx	SD24CAL	SD24SNGL		
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved	SD24ALGN	SD24DFx	SD24SCSx	SD24SC			
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 29-12. SD24BCCTLx Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-16	SD24MCx	RW	0h	<p>Manchester Coding of bit stream.            00b = Bitstream synchronous to f(M). Output data changes with falling clock edge, input data is captured with rising clock edge.            01b = Bitstream synchronous to f(M). Output data changes with rising clock edge, input data is captured with falling clock edge.            10b = Output bit stream Manchester encoded. Input bit stream Manchester decoded with oversampling of the input signal. A logic 1 is represented by a high-low transition, a logic 0 is represented by a low-high transition.            11b = Output bit stream Manchester encoded. Input bit stream Manchester decoded with oversampling of the input signal. A logic 1 is represented by a low-high transition, a logic 0 is represented by a high-low transition.</p>
12	SD24DI	RW	0h	<p>Digital bit stream input            0b = Bitstream from modulator fed into digital filter            1b = External bit stream fed into digital filter. Modulator is off.</p>
11-10	SD24DFSx	RW	0h	<p>Digital filter select            00b = SINC3 filter            01b = Reserved (defaults to 00, SINC).            10b = Reserved (defaults to 00, SINC).            11b = Reserved (defaults to 00, SINC).</p>
9	SD24CAL	RW	0h	<p>Calibration            0b = Calibration disabled.            1b = Calibration enabled. With a single conversion the offset of the modulator and PGA can be measured.</p>
8	SD24SNGL	RW	0h	<p>Single conversion mode select            0b = Continuous conversion mode            1b = Single conversion mode</p>
7	Reserved	R	0h	Reserved. Always reads as 0.
6	SD24ALGN	RW	0h	<p>Data alignment            0b = Right-aligned. LSB of filter output is Bit 0.            1b = Left-aligned. MSB of filter output (depending on OSR) is Bit 31.</p>
5-4	SD24DFx	RW	0h	<p>Data format            00b = Offset binary            01b = 2s complement            10b = Reserved (defaults to 00, Offset binary)            11b = Reserved (defaults to 01, 2s complement)</p>

**Table 29-12. SD24BCCTLx Register Description (continued)**

Bit	Field	Type	Reset	Description
3-1	SD24SCSx	RW	0h	<p>Start of conversion select</p> <p>000b = SD24SC bit</p> <p>001b = External trigger 1. See the device-specific data sheet.</p> <p>010b = External trigger 2. See the device-specific data sheet.</p> <p>011b = External trigger 3. See the device-specific data sheet.</p> <p>100b = Group 0 - Start of conversion defined by SD24GRP0SC bits in register SD24BCTL1.</p> <p>101b = Group 1 - Start of conversion defined by SD24GRP1SC bits in register SD24BCTL1.</p> <p>110b = Group 2 - Start of conversion defined by SD24GRP2SC bits in register SD24BCTL1.</p> <p>111b = Group 3 - Start of conversion defined by SD24GRP3SC bits in register SD24BCTL1.</p>
0	SD24SC	RW	0h	<p>Start of conversion. Software controlled start of conversion if SD24SCS = 00b. Manual stop of conversion (independent of SD24SCS setting).</p> <p>If a conversion is ongoing (triggered by any source selected through SD24SCS) this is indicated by SD24SC = 1. When starting the conversion (also by setting SD24SC) this bit will only read 1 when the start of conversion was synchronization to the modulator clock and the actual conversion started.</p> <p>0b = No conversion start (that is, conversion stopped)</p> <p>1b = Conversion ongoing</p>

**29.3.8 SD24BINCTLx Register**

SD24\_B Converter x Input Control Register

**Figure 29-23. SD24BINCTLx 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
SD24INTDLYx		SD24GAINx			Reserved		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 29-13. SD24BINCTLx Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7-6	SD24INTDLYx	RW	0h	Interrupt delay generation after conversion start. These bits select the delay for the first interrupt after conversion start. 00b = Fourth sample causes interrupt 01b = Third sample causes interrupt 10b = Second sample causes interrupt 11b = First sample causes interrupt
5-3	SD24GAINx	RW	0h	Gain 000b = 1 001b = 2 010b = 4 011b = 8 100b = 16 101b = 32 110b = 64 111b = 128
2-0	Reserved	RW	0h	Reserved

### 29.3.9 SD24BOSRx Register

SD24\_B Converter x Oversampling Control Register

**Figure 29-24. SD24BOSRx Register**

15	14	13	12	11	10	9	8
Reserved						SD24OSRx	
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD24OSRx							
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

**Table 29-14. SD24BOSRx Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	SD24OSRx	RW	0h	Oversampling rate. The oversampling rate is defined as OSRx + 1. Valid oversampling rates are 1 to 1024. Default is 256.

### 29.3.10 SD24BTRGOSR Register

SD24\_B Trigger Oversampling Control Register

**Figure 29-25. SD24BTRGOSR Register**

15	14	13	12	11	10	9	8
Reserved						SD24OSRx	
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD24OSRx							
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

**Table 29-15. SD24BTRGOSR Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	SD24OSRx	RW	0h	Oversampling rate. The oversampling rate is defined as OSRx + 1. Valid oversampling rates are 1 to 1024. Default is 256.

**29.3.11 SD24BPREx Register**

SD24\_B Converter x Preload Register

**Figure 29-26. SD24BPREx Register**

15	14	13	12	11	10	9	8
Reserved						SD24BPREx	
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD24BPREx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 29-16. SD24BPREx Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	SD24BPREx	RW	0h	Digital filter preload value

**29.3.12 SD24BTRGPRE Register**

SD24\_B Trigger Preload Register

**Figure 29-27. SD24BTRGPRE Register**

15	14	13	12	11	10	9	8
Reserved						SD24BPREx	
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD24BPREx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 29-17. SD24BTRGPRE Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	SD24BPREx	RW	0h	Digital filter preload value

### 29.3.13 SD24BMEMLx Register

SD24\_B Converter x Conversion Memory Low Word Register

**Figure 29-28. SD24BMEMLx Register**

15	14	13	12	11	10	9	8
Conversion Results - Low Word							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
Conversion Results - Low Word							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

**Table 29-18. SD24BMEMLx Register Description**

Bit	Field	Type	Reset	Description
15-0	Conversion Results - Low Word	R	0h	Conversion results. Actual format depends on selected data format, and oversampling rate.

### 29.3.14 SD24BMEMHx Register

SD24\_B Converter x Conversion Memory High Word Register

**Figure 29-29. SD24BMEMHx Register**

15	14	13	12	11	10	9	8
Conversion Results - High Word							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
Conversion Results - High Word							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

**Table 29-19. SD24BMEMHx Register Description**

Bit	Field	Type	Reset	Description
15-0	Conversion Results - High Word	R	0h	Conversion results. Actual format depends on selected data format, and oversampling rate.

## **DAC12\_A**

The DAC12\_A module is a 12-bit, voltage output digital-to-analog converter. This chapter describes the operation and use of the DAC12\_A module.

Topic	Page
30.1 <b>DAC12_A Introduction</b> .....	<b>789</b>
30.2 <b>DAC12_A Operation</b> .....	<b>792</b>
30.3 <b>DAC Outputs</b> .....	<b>796</b>
30.4 <b>DAC12_A Registers</b> .....	<b>797</b>

### 30.1 DAC12\_A Introduction

The DAC12\_A module is a 12-bit, voltage output DAC. The DAC12\_A can be configured in 8-bit or 12-bit mode and may be used in conjunction with the DMA controller. When multiple DAC12\_A modules are present, they may be grouped together for synchronous update operation.

Features of the DAC12\_A include:

- 12-bit monotonic output
- 8-bit or 12-bit voltage output resolution
- Programmable settling time vs power consumption
- Internal or external reference selection
- Straight binary or 2's complement data format, right or left justified
- Self-calibration option for offset correction
- Synchronized update capability for multiple DAC12\_A modules

---

**NOTE: Multiple DAC12\_A Modules**

Some devices may integrate more than one DAC12\_A module. In the case where more than one DAC12\_A is present on a device, the multiple DAC12\_A modules operate identically.

Throughout this chapter, nomenclature appears such as DAC12\_xDAT or DAC12\_xCTL to describe register names. When this occurs, the x is used to indicate which DAC12 module is being discussed. In cases where operation is identical, the register is simply referred to as DAC12\_xCTL.

---

The block diagram of the two DAC12\_A modules is shown in [Figure 30-1](#). The block diagram for the DAC12\_A module is shown in [Figure 30-2](#).

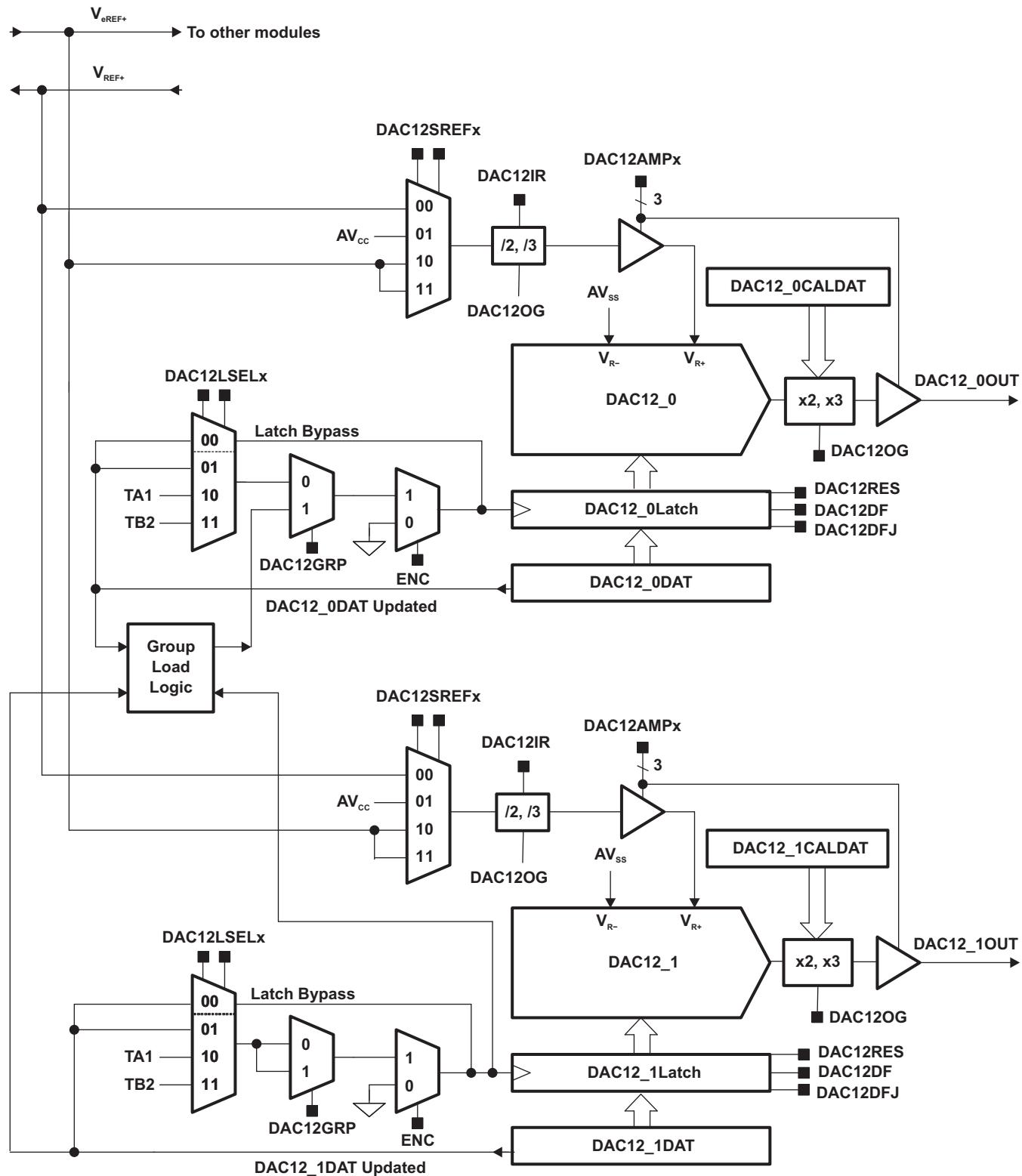
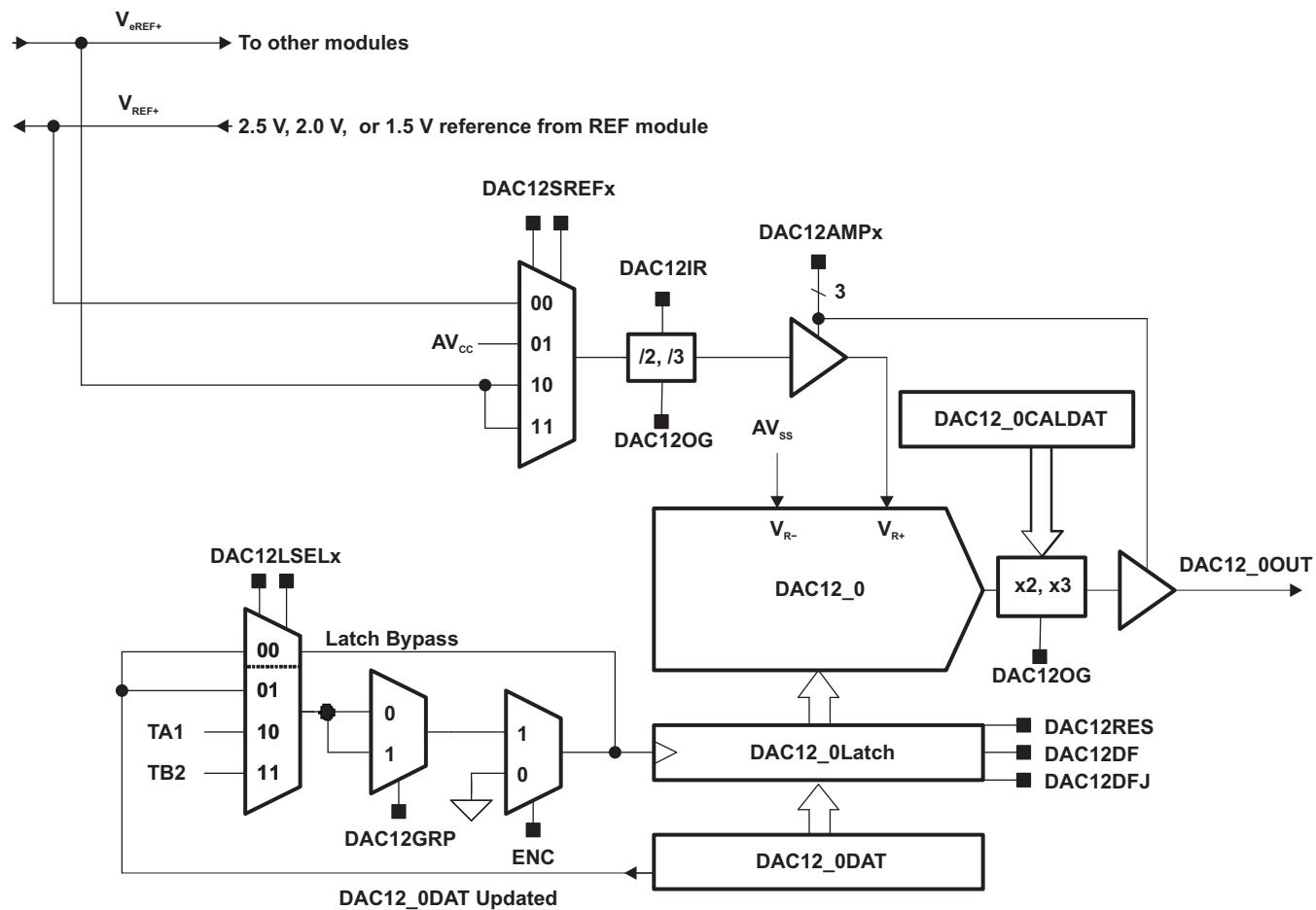


Figure 30-1. DAC12\_A Block Diagram for Two Module Devices



**Figure 30-2. DAC12\_A Block Diagram For Single Module Devices**

## 30.2 DAC12\_A Operation

The DAC12\_A module is configured with user software. The setup and operation of the DAC12\_A is discussed in the following sections.

### 30.2.1 DAC12\_A Core

The DAC12\_A can be configured to operate in 8-bit or 12-bit mode using the DAC12RES bit. The full-scale output is programmable to be 1x, 2x, or 3x the selected reference voltage via the DAC12IR and DAC12OG bits. This feature allows the user to control the dynamic range of the DAC. The DAC12DF bit allows the user to select between straight binary data and 2's complement data for the DAC. When using straight binary data format, the formula for the output voltage is given in [Table 30-1](#).

**Table 30-1. DAC Full-Scale Range ( $V_{ref} = V_{e_{REF+}}$  or  $V_{REF+}$ )**

Resolution	DAC12RES	DAC12OG	DAC12IR	Output Voltage Formula
12 bit	0	0	0	$V_{out} = V_{ref} \times 3 \times (DAC12_xDAT/4096)$
12 bit	0	1	0	$V_{out} = V_{ref} \times 2 \times (DAC12_xDAT/4096)$
12 bit	0	x	1	$V_{out} = V_{ref} \times (DAC12_xDAT/4096)$
8 bit	1	0	0	$V_{out} = V_{ref} \times 3 \times (DAC12_xDAT/256)$
8 bit	1	1	0	$V_{out} = V_{ref} \times 2 \times (DAC12_xDAT/256)$
8 bit	1	x	1	$V_{out} = V_{ref} \times (DAC12_xDAT/256)$

In 8-bit mode the maximum useable value for DAC12\_xDAT is 0FFh and in 12-bit mode the maximum useable value for DAC12\_xDAT is 0FFFh. Values greater than these may be written to the register, but all leading bits are ignored.

### 30.2.2 DAC12\_A Port Selection

On most devices, the DAC outputs are multiplexed with the port Px pins and other potentially other secondary functions. When DAC12AMPx > 0, the DAC function is automatically selected for the pin, regardless of the state of the associated PxSELx and PxDIRx bits. See the port pin schematic in the device-specific data sheet for more details.

### 30.2.3 DAC12\_A Reference

The reference for the DAC12\_A is configured to use AV<sub>CC</sub>, an external reference voltage, or the internal 1.5-V/2.0-V/2.5-V reference from the REF module with the DAC12SREFx bits. When DAC12SREFx = {0} and DAC12AMPx > {1}, V<sub>REF+</sub> is used as the reference, which is supplied via the REF module. See the [REF chapter](#) for further information. When DAC12SREFx = {1}, AV<sub>CC</sub> is used as the reference and when DAC12SREFx = {2,3} the Ve<sub>REF+</sub> signal is used as the reference.

#### 30.2.3.1 DAC12\_A Reference Input and Voltage Output Buffers

The reference input and voltage output buffers of the DAC can be configured for optimized settling time vs power consumption. Eight combinations are selected using the DAC12AMPx bits. In the low/low setting, the settling time is the slowest, and the current consumption of both buffers is the lowest. The medium and high settings have faster settling times, but the current consumption increases. See the device-specific data sheet for parameters.

### 30.2.4 Updating the DAC12\_A Voltage Output

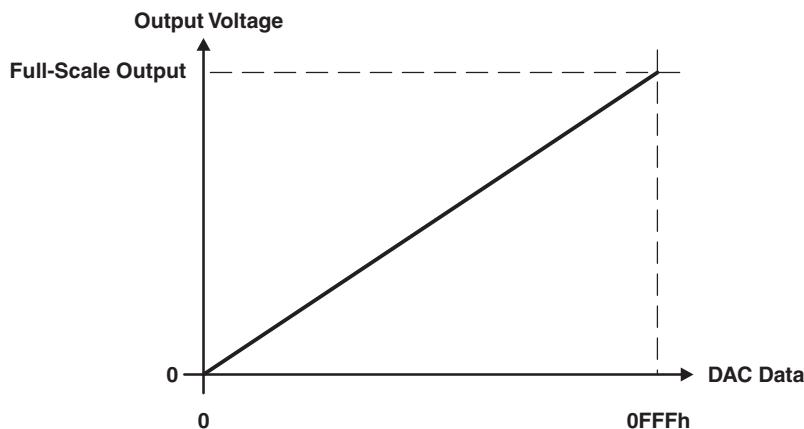
The DAC12\_xDAT register can be connected directly to the DAC core or double buffered. The trigger for updating the DAC voltage output is selected with the DAC12LSELx bits.

When DAC12LSELx = 0 the data latch is transparent and the DAC12\_xDAT register is applied directly to the DAC core. the DAC output updates immediately when new DAC data is written to the DAC12\_xDAT register, regardless of the state of the DAC12ENC bit.

When  $\text{DAC12LSELx} = 1$ , DAC data is latched and applied to the DAC core after new data is written to  $\text{DAC12}_x\text{DAT}$ . When  $\text{DAC12LSELx} = 2$  or  $3$ , data is latched on the rising edge from the  $\text{Timer0}_A5$  CCR1 output or  $\text{Timer}_B$  CCR2 output, respectively.  $\text{DAC12ENC}$  must be set to latch the new data when  $\text{DAC12LSELx} > 0$ .

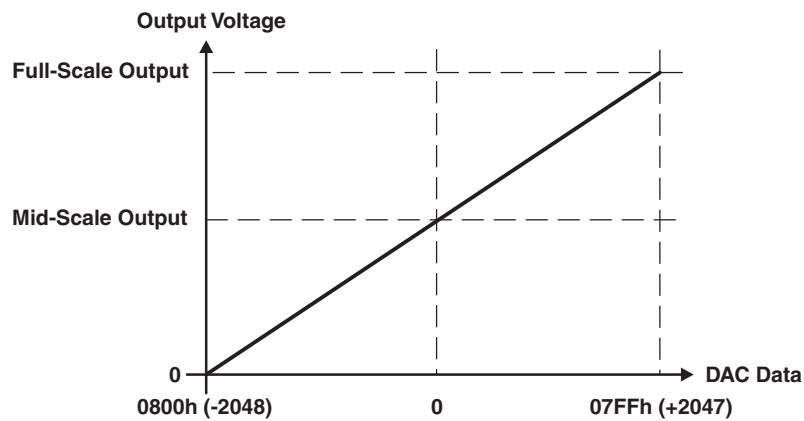
### 30.2.5 DAC12\_xDAT Data Formats

The DAC12\_A supports both straight binary and 2's complement data formats. When using straight binary data format, the full-scale output value is  $0\text{FFFh}$  in 12-bit mode ( $0\text{FFh}$  in 8-bit mode), as shown in [Figure 30-3](#).



**Figure 30-3. Output Voltage vs DAC Data, 12-Bit, Straight Binary Mode**

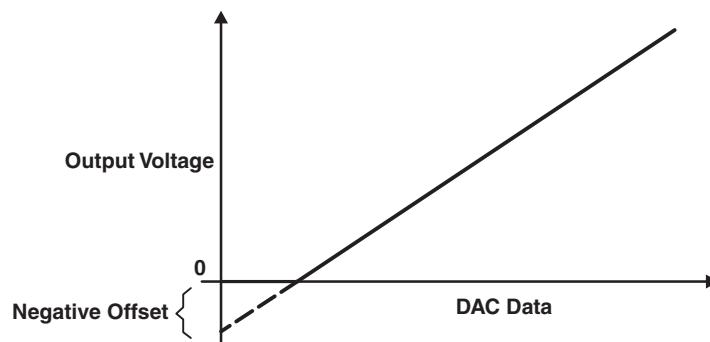
When using 2's complement data format, the range is shifted such that a  $\text{DAC12}_x\text{DAT}$  value of  $0800\text{h}$  ( $0080\text{h}$  in 8-bit mode) results in a zero output voltage,  $0000\text{h}$  is the mid-scale output voltage, and  $07FF\text{h}$  ( $007F\text{h}$  for 8-bit mode) is the full-scale voltage output, as shown in [Figure 30-4](#).



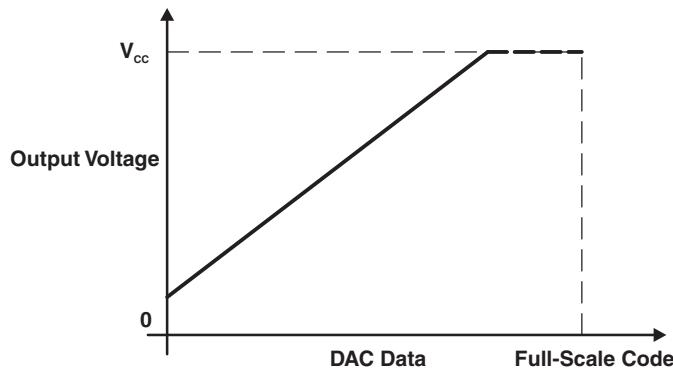
**Figure 30-4. Output Voltage vs DAC Data, 12-Bit, 2's complement Mode**

### 30.2.6 DAC12\_A Output Amplifier Offset Calibration

The offset voltage of the DAC output amplifier can be positive or negative. When the offset is negative, the output amplifier attempts to drive the voltage negative, but cannot do so. The output voltage remains at zero until the DAC digital input produces a sufficient positive output voltage to overcome the negative offset voltage, resulting in the transfer function shown in [Figure 30-5](#).

**Figure 30-5. Negative Offset**

When the output amplifier has a positive offset, a digital input of zero does not result in a zero output voltage. The DAC12\_A output voltage reaches the maximum output level before the DAC12\_A data reaches the maximum code. This is shown in [Figure 30-6](#).

**Figure 30-6. Positive Offset**

The DAC12\_A has the capability to calibrate the offset voltage of the output amplifier. Setting the DAC12CALON bit initiates the offset calibration. The calibration should complete before using the DAC. This can be checked by monitoring the DAC12CALON bit. When the calibration is complete, the DAC12CALON bit is automatically reset. The DAC12AMPx bits should be configured before calibration. For best calibration results, port and CPU activity should be minimized during calibration.

The contents of DAC12x\_CALDAT can be protected from inadvertent write access via a lock mechanism controlled by the DAC12x\_CALCTL register. At power up, the LOCK bit is set, and calibration is not possible and writing to the DAC12x\_CALDAT is ignored. To perform calibration, the LOCK bit must be cleared by writing the correct password to DAC12x\_CALCTL and clearing the LOCK bit. Once cleared calibration or writing to the DAC12x\_CALDAT can be performed. After calibration is performed, the user should lock the calibration registers by writing the correct password to DAC12x\_CALCTL and setting the LOCK bit.

Reading DAC12\_xCALDAT should only be performed while the DAC12CALON bit is cleared, otherwise incorrect values may be read. The DAC12xCAL data format is two's complement. Only the lower byte is used and the upper byte has no effect on the calibration.

### 30.2.7 Grouping Multiple DAC12\_A Modules

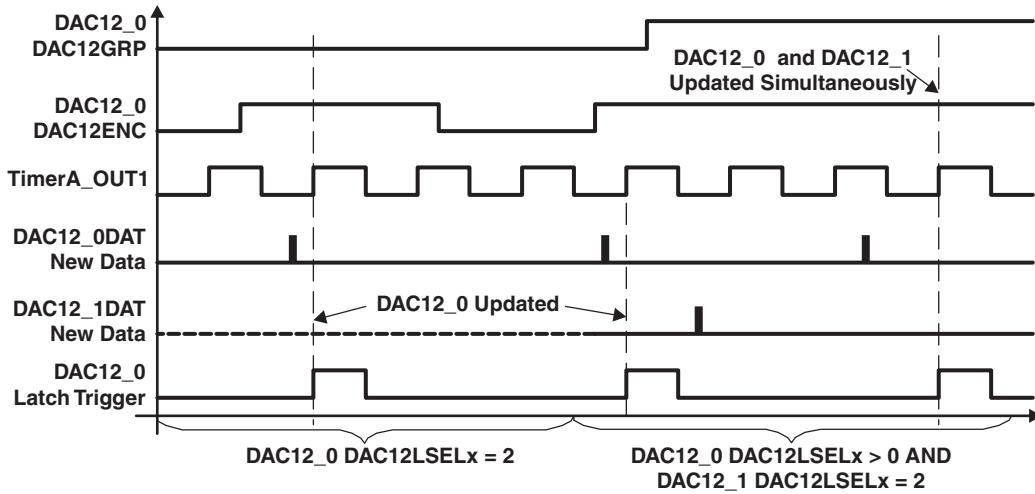
Multiple DAC12\_A modules can be grouped together with the DAC12GRP bit to synchronize the update of each DAC output. Hardware ensures that all DAC12\_A modules in a group update simultaneously independent of any interrupt or NMI event.

On devices that contain more than one DAC, DAC12\_0 and DAC12\_1 are grouped by setting the DAC12GRP bit of DAC12\_0. The DAC12GRP bit of DAC12\_1 is don't care. When DAC12\_0 and DAC12\_1 are grouped:

- The DAC12\_0 and DAC12\_1 DAC12LSELx bits select the update trigger for both DACs.
- The DAC12LSELx bits for both DACs must be the same.
- The DAC12LSELx bits for both DACs must be  $> 0$
- The DAC12ENC bits of both DACs must be set to 1

When DAC12\_0 and DAC12\_1 are grouped, both DAC12\_xDAT registers must be written to before the outputs update, even if data for one or both of the DACs is not changed. [Figure 30-7](#) shows a latch-update timing example for grouped DAC12\_0 and DAC12\_1.

When DAC12\_0 DAC12GRP = 1 and both DAC12\_x DAC12LSELx  $> 0$  and either DAC12ENC = 0, neither DAC updates.



**Figure 30-7. DAC12\_A Group Update Example, Timer\_A3 Trigger**

**NOTE: DAC12\_A Settling Time**

The DMA controller is capable of transferring data to the DAC12\_A faster than the DAC12\_A output can settle. The user must assure the DAC12\_A settling time is not violated when using the DMA controller. See the device-specific data sheet for parameters.

### 30.2.8 DAC12\_A Interrupts

The DAC12IFG bit is set when  $\text{DAC12LSELx} > 0$  and DAC data is latched from the DAC12\_xDAT register into the data latch. When  $\text{DAC12LSELx} = 0$ , the DAC12IFG flag is not set.

A set DAC12IFG bit indicates that the DAC is ready for new data. If both the DAC12IE and GIE bits are set, the DAC12IFG generates an interrupt request. The DAC12IFG flag must be reset by software or can be reset automatically by accessing the DAC12IV register. See the DAC12IV description for further information.

For devices that contain a DMA, each DAC channel has a DMA trigger associated with it. When DAC12IFG is set, it can trigger a DMA transfer to the DAC12x\_DAT register. DAC12IFG is automatically be reset when the transfer begins. If the DAC12IE is also set, no DMA transfer occurs when the DAC12IFG is set.

#### 30.2.8.1 DAC12IV, Interrupt Vector Generator

The DAC12\_A flags are prioritized and combined to source a single interrupt vector. The interrupt vector register DAC12IV is used to determine which flag requested an interrupt.

The highest priority enabled interrupt generates a number in the DAC12IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled DAC interrupts do not affect the DAC12IV value.

Any access, read or write, of the DAC12IV 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 DAC12IFG\_0 and DAC12IFG\_1 flags are set when the interrupt service routine accesses the DAC12IV register, the DAC12IFG\_0 flag is reset automatically. After the RETI instruction of the interrupt service routine is executed, the DAC12IFG\_1 flag generates another interrupt.

### DAC12IV Software Example

The following software example shows the recommended use of DAC12IV and the handling overhead. The DAC12IV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

```
; Interrupt handler for DAC12_A.          Cycles
DAC12_HND      ; Interrupt latency       6
                ; Add offset to Jump table 3
                ; Vector 0: No interrupt   5
                ; Vector 2: DAC12_0        2
                ; Vector 4: DAC12_1        2
                ; Vector 6: Reserved       5
                ; Vector 8: Reserved       5
                ; Vector 8: Reserved       5
                ; Vector 10: Reserved      5

DAC12IFG_1_HND ; Vector 4: DAC12_1
...             ; Task starts here
RETI           ; Back to main program

DAC12IFG_0_HND ; Vector 2: DAC12_0
...             ; Task starts here
RETI           ; Back to main program      5
```

### 30.3 DAC Outputs

On most devices, each DAC channel can be output to two different port pins selected via the DAC12OPS bit. When DAC12OPS = 0, one of the two ports is selected as the DAC output. Similarly, when DAC12OPS =1, the other port is selected as the DAC output. [Table 30-2](#) summarizes this for a single DAC channel that can be output to either ports Pm.y or Pn.z.

**Table 30-2. DAC Output Selection**

DAC12OPS	DAC12AMP	Pm.y Function	Pn.z Function
0	{0}	I/O	I/O
0	{1}	I/O	DAC output, 0 V
0	{>1}	I/O	DAC output
1	{0}	I/O	I/O
1	{1}	DAC output, 0 V	I/O
1	{>1}	DAC output	I/O

### 30.4 DAC12\_A Registers

The DAC12\_A registers are listed in [Table 30-3](#). The base address can be found in the device specific data sheet. The address offset is listed in [Table 30-3](#).

**Table 30-3. DAC12\_A Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	DAC12_0CTL0	DAC12_0 Control 0	Read/write	Word	0000h	<a href="#">Section 30.4.1</a>
02h	DAC12_0CTL1	DAC12_0 Control 1	Read/write	Word	0000h	<a href="#">Section 30.4.2</a>
04h	DAC12_0DAT	DAC12_0 Data	Read/write	Word	0000h	<a href="#">Section 30.4.3</a> through <a href="#">Section 30.4.10</a>
06h	DAC12_0CALCTL	DAC12_0 Calibration Control	Read/write	Word	9601h	<a href="#">Section 30.4.11</a>
08h	DAC12_0CALDAT	DAC12_0 Calibration Data	Read/write	Word	0000h	<a href="#">Section 30.4.12</a>
10h	DAC12_1CTL0	DAC12_1 Control 0	Read/write	Word	0000h	<a href="#">Section 30.4.1</a>
12h	DAC12_1CTL1	DAC12_1 Control 1	Read/write	Word	0000h	<a href="#">Section 30.4.2</a>
14h	DAC12_1DAT	DAC12_1 Data	Read/write	Word	0000h	<a href="#">Section 30.4.3</a> through <a href="#">Section 30.4.10</a>
16h	DAC12_1CALCTL	DAC12_1 Calibration Control	Read/write	Word	0000h	<a href="#">Section 30.4.11</a>
18h	DAC12_1CALDAT	DAC12_1 Calibration Data	Read/write	Word	0000h	<a href="#">Section 30.4.12</a>
1Eh	DAC12IV	DAC12IV	Read	Word	0000h	<a href="#">Section 30.4.13</a>

**30.4.1 DAC12\_xCTL0 Register**

DAC12 Control Register 0

**Figure 30-8. DAC12\_xCTL0 Register**

15	14	13	12	11	10	9	8
DAC12OPS	DAC12SREFx	DAC12RES	DAC12LSELx	DAC12CALON	DAC12IR		
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DAC12AMPx	DAC12DF	DAC12IE	DAC12IFG	DAC12ENC	DAC12GRP		
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
Can be modified only when DAC12ENC = 0							

**Table 30-4. DAC12\_xCTL0 Register Description**

Bit	Field	Type	Reset	Description
15	DAC12OPS	RW	0h	DAC output select 0b = DAC12_x channel output on Pm.y 1b = DAC12_x channel output on Pn.z
14-13	DAC12SREFx	RW	0h	DAC select reference voltage 00b = VREF+ 01b = AVCC 10b = VEREF+ 11b = VEREF+
12	DAC12RES	RW	0h	DAC resolution select 0b = 12-bit resolution 1b = 8-bit resolution
11-10	DAC12LSELx	RW	0h	DAC load select. Selects the load trigger for the DAC latch. DAC12ENC must be set for the DAC to update, except when DAC12LSELx = 0. 00b = DAC latch loads when DAC12_xDAT written (DAC12ENC is ignored) 01b = DAC latch loads when DAC12_xDAT written, or, when grouped, when all DAC12_xDAT registers in the group have been written. 10b = Rising edge of Timer_A.OUT1 (TA1) 11b = Rising edge of Timer_B.OUT2 (TB2)
9	DAC12CALON	RW	0h	DAC calibration on. This bit initiates the DAC offset calibration sequence and is automatically reset when the calibration completes. 0b = Calibration is not active 1b = Initiate calibration or calibration in progress
8	DAC12IR	RW	0h	DAC input range. The DAC12IR bit along with the DAC12OG bit set the reference input and voltage output range. 0b = If DAC12I0G = 0, then DAC12 full-scale output = 3x reference voltage; if DAC12I0G = 1, then DAC12 full-scale output = 2x reference voltage 1b = DAC12 full-scale output = 1x reference voltage

**Table 30-4. DAC12\_xCTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
7-5	DAC12AMPx	RW	0h	<p>DAC amplifier setting. These bits select settling time vs current consumption for the DAC input and output amplifiers.</p> <p>000b = Input Buffer: Off; Output Buffer: DAC off, output high impedance</p> <p>001b = Input Buffer: Off; Output Buffer: DAC off, output 0 V</p> <p>010b = Input Buffer: Low speed and current; Output Buffer: Low speed and current</p> <p>011b = Input Buffer: Low speed and current; Output Buffer: Medium speed and current</p> <p>100b = Input Buffer: Low speed and current; Output Buffer: High speed and current</p> <p>101b = Input Buffer: Medium speed and current; Output Buffer: Medium speed and current</p> <p>110b = Input Buffer: Medium speed and current; Output Buffer: High speed and current</p> <p>111b = Input Buffer: High speed and current; Output Buffer: High speed and current</p>
4	DAC12DF	RW	0h	<p>DAC data format</p> <p>0b = Straight binary</p> <p>1b = 2s complement</p>
3	DAC12IE	RW	0h	<p>DAC interrupt enable</p> <p>0b = Disabled</p> <p>1b = Enabled</p>
2	DAC12IFG	RW	0h	<p>DAC interrupt flag</p> <p>0b = No interrupt pending</p> <p>1b = Interrupt pending</p>
1	DAC12ENC	RW	0h	<p>DAC enable conversion. This bit enables the DAC12_A module when DAC12LSELx &gt; 0. when DAC12LSELx = 0, DAC12ENC is ignored.</p> <p>0b = DAC disabled</p> <p>1b = DAC enabled</p>
0	DAC12GRP	RW	0h	<p>DAC group. Groups DAC12_x with the next higher DAC12_x. Not used for DAC12_1 on dual DAC devices.</p> <p>0b = Not grouped</p> <p>1b = Grouped</p>

**30.4.2 DAC12\_xCTL1 Register**

DAC12 Control Register 1

**Figure 30-9. DAC12\_xCTL1 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							
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)

Can be modified only when DAC12ENC = 0

**Table 30-5. DAC12\_xCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved. Always reads as 0.
1	DAC12OG	RW	0h	DAC output buffer gain. Can be modified only when DAC12ENC = 0. 0b = 3x gain 1b = 2x gain
0	DAC12DJ	RW	0h	DAC data format justification. Can be modified only when DAC12ENC = 0. 0b = Data format right justified 1b = Data format left justified

### 30.4.3 DAC12\_xDAT Register, Unsigned 12-Bit Binary Format, Right Justified

DAC12 Data Register unsigned 12-bit binary format, right justified (DAC12RES = 0, DAC12DF = 0, DAC12DFJ = 0)

**Figure 30-10. DAC12\_xDAT Register**

15	14	13	12	11	10	9	8
Reserved				DAC12 Data			
r(0)	r(0)	r(0)	r(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 30-6. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-0	DAC12 Data	RW	0h	DAC data in unsigned format. Bit 11 represents the MSB.

### 30.4.4 DAC12\_xDAT Register, Unsigned 12-Bit Binary Format, Left Justified

DAC12 Data Register unsigned 12-bit binary format, left justified (DAC12RES = 0, DAC12DF = 0, DAC12DFJ = 1)

**Figure 30-11. DAC12\_xDAT Register**

15	14	13	12	11	10	9	8
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DAC12 Data				Reserved			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r(0)	r(0)	r(0)	r(0)

**Table 30-7. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-4	DAC12 Data	RW	0h	DAC data in unsigned format. Bit 15 represents the MSB.
3-0	Reserved	R	0h	Reserved. Always reads as 0.

**30.4.5 DAC12\_xDAT Register, 2s-Complement 12-Bit Binary Format, Right Justified**

DAC12 Data Register 2s complement 12-bit binary format, right justified (DAC12RES = 0, DAC12DF = 1, DAC12DFJ = 0)

**Figure 30-12. DAC12\_xDAT Register**

15	14	13	12	11	10	9	8
DAC12 Data							
Bit 11	Bit 11	Bit 11	Bit 11				
r(0)	r(0)	r(0)	r(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 30-8. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-12	Bit 11	R	0h	These bits are sign extension bits and are equal to the contents of bit 11. These bits are automatically updated with the contents of bit 11.
11-0	DAC12 Data	RW	0h	DAC data in twos complement format. Bit 11 represents the sign bit of the twos complement value.

**30.4.6 DAC12\_xDAT Register, 2s-Complement 12-Bit Binary Format, Left Justified**

DAC12 Data Register 2s complement 12-bit binary format, left justified (DAC12RES = 0, DAC12DF = 1, DAC12DFJ = 1)

**Figure 30-13. DAC12\_xDAT Register**

15	14	13	12	11	10	9	8
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DAC12 Data				Reserved			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r(0)	r(0)	r(0)	r(0)

**Table 30-9. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-4	DAC12 Data	RW	0h	DAC data in twos complement format. Bit 15 represents the sign bit of the twos complement value.
3-0	Reserved	R	0h	Reserved. Always reads as 0.

### 30.4.7 DAC12\_xDAT Register, Unsigned 8-Bit Binary Format, Right Justified

DAC12 Data Register unsigned 8-bit binary format, right justified (DAC12RES = 1, DAC12DF = 0, DAC12DFJ = 0)

**Figure 30-14. DAC12\_xDAT 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
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 30-10. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7-0	DAC12 Data	RW	0h	DAC data in unsigned format. Bit 7 represents the MSB.

### 30.4.8 DAC12\_xDAT Register, Unsigned 8-Bit Binary Format, Left Justified

DAC12 Data Register unsigned 8-bit binary format, left justified (DAC12RES = 1, DAC12DF = 0, DAC12DFJ = 1)

**Figure 30-15. DAC12\_xDAT Register**

15	14	13	12	11	10	9	8
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved							
r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)

**Table 30-11. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-8	DAC12 Data	RW	0h	DAC data in unsigned format. Bit 15 represents the MSB.
7-0	Reserved	R	0h	Reserved. Always reads as 0.

**30.4.9 DAC12\_xDAT Register, 2s-Complement 8-Bit Binary Format, Right Justified**

DAC12 Data Register 2s complement 8-bit binary format, right justified (DAC12RES = 1, DAC12DF = 1, DAC12DFJ = 0)

**Figure 30-16. DAC12\_xDAT Register**

15	14	13	12	11	10	9	8
Bit 7	Bit 7	Bit 7	Bit 7	Bit 7	Bit 7	Bit 7	Bit 7
r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)
7	6	5	4	3	2	1	0
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 30-12. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-8	Bit 7	R	0h	These bits are sign extension bits and are equal to the contents of bit 7. These bits are automatically updated with the contents of bit 7.
7-0	DAC12 Data	RW	0h	DAC data in twos complement format. Bit 7 represents the sign bit of the twos complement value.

**30.4.10 DAC12\_xDAT Register, 2s-Complement 8-Bit Binary Format, Left Justified**

DAC12 Data Register 2s complement 8-bit binary format, left justified (DAC12RES = 1, DAC12DF = 1, DAC12DFJ = 1)

**Figure 30-17. DAC12\_xDAT Register**

15	14	13	12	11	10	9	8
DAC12 Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved							
r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)

**Table 30-13. DAC12\_xDAT Register Description**

Bit	Field	Type	Reset	Description
15-8	DAC12 Data	RW	0h	DAC data in twos complement format. Bit 15 represents the sign bit of the twos complement value.
7-0	Reserved	R	0h	Reserved. Always reads as 0.

### 30.4.11 DAC12\_xCALCTL Register

DAC12 Calibration Control Register

**Figure 30-18. DAC12\_xCALCTL Register**

15	14	13	12	11	10	9	8
<b>DAC12KEY</b>							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	rw-(1)

**Table 30-14. DAC12\_xCALCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	DAC12KEY	RW	96h	DAC calibration lock password. Always reads as 0x96. Must be written with 0xA5 for LOCK bit to be set or cleared. An incorrect key results in the LOCK bit being set, thereby disabling write access to DAC12_xCALDAT.
7-1	Reserved	R	0h	Reserved. Always reads as 0.
0	LOCK	RW	1h	DAC calibration lock. To enable write access to the DAC12 calibration register, this bit must be cleared by writing 0xA5 to DAC12KEY along with LOCK = 0. Writing an incorrect key or writing to DAC12x_CALCTL using byte mode causes the LOCK bit to be automatically set. If the LOCK bit is set, write access to the calibration registers and hardware calibration is not possible. All previous values in the DAC12_xCALDAT are retained. 0b = Calibration register write access enabled, calibration can be performed. 1b = Calibration register write access disabled, calibration disabled.

### 30.4.12 DAC12\_xCALDAT Register

DAC12 Calibration DATA Register

**Figure 30-19. DAC12\_xCALDAT 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
DAC12 Calibration Data							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**Table 30-15. DAC12\_xCALDAT Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7-0	DAC12 Calibration Data	RW	0h	DAC calibration data. The DAC calibration data is represented in twos complement format providing a range of -128 to +127.

**30.4.13 DAC12IV Register**

DAC12 Interrupt Vector Register

**Figure 30-20. DAC12IV Register**

15	14	13	12	11	10	9	8
DAC12IVx							
r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)
7	6	5	4	3	2	1	0
DAC12IVx							
r(0)	r(0)	r(0)	r(0)	r-(0)	r-(0)	r-(0)	r(0)

**Table 30-16. DAC12IV Register Description**

Bit	Field	Type	Reset	Description
15-0	DAC12IVx	R	0h	<p>DAC interrupt vector value</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: DAC12 channel 0; Interrupt Flag: DAC12IFG_0; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: DAC12 channel 1; Interrupt Flag: DAC12IFG_1; Interrupt Priority: Lowest</p> <p>06h = Reserved</p> <p>08h = Reserved</p> <p>0Ah = Reserved</p> <p>0Ch = Reserved</p> <p>0Eh = Reserved</p>

**Comp\_B**

Comp\_B is an analog voltage comparator. This chapter describes the Comp\_B. Comp\_B covers general comparator functionality for up to 16 channels.

Topic	Page
31.1 Comp_B Introduction .....	808
31.2 Comp_B Operation .....	809
31.3 Comp_B Registers .....	814

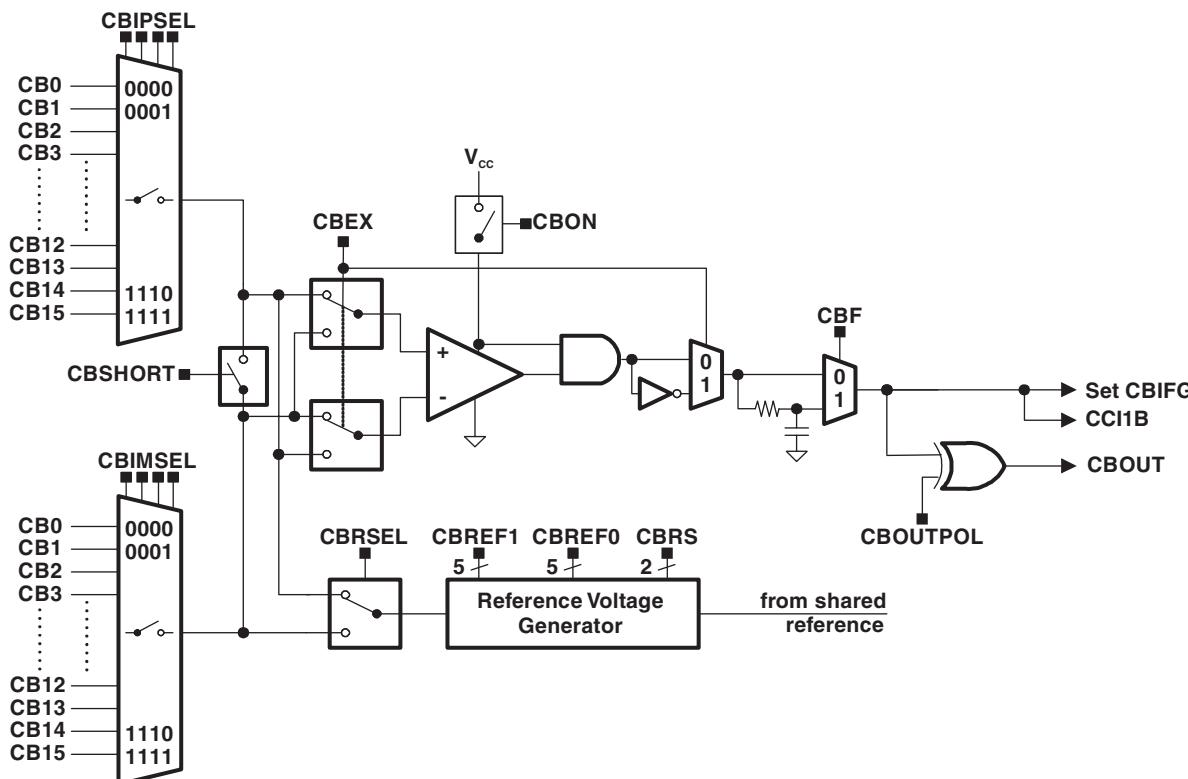
### 31.1 Comp\_B Introduction

The Comp\_B module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals.

Features of Comp\_B include:

- Inverting and noninverting terminal input multiplexer
- Software-selectable RC filter for the comparator output
- Output provided to Timer\_A capture input
- Software control of the port input buffer
- Interrupt capability
- Selectable reference voltage generator, voltage hysteresis generator
- Reference voltage input from shared reference
- Ultra-low-power comparator mode
- Interrupt driven measurement system – low-power operation support

The Comp\_B block diagram is shown in [Figure 31-1](#).



**Figure 31-1. Comp\_B Block Diagram**

## 31.2 Comp\_B Operation

The Comp\_B module is configured by user software. The setup and operation of Comp\_B is discussed in the following sections.

### 31.2.1 Comparator

The comparator compares the analog voltages at the + and – input terminals. If the + terminal is more positive than the – terminal, the comparator output CBOU is high. The comparator can be switched on or off using control bit CBON. The comparator should be switched off when not in use to reduce current consumption. When the comparator is switched off, CBOU is always low. The bias current of the comparator is programmable.

### 31.2.2 Analog Input Switches

The analog input switches connect or disconnect the two comparator input terminals to associated port pins using the CBIPSELx and CBIMSELx bits. The comparator terminal inputs can be controlled individually. The CBIPSELx/CBIMSELx bits allow:

- Application of an external signal to the + and – terminals of the comparator
- Routing of an internal reference voltage to an associated output port pin
- Application of an external current source (for example, a resistor) to the + or – terminal of the comparator
- The mapping of both terminals of the internal multiplexer to the outside

Internally, the input switch is constructed as a T-switch to suppress distortion in the signal path.

---

**NOTE: Comparator Input Connection**

When the comparator is on, the input terminals should be connected to a signal, power, or ground. Otherwise, floating levels may cause unexpected interrupts and increased current consumption.

---

The CBEX bit controls the input multiplexer, permuting the input signals of the comparator's + and – terminals. Additionally, when the comparator terminals are permuted, the output signal from the comparator is inverted too. This allows the user to determine or compensate for the comparator input offset voltage.

### 31.2.3 Port Logic

The Px.y pins associated with a comparator channel are enabled by the CBIPSELx or CBIMSELx bits to disable its digital components while used as comparator input. Only one of the comparator input pins is selected as input to the comparator by the input multiplexer at a time.

### 31.2.4 Input Short Switch

The CBSHORT bit shorts the Comp\_B inputs. This can be used to build a simple sample-and-hold for the comparator as shown in [Figure 31-2](#).

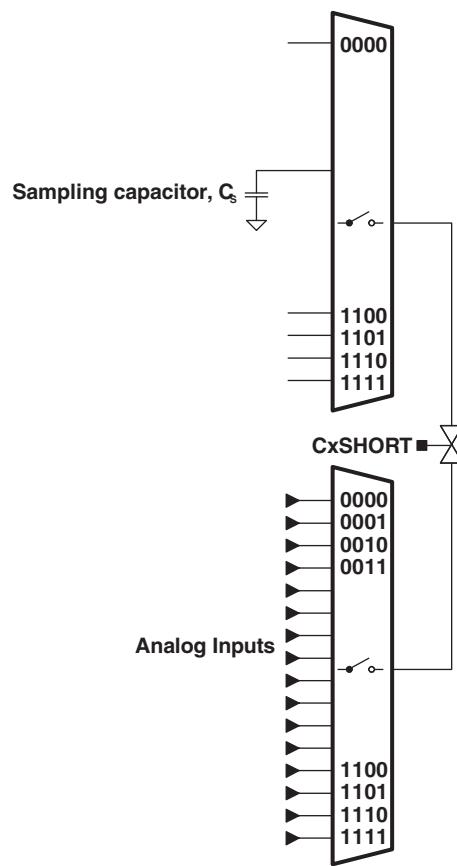


Figure 31-2. Comp\_B Sample-And-Hold

The required sampling time is proportional to the size of the sampling capacitor ( $C_s$ ), the resistance of the input switches in series with the short switch ( $R_i$ ), and the resistance of the external source ( $R_s$ ). The total internal resistance ( $R_i$ ) is typically in the range of 1 kΩ. The sampling capacitor  $C_s$  should be greater than 100 pF. The time constant, Tau, to charge the sampling capacitor  $C_s$  can be calculated with the following equation:

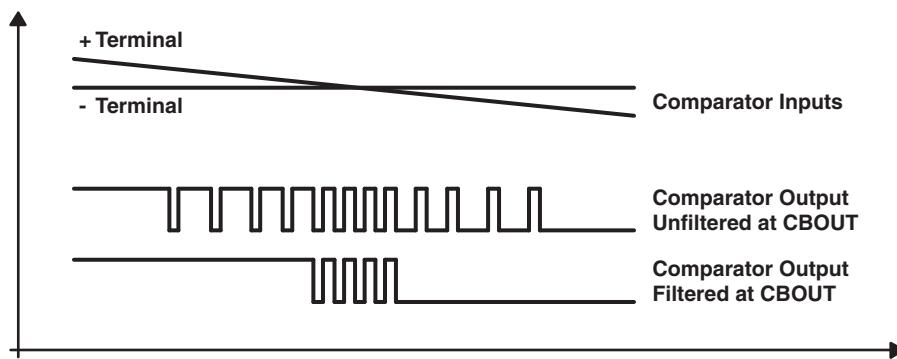
$$\text{Tau} = (R_i + R_s) \times C_s$$

Depending on the required accuracy, 3 to 10 Tau should be used as a sampling time. With 3 Tau the sampling capacitor is charged to approximately 95% of the input signals voltage level, with 5 Tau it is charged to more than 99%, and with 10 Tau the sampled voltage is sufficient for 12-bit accuracy.

### 31.2.5 Output Filter

The output of the comparator can be used with or without internal filtering. When control bit CBF is set, the output is filtered with an on-chip RC filter. The delay of the filter can be adjusted in four different steps.

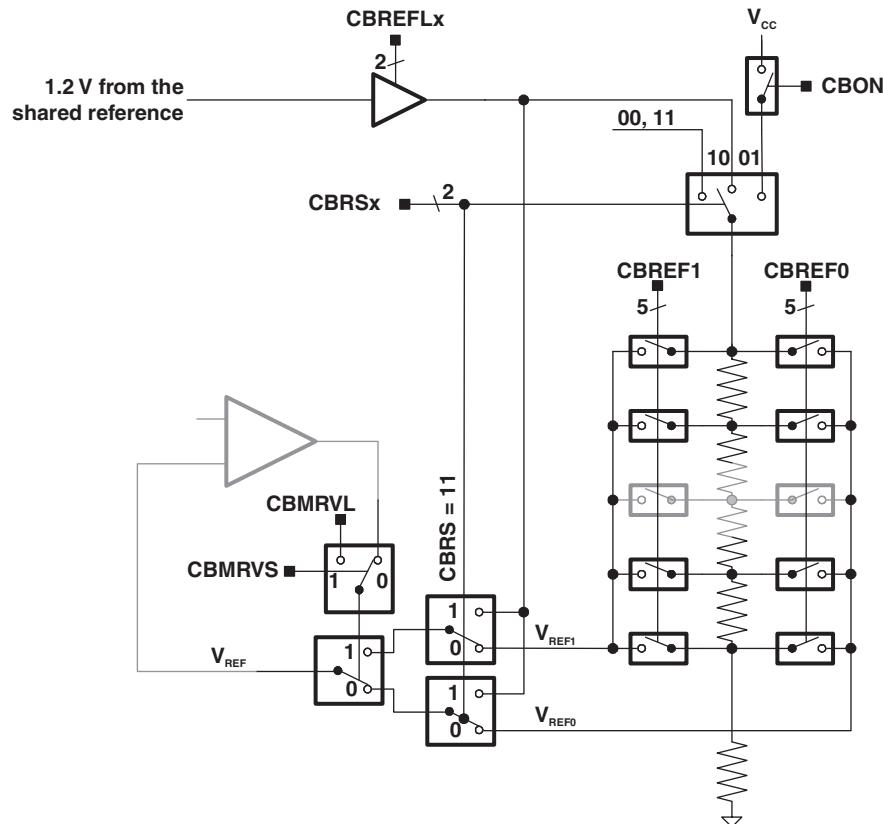
All comparator outputs are oscillating if the voltage difference across the input terminals is small. Internal and external parasitic effects and cross coupling on and between signal lines, power supply lines, and other parts of the system are responsible for this behavior as shown in Figure 31-3. The comparator output oscillation reduces the accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.



**Figure 31-3. RC-Filter Response at the Output of the Comparator**

### 31.2.6 Reference Voltage Generator

The Comp\_B reference block diagram is shown in Figure 31-4.



**Figure 31-4. Reference Generator Block Diagram**

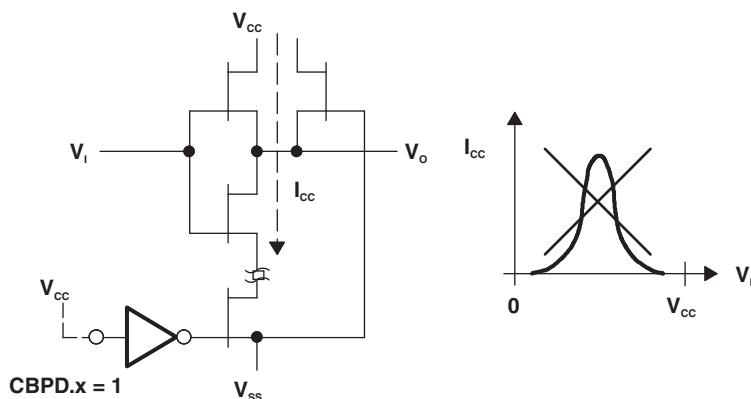
The voltage reference generator is used to generate VREF, which can be applied to either comparator input terminal. The CBREF1x (VREF1) and CBREF0x (VREF0) bits control the output of the voltage generator. The CBRSEL bit selects the comparator terminal to which VREF is applied. If external signals are applied to both comparator input terminals, the internal reference generator should be turned off to reduce current consumption. The voltage reference generator can generate a fraction of the device's  $V_{CC}$  or of the voltage reference of the integrated precision voltage reference source. Vref1 is used while CBOUT is 1 and Vref0 is used while CBOUT is 0. This allows the generation of a hysteresis without using external components.

### 31.2.7 Comp\_B, Port Disable Register CBPD

The comparator input and output functions are multiplexed with the associated I/O port pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from  $V_{cc}$  to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption.

The CBPDx bits, when set, disable the corresponding Px.y input buffer as shown in [Figure 31-5](#). When current consumption is critical, any Px.y pin connected to analog signals should be disabled with their associated CBPDx bits.

Selecting an input pin to the comparator multiplexer with the CBIPSEL or CBIMSEL bits automatically disables the input buffer for that pin, regardless of the state of the associated CBPDx bit.



**Figure 31-5. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer**

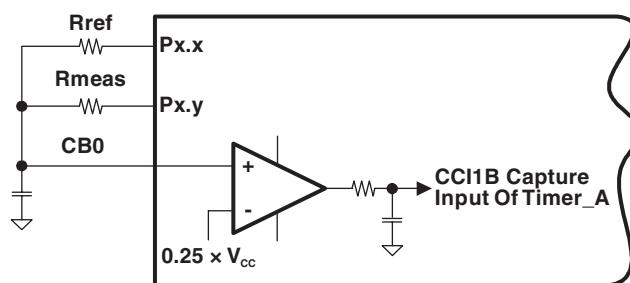
### 31.2.8 Comp\_B Interrupts

One interrupt flag and one interrupt vector is associated with the Comp\_B.

The interrupt flag CBIFG is set on either the rising or falling edge of the comparator output, selected by the CBIES bit. If both the CBIE and the GIE bits are set, then the CBIFG interrupt flag generates an interrupt request.

### 31.2.9 Comp\_B Used to Measure Resistive Elements

The Comp\_B can be optimized to precisely measure resistive elements using single slope analog-to-digital conversion. For example, temperature can be converted into digital data using a thermistor, by comparing the thermistor's capacitor discharge time to that of a reference resistor as shown in [Figure 31-6](#). A reference resistor Rref is compared to Rmeas.



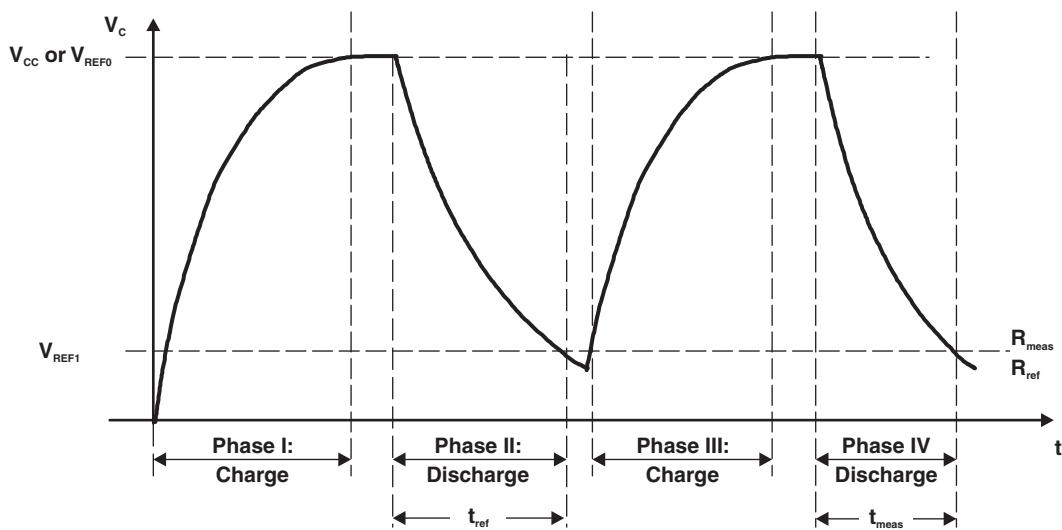
**Figure 31-6. Temperature Measurement System**

The resources used to calculate the temperature sensed by Rmeas are:

- Two digital I/O pins charge and discharge the capacitor.
- I/O is set to output high ( $V_{CC}$ ) to charge capacitor, reset to discharge.
- I/O is switched to high-impedance input with CBPDx set when not in use.
- One output charges and discharges the capacitor via Rref.
- One output discharges capacitor via Rmeas.
- The + terminal is connected to the positive terminal of the capacitor.
- The – terminal is connected to a reference level, for example  $0.25 \times V_{CC}$ .
- The output filter should be used to minimize switching noise.
- COUT is used to gate Timer\_A CCI1B, capturing capacitor discharge time.

More than one resistive element can be measured. Additional elements are connected to CB0 with available I/O pins and switched to high impedance when not being measured.

The thermistor measurement is based on a ratiometric conversion principle. The ratio of two capacitor discharge times is calculated as shown in [Figure 31-7](#).



**Figure 31-7. Timing for Temperature Measurement Systems**

The  $V_{CC}$  voltage and the capacitor value should remain constant during the conversion, but are not critical since they cancel in the ratio:

$$\frac{N_{\text{meas}}}{N_{\text{ref}}} = \frac{-R_{\text{meas}} \times C \times \ln \frac{V_{\text{ref1}}}{V_{\text{CC}}}}{-R_{\text{ref}} \times C \times \ln \frac{V_{\text{ref1}}}{V_{\text{CC}}}}$$

$$\frac{N_{\text{meas}}}{N_{\text{ref}}} = \frac{R_{\text{meas}}}{R_{\text{ref}}}$$

$$R_{\text{meas}} = R_{\text{ref}} \times \frac{N_{\text{meas}}}{N_{\text{ref}}}$$

### 31.3 Comp\_B Registers

The Comp\_B registers are listed in [Table 31-1](#). The base address of the Comp\_B module can be found in the device-specific data sheet.

**Table 31-1. Comp\_B Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	CBCTL0	Comp_B control register 0	Read/write	Word	0000h	<a href="#">Section 31.3.1</a>
02h	CBCTL1	Comp_B control register 1	Read/write	Word	0000h	<a href="#">Section 31.3.2</a>
04h	CBCTL2	Comp_B control register 2	Read/write	Word	0000h	<a href="#">Section 31.3.3</a>
06h	CBCTL3	Comp_B control register 3	Read/write	Word	0000h	<a href="#">Section 31.3.4</a>
0Ch	CBINT	Comp_B interrupt register	Read/write	Word	0000h	<a href="#">Section 31.3.5</a>
0Eh	CBIV	Comp_B interrupt vector word	Read	Word	0000h	<a href="#">Section 31.3.6</a>

### 31.3.1 CBCTL0 Register

Comp\_B Control Register 0

**Figure 31-8. CBCTL0 Register**

15	14	13	12	11	10	9	8
CBIMEN	Reserved				CBIMSEL		
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CBIPEN	Reserved				CBIPSEL		
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

**Table 31-2. CBCTL0 Register Description**

Bit	Field	Type	Reset	Description
15	CBIMEN	RW	0h	Channel input enable for the V– terminal of the comparator. 0b = Selected analog input channel for V– terminal is disabled. 1b = Selected analog input channel for V– terminal is enabled.
14-12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	CBIMSEL	RW	0h	Channel input selected for the V– terminal of the comparator if CBIMEN is set to 1.
7	CBIPEN	RW	0h	Channel input enable for the V+ terminal of the comparator. 0b = Selected analog input channel for V+ terminal is disabled. 1b = Selected analog input channel for V+ terminal is enabled.
6-4	Reserved	R	0h	Reserved. Always reads as 0.
3-0	CBIPSEL	RW	0h	Channel input selected for the V+ terminal of the comparator if CBIPEN is set to 1.

**31.3.2 CBCTL1 Register**

Comp\_B Control Register 1

**Figure 31-9. CBCTL1 Register**

15	14	13	12	11	10	9	8
Reserved		CBMRVS	CBMRVL	CBON	CBPWRMD		
r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CBFDLY	CBEX	CBSHORT	CBIES	CBF	CBOUTPOL	CBOUT	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

**Table 31-3. CBCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12	CBMRVS	RW	0h	This bit defines if the comparator output selects between VREF0 or VREF1 if CBRS = 00, 01, or 10. 0b = Comparator output state selects between VREF0 or VREF1. 1b = CBMRVL selects between VREF0 or VREF1.
11	CBMRVL	RW	0h	This bit is valid of CBMRVS is set to 1. 0b = VREF0 is selected if CBRS = 00, 01, or 10. 1b = VREF1 is selected if CBRS = 00, 01, or 10.
10	CBON	RW	0h	On. This bit turns the comparator on. When the comparator is turned off the Comp_B consumes no power. 0b = Off 1b = On
9-8	CBPWRMD	RW	0h	Power mode. Not all modes are supported in all products. See devices specific data sheet for details. 00b = High-speed mode (optional) 01b = Normal mode (optional) 10b = Ultra-low-power mode (optional) 11b = Reserved
7-6	CBFDLY	RW	0h	Filter delay. The filter delay can be selected in 4 steps. See the device-specific data sheet for details. 00b = Typical filter delay of 450 ns 01b = Typical filter delay of 900 ns 10b = Typical filter delay of 1800 ns 11b = Typical filter delay of 3600 ns
5	CBEX	RW	0h	Exchange. This bit permutes the comparator 0 inputs and inverts the comparator 0 output.
4	CBSHORT	RW	0h	Input short. This bit shorts the + and – input terminals. 0b = Inputs not shorted 1b = Inputs shorted
3	CBIES	RW	0h	Interrupt edge select for CBIIFG and CBIFG 0b = Rising edge for CBIFG, falling edge for CBIIFG 1b = Falling edge for CBIFG, rising edge for CBIIFG
2	CBF	RW	0h	Output filter 0b = Comp_B output is not filtered 1b = Comp_B output is filtered
1	CBOUTPOL	RW	0h	Output polarity. This bit defines the CBOUT polarity. 0b = Noninverted 1b = Inverted

**Table 31-3. CBCTL1 Register Description (continued)**

Bit	Field	Type	Reset	Description
0	CBOUT	R	0h	Output value. This bit reflects the value of the Comp_B output. Writing this bit has no effect on the comparator output.

**31.3.3 CBCTL2 Register**

Comp\_B Control Register 2

**Figure 31-10. CBCTL2 Register**

15	14	13	12	11	10	9	8
CBREFACC	CBREFL		CBREF1				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CBRS		CBRSEL		CBREFO			
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 31-4. CBCTL2 Register Description**

Bit	Field	Type	Reset	Description
15	CBREFACC	RW	0h	Reference accuracy. A reference voltage is requested only if CBREFL > 0. 0b = Static mode 1b = Clocked (low-power, low-accuracy) mode
14-13	CBREFL	RW	0h	Reference voltage level 00b = Reference voltage is disabled. No reference voltage is requested. 01b = 1.5 V 10b = 2.0 V 11b = 2.5 V
12-8	CBREF1	RW	0h	Reference resistor tap 1. This register defines the tap of the resistor string while CBOUT = 1.
7-6	CBRS	RW	0h	Reference source. This bit define if the reference voltage is derived from VCC or from the precise shared reference. 00b = No current is drawn by the reference curcuitry. 01b = VCC applied to the resistor ladder 10b = Shared reference voltage applied to the resistor ladder. 11b = Shared reference voltage supplied to V(CREF). Resistor ladder is off.
5	CBRSEL	RW	0h	Reference select. This bit selects which terminal the V(CCREF) is applied to. 0b = When CBEX = 0: V(REF) is applied to the + terminal; When CBEX = 1: V(REF) is applied to the – terminal 1b = When CBEX = 0: V(REF) is applied to the – terminal; When CBEX = 1: V(REF) is applied to the + terminal
4-0	CBREFO	RW	0h	Reference resistor tap 0. This register defines the tap of the resistor string while CBOUT = 0.

### 31.3.4 CBCTL3 Register

Comp\_B Control Register 3

**Figure 31-11. CBCTL3 Register**

15	14	13	12	11	10	9	8
CBPD15	CBPD14	CBPD13	CBPD14	CBPD11	CBPD10	CBPD9	CBPD8
rw-(0)							
7	6	5	4	3	2	1	0
CBPD7	CBPD6	CBPD5	CBPD4	CBPD3	CBPD2	CBPD1	CBPD0
rw-(0)							

**Table 31-5. CBCTL3 Register Description**

Bit	Field	Type	Reset	Description
15	CBPD15	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD15 disables the port of the comparator channel 15. 0b = Input buffer enabled 1b = Input buffer disabled
14	CBPD14	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD14 disables the port of the comparator channel 14. 0b = Input buffer enabled 1b = Input buffer disabled
13	CBPD13	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD13 disables the port of the comparator channel 13. 0b = Input buffer enabled 1b = Input buffer disabled
12	CBPD12	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD12 disables the port of the comparator channel 12. 0b = Input buffer enabled 1b = Input buffer disabled
11	CBPD11	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD11 disables the port of the comparator channel 11. 0b = Input buffer enabled 1b = Input buffer disabled
10	CBPD10	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD10 disables the port of the comparator channel 10. 0b = Input buffer enabled 1b = Input buffer disabled
9	CBPD9	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD9 disables the port of the comparator channel 9. 0b = Input buffer enabled 1b = Input buffer disabled
8	CBPD8	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD8 disables the port of the comparator channel 8. 0b = Input buffer enabled 1b = Input buffer disabled

**Table 31-5. CBCTL3 Register Description (continued)**

Bit	Field	Type	Reset	Description
7	CBPD7	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD7 disables the port of the comparator channel 7. 0b = Input buffer enabled 1b = Input buffer disabled
6	CBPD6	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD6 disables the port of the comparator channel 6. 0b = Input buffer enabled 1b = Input buffer disabled
5	CBPD5	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD5 disables the port of the comparator channel 5. 0b = Input buffer enabled 1b = Input buffer disabled
4	CBPD4	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD4 disables the port of the comparator channel 4. 0b = Input buffer enabled 1b = Input buffer disabled
3	CBPD3	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD3 disables the port of the comparator channel 3. 0b = Input buffer enabled 1b = Input buffer disabled
2	CBPD2	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD2 disables the port of the comparator channel 2. 0b = Input buffer enabled 1b = Input buffer disabled
1	CBPD1	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD1 disables the port of the comparator channel 1. 0b = Input buffer enabled 1b = Input buffer disabled
0	CBPD0	RW	0h	Port disable. This bit individually disables the input buffer for the pins of the port associated with Comp_B. The bit CBPD0 disables the port of the comparator channel 0. 0b = Input buffer enabled 1b = Input buffer disabled

### 31.3.5 CBINT Register

Comp\_B Interrupt Control Register

**Figure 31-12. CBINT Register**

15	14	13	12	11	10	9	8
Reserved						CBIIE	CBIE
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved						CBIIFG	CBIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

**Table 31-6. CBINT Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9	CBIIE	RW	0h	Comp_B output interrupt enable inverted polarity 0b = Interrupt is disabled 1b = Interrupt is enabled
8	CBIE	RW	0h	Comp_B output interrupt enable 0b = Interrupt is disabled 1b = Interrupt is enabled
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1	CBIIFG	RW	0h	Comp_B output inverted interrupt flag. The bit CBIES defines the transition of the output setting this bit. 0b = No interrupt pending 1b = Output interrupt pending
0	CBIFG	RW	0h	Comp_B output interrupt flag. The bit CBIES defines the transition of the output setting this bit. 0b = No interrupt pending 1b = Output interrupt pending

**31.3.6 CBIV Register**

Comp\_B Interrupt Vector Word Register

**Figure 31-13. CBIV Register**

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

**Table 31-7. CBIV Register Description**

Bit	Field	Type	Reset	Description
15-0	CBIV	R	0h	Comp_B interrupt vector word register. The interrupt vector register reflects only interrupt flags whose interrupt enable bit are set. Reading the CBIV register clears the pending interrupt flag with the highest priority. 00h = No interrupt pending 02h = Interrupt Source: CBOUT interrupt; Interrupt Flag: CBIFG; Interrupt Priority: Highest 04h = Interrupt Source: CBOUT interrupt inverted polarity; Interrupt Flag: CBIIFG; Interrupt Priority: Lowest

## **LCD\_B Controller**

The LCD\_B controller drives static, 2-mux, 3-mux, or 4-mux LCDs. This chapter describes the LCD\_B controller.

Topic	Page
32.1 LCD_B Controller Introduction .....	824
32.2 LCD_B Controller Operation .....	826
32.3 LCD_B Registers .....	844

### 32.1 LCD\_B Controller Introduction

The LCD\_B controller directly drives LCD displays by creating the ac segment and common voltage signals automatically. The LCD\_B controller can support static, 2-mux, 3-mux, and 4-mux LCD glasses.

The LCD\_B controller features are:

- Display memory
- Automatic signal generation
- Configurable frame frequency
- Blinking of individual segments with separate blinking memory
- Regulated charge pump
- Contrast control by software
- Support for four types of LCDs
  - Static
  - 2-mux, 1/2 bias or 1/3 bias
  - 3-mux, 1/2 bias or 1/3 bias
  - 4-mux, 1/2 bias or 1/3 bias

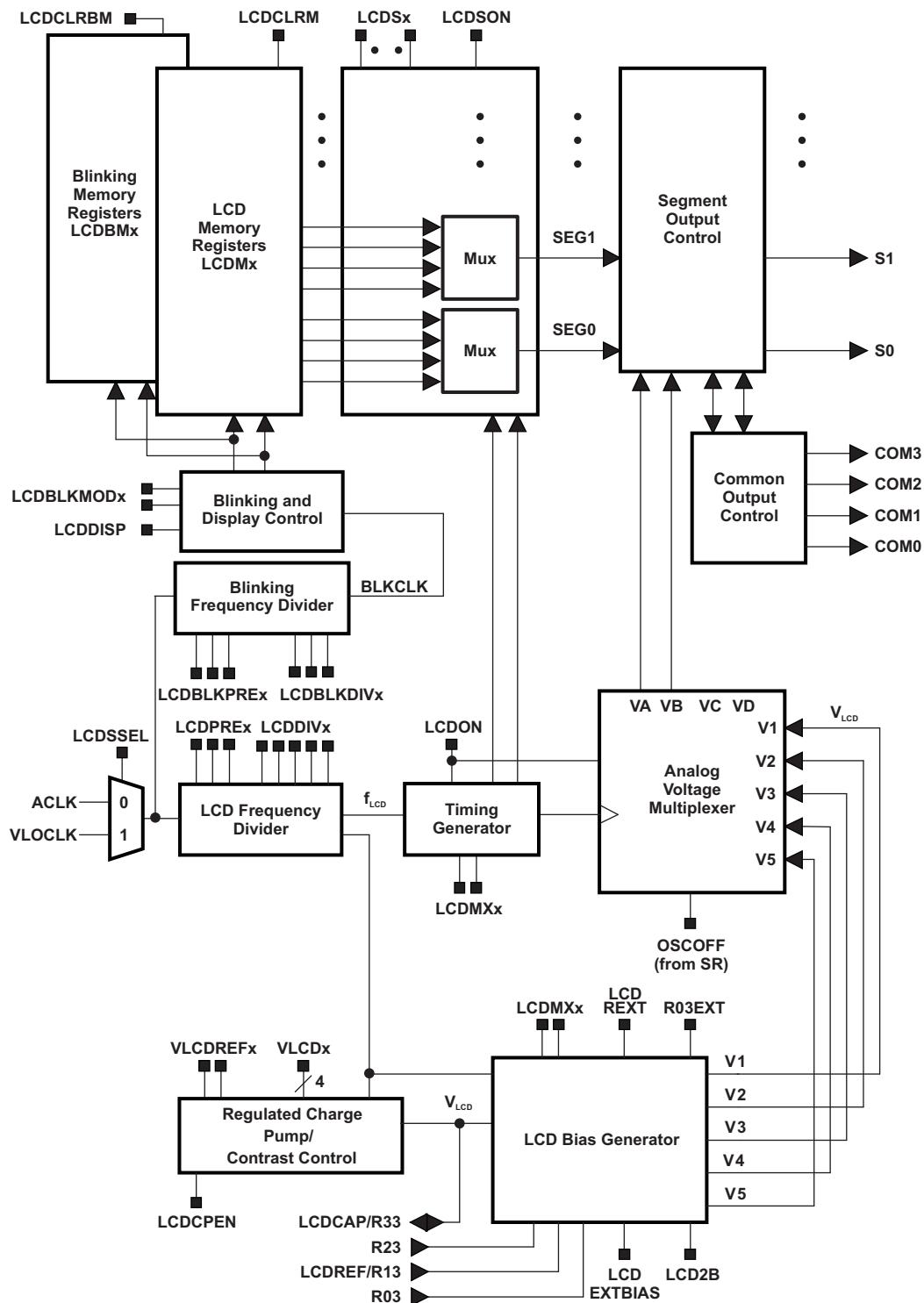
The LCD\_B controller block diagram for a configuration with a maximum of 160 segments is shown in [Figure 32-1](#).

---

**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.

---


**Figure 32-1. LCD\_B Controller Block Diagram**

## 32.2 LCD\_B Controller Operation

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

### **32.2.1 LCD Memory**

The LCD memory map for a device with a 160-segment maximum is shown in Figure 32-2. Each memory bit corresponds to one LCD segment 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 LCDM1, LCDM3, etc.

Setting the bit LCDCLRM clears all LCD memory registers at the next frame boundary. It is reset automatically after the registers are cleared.

**Figure 32-2. LCD Memory - Example for 160 Segments Maximum**

### **32.2.2 LCD Timing Generation**

The LCD\_B controller uses the  $f_{LCD}$  signal from the integrated clock divider to generate the timing for common and segment lines. With the LCDSEL bit ACLK with a frequency between 30 kHz and 40 kHz or VLOCLK can be selected as clock source into the divider. The  $f_{LCD}$  frequency is selected with the LCDPREx and LCDDIVx bits. The resulting  $f_{LCD}$  frequency is calculated by:

$$f_{LCD} = \frac{f_{ACLK/VLOCLK}}{(LCDDIVx + 1) \times 2^{LCDPRE}}$$

The proper  $f_{LCD}$  frequency depends on the LCD's requirement for framing frequency and the LCD multiplex rate and 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 30 Hz to 100 Hz:

$$f_{Frame} \text{ (from LCD data sheet)} = 30 \text{ Hz to } 100 \text{ Hz}$$

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

$$f_{LCD}(\min) = 180 \text{ Hz}$$

$$f_{LCD}(\max) = 600 \text{ Hz}$$

With  $f_{ACLK/VLOCLK} = 32768 \text{ Hz}$ ,  $LCDPREx = 011$ , and  $LCDDIVx = 10101$ :

$$f_{LCD} = 32768 \text{ Hz} / ((21+1) \times 2^3) = 32768 \text{ Hz} / 176 = 186 \text{ Hz}$$

With  $LCDPREx = 001$  and  $LCDDIVx = 11011$ :

$$f_{LCD} = 32768 \text{ Hz} / ((27+1) \times 2^1) = 32768 \text{ Hz} / 56 = 585 \text{ Hz}$$

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

### **32.2.3 Blanking the LCD**

The LCD controller allows to blank 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.

### **32.2.4 LCD Blinking**

The LCD\_B controller also supports blinking. The blinking mode  $LCDBLKMODx = 01$  allows to blink individual segments, with  $LCDBLKMODx = 10$  all segments are blinking, and with  $LCDBLKMODx = 00$  blinking is disabled.

#### **32.2.4.1 Blinking Memory**

To enable individual segments for blinking the corresponding bit in the blinking memory  $LCDBMx$  registers needs to be set. The memory uses the same structure as the LCD memory shown in [Figure 32-2](#). 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  $LCDBM1$ ,  $LCDBM3$ , etc.

Setting the bit  $LCDCLRB$  clears all blinking memory registers at the next frame boundary. It is automatically reset after the registers are cleared.

#### **32.2.4.2 Blinking Frequency**

The blinking frequency  $f_{BLINK}$  is selected with the  $LCDBLKPREx$  and  $LCDBLKDIVx$  bits. The same clock is used as selected for the LCD frequency  $f_{LCD}$ . The resulting  $f_{BLINK}$  frequency is calculated by:

$$f_{Blink} = \frac{f_{ACLKVLO}}{(LCDBLKDIVx + 1) \times 2^{9+LCDBLKPREx}}$$

The divider generating the blinking frequency  $f_{BLINK}$  is reset while  $LCDBLKMODx = 00$ . 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 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 be changed only when  $LCDBLKMODx = 00$ .

---

#### **32.2.4.3 Dual Display Memory**

The blinking memory can also be used as a secondary display memory when no blinking mode  $LCDBLKMODx = 01$  or  $10$  is selected. The memory to be displayed can be selected either manually using the  $LCDDISP$  bit or automatically with  $LCDBLKMODx = 11$ .

With LCDDISP = 0 the LCD memory is selected, with LCDDISP = 1 the blinking memory 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. 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.

### 32.2.5 LCD\_B Voltage And Bias Generation

The LCD\_B module allows selectable sources for the peak output waveform voltage, V1, as well as the fractional LCD biasing voltages V2 to V5.  $V_{LCD}$  may be sourced from  $V_{CC}$ , an internal charge pump, or externally.

All internal voltage generation is disabled if the selected clock source (ACLK or VLOCLK) is turned off (OSCOFF = 1) or the LCD\_B module is disabled (LCDON = 0).

#### 32.2.5.1 LCD Voltage Selection

$V_{LCD}$  is sourced from  $V_{CC}$  when VLCDEXT = 0, VLCDx = 0, and VREFx = 0.  $V_{LCD}$  is sourced from the internal charge pump when VLCDEXT = 0, VLCDCPEN = 1, and VLCDx > 0. The charge pump is always sourced from  $DV_{CC}$ . The VLCDx bits provide a software selectable LCD voltage from 2.6 V to 3.44 V (typical) independent of  $DV_{CC}$ . See the device-specific data sheet for specifications.

When the internal charge pump is used, a 4.7- $\mu$ F or larger capacitor must be connected between pin LCDCAP and ground. If no capacitor is connected and the charge pump is enabled, the LCDNOCAPIFG interrupt flag is set, and the charge pump is disabled to prevent damage to the device. The charge pump may be temporarily disabled by setting LCDCPEN = 0 with VLCDx > 0 to reduce system noise, or it can be automatically disabled during certain periods by setting the corresponding bits in the LCDBCPCTL 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 4.7- $\mu$ F or larger capacitor must be connected from pin LCDCAP to ground when the internal charge pump is enabled. If no capacitor is connected, the LCDNOCAPIFG interrupt flag is set and the charge pump is disabled.

The internal charge pump may use an external reference voltage when VLCDREFx = 01 (and LCDREXT = 0 and LCDEXTBIAS = 0). In this case, the charge pump voltage is set to a multiply of the external reference voltage according to the VLCDx bits setting.

When VLCDEXT = 1,  $V_{LCD}$  is sourced externally from the LCDAP, pin and the internal charge pump is disabled.

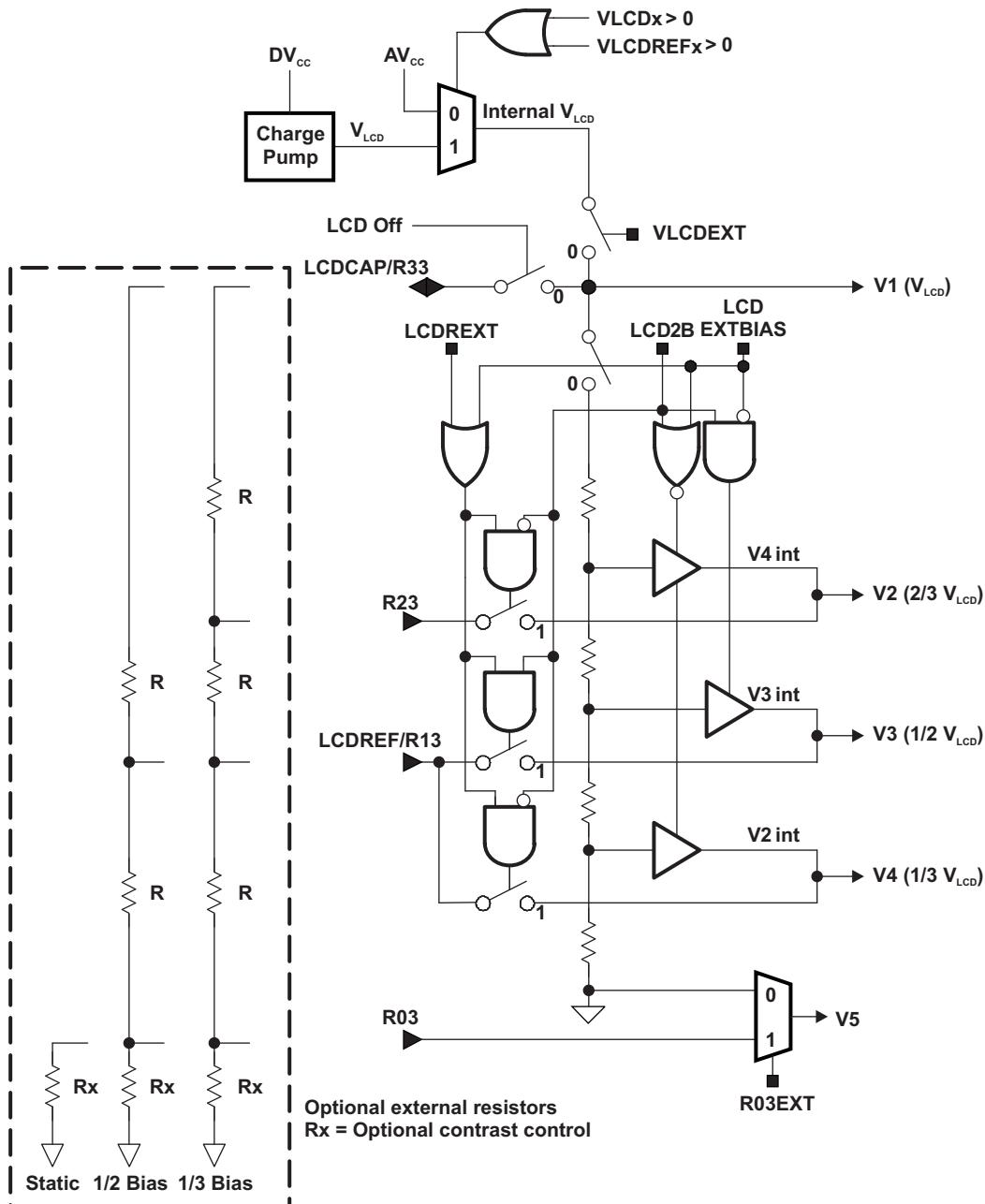
#### 32.2.5.2 LCD Bias Generation

The fractional LCD biasing voltages, V2 to V5 can be generated internally or externally, independent of the source for  $V_{LCD}$ . The LCD bias generation block diagram is shown in [Figure 32-3](#).

The internally generated bias voltages V2 to V4 are switched to external pins with LCDREXT = 1.

To source the bias voltages V2 to V4 externally, LCDEXTBIAS is set. This also disables the internal bias generation. Typically, an equally weighted resistor divider is used with resistors ranging from a few k $\Omega$  to 1 M $\Omega$ , depending on the size of the display. When using an external resistor divider, the  $V_{LCD}$  voltage may be sourced from the internal charge pump when VLCDEXT = 0 taking the maximum charge pump load current into account. V5 can also be sourced externally when R03EXT is set to control the contrast of the connected display by changing the voltage at the low end of the external resistor divider as shown in the left part of [Figure 32-3](#).

When using an external resistor divider R33 may serve as a switched  $V_{LCD}$  output when VLCDEXT = 0. This allows the power to the resistor ladder to be turned off, eliminating current consumption when the LCD is not used. When VLCDEXT = 1, R33 serves as a  $V_{LCD}$  input.


**Figure 32-3. Bias Generation**

The internal bias generator supports 1/2 bias LCDs when  $LCD2B = 1$ , and 1/3 bias LCDs when  $LCD2B = 0$  in 2-mux, 3-mux, and 4-mux modes. In static mode, the internal divider is disabled.

Some devices share the **LCDCAP**, **R33**, and **R23** functions. In this case, the charge pump cannot be used together with an external resistor divider with 1/3 biasing. When  $R03$  is not available externally,  $V5$  is always  $V_{ss}$ .

### 32.2.5.3 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 and the selected biasing scheme. Table 32-1 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 32-1. LCD Voltage and Biasing Characteristics

Mode	Bias Config	LCDMx	LCD2B	COM Lines	Voltage Levels	$V_{RMS,OFF}/V_{LCD}$	$V_{RMS,ON}/V_{LCD}$	Contrast Ratio $V_{RMS,ON}/V_{RMS,OFF}$
Static	Static	0	X	1	$V_1, V_5$	0	1	1/0
2-mux	1/2	1	1	2	$V_1, V_3, V_5$	0.354	0.791	2.236
2-mux	1/3	1	0	2	$V_1, V_2, V_4, V_5$	0.333	0.745	2.236
3-mux	1/2	10	1	3	$V_1, V_3, V_5$	0.408	0.707	1.732
3-mux	1/3	10	0	3	$V_1, V_2, V_4, V_5$	0.333	0.638	1.915
4-mux	1/2	11	1	4	$V_1, V_3, V_5$	0.433	0.661	1.528
4-mux	1/3	11	0	4	$V_1, V_2, V_4, V_5$	0.333	0.577	1.732

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 or equal than 3 times  $V_{th,10\%}$ .

In 3-mux and 4-mux mode typically a 1/3 biasing is used but a 1/2 biasing scheme is also possible. The 1/2 bias reduces the contrast ratio but the advantage is a reduction of the required full-scale LCD voltage  $V_{LCD}$ .

### 32.2.6 LCD Outputs

Some LCD segment, common, and Rxx functions are multiplexed with digital I/O functions. These pins can function either as digital I/O or as LCD functions.

The LCD segment functions, when multiplexed with digital I/O, are selected using the LCDSx bits in the LCDBPCTLx registers. The LCDSx bits select the LCD function for each segment line. When LCDSx = 0, a multiplexed pin is set to digital I/O function. When LCDSx = 1, a multiplexed pin is selected as LCD function.

The pin functions for COMx and Rxx, when multiplexed with digital I/O, are selected as described in the port schematic section of the device-specific datasheet. The COM1 to COM3 pins are shared with segment lines. If these pins are required as COM pins due to the selected LCD multiplexing mode the COM functionality takes precedence over the segment function that can be selected for those pins with the LCDSx bits as for all other segment pins.

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 Pins**

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

---

### 32.2.7 LCD\_B Interrupts

The LCD\_B module has four interrupt sources available, each with independent enables and flags.

The four interrupt flags, namely LCDFRMIFG, LCDLKOFFIFG, LCDLKONIFG, and LCDNOCAPIFG, are prioritized and combined to source a single interrupt vector. The interrupt vector register LCDBIV is used to determine which flag requested an interrupt.

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

Any read access of the LCDBIV 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 LCDBIV register automatically resets all pending interrupt flags. In addition, all flags can be cleared via software.

The LCDNOCAPIFG indicates that no capacitor is present at the LCDCAP pin when the charge pump is enabled. Setting the LCDNOCAPIE bit enables the interrupt.

The LCDBLKONIFG is set at the BLKCLK edge synchronized to the frame boundaries that turns on the segments when blinking is enabled with LCDBLKMODx = 01 or 10. It is also set at the BLKCLK edge synchronized to the frame boundaries 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 edge synchronized to the frame boundaries that blanks the segments when blinking is enabled with LCDBLKMODx = 01 or 10. It is also set at the BLKCLK edge synchronized to the frame boundaries 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.

### **32.2.7.1 LCDBIV Software Example**

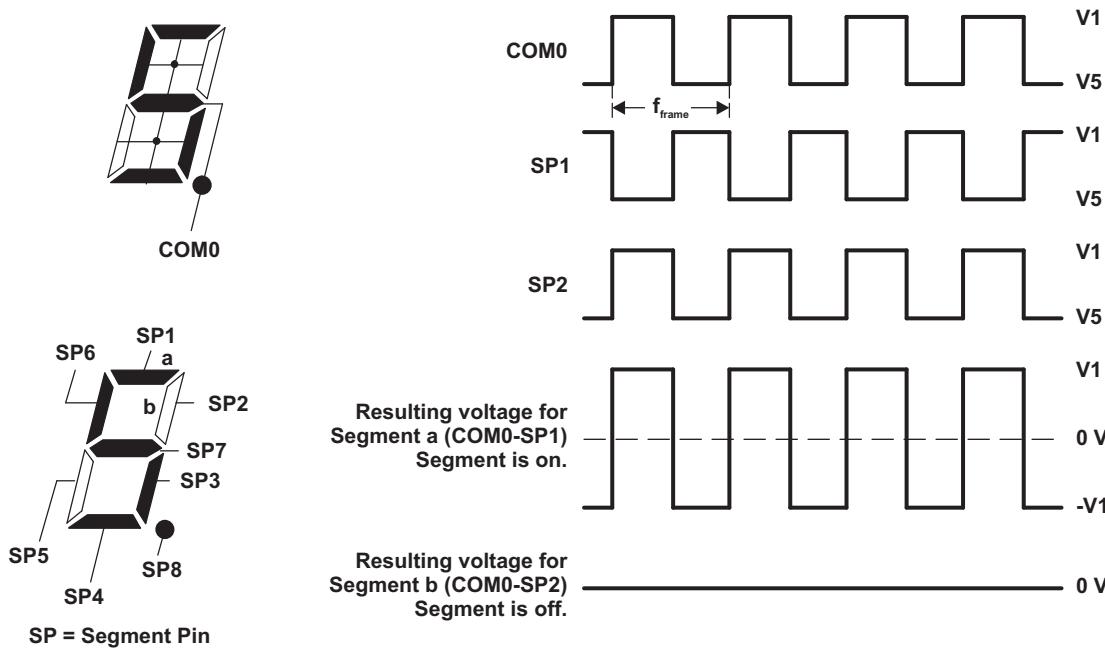
The following software example shows the recommended use of LCDBIV and the handling overhead. The LCDBIV value is added to the PC to automatically jump to the appropriate routine.

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_B interrupt flags.
LCDB_HND           ; Interrupt latency      6
    ADD &LCDBIV,PC ; Add offset to Jump table   3
    RETI             ; Vector 0: No interrupt     5
    JMP LCDNOCAP_HND ; Vector 2: LCDNOCAPIFG     2
    JMP LCDBLKON_HND ; Vector 4: LCDBLKONIFG     2
    JMP LCDBLKOFF_HND ; Vector 6: LCDBLKOFFIFG    2
    LCDFRM_HND       ; Vector 8: LCDFRMIFG      2
    ...              ; Task starts here
    RETI             ;                           5
LCDNOCAP_HND ; Vector 2: LCDNOCAPIFG
    ... ; 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
```

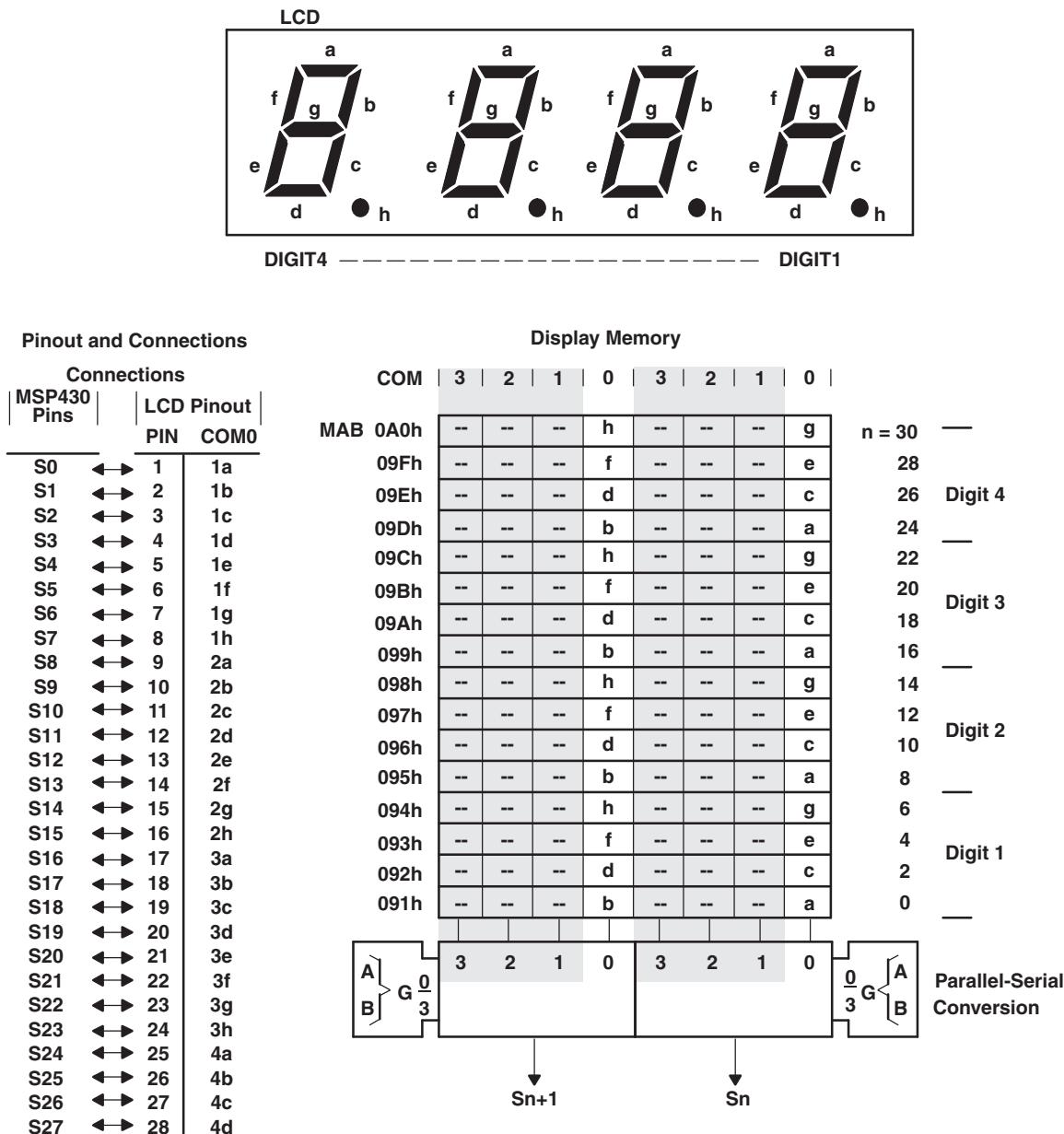
### 32.2.8 Static Mode

In static mode, each MSP430 segment pin drives one LCD segment and one common line, COM0, is used. [Figure 32-4](#) shows some example static waveforms.



**Figure 32-4. Example Static Waveforms**

Figure 32-5 shows an example static LCD, pinout, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user's application depends on the LCD pinout and on the MSP430-to-LCD connections.



**Figure 32-5. Static LCD Example (MAB addresses need to be replaced with LCDMx)**

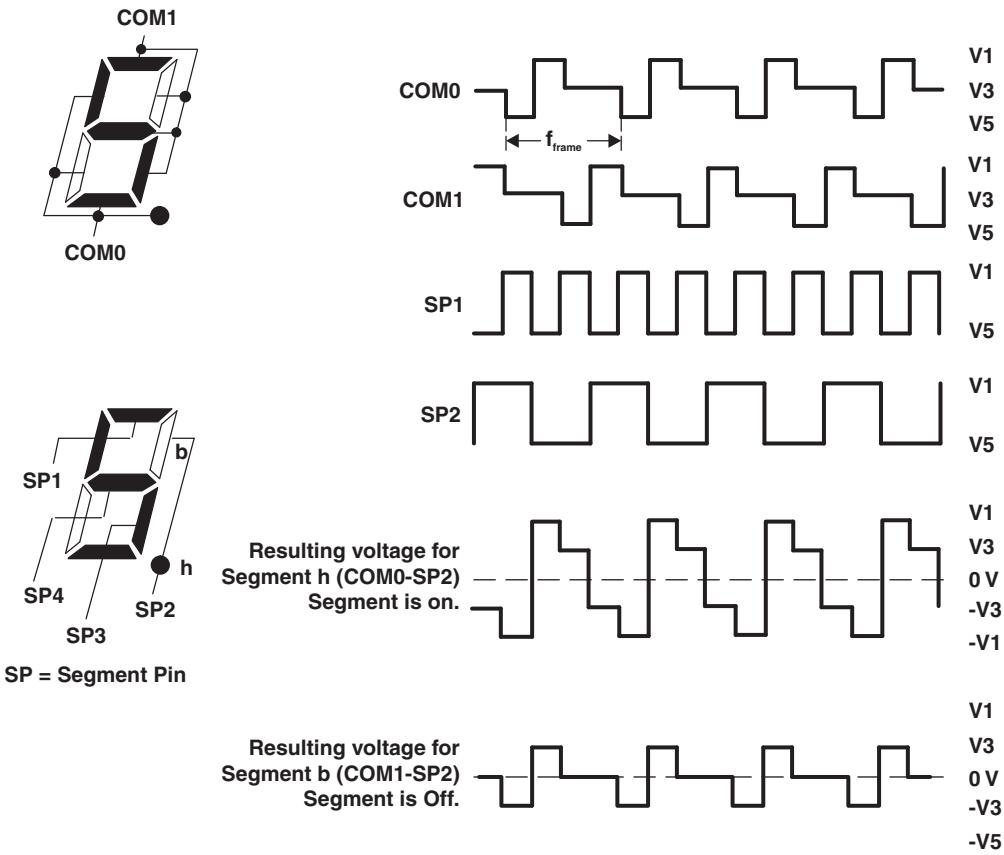
### 32.2.8.1 Static Mode Software Example

```

; All eight segments of a digit are often located in four
; display memory bytes with the static display method.
;
a EQU 001h
b EQU 010h
c EQU 002h
d EQU 020h
e EQU 004h
f EQU 040h
g EQU 008h
h EQU 080h
; The register content of Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
MOV.B Table (Rx),RY ; Load segment information
; into temporary memory.
; (Ry) = 0000 0000 hfdb geca
MOV.B Ry,&LCDn ; Note:
; All bits of an LCD memory
; byte are written
RRA Ry ; (Ry) = 0000 0000 0hfd bgec
MOV.B Ry,&LCDn+1 ; Note:
; All bits of an LCD memory
; byte are written
RRA Ry ; (Ry) = 0000 0000 00hf dbge
MOV.B Ry,&LCDn+2 ; Note:
; All bits of an LCD memory
; byte are written
RRA Ry ; (Ry) = 0000 0000 000h fdbg
MOV.B Ry,&LCDn+3 ; Note:
; All bits of an LCD memory
; byte are written
..... ; Table
DB a+b+c+d+e+f ; displays "0"
DB b+c; ; displays "1"
..... .
DB .....
```

### 32.2.9 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 32-6](#) shows some example 2-mux, 1/2 bias waveforms.



**Figure 32-6. Example 2-Mux Waveforms**

Figure 32-7 shows an example 2-mux LCD, pinout, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user's application completely depends on the LCD pinout and on the MSP430-to-LCD connections.

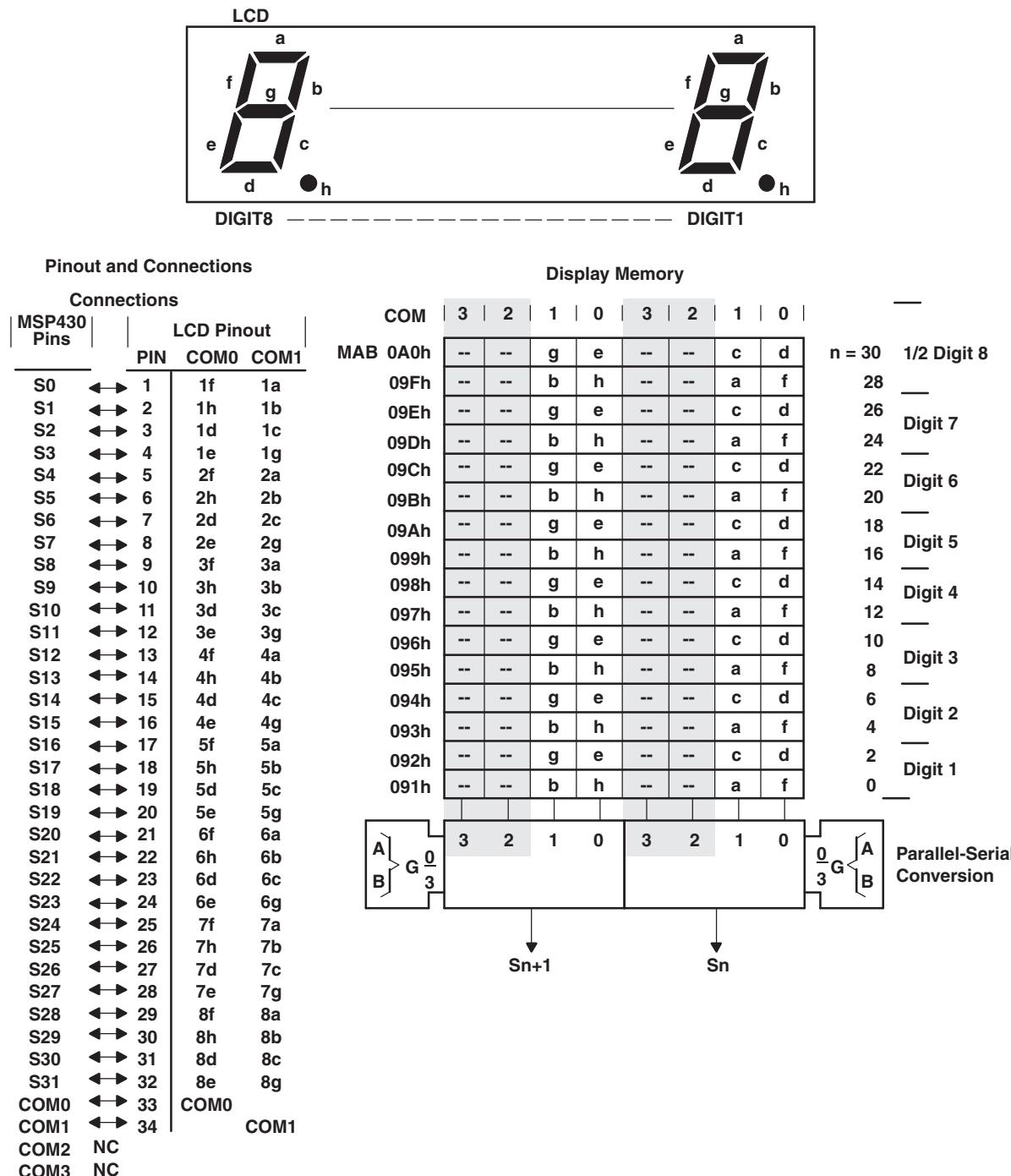


Figure 32-7. 2-Mux LCD Example (MAB addresses need to be replaced with LCDMx)

### **32.2.9.1 2-Mux Mode Software Example**

```

; All eight segments of a digit are often located in two
; display memory bytes with the 2-mux display rate ;
a EQU 002h
b EQU 020h
c EQU 008h
d EQU 004h
e EQU 040h
f EQU 001h
g EQU 080h
h EQU 010h
; The register content of Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx. ;
..... .
MOV.B Table(Rx),Ry ; Load segment information into
; temporary memory.
MOV.B Ry,&LCDn ; (Ry) = 0000 0000 gebh cda
; Note:
; All bits of an LCD memory byte
; are written
RRA Ry ; (Ry) = 0000 0000 0geb hcda
RRA Ry ; (Ry) = 0000 0000 00ge bhcd
MOV.B Ry,&LCDn+1 ; Note:
; All bits of an LCD memory byte
; are written
..... .
Table
DB a+b+c+d+e+f ; displays "0"
..... .
DB a+b+c+d+e+f+g ; displays "8"
..... .
DB ..... ;
```

### 32.2.10 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 32-8 shows some example 3-mux, 1/3 bias waveforms.

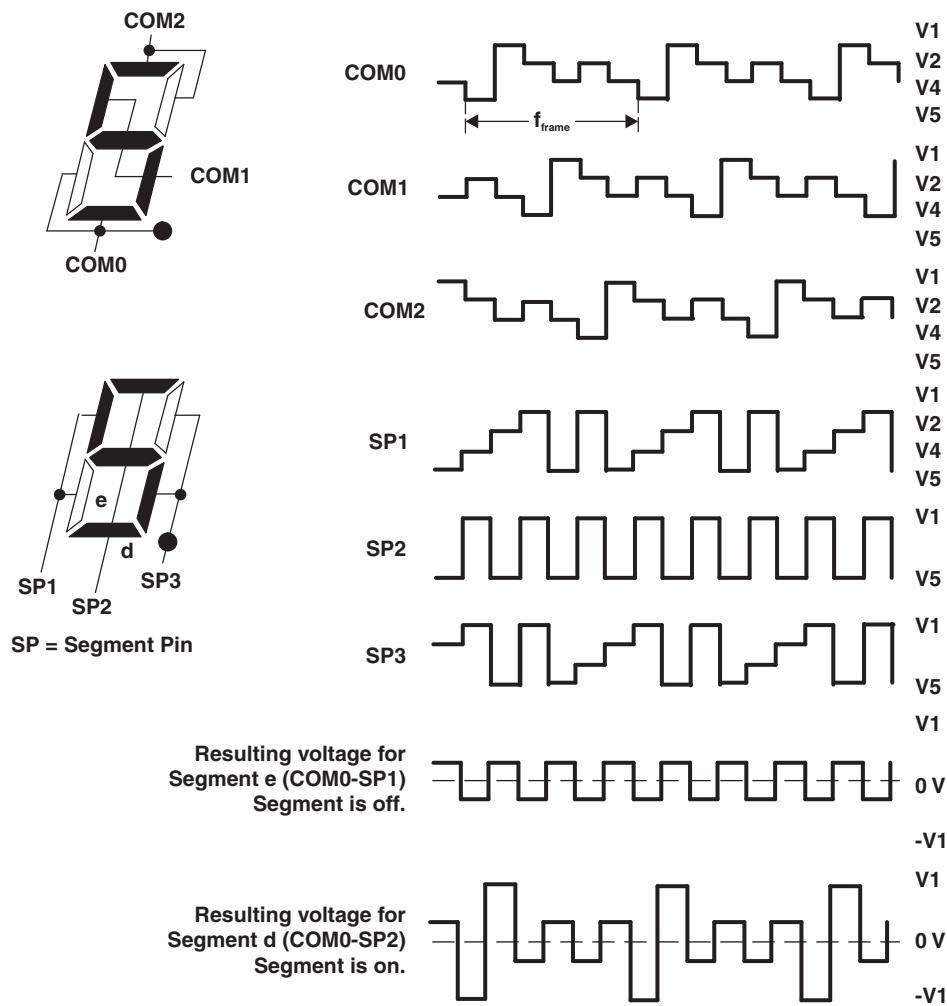
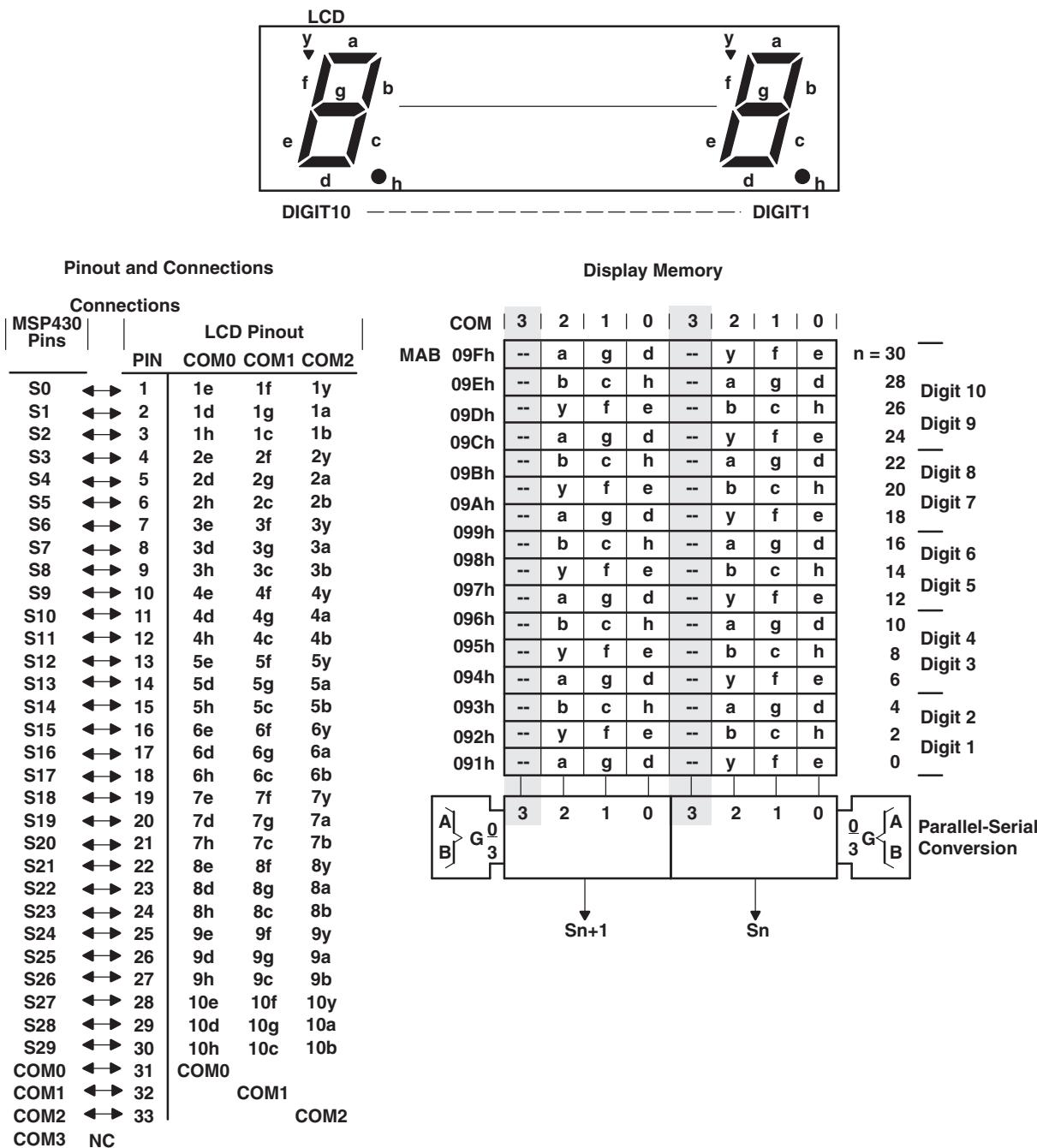


Figure 32-8. Example 3-Mux Waveforms

**Figure 32-9** shows an example 3-mux LCD, pinout, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user's application depends on the LCD pinout and on the MSP430-to-LCD connections.



**Figure 32-9. 3-Mux LCD Example (MAB addresses need to be replaced with LCDMx)**

### 32.2.10.1 3-Mux Mode Software Example

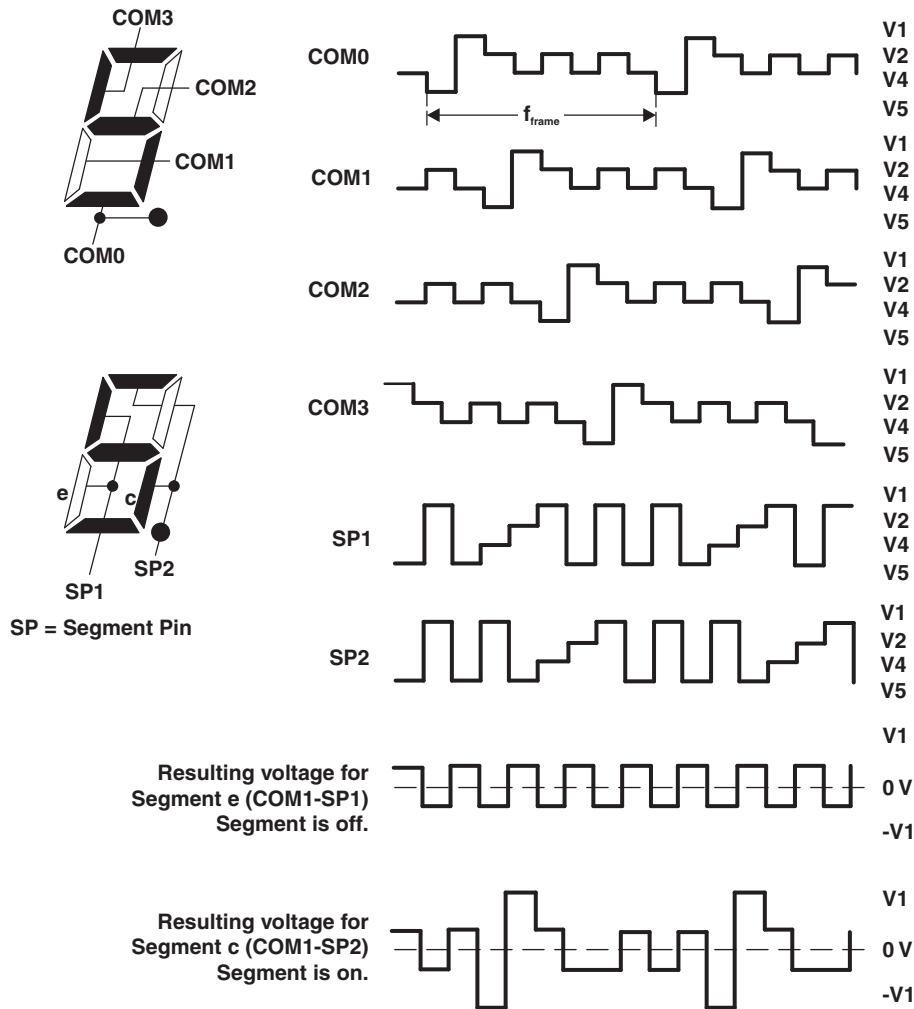
```

; The 3-mux rate can support nine segments for each
; digit. The nine segments of a digit are located in
; 1 1/2 display memory bytes.
;
a EQU 0040h
b EQU 0400h
c EQU 0200h
d EQU 0010h
e EQU 0001h
f EQU 0002h
g EQU 0020h
h EQU 0100h
Y EQU 0004h
; The LSDigit of register Rx should be displayed.
; The Table represents the 'on'-segments according to the
; LSDigit of register of Rx.
; The register Ry is used for temporary memory
;
ODDDIG
    RLA Rx ; LCD in 3-mux has 9 segments per
            ; digit
            ; word table required for
            ; displayed characters.
    MOV Table(Rx),Ry ; Load segment information to
                    ; temporary mem.
                    ; (Ry) = 0000 0bch 0agd 0yfe
    MOV.B Ry,&LCDn ; write 'a, g, d, y, f, e' of
                    ; Digit n (LowByte)
    SWPB Ry ; (Ry) = 0agd 0yfe 0000 0bch
    BIC.B #07h,&LCDn+1 ; write 'b, c, h' of Digit n
                    ; (HighByte)
    BIS.B Ry,&LCDn+1
    .....
EVNDIG
    RLA Rx ; LCD in 3-mux has 9 segments per
            ; digit
            ; word table required for
            ; displayed characters.
    MOV Table(Rx),Ry ; Load segment information to
                    ; temporary mem.
                    ; (Ry) = 0000 0bch 0agd 0yfe
    RLA Ry ; (Ry) = 0000 bch0 agd0 yfe0
    RLA Ry ; (Ry) = 000b ch0a gd0y fe00
    RLA Ry ; (Ry) = 00bc h0ag d0yf e000
    RLA Ry ; (Ry) = 0bch 0agd 0yfe 0000
    BIC.B #070h,&LCDn+1
    BIS.B Ry,&LCDn+1 ; write 'y, f, e' of Digit n+1
                    ; (LowByte)
    SWPB Ry ; (Ry) = 0yfe 0000 0bch 0agd
    MOV.B Ry,&LCDn+2 ; write 'b, c, h, a, g, d' of
                    ; Digit n+1 (HighByte)
    .....
Table
    DW a+b+c+d+e+f ; displays "0"
    DW b+c           ; displays "1"
    .....
    DW a+e+f+g ; displays "F"

```

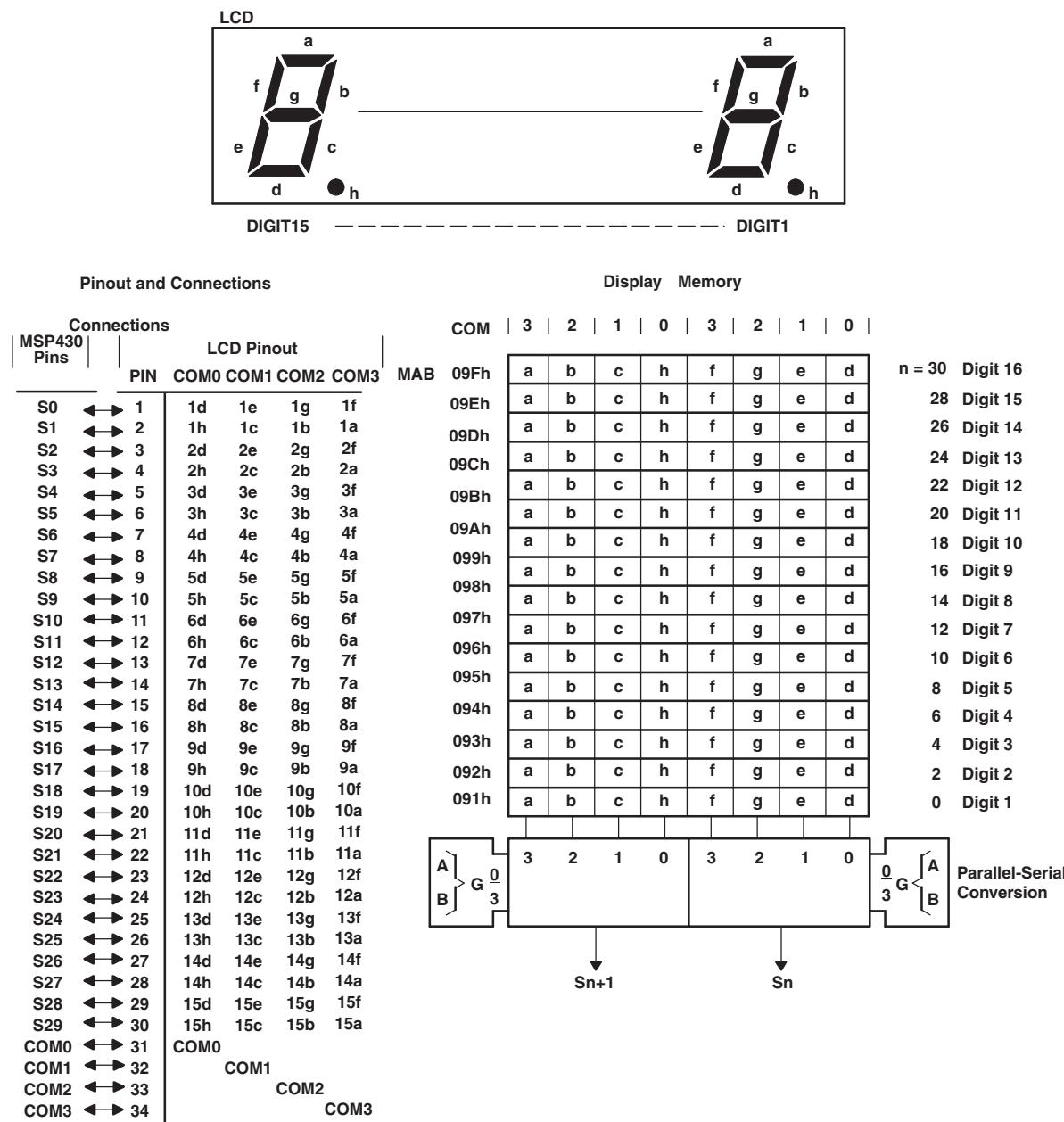
### 32.2.11 4-Mux Mode

In 4-mux mode, each MSP430 segment pin drives four LCD segments and all four common lines (COM0, COM1, COM2, and COM3) are used. [Figure 32-10](#) shows some example 4-mux, 1/3 bias waveforms.



**Figure 32-10. Example 4-Mux Waveforms**

**Figure 32-11** shows an example 4-mux LCD, pinout, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user's application depends on the LCD pinout and on the MSP430-to-LCD connections.



**Figure 32-11. 4-Mux LCD Example (MAB addresses need to be replaced with LCDMx)**

### **32.2.11.1 4-Mux Mode Software Example**

```

; The 4-mux rate supports eight segments for each digit.
; All eight segments of a digit can often be located in
; one display memory byte
a EQU 080h
b EQU 040h
c EQU 020h
d EQU 001h
e EQU 002h
f EQU 008h
g EQU 004h
h EQU 010h
;
; The LSDigit of register Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
;
MOV.B Table(Rx),&LCDn ; n = 1 ..... 15
; all eight segments are
; written to the display
; memory
..... .
Table
DB a+b+c+d+e+f ; displays "0"
DB b+c ; displays "1"
..... .
DB b+c+d+e+g ; displays "d"
DB a+d+e+f+g ; displays "E"
DB a+e+f+g ; displays "F"

```

### 32.3 LCD\_B Registers

The LCD\_B registers are listed in [Table 32-2](#) to [Table 32-4](#). The LCD memory and blinking memory registers can also be accessed as word.

**Table 32-2. LCD\_B Registers**

Offset	Acronym	Register Name	Type	Reset	Section
000h	LCDBCTL0	LCD_B control register 0	Read/write	0000h	<a href="#">Section 32.3.1</a>
002h	LCDBCTL1	LCD_B control register 1	Read/write	0000h	<a href="#">Section 32.3.2</a>
004h	LCDBBLKCTL	LCD_B blinking control register	Read/write	0000h	<a href="#">Section 32.3.3</a>
006h	LCDBMEMCTL	LCD_B memory control register	Read/write	0000h	<a href="#">Section 32.3.4</a>
008h	LCDBVCTL	LCD_B voltage control register	Read/write	0000h	<a href="#">Section 32.3.5</a>
00Ah	LCDBPCTL0	LCD_B port control 0	Read/write	0000h	<a href="#">Section 32.3.6</a>
00Ch	LCDBPCTL1	LCD_B port control 1	Read/write	0000h	<a href="#">Section 32.3.7</a>
00Eh	LCDBPCTL2	LCD_B port control 2 ( $\geq 128$ segments)	Read/write	0000h	<a href="#">Section 32.3.8</a>
010h	LCDBPCTL3	LCD_B port control 3 (192 segments)	Read/write	0000h	<a href="#">Section 32.3.9</a>
012h	LCDBCPCTL	LCD_B charge pump control	Read/write	0000h	<a href="#">Section 32.3.10</a>
014h		Reserved			
016h		Reserved			
018h		Reserved			
01Ah		Reserved			
01Ch		Reserved			
01Eh	LCDBIV	LCD_B interrupt vector	Read/write	0000h	<a href="#">Section 32.3.11</a>

**Table 32-3. LCD\_B Memory Registers<sup>(1)</sup>**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Reset</b>
020h	LCDM1	LCD memory 1 (S1/S0)	Read/write	Unchanged
021h	LCDM2	LCD memory 2 (S3/S2)	Read/write	Unchanged
022h	LCDM3	LCD memory 3 (S5/S4)	Read/write	Unchanged
023h	LCDM4	LCD memory 4 (S7/S6)	Read/write	Unchanged
024h	LCDM5	LCD memory 5 (S9/S8)	Read/write	Unchanged
025h	LCDM6	LCD memory 6 (S11/S10)	Read/write	Unchanged
026h	LCDM7	LCD memory 7 (S13/S12)	Read/write	Unchanged
027h	LCDM8	LCD memory 8 (S15/S14)	Read/write	Unchanged
028h	LCDM9	LCD memory 9 (S17/S16)	Read/write	Unchanged
029h	LCDM10	LCD memory 10 (S19/S18)	Read/write	Unchanged
02Ah	LCDM11	LCD memory 11 (S21/S20)	Read/write	Unchanged
02Bh	LCDM12	LCD memory 12 (S23/S22)	Read/write	Unchanged
02Ch	LCDM13	LCD memory 13 (S25/S24)	Read/write	Unchanged
02Dh	LCDM14	LCD memory 14 (S27/S26)	Read/write	Unchanged
02Eh	LCDM15	LCD memory 15 (S29/S28, ≥128 segments)	Read/write	Unchanged
02Fh	LCDM16	LCD memory 16 (S31/S30, ≥128 segments)	Read/write	Unchanged
030h	LCDM17	LCD memory 17 (S33/S32, ≥128 segments)	Read/write	Unchanged
031h	LCDM18	LCD memory 18 (S35/S34, ≥128 segments)	Read/write	Unchanged
032h	LCDM19	LCD memory 19 (S37/S36, ≥160 segments)	Read/write	Unchanged
033h	LCDM20	LCD memory 20 (S39/S38, ≥160 segments)	Read/write	Unchanged
034h	LCDM21	LCD memory 21 (S41/S40, ≥160 segments)	Read/write	Unchanged
035h	LCDM22	LCD memory 22 (S43/S42, ≥160 segments)	Read/write	Unchanged
036h	LCDM23	LCD memory 23 (S45/S44, 192 segments)	Read/write	Unchanged
037h	LCDM24	LCD memory 24 (S47/S46, 192 segments)	Read/write	Unchanged
038h	LCDM25	LCD memory 25 (S49/S48, 192 segments)	Read/write	Unchanged
039h	LCDM26	LCD memory 26 (S50, 192 segments)	Read/write	Unchanged
03Ah		Reserved	Read/write	Unchanged
03Bh		Reserved	Read/write	Unchanged
03Ch		Reserved	Read/write	Unchanged
03Dh		Reserved	Read/write	Unchanged
03Eh		Reserved	Read/write	Unchanged
03Fh		Reserved	Read/write	Unchanged

<sup>(1)</sup> The LCD memory registers can also be accessed as word.

**Table 32-4. LCD\_B Blinking Memory Registers<sup>(1)</sup>**

Offset	Acronym	Register Name	Type	Reset
040h	LCDBM1	LCD blinking memory 1	Read/write	Unchanged
041h	LCDBM2	LCD blinking memory 2	Read/write	Unchanged
042h	LCDBM3	LCD blinking memory 3	Read/write	Unchanged
043h	LCDBM4	LCD blinking memory 4	Read/write	Unchanged
044h	LCDBM5	LCD blinking memory 5	Read/write	Unchanged
045h	LCDBM6	LCD blinking memory 6	Read/write	Unchanged
046h	LCDBM7	LCD blinking memory 7	Read/write	Unchanged
047h	LCDBM8	LCD blinking memory 8	Read/write	Unchanged
048h	LCDBM9	LCD blinking memory 9	Read/write	Unchanged
049h	LCDBM10	LCD blinking memory 10	Read/write	Unchanged
04Ah	LCDBM11	LCD blinking memory 11	Read/write	Unchanged
04Bh	LCDBM12	LCD blinking memory 12	Read/write	Unchanged
04Ch	LCDBM13	LCD blinking memory 13	Read/write	Unchanged
04Dh	LCDBM14	LCD blinking memory 14	Read/write	Unchanged
04Eh	LCDBM15	LCD blinking memory 15 ( $\geq 128$ segments)	Read/write	Unchanged
04Fh	LCDBM16	LCD blinking memory 16 ( $\geq 128$ segments)	Read/write	Unchanged
050h	LCDBM17	LCD blinking memory 17 ( $\geq 128$ segments)	Read/write	Unchanged
051h	LCDBM18	LCD blinking memory 18 ( $\geq 128$ segments)	Read/write	Unchanged
052h	LCDBM19	LCD blinking memory 19 ( $\geq 160$ segments)	Read/write	Unchanged
053h	LCDBM20	LCD blinking memory 20 ( $\geq 160$ segments)	Read/write	Unchanged
054h	LCDBM21	LCD blinking memory 21 ( $\geq 160$ segments)	Read/write	Unchanged
055h	LCDBM22	LCD blinking memory 22 ( $\geq 160$ segments)	Read/write	Unchanged
056h	LCDBM23	LCD blinking memory 23 (190 segments)	Read/write	Unchanged
057h	LCDBM24	LCD blinking memory 24 (190 segments)	Read/write	Unchanged
058h	LCDBM25	LCD blinking memory 25 (190 segments)	Read/write	Unchanged
059h	LCDBM26	LCD blinking memory 26 (190 segments)	Read/write	Unchanged
05Ah		Reserved	Read/write	Unchanged
05Bh		Reserved	Read/write	Unchanged
05Ch		Reserved	Read/write	Unchanged
05Dh		Reserved	Read/write	Unchanged
05Eh		Reserved	Read/write	Unchanged
05Fh		Reserved	Read/write	Unchanged

<sup>(1)</sup> The LCD blinking memory registers can also be accessed as word.

### 32.3.1 LCDBCTL0 Register

LCD\_B Control Register 0

**Figure 32-12. LCDBCTL0 Register**

15	14	13	12	11	10	9	8
LCDDIVx						LCDPREx	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
LCDSEL	Reserved		LCDMXx	LCDSON	Reserved	LCDON	
rw-0	r0	r0	rw-0	rw-0	rw-0	r0	rw-0

**Table 32-5. LCDBCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-11	LCDDIVx	RW	0h	LCD frequency divider. Together with LCDPREx the LCD frequency f(LCD) is calculated as $f(LCD) = f(ACLK/VLO) / [(LCDDIVx + 1) \times 2^{(LCDPREx)}]$ . Settings for this bit should be changed only while LCDON = 0. 00000b = Divide by 1 00001b = Divide by 2 ⋮ 11110b = Divide by 31 11111b = Divide by 32
10-8	LCDPREx	RW	0h	LCD frequency pre-scaler. Together with LCDDIVx the LCD frequency f(LCD) is calculated as $f(LCD) = f(ACLK/VLO) / [(LCDDIVx + 1) \times 2^{(LCDPREx)}]$ . Settings for this bit should be changed only while LCDON = 0. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 4 011b = Divide by 8 100b = Divide by 16 101b = Divide by 32 110b = Reserved - Defaults to divide by 32 111b = Reserved - Defaults to divide by 32
7	LCDSEL	RW	0h	Clock source select for LCD and blinking frequency Settings for this bit should be changed only while LCDON = 0. 0b = ACLK (30 kHz to 40 kHz) 1b = VLOCLK
6-5	Reserved	R	0h	Reserved. Always reads as 0.
4-3	LCDMXx	RW	0h	LCD mux rate. These bits select the LCD mode. Settings for this bit should be changed only while LCDON = 0. 00b = Static 01b = 2-mux 10b = 3-mux 11b = 4-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	Reserved	R	0h	Reserved. Always reads as 0.
0	LCDON	RW	0h	LCD on. This bit turns the LCD_B module on or off. 0b = LCD_B module off 1b = LCD_B module on

**32.3.2 LCDBCTL1 Register**

LCD\_B Control Register 1

**Figure 32-13. LCDBCTL1 Register**

15	14	13	12	11	10	9	8
		Reserved		LCDNOCAPIE	LCDBLKONIE	LCDBLKOFFIE	LCDFRMIE
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
		Reserved		LCDNOCAPIFG	LCDBLKONIFG	LCDBLKOFFIFG	LCDFRMIFG
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

**Table 32-6. LCDBCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11	LCDNOCAPIE	RW	0h	No capacitance connected interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
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
7-4	Reserved	R	0h	Reserved. Always reads as 0.
3	LCDNOCAPIFG	RW	0h	No capacitance connected interrupt flag. Set when charge pump is enabled but no capacitance is connected to LCDCAP pin. 0b = No interrupt pending 1b = Interrupt pending
2	LCDBLKONIFG	RW	0h	LCD blinking interrupt flag, segments switched on. 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, segments switched off. 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

### 32.3.3 LCDBBLKCTL Register

LCD\_B Blink Control Register

**Figure 32-14. LCDBBLKCTL 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
LCDBLKDIVx			LCDBLKPReX			LCDBLKMODx	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 32-7. LCDBBLKCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7-5	LCDBLKDIVx	RW	0h	<p>Clock divider for blinking frequency. Together with LCDBLKPReX, the blinking frequency <math>f(\text{BLINK})</math> is calculated as <math>f(\text{BLINK}) = f(\text{ACLK/VLO}) / [(LCDBLKDIVx + 1) \times 2^{(9+LCDBLKPReX)}]</math>.</p> <p>Settings for this bit should be changed only while LCDBLKMODx = 00.</p> <ul style="list-style-type: none"> <li>000b = Divide by 1</li> <li>001b = Divide by 2</li> <li>010b = Divide by 3</li> <li>011b = Divide by 4</li> <li>100b = Divide by 5</li> <li>101b = Divide by 6</li> <li>110b = Divide by 7</li> <li>111b = Divide by 8</li> </ul>
4-2	LCDBLKPReX	RW	0h	<p>Clock pre-scaler for blinking frequency. Together with LCDBLKDIVx, the blinking frequency <math>f(\text{BLINK})</math> is calculated as <math>f(\text{BLINK}) = f(\text{ACLK/VLO}) / ((LCDBLKDIVx + 1) \times 2^{(9+LCDBLKPReX)})</math>.</p> <p>Settings for this bit should be changed only while LCDBLKMODx = 00.</p> <ul style="list-style-type: none"> <li>000b = Divide by 512</li> <li>001b = Divide by 1024</li> <li>010b = Divide by 2048</li> <li>011b = Divide by 4096</li> <li>100b = Divide by 8162</li> <li>101b = Divide by 16384</li> <li>110b = Divide by 32768</li> <li>111b = Divide by 65536</li> </ul>
1-0	LCDBLKMODx	RW	0h	<p>Blinking mode</p> <ul style="list-style-type: none"> <li>00b = Blinking disabled</li> <li>01b = Blinking of individual segments as enabled in blinking memory register LCDBMx</li> <li>10b = Blinking of all segments</li> <li>11b = Switching between display contents as stored in LCDMx and LCDBMx memory registers.</li> </ul>

**32.3.4 LCDBMEMCTL Register**

LCD\_B Memory Control Register

**Figure 32-15. LCDBMEMCTL 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					LCDCLRB <sub>M</sub>	LCDCLRM	LCDDISP
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

**Table 32-8. LCDBMEMCTL Register Description**

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Always reads as 0.
2	LCDCLRB <sub>M</sub>	RW	0h	<p>Clear LCD blinking memory</p> <p>Clears all blinking memory registers LCDBM<sub>x</sub>. The bit is automatically reset when the blinking memory is cleared.</p> <p>0b = Contents of blinking memory registers LCDBM<sub>x</sub> remain unchanged 1b = Clear content of all blinking memory registers LCDBM<sub>x</sub></p>
1	LCDCLRM	RW	0h	<p>Clear LCD memory</p> <p>Clears all LCD memory registers LCDM<sub>x</sub>. The bit is automatically reset when the LCD memory is cleared.</p> <p>0b = Contents of LCD memory registers LCDM<sub>x</sub> remain unchanged 1b = Clear content of all LCD memory registers LCDM<sub>x</sub></p>
0	LCDDISP	RW	0h	<p>Select LCD memory registers for display</p> <p>The bit is cleared in LCDBLKMOD<sub>x</sub> = 01 and LCDBLKMOD<sub>x</sub> = 10 and cannot be changed by software.</p> <p>When LCDBLKMOD<sub>x</sub> = 11, this bit reflects the currently displayed memory but cannot be changed by software. When returning to LCDBLKMOD<sub>x</sub> = 00 the bit is cleared.</p> <p>0b = Display content of LCD memory registers LCDM<sub>x</sub> 1b = Display content of LCD blinking memory registers LCDBM<sub>x</sub></p>

### 32.3.5 LCDBVCTL Register

LCD\_B Voltage Control Register

**Figure 32-16. LCDBVCTL Register**

15	14	13	12	11	10	9	8
Reserved		VLCDx					
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	r0
7	6	5	4	3	2	1	0
LCDREXT	R03EXT	LCDEXTBIAS	VLCDEXT	LCDCPEN	VLCDREFx	LCD2B	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 32-9. LCDBVCTL Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-9	VLCDx	RW	0h	<p>Charge pump voltage select. LCDCPEN must be 1 for the charge pump to be enabled. V(CC) is used for V(LCD) when VLCDx = 0000 and VLCDREFx = 00 and VLCDEXT = 0.</p> <p>If VLCDREFx = 00 or 10:</p> <ul style="list-style-type: none"> <li>0000b = Charge pump disabled</li> <li>0001b = V(LCD) = 2.60 V</li> <li>0010b = V(LCD) = 2.66 V</li> <li>0011b = V(LCD) = 2.72 V</li> <li>0100b = V(LCD) = 2.78 V</li> <li>0101b = V(LCD) = 2.84 V</li> <li>0110b = V(LCD) = 2.90 V</li> <li>0111b = V(LCD) = 2.96 V</li> <li>1000b = V(LCD) = 3.02 V</li> <li>1001b = V(LCD) = 3.08 V</li> <li>1010b = V(LCD) = 3.14 V</li> <li>1011b = V(LCD) = 3.20 V</li> <li>1100b = V(LCD) = 3.26 V</li> <li>1101b = V(LCD) = 3.32 V</li> <li>1110b = V(LCD) = 3.38 V</li> <li>1111b = V(LCD) = 3.44 V</li> </ul> <p>If VLCDREFx = 01 or 11:</p> <ul style="list-style-type: none"> <li>0000b = Charge pump disabled</li> <li>0001b = V(LCD) = <math>2.17 \times V(REF)</math></li> <li>0010b = V(LCD) = <math>2.22 \times V(REF)</math></li> <li>0011b = V(LCD) = <math>2.27 \times V(REF)</math></li> <li>0100b = V(LCD) = <math>2.32 \times V(REF)</math></li> <li>0101b = V(LCD) = <math>2.37 \times V(REF)</math></li> <li>0110b = V(LCD) = <math>2.42 \times V(REF)</math></li> <li>0111b = V(LCD) = <math>2.47 \times V(REF)</math></li> <li>1000b = V(LCD) = <math>2.52 \times V(REF)</math></li> <li>1001b = V(LCD) = <math>2.57 \times V(REF)</math></li> <li>1010b = V(LCD) = <math>2.62 \times V(REF)</math></li> <li>1011b = V(LCD) = <math>2.67 \times V(REF)</math></li> <li>1100b = V(LCD) = <math>2.72 \times V(REF)</math></li> <li>1101b = V(LCD) = <math>2.77 \times V(REF)</math></li> <li>1110b = V(LCD) = <math>2.82 \times V(REF)</math></li> <li>1111b = V(LCD) = <math>2.87 \times V(REF)</math></li> </ul>
8	Reserved	R	0h	Reserved. Always reads as 0.

**Table 32-9. LCDBVCTL Register Description (continued)**

Bit	Field	Type	Reset	Description
7	LCDREXT	RW	0h	V2 to V4 voltage on external Rx3 pins. This bit selects the external connections for voltages V2 to V4 with internal bias generation (LCDEXTBIAS = 0). The bit is don't care if external biasing is selected (LCDEXTBIAS = 1). Settings for this bit should be changed only while LCDON = 0. 0b = Internally generated V2 to V4 are not switched to pins (LCDEXTBIAS = 0). 1b = Internally generated V2 to V4 are switched to pins (LCDEXTBIAS = 0).
6	R03EXT	RW	0h	V5 voltage select. This bit selects the external connection for the lowest LCD voltage. R03EXT is ignored if there is no R03 pin available. Settings for this bit should be changed only while LCDON = 0. 0b = V5 is V(SS) 1b = V5 is sourced from the R03 pin
5	LCDEXTBIAS	RW	0h	V2 to V4 voltage select. This bit selects the generation for voltages V2 to V4. Settings for this bit should be changed only while LCDON = 0. 0b = V2 to V4 are generated internally. 1b = V2 to V4 are sourced externally and the internal bias generator is switched off.
4	VLCDEXT	RW	0h	V(LCD) source select Settings for this bit should be changed only while LCDON = 0. 0b = V(LCD) is generated internally. 1b = V(LCD) is sourced externally.
3	LDCDPEN	RW	0h	Charge pump enable 0b = Charge pump disabled 1b = Charge pump enabled when V(LCD) is generated internally (VLCDEXT = 0) and VLCDx > 0 or VLCDREFx > 0.
2-1	VLCDREFx	RW	0h	Charge pump reference select If LCDEXTBIAS = 1 or LCDREXT = 1 settings 01, 10 and 11 are not supported. Internal reference voltage used instead. Settings for this bit should be changed only while LCDON = 0. 00b = Internal reference voltage 01b = External reference voltage 10b = Internal reference voltage switched to external pin LCDREF/R13. 11b = Reserved. Defaults to external reference voltage.
0	LCD2B	RW	0h	Bias select. LCD2B is ignored when LCDMx = 00. 0b = 1/3 bias 1b = 1/2 bias

### 32.3.6 LCDBPCTL0 Register

LCD\_B Port Control Register 0

**Figure 32-17. LCDBPCTL0 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 32-10. LCDBPCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDSx	RW	0h	<p>LCD segment line x enable</p> <p>This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0b = Multiplexed pins are port functions.</p> <p>1b = Pins are LCD functions.</p>

### 32.3.7 LCDBPCTL1 Register

LCD\_B Port Control Register 1

**Figure 32-18. LCDBPCTL1 Register**

15	14	13	12	11	10	9	8
LCDS31	LCDS30	LCDS29	LCDS28	LCDS27	LCDS26	LCDS25	LCDS24
rw-0							
7	6	5	4	3	2	1	0
LCDS23	LCDS22	LCDS21	LCDS20	LCDS19	LCDS18	LCDS17	LCDS16
rw-0							

**Table 32-11. LCDBPCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDSx	RW	0h	<p>LCD segment line x enable</p> <p>LCDS27 to LCDS31 are reserved on devices supporting a maximum of 96 segments.</p> <p>This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0b = Multiplexed pins are port functions.</p> <p>1b = Pins are LCD functions.</p>

**32.3.8 LCDBPCTL2 Register**LCD\_B Port Control Register 2 ( $\geq$  128 Segments)**Figure 32-19. LCDBPCTL2 Register**

15	14	13	12	11	10	9	8
LCDS47	LCDS46	LCDS45	LCDS44	LCDS43	LCDS42	LCDS41	LCDS40
rw-0							
7	6	5	4	3	2	1	0
LCDS39	LCDS38	LCDS37	LCDS36	LCDS35	LCDS34	LCDS33	LCDS32
rw-0							

**Table 32-12. LCDBPCTL2 Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDSx	RW	0h	LCD segment line x enable LCDS35 to LCDS47 are reserved on devices supporting a maximum of 128 segments. LCDS43 to LCDS47 are reserved on devices supporting a maximum of 160 segments. 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.

**32.3.9 LCDBPCTL3 Register**

LCD\_B Port Control Register 2 (192 Segments)

**Figure 32-20. LCDBPCTL3 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					LCDS50	LCDS49	LCDS48
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

**Table 32-13. LCDBPCTL3 Register Description**

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	LCDSx	RW	0h	LCD segment line x enable This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. NOTE: Settings for LCDSx should be changed only while LCDON = 0. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

### 32.3.10 LCDBCPCTL Register

LCD\_B Charge Pump Control Register

**Figure 32-21. LCDBCPCTL Register**

15	14	13	12	11	10	9	8
Reserved							
r0							
7	6	5	4	3	2	1	0
LCDCPDIS7	LCDCPDIS6	LCDCPDIS5	LCDCPDIS4	LCDCPDIS3	LCDCPDIS2	LCDCPDIS1	LCDCPDIS0
rw-0							

**Table 32-14. LCDBCPCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	LCDCPDIS7	RW	0h	Reserved
6	LCDCPDIS6	RW	0h	Reserved
5	LCDCPDIS5	RW	0h	Reserved
4	LCDCPDIS4	RW	0h	Reserved
3	LCDCPDIS3	RW	0h	Reserved
2	LCDCPDIS2	RW	0h	LCD charge pump disable during ADC12 conversion 0b = LCD charge pump not automatically disabled during conversion. 1b = LCD charge pump automatically disabled during conversion.
1	LCDCPDIS1	RW	0h	Reserved
0	LCDCPDIS0	RW	0h	Reserved

**32.3.11 LCDBIV Register**

LCD\_B Interrupt Vector Register

**Figure 32-22. LCDBIV Register**

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

**Table 32-15. LCDBIV Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDBIVx	R	0h	LCD_B interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: No capacitor connected; Interrupt Flag: LCDNOCAPIFG; Interrupt Priority: Highest 04h = Interrupt Source: Blink, segments off; Interrupt Flag: LCDBLKOFFIFG 06h = Interrupt Source: Blink, segments on; Interrupt Flag: LCDBLKONIFG 08h = Interrupt Source: Frame interrupt; Interrupt Flag: LCDFRMIFG; Interrupt Priority: Lowest

## **LCD\_C Controller**

The LCD\_C controller drives static and 2-mux to 8-mux LCDs. This chapter describes the LCD\_C controller. The differences between LCD\_B and LCD\_C are listed in [Table 33-1](#).

Topic	Page
<b>33.1 LCD_C Introduction .....</b>	<b>858</b>
<b>33.2 LCD_C Operation .....</b>	<b>860</b>
<b>33.3 LCD_C Registers .....</b>	<b>876</b>

### 33.1 LCD\_C Introduction

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

The LCD\_C controller features are:

- Display memory
- 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, 1/2 bias or 1/3 bias
  - 3-mux, 1/2 bias or 1/3 bias
  - 4-mux, 1/2 bias or 1/3 bias
  - 5-mux, 1/3 bias
  - 6-mux, 1/3 bias
  - 7-mux, 1/3 bias
  - 8-mux, 1/3 bias

The differences between LCD\_B and LCD\_C are listed in [Table 33-1](#).

**Table 33-1. Differences Between LCD\_B and LCD\_C**

Feature	LCD_B	LCD_C
Supported types of LCDs	Static, 2-, 3-, 4-mux	Static, 2-, 3-, 4-, 5-, 6-, 7, 8-mux
Maximum VLCDx settings	001111b	001111b
Maximum LCD voltage ( $V_{LCD,typ}$ )	3.44 V	3.44 V
Supported biasing schemes for 5-mux to 8-mux	5- to 8-mux not supported	1/3 biasing

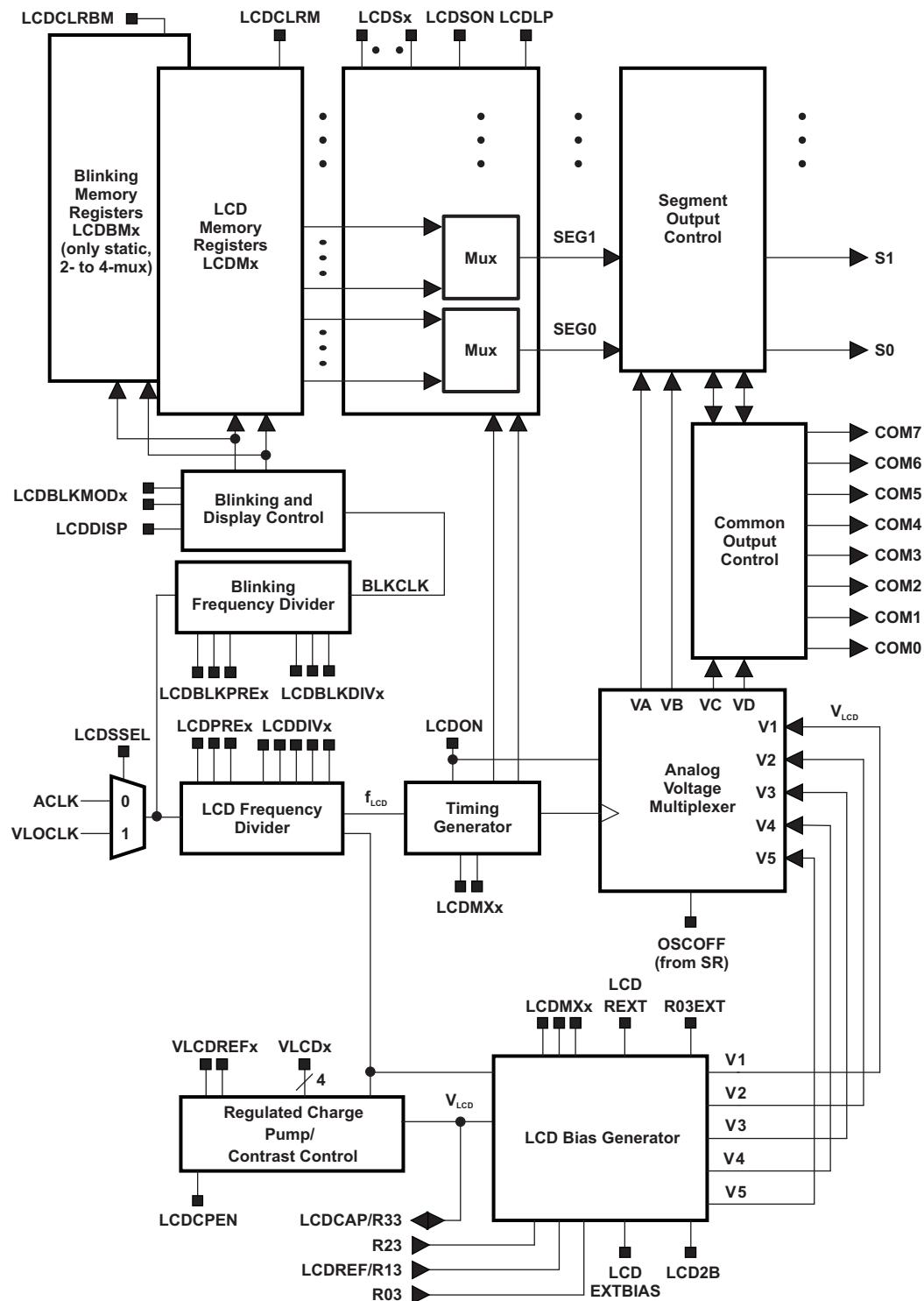
The LCD controller block diagram is shown in [Figure 33-1](#).

---

**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.

---



**Figure 33-1. LCD Controller Block Diagram**

## 33.2 LCD\_C Operation

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

### **33.2.1 LCD Memory**

The LCD memory organization differs slightly depending on the mode.

Each memory bit corresponds to one LCD segment 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 LCDM1, LCDM3, ...

Setting the bit LCDCLRM clears all LCD memory registers at the next frame boundary. It is reset automatically after the registers are cleared.

### **33.2.1.1 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. An example LCD memory map for these modes with 160 segments is shown in [Figure 33-2](#).

**Figure 33-2. LCD Memory for Static and 2-Mux to 4-Mux Mode - Example for 160 Segments**

### 33.2.1.2 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. An example LCD memory map for these modes with 160 segments is shown in [Figure 33-3](#)

Associated Common Pins	7	6	5	4	3	2	1	0	Associated Segment Pins
Register	7						0	n	
LCDM20	--	--	--	--	--	--	--	--	19 19
LCDM19	--	--	--	--	--	--	--	--	18 18
LCDM18	--	--	--	--	--	--	--	--	17 17
LCDM17	--	--	--	--	--	--	--	--	16 16
LCDM16	--	--	--	--	--	--	--	--	15 15
LCDM15	--	--	--	--	--	--	--	--	14 14
LCDM14	--	--	--	--	--	--	--	--	13 13
LCDM13	--	--	--	--	--	--	--	--	12 12
LCDM12	--	--	--	--	--	--	--	--	11 11
LCDM11	--	--	--	--	--	--	--	--	10 10
LCDM10	--	--	--	--	--	--	--	--	9 9
LCDM9	--	--	--	--	--	--	--	--	8 8
LCDM8	--	--	--	--	--	--	--	--	7 7
LCDM7	--	--	--	--	--	--	--	--	6 6
LCDM6	--	--	--	--	--	--	--	--	5 5
LCDM5	--	--	--	--	--	--	--	--	4 4
LCDM4	--	--	--	--	--	--	--	--	3 3
LCDM3	--	--	--	--	--	--	--	--	2 2
LCDM2	--	--	--	--	--	--	--	--	1 1
LCDM1	--	--	--	--	--	--	--	--	0 0

**Figure 33-3. LCD Memory for 5-Mux to 8-Mux Mode - Example for 160 Segments**

### 33.2.2 LCD Timing Generation

The LCD\_C controller uses the  $f_{LCD}$  signal from the integrated clock divider to generate the timing for common and segment lines. With the LCDSSEL bit, ACLK with a frequency between 30 kHz and 40 kHz or VLOCLK can be selected as clock source into the divider. The  $f_{LCD}$  frequency is selected with the LCDDPREx and LCDDIVx bits. The resulting  $f_{LCD}$  frequency is calculated by:

$$f_{LCD} = \frac{f_{ACLK/VLOCLK}}{(LCDDIVx + 1) \times 2^{LCDDPRE}}$$

The proper  $f_{LCD}$  frequency depends on the LCD's requirement for framing frequency and the LCD multiplex rate. It is calculated by:

$$f_{LCD} = 2 \times \text{mux} \times f_{Frame}$$

For example, to calculate  $f_{LCD}$  for a 3-mux LCD with a frame frequency of 30 Hz to 100 Hz:

$$f_{Frame} \text{ (from LCD data sheet)} = 30 \text{ Hz to } 100 \text{ Hz}$$

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

$$f_{LCD}(\min) = 180 \text{ Hz}$$

$$f_{LCD}(\max) = 600 \text{ Hz}$$

With  $f_{ACLK/VLOCLK} = 32768 \text{ Hz}$ , LCDDPRE = 011, and LCDDIVx = 10101:

$$f_{LCD} = 32768 \text{ Hz} / ((21+1) \times 2^3) = 32768 \text{ Hz} / 176 = 186 \text{ Hz}$$

With LCDPREx = 001 and LCDDIVx = 11011:

$$f_{LCD} = 32768 \text{ Hz} / ((27+1) \times 2^1) = 32768 \text{ Hz} / 56 = 585 \text{ Hz}$$

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

### 33.2.3 Blanking the LCD

The LCD controller allows blanking the complete LCD. The LCDSON bit is combined with a logical AND 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.

### 33.2.4 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.

#### 33.2.4.1 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 33-2](#). 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 LCDBM1, LCDBM3, ...

Setting the bit LCDCLRBM clears all blinking memory registers at the next frame boundary. It is automatically reset after the registers are cleared.

#### 33.2.4.2 Blinking Frequency

The blinking frequency  $f_{BLINK}$  is selected with the LCDBLKPREx and LCDBLKDIVx bits. The same clock is used as selected for the LCD frequency  $f_{LCD}$ . The resulting  $f_{BLINK}$  frequency is calculated by:

$$f_{Blink} = \frac{f_{ACLKVLO}}{(LCDBLKDIVx + 1) \times 2^{9+LCDBLKPREx}}$$

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.

---

#### 33.2.4.3 Dual Display Memory

In static and 2-mux to 4-mux mode, the blinking memory can also be used as a secondary display memory when no blinking mode LCDBLKMODx = 01 or 10 is selected. The memory to be displayed can be selected either manually using the LCDDISP bit or automatically with LCDBLKMODx = 11.

With LCDDISP = 0, the LCD memory is selected, and with LCDDISP = 1 the blinking memory is selected as display memory. Switching between the memories is synchronized to the frame boundaries.

With  $\text{LCDBLKMODx} = 11$  the LCD controller switches automatically between the memories using the divider to generate the blinking frequency. After  $\text{LCDBLKMODx} = 11$  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.

### 33.2.5 LCD Voltage And Bias Generation

The LCD\_C module allows selectable sources for the peak output waveform voltage,  $V_1$ , as well as the fractional LCD biasing voltages  $V_2$  to  $V_5$ .  $V_{\text{LCD}}$  may be sourced from  $V_{\text{CC}}$ , an internal charge pump, or externally.

All internal voltage generation is disabled if the selected clock source (ACLK or VLOCLK) is turned off ( $\text{OSCOFF} = 1$ ) or the LCD\_C module is disabled ( $\text{LCDON} = 0$ ).

#### 33.2.5.1 LCD Voltage Selection

$V_{\text{LCD}}$  is sourced from  $V_{\text{CC}}$  when  $\text{VLCDEXT} = 0$ ,  $\text{VLCDx} = 0$ , and  $\text{VREFx} = 0$ .  $V_{\text{LCD}}$  is sourced from the internal charge pump when  $\text{VLCDEXT} = 0$ ,  $\text{VLCDCPEN} = 1$ , and  $\text{VLCDx} > 0$ . The charge pump is always sourced from  $DV_{\text{CC}}$ . The VLCDx bits provide a software selectable LCD voltage from 2.6 V to 3.44 V (typical) independent of  $DV_{\text{CC}}$ . See the device-specific data sheet for specifications.

When the internal charge pump is used, a 4.7- $\mu\text{F}$  or larger capacitor must be connected between the  $\text{LCDCAP}$  pin and ground. If no capacitor is connected and the charge pump is enabled, the  $\text{LCDNOCAPIFG}$  interrupt flag is set, and the charge pump is disabled to prevent damage to the device. The charge pump may be temporarily disabled by setting  $\text{LCDCPEN} = 0$  with  $\text{VLCDx} > 0$  to reduce system noise. It can be automatically disabled during certain periods by setting the corresponding bits in the  $\text{LCDCCPCTL}$  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 4.7- $\mu\text{F}$  or larger capacitor must be connected from the  $\text{LCDCAP}$  pin to ground when the internal charge pump is enabled. If no capacitor is connected, the  $\text{LCDNOCAPIFG}$  interrupt flag is set and the charge pump is disabled.

---

The internal charge pump may use an external reference voltage when  $\text{VLCDREFx} = 01$  (and  $\text{LCDREXT} = 0$  and  $\text{LCDEXTBIAS} = 0$ ). In this case, the charge pump voltage is set to a multiply of the external reference voltage according to the VLCDx bits setting.

When  $\text{VLCDEXT} = 1$ ,  $V_{\text{LCD}}$  is sourced externally from the  $\text{LCDCAP}$  pin and the internal charge pump is disabled.

### 33.2.5.2 LCD Bias Generation

The fractional LCD biasing voltages, V2 to V5 can be generated internally or externally, independent of the source for  $V_{LCD}$ .

The bias generation block diagram for LCD\_C static and 2-mux to 8-mux modes is shown in Figure 33-4.

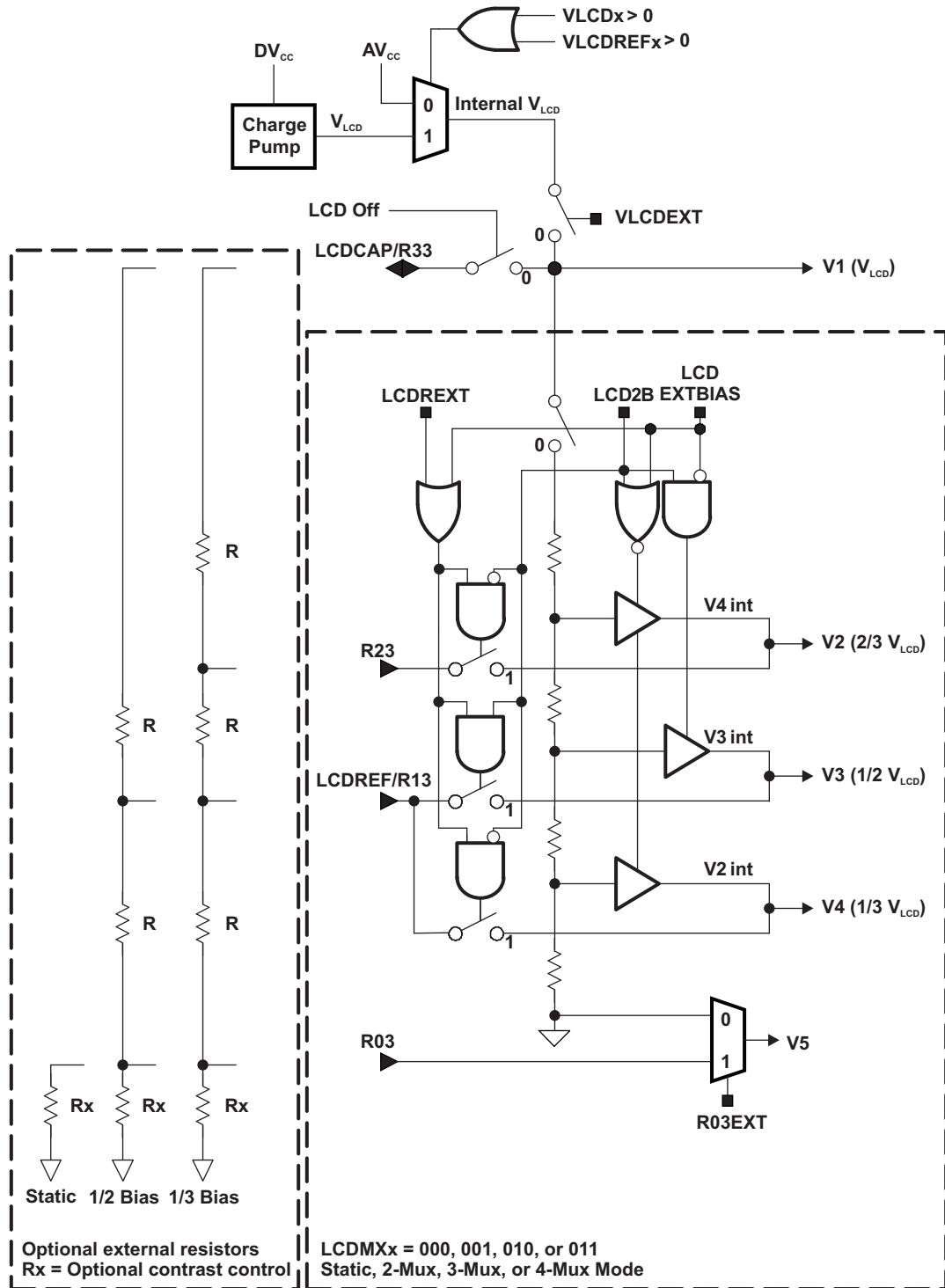


Figure 33-4. Bias Generation

The internally generated bias voltages V2 to V4 are switched to external pins with LCDREEXT = 1.

To source the bias voltages V2 to V4 externally, LCDEXTBIAS is set. This also disables the internal bias generation. Typically, an equally weighted resistor divider is used with resistors ranging from a few  $k\Omega$  to  $1 M\Omega$ , depending on the size of the display. When using an external resistor divider, the  $V_{LCD}$  voltage may be sourced from the internal charge pump when VLCDEXT = 0 taking the maximum charge pump load current into account. In static, 2-mux, 3-mux and 4-mux mode V5 can also be sourced externally when R03EXT is set to control the contrast of the connected display by changing the voltage at the low end of the external resistor divider as shown in the left part of [Figure 33-4](#).

When using an external resistor divider, R33 may serve as a switched  $V_{LCD}$  output when VLCDEXT = 0. This allows the power to the resistor ladder to be turned off, which eliminates current consumption when the LCD is not used. When VLCDEXT = 1, R33 serves as a  $V_{LCD}$  input.

In 2-mux, 3-mux, and 4-mux modes, the bias generator supports 1/2 biasing when LCD2B = 1 and 1/3 biasing when LCD2B = 0. In 5-mux to 8-mux modes, the bias generator supports 1/4 biasing when LCD2B = 1 and 1/3 biasing when LCD2B = 0. In static mode, the internal divider is disabled.

**Table 33-2. Bias Voltages and external Pins**

Mode	Bias Config	LCD2B	Voltage Level	Pin	Condition
Static	Static	X	V1 ("1")	R33	if LCDREXT=1 or LCDEXTBIAS=1
			V5 ("0")	R03	if R03EXT=1
2-mux, 3-mux, 4-mux	1/2	1	V1 ("1")	R33	if LCDREXT=1 or LCDEXTBIAS=1
			V3 ("1/2")	R13	if LCDREXT=1 or LCDEXTBIAS=1
			V5 ("0")	R03	if R03EXT=1
2-mux, 3-mux, 4-mux	1/3	0	V1 ("1")	R33	if LCDREXT=1 or LCDEXTBIAS=1
			V2 ("2/3")	R23	if LCDREXT=1 or LCDEXTBIAS=1
			V4 ("1/3")	R13	if LCDREXT=1 or LCDEXTBIAS=1
			V5 ("0")	R03	if R03EXT=1
5-mux to 8-mux	1/4	1	V1 ("1")	R33	if LCDREXT=1 or LCDEXTBIAS=1
			V2 ("3/4")	R23	if LCDREXT=1 or LCDEXTBIAS=1
			V3 ("2/4")	R13	if LCDREXT=1 or LCDEXTBIAS=1
			V4 ("1/4")	R03	if LCDREXT=1 or LCDEXTBIAS=1
			V5 ("0")	not available	
5-mux to 8-mux	1/3	0	V1 ("1")	R33	if LCDREXT=1 or LCDEXTBIAS=1
			V2 ("2/3")	R23	if LCDREXT=1 or LCDEXTBIAS=1
			V4 ("1/3")	R13	if LCDREXT=1 or LCDEXTBIAS=1
			V5 ("0")	R03	if R03EXT=1

### 33.2.5.3 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 and the selected biasing scheme. [Table 33-3](#) 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 33-3. LCD Voltage and Biasing Characteristics**

Mode	Bias Config	LCDMx	LCD2B	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	X	1	V1, V5	0	1	1/0
2-mux	1/2 <sup>(1)</sup>	0001	1	2	V1, V3, V5	0.354	0.791	2.236
	1/3	0001	0	2	V1, V2, V4, V5	0.333	0.745	2.236
3-mux	1/2	0010	1	3	V1, V3, V5	0.408	0.707	1.732
	1/3 <sup>(1)</sup>	0010	0	3	V1, V2, V4, V5	0.333	0.638	1.915
4-mux	1/2	0011	1	4	V1, V3, V5	0.433	0.661	1.528
	1/3 <sup>(1)</sup>	0011	0	4	V1, V2, V4, V5	0.333	0.577	1.732
5-mux	1/4	0100	1	5	V1, V2, V3, V4, V5	0.316	0.500	1.581
	1/3 <sup>(1)</sup>	0100	0	5	V1, V2, V4, V5	0.333	0.537	1.612
6-mux	1/4	0101	1	6	V1, V2, V3, V4, V5	0.306	0.468	1.528
	1/3 <sup>(1)</sup>	0101	0	6	V1, V2, V4, V5	0.333	0.509	1.528
7-mux	1/4 <sup>(1)</sup>	0110	1	7	V1, V2, V3, V4, V5	0.299	0.443	1.483
	1/3	0110	0	7	V1, V2, V4, V5	0.333	0.488	1.464
8-mux	1/4 <sup>(1)</sup>	0111	1	8	V1, V2, V3, V4, V5	0.293	0.424	1.446
	1/3	0111	0	8	V1, V2, V4, V5	0.333	0.471	1.414

<sup>(1)</sup> Recommended setting to achieve best contrast

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  $V_{th,10\%}$ .

In 3-mux and 4-mux mode, a 1/3 biasing is typically used, but a 1/2 biasing scheme is also possible. The 1/2 bias reduces the contrast ratio, but the advantage is a reduction of the required full-scale LCD voltage  $V_{LCD}$ .

In 7-mux and 8-mux mode, a 1/4 biasing is typically used, but a 1/3 biasing scheme is also possible. The 1/3 bias reduces the contrast ratio, but the advantage is a reduction of the required full-scale LCD voltage  $V_{LCD}$ .

### 33.2.6 LCD Outputs

Some LCD segment, common, and Rxx functions are multiplexed with digital I/O functions. These pins can function either as digital I/O or as LCD functions.

The LCD segment functions, when multiplexed with digital I/O, are selected using the LCDSx bits in the LCDCPCTLx registers. The LCDSx bits select the LCD function for each segment line. When LCDSx = 0, a multiplexed pin is set to digital I/O function. When LCDSx = 1, a multiplexed pin is selected as LCD function.

The pin functions for COMx and Rxx, when multiplexed with digital I/O, are selected as described in the port schematic section of the device-specific data sheet. An some devices the COM1 to COM7 pins are shared with segment lines, refer to the device-specific data sheet. If these pins are required as COM pins due to the selected LCD multiplexing mode, the COM functionality takes precedence over the segment function that can be selected for those pins with the LCDSx bits as for all other segment pins.

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 Pins**

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

---

### 33.2.7 LCD Interrupts

The LCD\_C module has four interrupt sources available, each with independent enables and flags.

The four interrupt flags, namely LCDFRMIFG, LCDBLKOFFIFG, LCDBLKONIFG, and LCDNOCAPIFG, are prioritized and combined to source a single interrupt vector. The interrupt vector register LCDCIV is used to determine which flag requested an interrupt.

The highest priority enabled interrupt generates a number in the LCDCIV 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 LCDCIV value.

Any read access of the LCDCIV 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 LCDCIV register automatically resets all pending interrupt flags. In addition, all flags can be cleared via software.

The LCDNOCAPIFG indicates that no capacitor is present at the LCDCAP pin when the charge pump is enabled. Setting the LCDNOCAPIE bit enables the interrupt.

The LCDBLKONIFG is set at the BLKCLK edge synchronized to the frame boundaries that turns on the segments when blinking is enabled with LCDBLKMODx = 01 or 10. It is also set at the BLKCLK edge synchronized to the frame boundaries 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 edge synchronized to the frame boundaries that blanks the segments when blinking is enabled with LCDBLKMODx = 01 or 10. It is also set at the BLKCLK edge synchronized to the frame boundaries 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.

### 33.2.7.1 LCDCIV Software Example

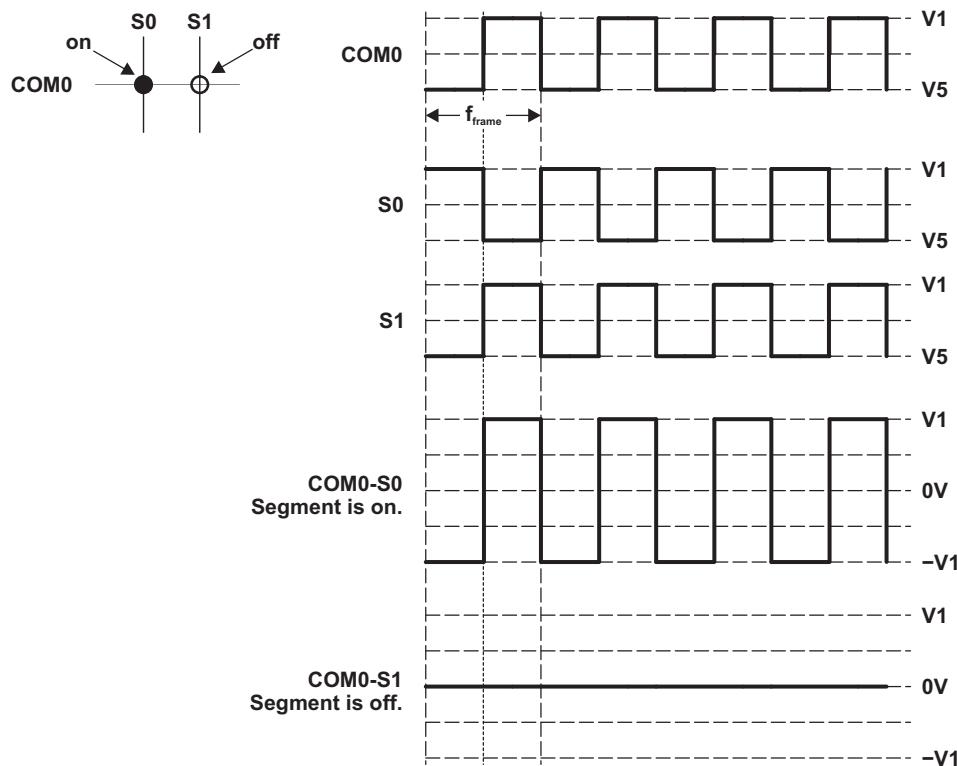
The following software example shows the recommended use of LCDCIV and the handling overhead. The LCDCIV value is added to the PC to automatically jump to the appropriate routine.

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_B interrupt flags.
LCDB_HND           ; Interrupt latency      6
    ADD &LCDBIV,PC ; Add offset to Jump table   3
    RETI             ; Vector 0: No interrupt     5
    JMP LCDNOCAP_HND ; Vector 2: LCDNOCAPIFG     2
    JMP LCDBLKON_HND ; Vector 4: LCDBLKONIFG     2
    JMP LCDBLKOFF_HND; Vector 6: LCDBLKOFFIFG     2
    LCDFRM_HND       ; Vector 8: LCDFRMIFG      2
    ...              ; Task starts here          5
    RETI
LCDNOCAP_HND ; Vector 2: LCDNOCAPIFG
    ... ; Task starts here          5
    RETI
LCDBLKON_HND ; Vector 4: LCDBLKONIFG
    ... ; Task starts here          5
    RETI ; Back to main program
LCDBLKOFF_HND ; Vector 6: LCDBLKOFFIFG
    ... ; Task starts here          5
    RETI ; Back to main program      5
```

### 33.2.8 Static Mode

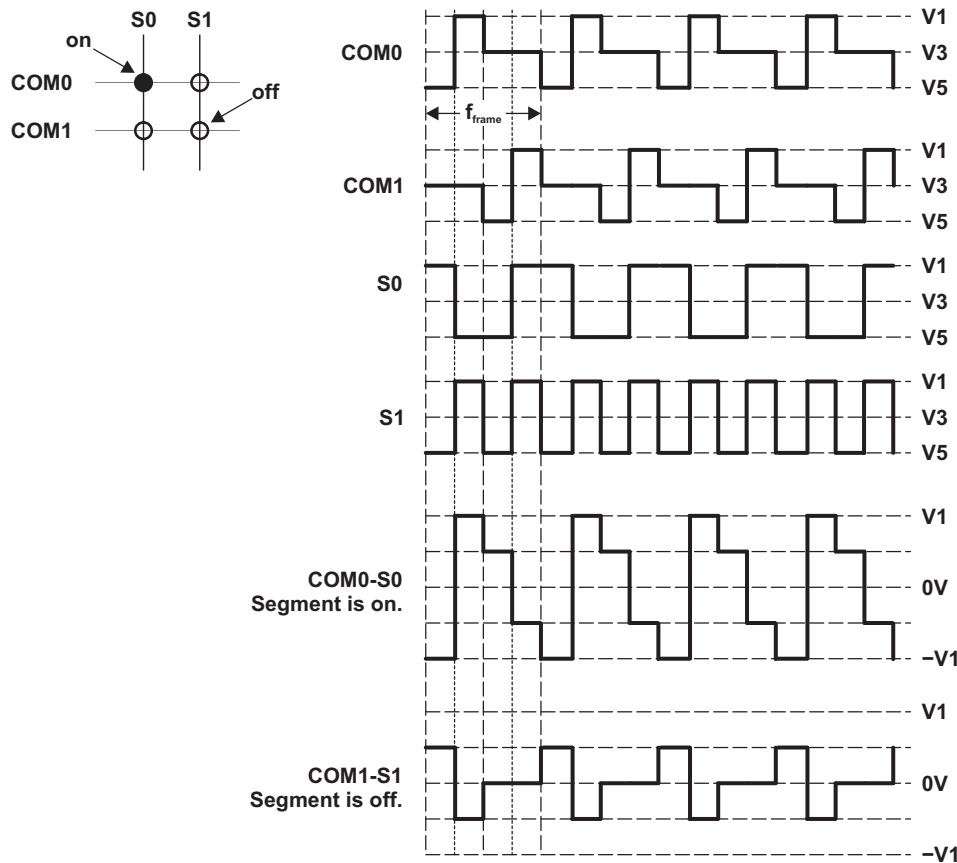
In static mode, each MSP430 segment pin drives one LCD segment, and one common line (COM0) is used. [Figure 33-5](#) shows some example static waveforms.



**Figure 33-5. Example Static Waveforms**

### 33.2.9 2-Mux Mode

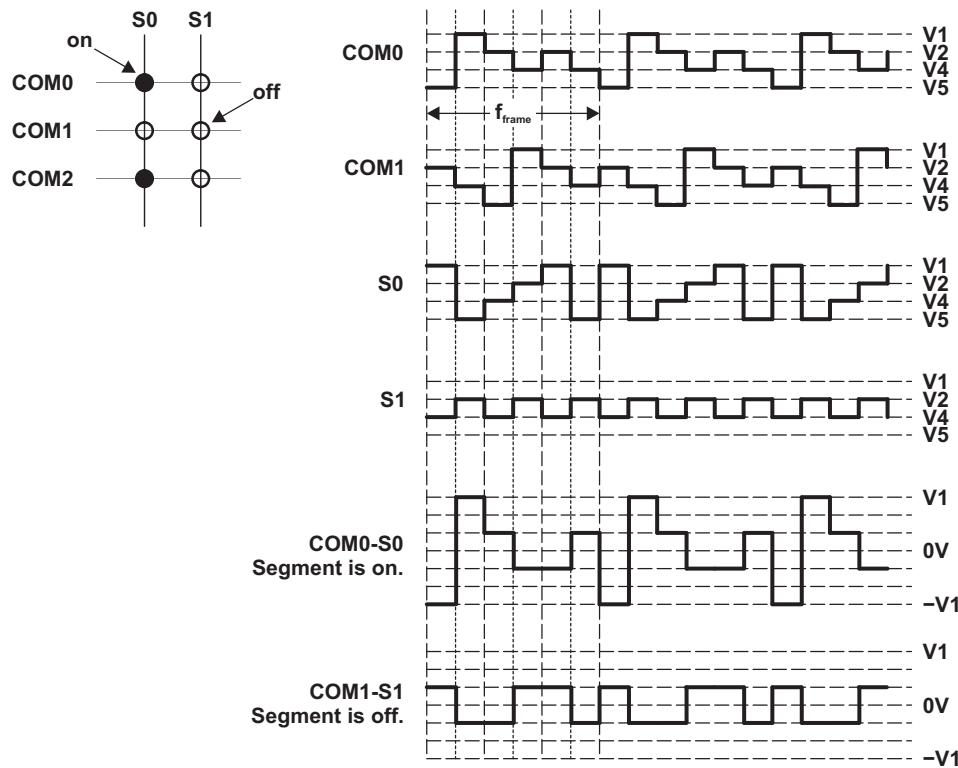
In 2-mux mode, each MSP430 segment pin drives two LCD segments, and two common lines (COM0 and COM1) are used. Figure 33-6 shows some example 2-mux 1/2-bias waveforms.



**Figure 33-6. Example 2-Mux Waveforms**

### 33.2.10 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 33-7](#) shows some example 3-mux 1/3-bias waveforms.



**Figure 33-7. Example 3-Mux Waveforms**

### 33.2.11 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 33-8 shows some example 4-mux 1/3-bias waveforms.

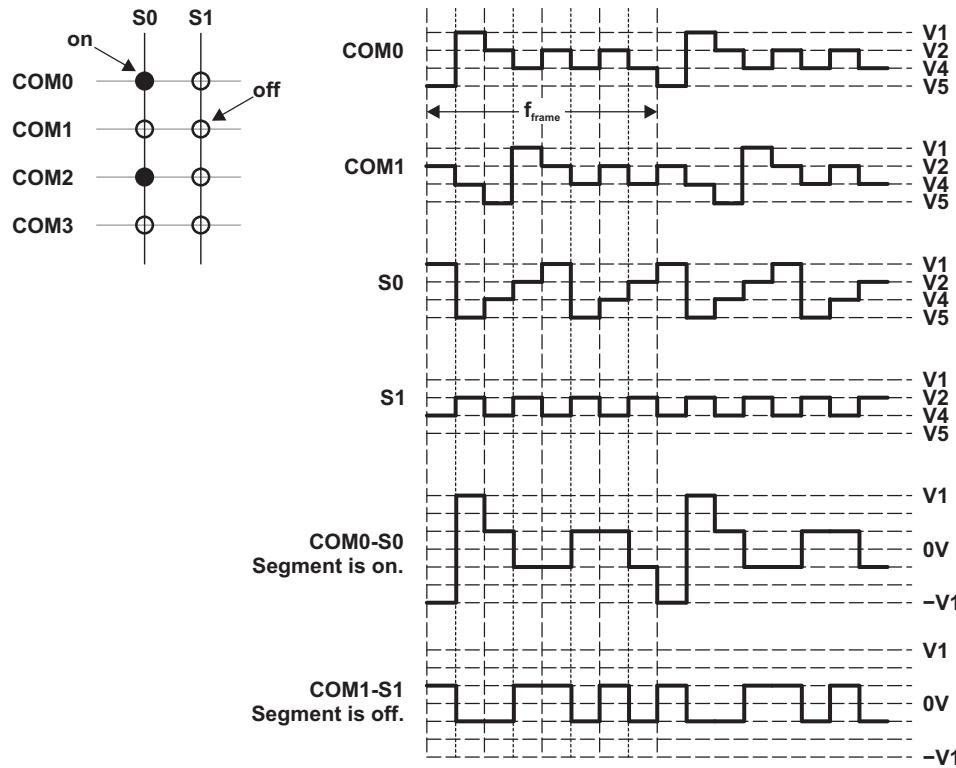
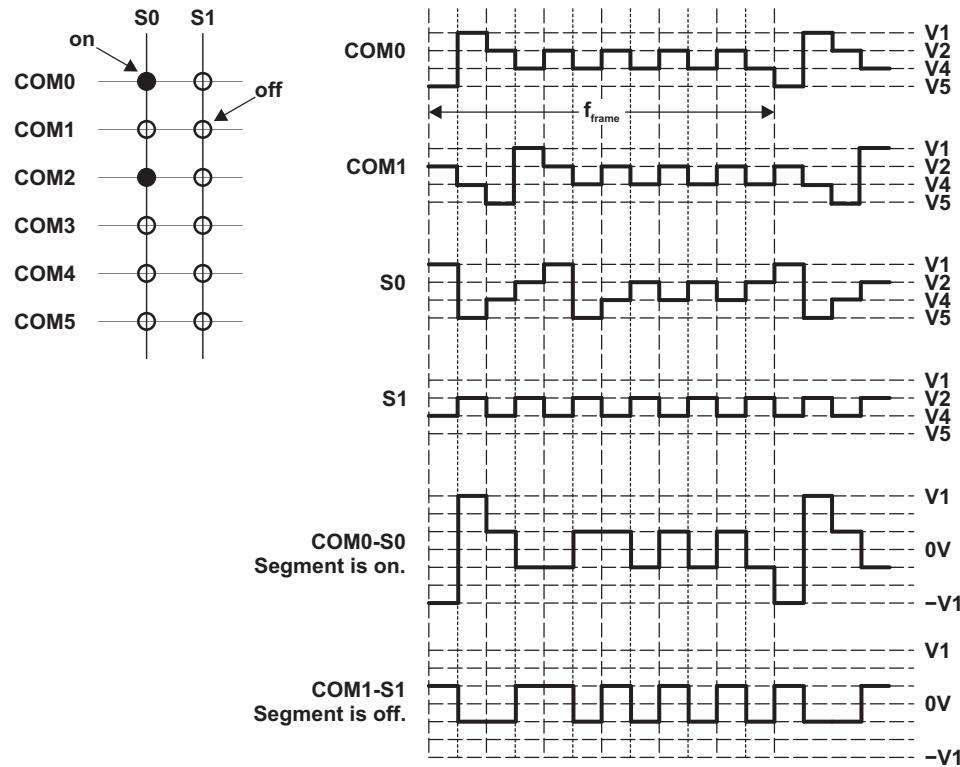


Figure 33-8. Example 4-Mux Waveforms

### 33.2.12 6-Mux Mode

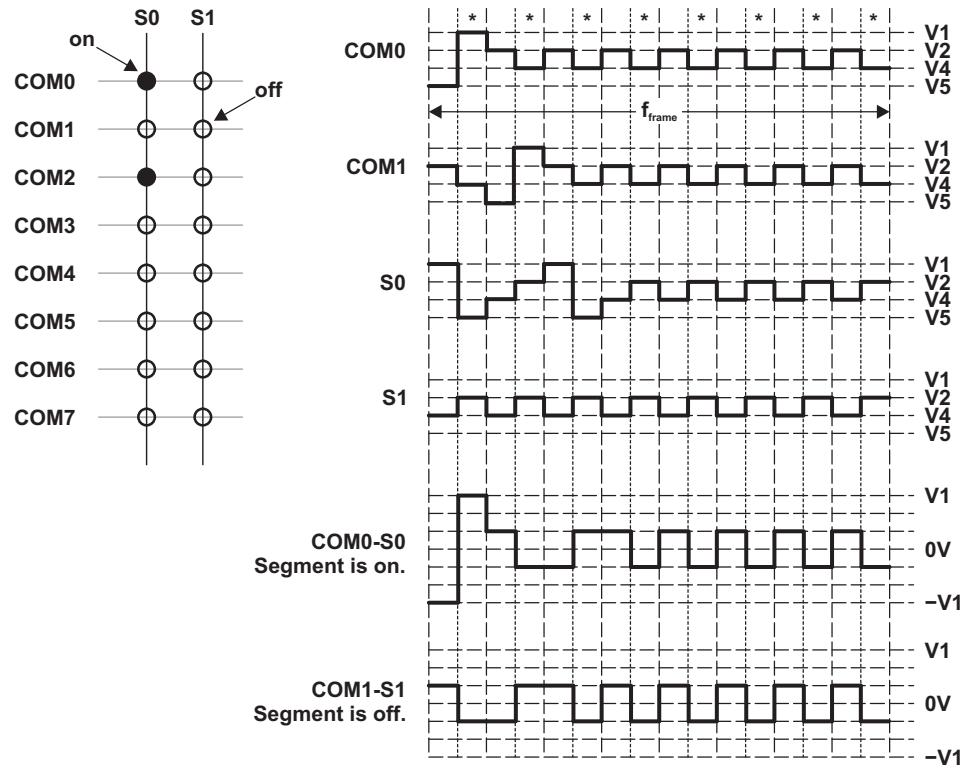
In 6-mux mode, each MSP430 segment pin drives six LCD segments, and six common lines (COM0, COM1, COM2, COM3, COM4, and COM5) are used. [Figure 33-9](#) shows some example 6-mux 1/3-bias waveforms.



**Figure 33-9. Example 6-Mux Waveforms**

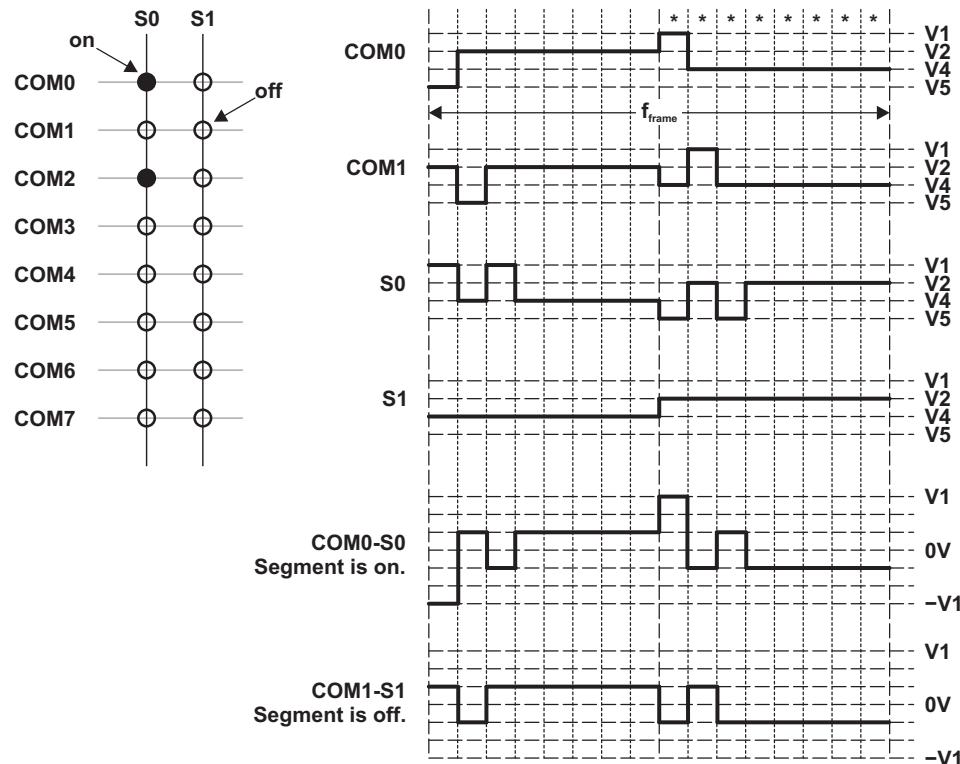
### 33.2.13 8-Mux Mode

In 8-mux mode, each MSP430 segment pin drives eight LCD segments, and eight common lines (COM0 through COM7) are used. [Figure 33-10](#) shows some example 8-mux 1/3-bias waveforms.



**Figure 33-10. Example 8-Mux, 1/3 Bias Waveforms (LCDLP = 0)**

Figure 33-11 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 33-10 are grouped together. The same principle applies to all mux modes.



**Figure 33-11. Example 8-Mux, 1/3 Bias Low-Power Waveforms (LCDLP = 1)**

### 33.3 LCD\_C Registers

The LCD\_C registers are listed in [Table 33-4](#) to [Table 33-7](#). 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 33-4. LCD\_C Control Registers**

Offset	Acronym	Register Name	Type	Reset	Section
000h	LCDCCTL0	LCD_C control 0	Read/write	0000h	<a href="#">Section 33.3.1</a>
002h	LCDCCTL1	LCD_C control 1	Read/write	0000h	<a href="#">Section 33.3.2</a>
004h	LCDCBLKCTL	LCD_C blinking control	Read/write	0000h	<a href="#">Section 33.3.3</a>
006h	LCDCMEMCTL	LCD_C memory control	Read/write	0000h	<a href="#">Section 33.3.4</a>
008h	LCDCVCTL	LCD_C voltage control	Read/write	0000h	<a href="#">Section 33.3.5</a>
00Ah	LCDCPCTL0	LCD_C port control 0	Read/write	0000h	<a href="#">Section 33.3.6</a>
00Ch	LCDCPCTL1	LCD_C port control 1	Read/write	0000h	<a href="#">Section 33.3.7</a>
00Eh	LCDCPCTL2	LCD_C port control 2 ( $\geq 256$ segments)	Read/write	0000h	<a href="#">Section 33.3.8</a>
010h	LCDCPCTL3	LCD_C port control 3 (384 segments)	Read/write	0000h	<a href="#">Section 33.3.9</a>
012h	LCDCCPCTL	LCD_C charge pump control	Read/write	0000h	<a href="#">Section 33.3.10</a>
014h		Reserved			
016h		Reserved			
018h		Reserved			
01Ah		Reserved			
01Ch		Reserved			
01Eh	LCDCIV	LCD_C interrupt vector	Read/write	0000h	<a href="#">Section 33.3.11</a>

**Table 33-5. LCD\_C Memory Registers for Static and 2-Mux to 4-Mux Modes<sup>(1)(2)</sup>**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Reset</b>
020h	LCDM1	LCD memory 1 (S1/S0)	Read/write	Unchanged
021h	LCDM2	LCD memory 2 (S3/S2)	Read/write	Unchanged
022h	LCDM3	LCD memory 3 (S5/S4)	Read/write	Unchanged
023h	LCDM4	LCD memory 4 (S7/S6)	Read/write	Unchanged
024h	LCDM5	LCD memory 5 (S9/S8)	Read/write	Unchanged
025h	LCDM6	LCD memory 6 (S11/S10)	Read/write	Unchanged
026h	LCDM7	LCD memory 7 (S13/S12)	Read/write	Unchanged
027h	LCDM8	LCD memory 8 (S15/S14)	Read/write	Unchanged
028h	LCDM9	LCD memory 9 (S17/S16)	Read/write	Unchanged
029h	LCDM10	LCD memory 10 (S19/S18)	Read/write	Unchanged
02Ah	LCDM11	LCD memory 11 (S21/S20)	Read/write	Unchanged
02Bh	LCDM12	LCD memory 12 (S23/S22)	Read/write	Unchanged
02Ch	LCDM13	LCD memory 13 (S25/S24)	Read/write	Unchanged
02Dh	LCDM14	LCD memory 14 (S27/S26)	Read/write	Unchanged
02Eh	LCDM15	LCD memory 15 (S29/S28)	Read/write	Unchanged
02Fh	LCDM16	LCD memory 16 (S31/S30)	Read/write	Unchanged
030h	LCDM17	LCD memory 17 (S33/S32)	Read/write	Unchanged
031h	LCDM18	LCD memory 18 (S35/S34)	Read/write	Unchanged
032h	LCDM19	LCD memory 19 (S37/S36)	Read/write	Unchanged
033h	LCDM20	LCD memory 20 (S39/S38)	Read/write	Unchanged
034h	LCDM21	LCD memory 21 (S41/S40)	Read/write	Unchanged
035h	LCDM22	LCD memory 22 (S43/S42)	Read/write	Unchanged
036h	LCDM23	LCD memory 23 (S45/S44)	Read/write	Unchanged
037h	LCDM24	LCD memory 24 (S47/S46)	Read/write	Unchanged
038h	LCDM25	LCD memory 25 (S49/S48)	Read/write	Unchanged
039h	LCDM26	LCD memory 26 (S51/S50)	Read/write	Unchanged
03Ah	LCDM27	LCD memory 27 (S53/S52)	Read/write	Unchanged
03Bh	LCDM28	LCD memory 28 (S54)	Read/write	Unchanged
03Ch		Reserved		
03Dh		Reserved		
03Eh		Reserved		
03Fh		Reserved		

<sup>(1)</sup> The LCD memory registers can also be accessed as word.

<sup>(2)</sup> 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 33-6. LCD Blinking Memory Registers for Static and 2-Mux to 4-Mux Modes<sup>(1)(2)</sup>**

Offset	Acronym	Register Name	Type	Reset
040h	LCDBM1	LCD blinking memory 1	Read/write	Unchanged
041h	LCDBM2	LCD blinking memory 2	Read/write	Unchanged
042h	LCDBM3	LCD blinking memory 3	Read/write	Unchanged
043h	LCDBM4	LCD blinking memory 4	Read/write	Unchanged
044h	LCDBM5	LCD blinking memory 5	Read/write	Unchanged
045h	LCDBM6	LCD blinking memory 6	Read/write	Unchanged
046h	LCDBM7	LCD blinking memory 7	Read/write	Unchanged
047h	LCDBM8	LCD blinking memory 8	Read/write	Unchanged
048h	LCDBM9	LCD blinking memory 9	Read/write	Unchanged
049h	LCDBM10	LCD blinking memory 10	Read/write	Unchanged
04Ah	LCDBM11	LCD blinking memory 11	Read/write	Unchanged
04Bh	LCDBM12	LCD blinking memory 12	Read/write	Unchanged
04Ch	LCDBM13	LCD blinking memory 13	Read/write	Unchanged
04Dh	LCDBM14	LCD blinking memory 14	Read/write	Unchanged
04Eh	LCDBM15	LCD blinking memory 15	Read/write	Unchanged
04Fh	LCDBM16	LCD blinking memory 16	Read/write	Unchanged
050h	LCDBM17	LCD blinking memory 17	Read/write	Unchanged
051h	LCDBM18	LCD blinking memory 18	Read/write	Unchanged
052h	LCDBM19	LCD blinking memory 19	Read/write	Unchanged
053h	LCDBM20	LCD blinking memory 20	Read/write	Unchanged
054h	LCDBM21	LCD blinking memory 21	Read/write	Unchanged
055h	LCDBM22	LCD blinking memory 22	Read/write	Unchanged
056h	LCDBM23	LCD blinking memory 23	Read/write	Unchanged
057h	LCDBM24	LCD blinking memory 24	Read/write	Unchanged
058h	LCDBM25	LCD blinking memory 25	Read/write	Unchanged
059h	LCDBM26	LCD blinking memory 26	Read/write	Unchanged
05Ah	LCDBM27	LCD blinking memory 27	Read/write	Unchanged
05Bh	LCDBM28	LCD blinking memory 28	Read/write	Unchanged
05Ch		Reserved		
05Dh		Reserved		
05Eh		Reserved		
05Fh		Reserved		

<sup>(1)</sup> The LCD blinking memory registers can also be accessed as word.

<sup>(2)</sup> 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 33-7. LCD Memory Registers for 5-Mux to 8-Mux<sup>(1)(2)</sup>**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Reset</b>
020h	LCDM1	LCD memory 1 (S0)	Read/write	Unchanged
021h	LCDM2	LCD memory 2 (S1)	Read/write	Unchanged
022h	LCDM3	LCD memory 3 (S2)	Read/write	Unchanged
023h	LCDM4	LCD memory 4 (S3)	Read/write	Unchanged
024h	LCDM5	LCD memory 5 (S4)	Read/write	Unchanged
025h	LCDM6	LCD memory 6 (S5)	Read/write	Unchanged
026h	LCDM7	LCD memory 7 (S6)	Read/write	Unchanged
027h	LCDM8	LCD memory 8 (S7)	Read/write	Unchanged
028h	LCDM9	LCD memory 9 (S8)	Read/write	Unchanged
029h	LCDM10	LCD memory 10 (S9)	Read/write	Unchanged
02Ah	LCDM11	LCD memory 11 (S10)	Read/write	Unchanged
02Bh	LCDM12	LCD memory 12 (S11)	Read/write	Unchanged
02Ch	LCDM13	LCD memory 13 (S12)	Read/write	Unchanged
02Dh	LCDM14	LCD memory 14 (S13)	Read/write	Unchanged
02Eh	LCDM15	LCD memory 15 (S14)	Read/write	Unchanged
02Fh	LCDM16	LCD memory 16 (S15)	Read/write	Unchanged
030h	LCDM17	LCD memory 17 (S16)	Read/write	Unchanged
031h	LCDM18	LCD memory 18 (S17)	Read/write	Unchanged
032h	LCDM19	LCD memory 19 (S18)	Read/write	Unchanged
033h	LCDM20	LCD memory 20 (S19)	Read/write	Unchanged
034h	LCDM21	LCD memory 21 (S20)	Read/write	Unchanged
035h	LCDM22	LCD memory 22 (S21)	Read/write	Unchanged
036h	LCDM23	LCD memory 23 (S22)	Read/write	Unchanged
037h	LCDM24	LCD memory 24 (S23)	Read/write	Unchanged
038h	LCDM25	LCD memory 25 (S24)	Read/write	Unchanged
039h	LCDM26	LCD memory 26 (S25)	Read/write	Unchanged
03Ah	LCDM27	LCD memory 27 (S26)	Read/write	Unchanged
03Bh	LCDM28	LCD memory 28 (S27)	Read/write	Unchanged
03Ch	LCDM29	LCD memory 29 (S28)	Read/write	Unchanged
03Dh	LCDM30	LCD memory 30 (S29)	Read/write	Unchanged
03Eh	LCDM31	LCD memory 31 (S30)	Read/write	Unchanged
03Fh	LCDM32	LCD memory 32 (S31)	Read/write	Unchanged
040h	LCDM33	LCD memory 33 (S32)	Read/write	Unchanged
041h	LCDM34	LCD memory 34 (S33)	Read/write	Unchanged
042h	LCDM35	LCD memory 35 (S34)	Read/write	Unchanged
043h	LCDM36	LCD memory 36 (S35)	Read/write	Unchanged
044h	LCDM37	LCD memory 37 (S36)	Read/write	Unchanged
045h	LCDM38	LCD memory 38 (S37)	Read/write	Unchanged
046h	LCDM39	LCD memory 39 (S38)	Read/write	Unchanged
047h	LCDM40	LCD memory 40 (S39)	Read/write	Unchanged
048h	LCDM41	LCD memory 41 (S40)	Read/write	Unchanged
049h	LCDM42	LCD memory 42 (S41)	Read/write	Unchanged
04Ah	LCDM43	LCD memory 43 (S42)	Read/write	Unchanged
04Bh	LCDM44	LCD memory 44 (S43)	Read/write	Unchanged
04Ch	LCDM45	LCD memory 45 (S44)	Read/write	Unchanged

<sup>(1)</sup> The LCD memory registers can also be accessed as word.

<sup>(2)</sup> 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 33-7. LCD Memory Registers for 5-Mux to 8-Mux<sup>(1)(2)</sup> (continued)**

Offset	Acronym	Register Name	Type	Reset
04Dh	LCDM46	LCD memory 46 (\$45)	Read/write	Unchanged
04Eh	LCDM47	LCD memory 47 (\$46)	Read/write	Unchanged
04Fh	LCDM48	LCD memory 48 (\$47)	Read/write	Unchanged
050h	LCDM49	LCD memory 49 (\$48)	Read/write	Unchanged
051h	LCDM50	LCD memory 50 (\$49)	Read/write	Unchanged
052h	LCDM51	LCD memory 51 (\$50)	Read/write	Unchanged
053h	LCDM52	LCD memory 52 (\$51)	Read/write	Unchanged
054h		Reserved		
055h		Reserved		
056h		Reserved		
057h		Reserved		
058h		Reserved		
059h		Reserved		
05Ah		Reserved		
05Bh		Reserved		
05Ch		Reserved		
05Dh		Reserved		
05Eh		Reserved		
05Fh		Reserved		

### 33.3.1 LCDCCTL0 Register

LCD\_C Control Register 0

NOTE: Settings for LCDDIVx, LCDPREx, LCDSSEL, LCDLP and LCDMXx should be changed only while LCDON = 0.

**Figure 33-12. LCDCCTL0 Register**

15	14	13	12	11	10	9	8
LCDDIVx						LCDPREx	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
LCDSEL	Reserved	LCDMXx			LCDSON	LCDLP	LCDON
rw-0	r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 33-8. LCDCCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-11	LCDDIVx	RW	0h	LCD frequency divider. Together with LCDPREx the LCD frequency f(LCD) is calculated as $f(LCD) = f(ACLK/VLO) / ((LCDDIVx + 1) \times 2^{(LCDPREx)})$ . 00000b = Divide by 1 00001b = Divide by 2 : 11110b = Divide by 31 11111b = Divide by 32
10-8	LCDPREx	RW	0h	LCD frequency pre-scaler. Together with LCDDIVx the LCD frequency f(LCD) is calculated as $f(LCD) = f(ACLK/VLO) / ((LCDDIVx + 1) \times 2^{(LCDPREx)})$ . 000b = Divide by 1 001b = Divide by 2 010b = Divide by 4 011b = Divide by 8 100b = Divide by 16 101b = Divide by 32 110b = Reserved (defaults to divide by 32) 111b = Reserved (defaults to divide by 32)
7	LCDSEL	RW	0h	Clock source select for LCD and blinking frequency 0b = ACLK (30 kHz to 40 kHz) 1b = VLOCLK
6	Reserved	R	0h	Reserved
5-3	LCDMXx	RW	0h	LCD mux rate. These bits select the LCD mode. 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

**Table 33-8. LCDCCTL0 Register Description (continued)**

Bit	Field	Type	Reset	Description
0	LCDON	RW	0h	LCD on. This bit turns the LCD_C module on or off. 0b = LCD_C module off 1b = LCD_C module on

### 33.3.2 LCDCCTL1 Register

LCD\_C Control Register 1

**Figure 33-13. LCDCCTL1 Register**

15	14	13	12	11	10	9	8
		Reserved		LCDNOCAPIE	LCDBLKONIE	LCDBLKOFFIE	LCDFRMIE
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
		Reserved		LCDNOCAPIFG	LCDBLKONIFG	LCDBLKOFFIFG	LCDFRMIFG
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

**Table 33-9. LCDCCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved
11	LCDNOCAPIE	RW	0h	No capacitance connected interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
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
7-4	Reserved	R	0h	Reserved
3	LCDNOCAPIFG	RW	0h	No capacitance connected interrupt flag. Set when charge pump is enabled but no capacitance is connected to LCDCAP pin. 0b = No interrupt pending 1b = Interrupt pending
2	LCDBLKONIFG	RW	0h	LCD blinking interrupt flag, segments switched on. 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, segments switched off. 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

### 33.3.3 LCDCBLKCTL Register

LCD\_C Blink Control Register

NOTE: Settings for LCDBLKDIVx and LCDBLKPReX should only be changed while LCDBLKMODx = 00.

**Figure 33-14. LCDCBLKCTL 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
LCDBLKDIVx			LCDBLKPReX			LCDBLKMODx	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 33-10. LCDCBLKCTL Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-5	LCDBLKDIVx	RW	0h	<p>Clock divider for blinking frequency. Together with LCDBLKPReX, the blinking frequency <math>f(BLINK)</math> is calculated as <math>f(BLINK) = f(ACLK/VLO) / ((LCDBLKDIVx + 1) \times 2^{(9+LCDBLKPReX)})</math>.</p> <p>NOTE: Should only be changed while LCDBLKMODx = 00.</p> <p>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</p>
4-2	LCDBLKPReX	RW	0h	<p>Clock pre-scaler for blinking frequency. Together with LCDBLKDIVx, the blinking frequency <math>f(BLINK)</math> is calculated as <math>f(BLINK) = f(ACLK/VLO) / ((LCDBLKDIVx + 1) \times 2^{(9+LCDBLKPReX)})</math>.</p> <p>NOTE: Should only be changed while LCDBLKMODx = 00.</p> <p>000b = Divide by 512      001b = Divide by 1024      010b = Divide by 2048      011b = Divide by 4096      100b = Divide by 8162      101b = Divide by 16384      110b = Divide by 32768      111b = Divide by 65536</p>
1-0	LCDBLKMODx	RW	0h	<p>Blinking mode</p> <p>00b = Blinking disabled      01b = Blinking of individual segments as enabled in blinking memory register LCDBMx. In mux mode &gt;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 &gt;5 blinking is disabled.</p>

### 33.3.4 LCDMEMCTL Register

LCD\_C Memory Control Register

**Figure 33-15. 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					LCDCLRB <sub>M</sub>	LCDCLRM	LCDDISP
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

**Table 33-11. LCDMEMCTL Register Description**

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved
2	LCDCLRB <sub>M</sub>	RW	0h	<p>Clear LCD blinking memory</p> <p>Clears all blinking memory registers LCDBM<sub>x</sub>. The bit is automatically reset when the blinking memory is cleared.</p> <p>Setting this bit has in 5-mux mode and above has no effect. It's immediately reset again.</p> <p>0b = Contents of blinking memory registers LCDBM<sub>x</sub> remain unchanged 1b = Clear content of all blinking memory registers LCDBM<sub>x</sub></p>
1	LCDCLRM	RW	0h	<p>Clear LCD memory</p> <p>Clears all LCD memory registers LCDM<sub>x</sub>. The bit is automatically reset when the LCD memory is cleared.</p> <p>0b = Contents of LCD memory registers LCDM<sub>x</sub> remain unchanged 1b = Clear content of all LCD memory registers LCDM<sub>x</sub></p>
0	LCDDISP	RW	0h	<p>Select LCD memory registers for display</p> <p>The bit is cleared in LCDBLKMOD<sub>x</sub> = 01 and LCDBLKMOD<sub>x</sub> = 10 or if a mux mode <math>\geq 5</math> is selected and cannot be changed by software.</p> <p>When LCDBLKMOD<sub>x</sub> = 11, this bit reflects the currently displayed memory but cannot be changed by software. When returning to LCDBLKMOD<sub>x</sub> = 00 the bit is cleared.</p> <p>0b = Display content of LCD memory registers LCDM<sub>x</sub> 1b = Display content of LCD blinking memory registers LCDBM<sub>x</sub></p>

**33.3.5 LCDCVCTL Register**

LCD\_C Voltage Control Register

NOTE: Settings for LCDREXT, R03EXT, LCDEXTBIAS, VLCDEXT, VLCDREFx, and LCD2B should be changed only while LCDON = 0.

**Figure 33-16. LCDCVCTL Register**

15	14	13	12	11	10	9	8
		Reserved	VLCDx				Reserved
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0
7	6	5	4	3	2	1	0
LCDREXT	R03EXT	LCDEXTBIAS	VLCDEXT	LCDCPEN	VLCDREFx	LCD2B	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 33-12. LCDCVCTL Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved
12-9	VLCDx	RW	0h	Charge pump voltage select. LCDCPEN must be 1 for the charge pump to be enabled. VCC is used for V(LCD) when VLCDx = 0000 and VLCDREFx = 00 and VLCDEXT = 0. 0000b = Charge pump disabled 0001b = If VLCDREFx = 00 or 10: V(LCD) = 2.60 V; If VLCDREFx = 01 or 11: V(LCD) = 2.17 × V(REF) 0010b to 1110b = If VLCDREFx = 00 or 10: V(LCD) = 2.60 V + (VLCDx - 1) × 0.06 V; If VLCDREFx = 01 or 11: V(LCD) = 2.17 × V(REF) + (VLCDx - 1) × 0.05 × V(REF) 1111b = If VLCDREFx = 00 or 10: V(LCD) = 2.60 V + (15 - 1) × 0.06 V = 3.44 V; If VLCDREFx = 01 or 11: V(LCD) = 2.17 × V(REF) + (15 - 1) × 0.05 × V(REF) = 2.87 × V(REF)
8	Reserved	R	0h	Reserved
7	LCDREXT	RW	0h	V2 to V4 voltage on external Rx3 pins. This bit selects the external connections for voltages V2 to V4 with internal bias generation (LCDEXTBIAS = 0). The bit is don't care if external biasing is selected (LCDEXTBIAS = 1). NOTE: Should be changed only while LCDON = 0. 0b = Internally generated V2 to V4 are not switched to pins (LCDEXTBIAS = 0) 1b = Internally generated V2 to V4 are switched to pins (LCDEXTBIAS = 0)
6	R03EXT	RW	0h	V5 voltage select. This bit selects the external connection for the lowest LCD voltage. R03EXT is ignored if there is no R03 pin available. NOTE: Should be changed only while LCDON = 0. 0b = V5 is VSS 1b = V5 is sourced from the R03 pin
5	LCDEXTBIAS	RW	0h	V2 to V4 voltage select. This bit selects the generation for voltages V2 to V4. NOTE: Should be changed only while LCDON = 0. 0b = V2 to V4 are generated internally 1b = V2 to V4 are sourced externally and the internal bias generator is switched off
4	VLCDEXT	RW	0h	V(LCD) source select NOTE: Should be changed only while LCDON = 0. 0b = V(LCD) is generated internally 1b = V(LCD) is sourced externally
3	LCDCPEN	RW	0h	Charge pump enable 0b = Charge pump disabled 1b = Charge pump enabled when V(LCD) is generated internally (VLCDEXT = 0) and VLCDx > 0 or VLCDREFx > 0

**Table 33-12. LCDCVCTL Register Description (continued)**

Bit	Field	Type	Reset	Description
2-1	VLCDREFx	RW	0h	<p>Charge pump reference select. If LCDEXTBIAST = 1 or LCDREXT = 1, settings 01, 10, and 11 are not supported; the internal reference voltage is used instead.</p> <p>NOTE: Should be changed only while LCDON = 0.</p> <p>00b = Internal reference voltage</p> <p>01b = External reference voltage</p> <p>10b = Internal reference voltage switched to external pin LCDREF/R13</p> <p>11b = Reserved (defaults to external reference voltage)</p>
0	LCD2B	RW	0h	<p>Bias select. LCD2B is ignored in static mode or mux modes <math>\geq 5</math>.</p> <p>NOTE: Should be changed only while LCDON = 0.</p> <p>0b = 1/3 bias</p> <p>1b = 1/2 bias</p>

**33.3.6 LCDCPCTL0 Register**

LCD\_C Port Control Register 0

NOTE: Settings for LCDSx should be changed only while LCDON = 0.

**Figure 33-17. LCDCPCTL0 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 33-13. LCDCPCTL0 Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDSx	RW	0h	LCD segment line x enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. NOTE: Settings for LCDSx should be changed only while LCDON = 0. 0b = Multiplexed pins are port functions 1b = Pins are LCD functions

### 33.3.7 LCDCPCTL1 Register

LCD\_C Port Control Register 1

NOTE: Settings for LCDSx should be changed only while LCDON = 0.

**Figure 33-18. LCDCPCTL1 Register**

15	14	13	12	11	10	9	8
LCDS31	LCDS30	LCDS29	LCDS28	LCDS27	LCDS26	LCDS25	LCDS24
rw-0							
7	6	5	4	3	2	1	0
LCDS23	LCDS22	LCDS21	LCDS20	LCDS19	LCDS18	LCDS17	LCDS16
rw-0							

**Table 33-14. LCDCPCTL1 Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDSx	RW	0h	<p>LCD segment line x enable.</p> <p>On devices supporting a maximum of 192 segments LCDS31 is reserved, if COM7 to COM1 are shared with segments. If COM7 to COM1 are not shared with segments LCDS24 to LCDS31 are reserved.</p> <p>This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>NOTE: Settings for LCDSx should be changed only while LCDON = 0.</p> <p>0b = Multiplexed pins are port functions</p> <p>1b = Pins are LCD functions</p>

### 33.3.8 LCDCPCTL2 Register

LCD\_C Port Control Register 2 ( $\geq 256$  Segments)

NOTE: Settings for LCDSx should be changed only while LCDON = 0.

**Figure 33-19. LCDCPCTL2 Register**

15	14	13	12	11	10	9	8
LCDS47	LCDS46	LCDS45	LCDS44	LCDS43	LCDS42	LCDS41	LCDS40
rw-0							
7	6	5	4	3	2	1	0
LCDS39	LCDS38	LCDS37	LCDS36	LCDS35	LCDS34	LCDS33	LCDS32
rw-0							

**Table 33-15. LCDCPCTL2 Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDSx	RW	0h	<p>LCD segment line x enable.            On devices supporting a maximum of 256 segments LCDS39 to LCDS47 are reserved, if COM7 to COM1 are shared with segments. If COM7 to COM1 are not shared with segments the complete register LCDCPCTL2 is not available.            On devices supporting a maximum of 320 segments, LCDS47 is reserved if COM7 to COM1 are shared with segments. If COM7 to COM1 are not shared with segments, LCDS40 to LCDS47 are reserved.            This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function.            NOTE: Settings for LCDSx should be changed only while LCDON = 0.            0b = Multiplexed pins are port functions            1b = Pins are LCD functions</p>

### 33.3.9 LCDCPCTL3 Register

LCD\_C Port Control Register 2 (384 Segments, COMs Shared With Segments)

NOTE: Settings for LCDSx should be changed only while LCDON = 0.

**Figure 33-20. LCDCPCTL3 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	LCDS53	LCDS52	LCDS51	LCDS50	LCDS49	LCDS48	
r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 33-16. LCDCPCTL3 Register Description**

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved
5-0	LCDSx	RW	0h	<p>LCD segment line x enable.  This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>NOTE: Settings for LCDSx should be changed only while LCDON = 0.  0b = Multiplexed pins are port functions  1b = Pins are LCD functions</p>

**33.3.10 LCDCCPCTL Register**

LCD\_C Charge Pump Control Register

**Figure 33-21. LCDCCPCTL Register**

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

**Table 33-17. LCDCCPCTL Register Description**

Bit	Field	Type	Reset	Description
15	LCDCPCLKSYNC	RW	0h	LCD charge pump clock synchronization (device specific). The charge pump clock is synchronized to a device specific clock (device-specific) when the respective clock source is enabled and does not indicate a fault via its fault signal - if available. 0b = Synchronization disabled 1b = Synchronization enabled
14-8	Reserved	R	0h	Reserved
7-0	LCDCPDISx	RW	0h	LCD charge pump disable (number of implemented bits and connected function is device-specific) 0b = Connected function cannot disable charge pump 1b = Connected function can disable charge pump

**33.3.11 LCDCIV Register**

LCD\_C Interrupt Vector Register

**Figure 33-22. LCDCIV Register**

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

**Table 33-18. LCDCIV Register Description**

Bit	Field	Type	Reset	Description
15-0	LCDCIVx	R	0h	LCD_C interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: No capacitor connected; Interrupt Flag: LCDNOCAPIFG; Interrupt Priority: Highest 04h = Interrupt Source: Blink, segments off; Interrupt Flag: LCDBLKOFFIFG 06h = Interrupt Source: Blink, segments on; Interrupt Flag: LCDBLKONIFG 08h = Interrupt Source: Frame interrupt; Interrupt Flag: LCDFRMIFG; Interrupt Priority: Lowest

## ***Universal Serial Communication Interface – UART Mode***

---

---

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode.

Topic	Page
34.1 Universal Serial Communication Interface (USCI) Overview .....	894
34.2 USCI Introduction – UART Mode .....	895
34.3 USCI Operation – UART Mode .....	897
34.4 USCI_A UART Mode Registers .....	913

### 34.1 Universal Serial Communication Interface (USCI) Overview

The USCI modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI\_A is different from USCI\_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI\_A modules, they are named USCI\_A0 and USCI\_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on which devices.

USCI\_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud-rate detection for LIN communications
- SPI mode

USCI\_Bx modules support:

- I<sup>2</sup>C mode
- SPI mode

### 34.2 USCI Introduction – UART Mode

In asynchronous mode, the USCI\_Ax modules connect the device to an external system via two external pins, UCAxRXD and UCAxTXD. UART mode is selected when the UCSYNC bit is cleared.

UART mode features include:

- 7- or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first or MSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto wake up from LPMx modes (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 and transmit

Figure 34-1 shows the USCI\_Ax when configured for UART mode.

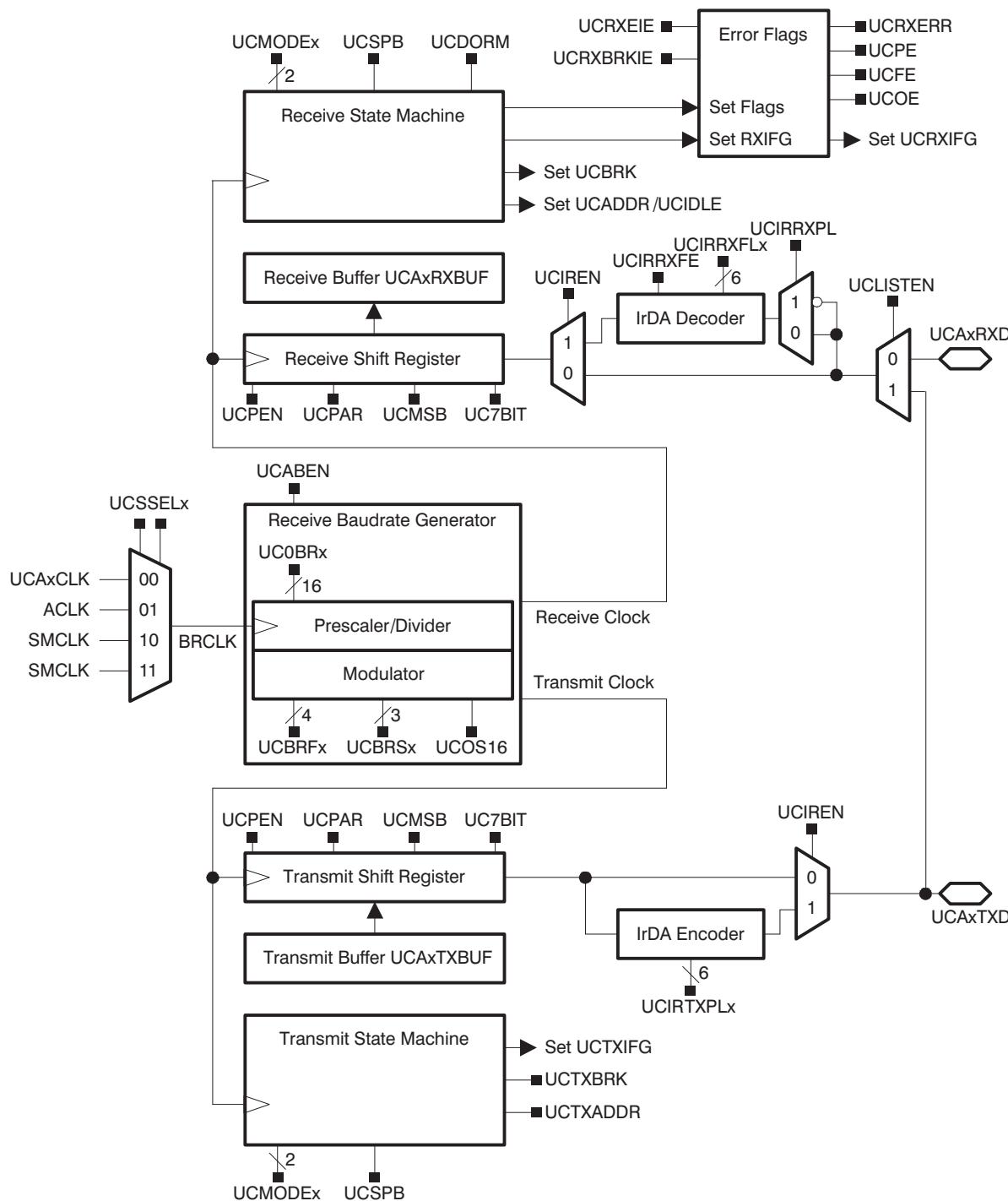


Figure 34-1. USCI\_Ax Block Diagram – UART Mode (UCSYNC = 0)

### 34.3 USCI Operation – UART Mode

In UART mode, the USCI transmits and receives characters at a bit rate asynchronous to another device. Timing for each character is based on the selected baud rate of the USCI. The transmit and receive functions use the same baud-rate frequency.

#### 34.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. When set, the UCSWRST bit resets the UCRXIE, UCTXIE, UCRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE, and UCBTOE bits, and sets the UCTXIFG bit. Clearing UCSWRST releases the USCI for operation.

To avoid unpredictable behavior, configure or reconfigure the USCI\_A module only when UCSWRST is set.

---

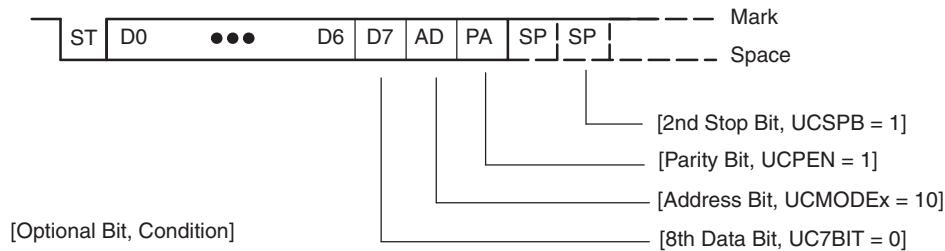
**NOTE: Initializing or reconfiguring the USCI module**

The recommended USCI initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST, &UCAxCTL1`).
  2. Initialize all USCI registers with UCSWRST = 1 (including UCAxCTL1).
  3. Configure ports.
  4. Clear UCSWRST via software (`BIC.B #UCSWRST, &UCAxCTL1`).
  5. Enable interrupts (optional) via UCRXIE and/or UCTXIE.
- 

#### 34.3.2 Character Format

The UART character format (see [Figure 34-2](#)) consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first. LSB first is typically required for UART communication.



**Figure 34-2. Character Format**

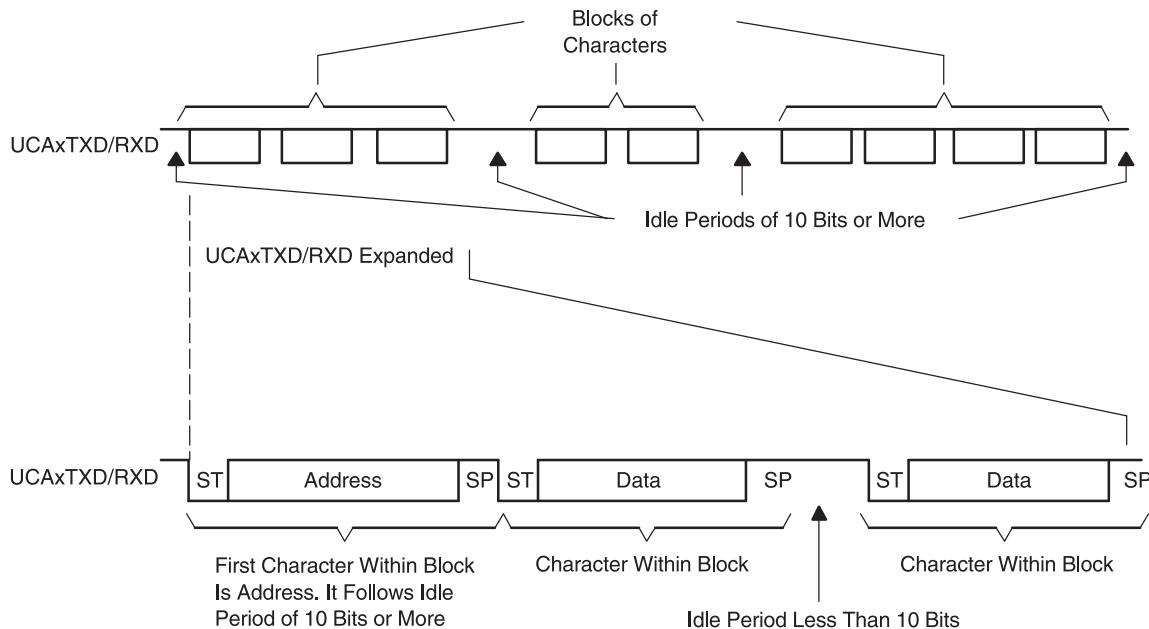
#### 34.3.3 Asynchronous Communication Format

When two devices communicate asynchronously, no multiprocessor format is required for the protocol. When three or more devices communicate, the USCI supports the idle-line and address-bit multiprocessor communication formats.

##### 34.3.3.1 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 34-3](#)). An idle receive line is detected when ten or more continuous ones (marks) are received after the one or two stop bits of a character. The baud-rate generator is switched off after reception of an idle line until the next start edge is detected. When an idle line is detected, the UCIDLE bit is set.

The first character received after an idle period is an address character. The UCIDLE bit is used as an address tag for each block of characters. In idle-line multiprocessor format, this bit is set when a received character is an address.



**Figure 34-3. Idle-Line Format**

The UCDORM bit is used to control data reception in the idle-line multiprocessor format. When UCDORM = 1, all non-address characters are assembled but not transferred into the UCAxRXBUF, and interrupts are not generated. When an address character is received, the character is transferred into UCAxRXBUF, UCRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and an address character is received but has a framing error or parity error, the character is not transferred into UCAxRXBUF and UCRXIFG is not set.

If an address is received, user software can validate the address and must reset UCDORM to continue receiving data. If UCDORM remains set, only address characters are received. When UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception completed. The UCDORM bit is not modified by the USCI hardware automatically.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the USCI to generate address character identifiers on UCAxTXD. The double-buffered UCTXADDR flag indicates if the next character loaded into UCAxTXBUF is preceded by an idle line of 11 bits. UCTXADDR is automatically cleared when the start bit is generated.

### Transmitting an Idle Frame

The following procedure sends out an idle frame to indicate an address character followed by associated data:

1. Set UCTXADDR, then write the address character to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCTXIFG = 1).

This generates an idle period of exactly 11 bits followed by the address character. UCTXADDR is reset automatically when the address character is transferred from UCAxTXBUF into the shift register.

2. Write desired data characters to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCTXIFG = 1).

The data written to UCAxTXBUF is transferred to the shift register and transmitted as soon as the shift register is ready for new data.

The idle-line time must not be exceeded between address and data transmission or between data transmissions. Otherwise, the transmitted data is misinterpreted as an address.

### 34.3.3.2 Address-Bit Multiprocessor Format

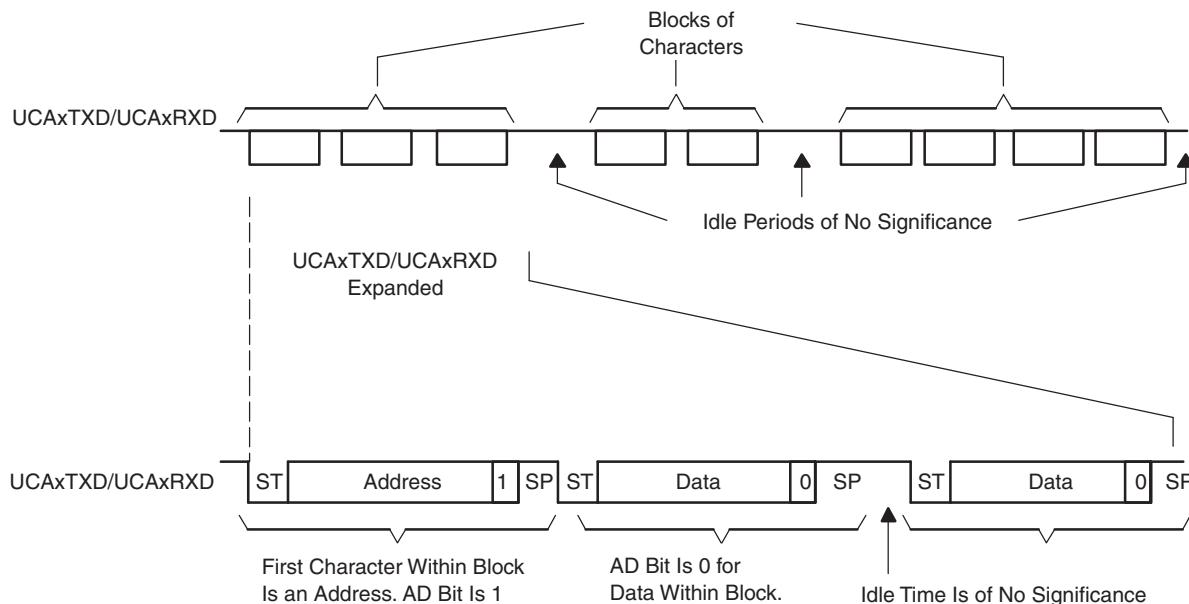
When UCMODEx = 10, the address-bit multiprocessor format is selected. Each processed character contains an extra bit used as an address indicator (see [Figure 34-4](#)). The first character in a block of characters carries a set address bit that indicates that the character is an address. The USCI UCADDR bit is set when a received character has its address bit set and is transferred to UCAXRXBUF.

The UCDORM bit is used to control data reception in the address-bit multiprocessor format. When UCDORM is set, data characters with address bit = 0 are assembled by the receiver but are not transferred to UCAXRXBUF and no interrupts are generated. When a character containing a set address bit is received, the character is transferred into UCAXRXBUF, UCRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and a character containing a set address bit is received but has a framing error or parity error, the character is not transferred into UCAXRXBUF and UCRXIFG is not set.

If an address is received, user software can validate the address and must reset UCDORM to continue receiving data. If UCDORM remains set, only address characters with address bit = 1 are received. The UCDORM bit is not modified by the USCI hardware automatically.

When UCDORM = 0, all received characters set the receive interrupt flag UCRXIFG. If UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception is completed.

For address transmission in address-bit multiprocessor mode, the address bit of a character is controlled by the UCTXADDR bit. The value of the UCTXADDR bit is loaded into the address bit of the character transferred from UCAXTXBUF to the transmit shift register. UCTXADDR is automatically cleared when the start bit is generated.



**Figure 34-4. Address-Bit Multiprocessor Format**

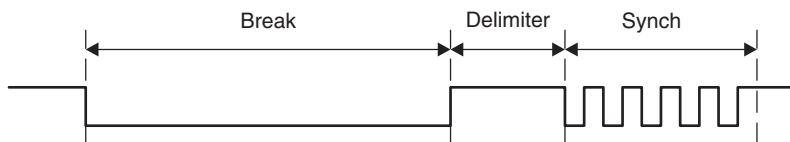
### Break Reception and Generation

When UCMODEx = 00, 01, or 10, the receiver detects a break when all data, parity, and stop bits are low, regardless of the parity, address mode, or other character settings. When a break is detected, the UCBRK bit is set. If the break interrupt enable bit (UCBRKIE) is set, the receive interrupt flag UCRXIFG is also set. In this case, the value in UCAXRXBUF is 0h, because all data bits were zero.

To transmit a break, set the UCTXBRK bit, then write 0h to UCAXTXBUF. UCAXTXBUF must be ready for new data (UCTXIFG = 1). This generates a break with all bits low. UCTXBRK is automatically cleared when the start bit is generated.

### 34.3.4 Automatic Baud-Rate Detection

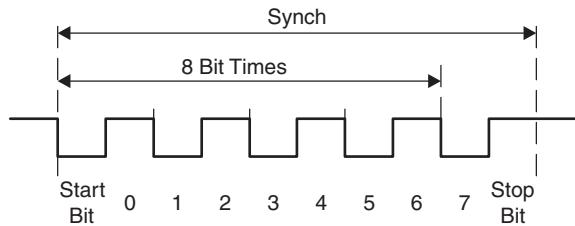
When UCMODE<sub>x</sub> = 11, UART mode with automatic baud-rate detection is selected. For automatic baud-rate detection, a data frame is preceded by a synchronization sequence that consists of a break and a synch field. A break is detected when 11 or more continuous zeros (spaces) are received. If the length of the break exceeds 21 bit times the break timeout error flag UCBTOE is set. The USCI can not transmit data while receiving the break/sync field. The synch field follows the break as shown in [Figure 34-5](#).



**Figure 34-5. 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 34-6](#)). The synchronization is based on the time measurement between the first falling edge and the last falling edge of the pattern. The transmit baud-rate generator is used for the measurement if automatic baud-rate detection is enabled by setting UCABDEN. Otherwise, the pattern is received but not measured. The result of the measurement is transferred into the baud-rate control registers (UCAxBR0, UCAxBR1, and UCAxMCTL). If the length of the synch field exceeds the measurable time, the synch timeout error flag UCSTOE is set.



**Figure 34-6. Auto Baud-Rate Detection – Synch Field**

The UCDORM bit is used to control data reception in this mode. When UCDORM is set, all characters are received but not transferred into the UCAxRXBUF, and interrupts are not generated. When a break/synch field is detected, the UCBRK flag is set. The character following the break/synch field is transferred into UCAxRXBUF and the UCRXIFG interrupt flag is set. Any applicable error flag is also set. If the UCBRKIE bit is set, reception of the break/synch sets the UCRXIFG. The UCBRK bit is reset by user software or by reading the receive buffer UCAxRXBUF.

When a break/synch field is received, user software must reset UCDORM to continue receiving data. If UCDORM remains set, only the character after the next reception of a break/synch field is received. The UCDORM bit is not modified by the USCI hardware automatically.

When UCDORM = 0, all received characters set the receive interrupt flag UCRXIFG. If UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception is complete.

The counter used to detect the baud rate is limited to 07FFFh (32767) counts. This means the minimum baud rate detectable is 488 baud in oversampling mode and 30 baud in low-frequency mode.

The automatic baud-rate detection mode can be used in a full-duplex communication system with some restrictions. The USCI can not transmit data while receiving the break/sync field and, if a 0h byte with framing error is received, any data transmitted during this time gets corrupted. The latter case can be discovered by checking the received data and the UCFE bit.

### 34.3.4.1 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.

### 34.3.5 IrDA Encoding and Decoding

When UCIREN is set, the IrDA encoder and decoder are enabled and provide hardware bit shaping for IrDA communication.

#### 34.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART (see Figure 34-7). The pulse duration is defined by UCIRTXPLx bits specifying the number of one-half clock periods of the clock selected by UCIRTXCLK.

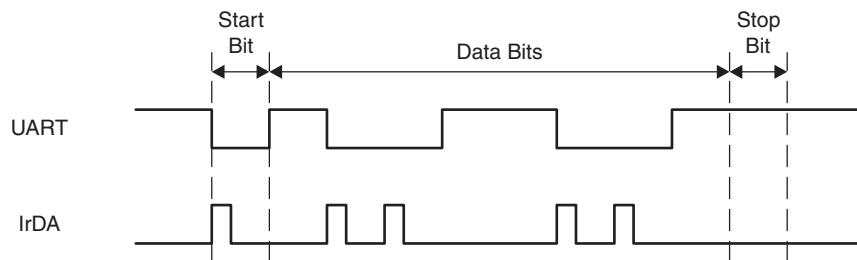


Figure 34-7. UART vs IrDA Data Format

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:

$$\text{UCIRTXPLx} = t_{PULSE} \times 2 \times f_{BRCLK} - 1$$

When UCIRTXCLK = 0 ,the prescaler UCBRx must to be set to a value greater or equal to 5.

#### 34.3.5.2 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:

$$\text{UCIRRXFLx} = (t_{PULSE} - t_{WAKE}) \times 2 \times f_{BRCLK} - 4$$

Where:

$t_{PULSE}$  = Minimum receive pulse width

$t_{WAKE}$  = Wake time from any low-power mode. Zero when the device is in active mode.

### 34.3.6 Automatic Error Detection

Glitch suppression prevents the USCI from being accidentally started. Any pulse on UCAXRXD shorter than the deglitch time  $t_g$  (approximately 150 ns) is ignored (see the device-specific data sheet for parameters).

When a low period on UCAXRXD exceeds  $t_g$ , a majority vote is taken for the start bit. If the majority vote fails to detect a valid start bit, the USCI halts character reception and waits for the next low period on UCAXRXD. The majority vote is also used for each bit in a character to prevent bit errors.

The USCI module automatically detects framing errors, parity errors, overrun errors, and break conditions when receiving characters. The bits UCFE, UCPE, UCOE, and UCBRK are set when their respective condition is detected. When the error flags UCFE, UCPE, or UCOE are set, UCRXERR is also set. The error conditions are described in [Table 34-1](#).

**Table 34-1. Receive Error Conditions**

Error Condition	Error Flag	Description
Framing error	UCFE	A framing error occurs when a low stop bit is detected. When two stop bits are used, both stop bits are checked for framing error. When a framing error is detected, the UCFE bit is set.
Parity error	UCPE	A parity error is a mismatch between the number of 1s in a character and the value of the parity bit. When an address bit is included in the character, it is included in the parity calculation. When a parity error is detected, the UCPE bit is set.
Receive overrun	UCOE	An overrun error occurs when a character is loaded into UCAXRXBUF before the prior character has been read. When an overrun occurs, the UCOE bit is set.
Break condition	UCBRK	When not using automatic baud-rate detection, a break is detected when all data, parity, and stop bits are low. When a break condition is detected, the UCBRK bit is set. A break condition can also set the interrupt flag UCRXIFG if the break interrupt enable UCBRKIE bit is set.

When UCRXEIE = 0 and a framing error or parity error is detected, no character is received into UCAXRXBUF. When UCRXEIE = 1, characters are received into UCAXRXBUF and any applicable error bit is set.

When any of the UCFE, UCPE, UCOE, UCBRK, or UCRXERR bit is set, the bit remains set until user software resets it or UCAXRXBUF is read. UCOE must be reset by reading UCAXRXBUF. Otherwise, it does not function properly. To detect overflows reliably the following flow is recommended. After a character was received and UCAXRXIFG is set, first read UCAXSTAT to check the error flags including the overflow flag UCOE. Read UCAXRXBUF next. This clears all error flags except UCOE, if UCAXRXBUF was overwritten between the read access to UCAXSTAT and to UCAXRXBUF. Therefore, the UCOE flag should be checked after reading UCAXRXBUF to detect this condition. Note that, in this case, the UCRXERR flag is not set.

### 34.3.7 USCI Receive Enable

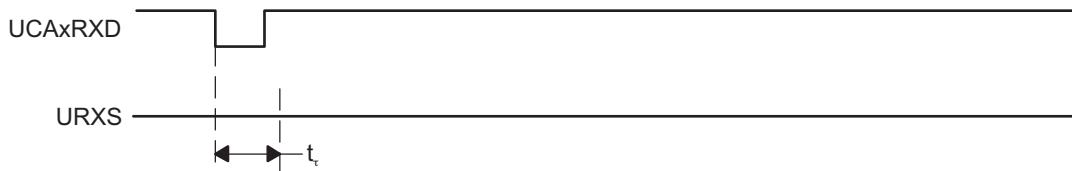
The USCI module is enabled by clearing the UCSWRST bit and the receiver is ready and in an idle state. The receive baud rate generator is in a ready state but is not clocked nor producing any clocks.

The falling edge of the start bit enables the baud rate generator and the UART state machine checks for a valid start bit. If no valid start bit is detected the UART state machine returns to its idle state and the baud rate generator is turned off again. If a valid start bit is detected, a character is received.

When the idle-line multiprocessor mode is selected with UCMODE<sub>x</sub> = 01 the UART state machine checks for an idle line after receiving a character. If a start bit is detected another character is received. Otherwise the UCIDLE flag is set after 10 ones are received and the UART state machine returns to its idle state and the baud rate generator is turned off.

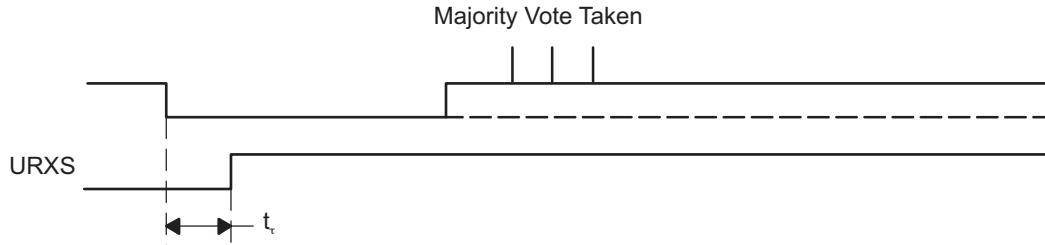
#### 34.3.7.1 Receive Data Glitch Suppression

Glitch suppression prevents the USCI from being accidentally started. Any glitch on UCAxRXD shorter than the deglitch time  $t_g$  (approximately 150 ns) is ignored by the USCI, and further action is initiated as shown in [Figure 34-8](#) (see the device-specific data sheet for parameters).



**Figure 34-8. Glitch Suppression, USCI Receive Not Started**

When a glitch is longer than  $t_g$ , or a valid start bit occurs on UCAxRXD, the USCI receive operation is started and a majority vote is taken (see [Figure 34-9](#)). If the majority vote fails to detect a start bit, the USCI halts character reception.



**Figure 34-9. Glitch Suppression, USCI Activated**

### 34.3.8 USCI Transmit Enable

The USCI module is enabled by clearing the UCSWRST bit and the transmitter is ready and in an idle state. The transmit baud-rate generator is ready but is not clocked nor producing any clocks.

A transmission is initiated by writing data to UCAxTXBUF. When this occurs, the baud-rate generator is enabled, and the data in UCAxTXBUF is moved to the transmit shift register on the next BITCLK after the transmit shift register is empty. UCTXIFG is set when new data can be written into UCAxTXBUF.

Transmission continues as long as new data is available in UCAxTXBUF at the end of the previous byte transmission. If new data is not in UCAxTXBUF when the previous byte has transmitted, the transmitter returns to its idle state and the baud-rate generator is turned off.

### 34.3.9 UART Baud-Rate Generation

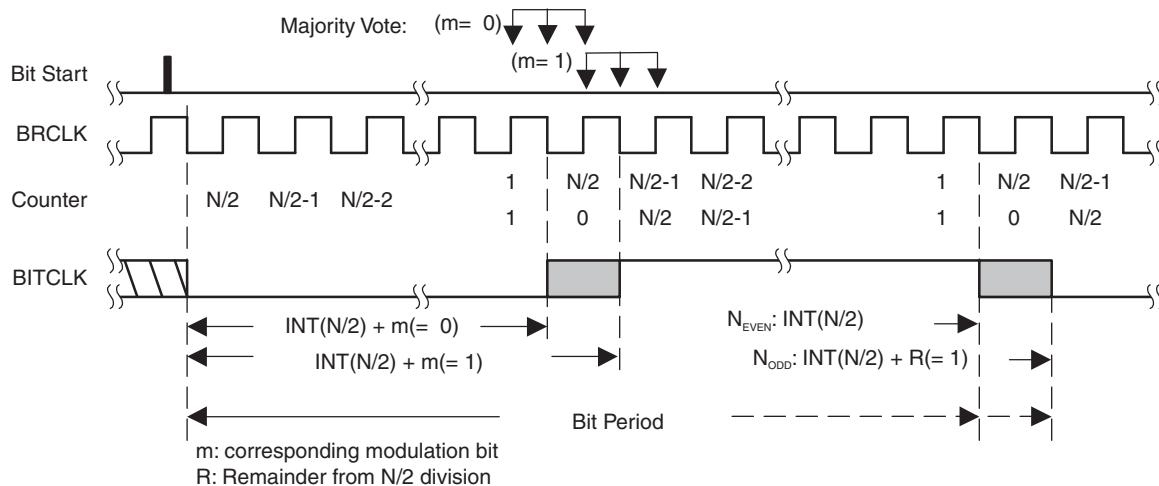
The USCI baud-rate generator is capable of producing standard baud rates from nonstandard source frequencies. It provides two modes of operation selected by the UCOS16 bit. The baud-rate is generated using the BRCLK that can be sourced by the external clock UCAXCLK, or the internal clocks ACLK or SMCLK depending on the UCSELx settings.

#### 34.3.9.1 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 USCI baud rate is one-third the UART source clock frequency BRCLK.

Timing for each bit is shown in [Figure 34-10](#). 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 34-10. BITCLK Baud-Rate Timing With UCOS16 = 0**

Modulation is based on the UCBRSx setting (see [Table 34-2](#)). A 1 in the table indicates that  $m = 1$  and the corresponding BITCLK period is one BRCLK period longer than a BITCLK period with  $m = 0$ . The modulation wraps around after eight bits but restarts with each new start bit.

**Table 34-2. BITCLK Modulation Pattern**

UCBRSx	Bit 0 (Start Bit)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0
3	0	1	0	1	0	1	0	0
4	0	1	0	1	0	1	0	1
5	0	1	1	1	0	1	0	1
6	0	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1

### 34.3.9.2 Oversampling Baud-Rate Generation

The oversampling mode is selected when UCOS16 = 1. This mode supports sampling a UART bit stream with higher input clock frequencies. This results in majority votes that are always 1/16 of a bit clock period apart. This mode also easily supports IrDA pulses with a 3/16 bit time when the IrDA encoder and decoder are enabled.

This mode uses one prescaler and one modulator to generate the BITCLK16 clock that is 16 times faster than the BITCLK. An additional divider and modulator stage generates BITCLK from BITCLK16. This combination supports fractional divisions of both BITCLK16 and BITCLK for baud-rate generation. In this mode, the maximum USCI baud rate is 1/16 the UART source clock frequency BRCLK. When UCBRx is set to 0 or 1, the first prescaler and modulator stage is bypassed and BRCLK is equal to BITCLK16 – in this case, no modulation for the BITCLK16 is possible and, thus, the UCBRFx bits are ignored.

Modulation for BITCLK16 is based on the UCBRFx setting (see [Table 34-3](#)). A 1 in the table indicates that the corresponding BITCLK16 period is one BRCLK period longer than the periods m = 0. The modulation restarts with each new bit timing.

Modulation for BITCLK is based on the UCBRSx setting (see [Table 34-2](#)) as previously described.

**Table 34-3. BITCLK16 Modulation Pattern**

UCBRFx	No. of BITCLK16 Clocks After Last Falling BITCLK Edge															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
03h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
04h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
05h	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
06h	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
07h	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
08h	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
09h	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1
0Ah	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
0Bh	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1
0Ch	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
0Dh	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
0Eh	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0Fh	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

### 34.3.10 Setting a Baud Rate

For a given BRCLK clock source, the baud rate used determines the required division factor N:

$$N = f_{BRCLK}/\text{Baudrate}$$

The division factor N is often a noninteger value, thus, at least one divider and one modulator stage is used to meet the factor as closely as possible.

If N is equal or greater than 16, the oversampling baud-rate generation mode can be chosen by setting UCOS16.

#### 34.3.10.1 Low-Frequency Baud-Rate Mode Setting

In low-frequency mode, the integer portion of the divisor is realized by the prescaler:

$$UCBRx = \text{INT}(N)$$

and the fractional portion is realized by the modulator with the following nominal formula:

$$UCBRSx = \text{round}[(N - \text{INT}(N)) \times 8]$$

Incrementing or decrementing the UCBRSx setting by one count may give a lower maximum bit error for any given bit. To determine if this is the case, a detailed error calculation must be performed for each bit for each UCBRSx setting.

#### 34.3.10.2 Oversampling Baud-Rate Mode Setting

In the oversampling mode, the prescaler is set to:

$$UCBRx = \text{INT}(N/16)$$

and the first stage modulator is set to:

$$UCBRFx = \text{round}[(N/16) - \text{INT}(N/16)] \times 16$$

When greater accuracy is required, the UCBRSx modulator can also be implemented with values from 0 to 7. To find the setting that gives the lowest maximum bit error rate for any given bit, a detailed error calculation must be performed for all settings of UCBRSx from 0 to 7 with the initial UCBRFx setting, and with the UCBRFx setting incremented and decremented by one.

### 34.3.11 Transmit Bit Timing

The timing for each character is the sum of the individual bit timings. Using the modulation features of the baud-rate generator reduces the cumulative bit error. The individual bit error can be calculated using the following steps.

#### 34.3.11.1 Low-Frequency Baud-Rate Mode Bit Timing

In low-frequency mode, calculate the length of bit i  $T_{bit,TX}[i]$  based on the UCBRx and UCBRSx settings:

$$T_{bit,TX}[i] = (1/f_{BRCLK})(UCBRx + m_{UCBRSx}[i])$$

Where:

$$m_{UCBRSx}[i] = \text{Modulation of bit } i \text{ from } \text{Table 34-2}$$

#### 34.3.11.2 Oversampling Baud-Rate Mode Bit Timing

In oversampling baud-rate mode, calculate the length of bit i  $T_{bit,TX}[i]$  based on the baud-rate generator UCBRx, UCBRFx and UCBRSx settings:

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}} \left( (16 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j] \right)$$

Where:

$$\sum_{j=0}^{15} m_{UCBRFx}[j] = \text{Sum of ones from the corresponding row in } \text{Table 34-3}$$

$$m_{UCBRSx}[i] = \text{Modulation of bit } i \text{ from } \text{Table 34-2}$$

This results in an end-of-bit time  $t_{bit,TX}[i]$  equal to the sum of all previous and the current bit times:

$$T_{bit,TX}[i] = \sum_{j=0}^i T_{bit,TX}[j]$$

To calculate bit error, this time is compared to the ideal bit time  $t_{bit,ideal,TX}[i]$ :

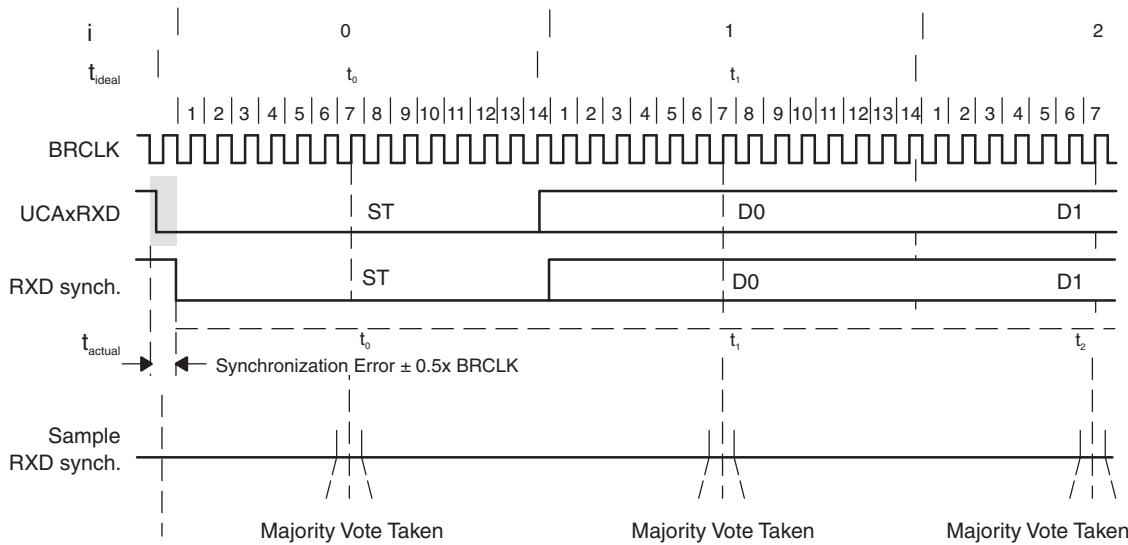
$$t_{bit,ideal,TX}[i] = (1/\text{Baudrate})(i + 1)$$

This results in an error normalized to one ideal bit time (1/baudrate):

$$\text{Error}_{TX}[i] = (t_{bit,TX}[i] - t_{bit,ideal,TX}[i]) \times \text{Baudrate} \times 100\%$$

### 34.3.12 Receive Bit Timing

Receive timing error consists of two error sources. The first is the bit-to-bit timing error similar to the transmit bit timing error. The second is the error between a start edge occurring and the start edge being accepted by the USCI module. [Figure 34-11](#) shows the asynchronous timing errors between data on the UCAxRXD pin and the internal baud-rate clock. This results in an additional synchronization error. The synchronization error  $t_{SYNC}$  is between  $-0.5$  BRCLKs and  $+0.5$  RCLKs, independent of the selected baud-rate generation mode.



**Figure 34-11. Receive Error**

The ideal sampling time  $t_{bit,ideal,RX}[i]$  is in the middle of a bit period:

$$t_{bit,ideal,RX}[i] = (1/\text{Baudrate})(i + 0.5)$$

The real sampling time,  $t_{bit,RX}[i]$ , is equal to the sum of all previous bits according to the formulas shown in the transmit timing section, plus one-half BITCLK for the current bit  $i$ , plus the synchronization error  $t_{SYNC}$ .

This results in the following  $t_{bit,RX}[i]$  for the low-frequency baud-rate mode:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left( \text{INT}(\frac{1}{2}\text{UCBRx}) + m_{UCBRSx}[i] \right)$$

Where:

$$T_{bit,RX}[i] = (1/f_{BRCLK})(UCBRx + m_{UCBRSx}[i])$$

$m_{UCBRSx}[i]$  = Modulation of bit  $i$  from [Table 34-2](#)

For the oversampling baud-rate mode, the sampling time  $t_{bit,RX}[i]$  of bit i is calculated by:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left( (8 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j] \right)$$

Where:

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}} \left( (16 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j] \right)$$

$\sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j]$  = Sum of ones from columns 0 to  $(7 + m_{UCBRSx}[i])$  from the corresponding row in [Table 34-3](#).

$m_{UCBRSx}[i]$  = Modulation of bit i from [Table 34-2](#)

This results in an error normalized to one ideal bit time (1/baudrate) according to the following formula:

$$\text{Error}_{RX}[i] = (t_{bit,RX}[i] - t_{bit,ideal,RX}[i]) \times \text{Baudrate} \times 100\%$$

### 34.3.13 Typical Baud Rates and Errors

Standard baud-rate data for UCBRx, UCBRSx, and UCBRFx are listed in [Table 34-4](#) and [Table 34-5](#) for a 32,768-Hz crystal sourcing ACLK and typical SMCLK frequencies. Please ensure that the selected BRCLK frequency does not exceed the device specific maximum USCI input frequency (see the device-specific data sheet).

The receive error is the accumulated time versus the ideal scanning time in the middle of each bit. The worst-case error is given for the reception of an 8-bit character with parity and one stop bit including synchronization error.

The transmit error is the accumulated timing error versus the ideal time of the bit period. The worst-case error is given for the transmission of an 8-bit character with parity and stop bit.

**Table 34-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0**

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)	Maximum RX Error (%)
32,768	1200	27	2	0	-2.8	1.4
32,768	2400	13	6	0	-4.8	6.0
32,768	4800	6	7	0	-12.1	5.7
32,768	9600	3	3	0	-21.1	15.2
1,000,000	9600	104	1	0	-0.5	0.6
1,000,000	19200	52	0	0	-1.8	0
1,000,000	38400	26	0	0	-1.8	0
1,000,000	57600	17	3	0	-2.1	4.8
1,000,000	115200	8	6	0	-7.8	6.4
1,048,576	9600	109	2	0	-0.2	0.7
1,048,576	19200	54	5	0	-1.1	1.0
1,048,576	38400	27	2	0	-2.8	1.4
1,048,576	57600	18	1	0	-4.6	3.3
1,048,576	115200	9	1	0	-1.1	10.7
4,000,000	9600	416	6	0	-0.2	0.2
4,000,000	19200	208	3	0	-0.2	0.5
4,000,000	38400	104	1	0	-0.5	0.6
4,000,000	57600	69	4	0	-0.6	0.8
4,000,000	115200	34	6	0	-2.1	0.6
4,000,000	230400	17	3	0	-2.1	4.8
4,194,304	9600	436	7	0	-0.3	0

**Table 34-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 (continued)**

<b>BRCLK Frequency (Hz)</b>	<b>Baud Rate (baud)</b>	<b>UCBRx</b>	<b>UCBRSx</b>	<b>UCBRFx</b>	<b>Maximum TX Error (%)</b>	<b>Maximum RX Error (%)</b>
4,194,304	19200	218	4	0	-0.2	0.2
4,194,304	57600	72	7	0	-1.1	0.6
4,194,304	115200	36	3	0	-1.9	1.5
8,000,000	9600	833	2	0	-0.1	0
8,000,000	19200	416	6	0	-0.2	0.2
8,000,000	38400	208	3	0	-0.2	0.5
8,000,000	57600	138	7	0	-0.7	0
8,000,000	115200	69	4	0	-0.6	0.8
8,000,000	230400	34	6	0	-2.1	0.6
8,000,000	460800	17	3	0	-2.1	4.8
8,388,608	9600	873	7	0	-0.1	0.06
8,388,608	19200	436	7	0	-0.3	0
8,388,608	57600	145	5	0	-0.5	0.3
8,388,608	115200	72	7	0	-1.1	0.6
12,000,000	9600	1250	0	0	0	0
12,000,000	19200	625	0	0	0	-0.2
12,000,000	38400	312	4	0	-0.2	0
12,000,000	57600	208	2	0	-0.5	0.2
12,000,000	115200	104	1	0	-0.5	0.6
12,000,000	230400	52	0	0	-1.8	0
12,000,000	460800	26	0	0	-1.8	0
16,000,000	9600	1666	6	0	-0.05	0.05
16,000,000	19200	833	2	0	-0.1	0.05
16,000,000	38400	416	6	0	-0.2	0.2
16,000,000	57600	277	7	0	-0.3	0.3
16,000,000	115200	138	7	0	-0.7	0
16,000,000	230400	69	4	0	-0.6	0.8
16,000,000	460800	34	6	0	-2.1	0.6
16,777,216	9600	1747	5	0	-0.04	0.03
16,777,216	19200	873	7	0	-0.09	0.06
16,777,216	57600	291	2	0	-0.2	0.2
16,777,216	115200	145	5	0	-0.5	0.3
20,000,000	9600	2083	2	0	-0.05	0.02
20,000,000	19200	1041	6	0	-0.06	0.06
20,000,000	38400	520	7	0	-0.2	0.06
20,000,000	57600	347	2	0	-0.06	0.2
20,000,000	115200	173	5	0	-0.4	0.3
20,000,000	230400	86	7	0	-1.0	0.6
20,000,000	460800	43	3	0	-1.4	1.3

**Table 34-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1**

<b>BRCLK Frequency (Hz)</b>	<b>Baud Rate (baud)</b>	<b>UCBRx</b>	<b>UCBRSx</b>	<b>UCBRFx</b>	<b>Maximum TX Error (%)</b>	<b>Maximum RX Error (%)</b>
1,000,000	9600	6	0	8	-1.8	0
1,000,000	19200	3	0	4	-1.8	0
1,048,576	9600	6	0	13	-2.3	0
1,048,576	19200	3	1	6	-4.6	3.2
4,000,000	9600	26	0	1	0	0.9
4,000,000	19200	13	0	0	-1.8	0
4,000,000	38400	6	0	8	-1.8	0
4,000,000	57600	4	5	3	-3.5	3.2
4,000,000	115200	2	3	2	-2.1	4.8
4,194,304	9600	27	0	5	0	0.2
4,194,304	19200	13	0	10	-2.3	0
4,194,304	57600	4	4	7	-2.5	2.5
4,194,304	115200	2	6	3	-3.9	2.0
8,000,000	9600	52	0	1	-0.4	0
8,000,000	19200	26	0	1	0	0.9
8,000,000	38400	13	0	0	-1.8	0
8,000,000	57600	8	0	11	0	0.88
8,000,000	115200	4	5	3	-3.5	3.2
8,000,000	230400	2	3	2	-2.1	4.8
8,388,608	9600	54	0	10	0	0.2
8,388,608	19200	27	0	5	0	0.2
8,388,608	57600	9	0	2	0	2.8
8,388,608	115200	4	4	7	-2.5	2.5
12,000,000	9600	78	0	2	0	0
12,000,000	19200	39	0	1	0	0
12,000,000	38400	19	0	8	-1.8	0
12,000,000	57600	13	0	0	-1.8	0
12,000,000	115200	6	0	8	-1.8	0
12,000,000	230400	3	0	4	-1.8	0
16,000,000	9600	104	0	3	0	0.2
16,000,000	19200	52	0	1	-0.4	0
16,000,000	38400	26	0	1	0	0.9
16,000,000	57600	17	0	6	0	0.9
16,000,000	115200	8	0	11	0	0.9
16,000,000	230400	4	5	3	-3.5	3.2
16,000,000	460800	2	3	2	-2.1	4.8
16,777,216	9600	109	0	4	0	0.2
16,777,216	19200	54	0	10	0	0.2
16,777,216	57600	18	0	3	-1.0	0
16,777,216	115200	9	0	2	0	2.8
20,000,000	9600	130	0	3	-0.2	0
20,000,000	19200	65	0	2	0	0.4
20,000,000	38400	32	0	9	0	0.4
20,000,000	57600	21	0	11	-0.7	0
20,000,000	115200	10	0	14	0	2.5
20,000,000	230400	5	0	7	0	2.5
						3.5

**Table 34-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1 (continued)**

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)	Maximum RX Error (%)
20,000,000	460800	2	6	10	-3.2	1.8

### 34.3.14 Using the USCI Module in UART Mode With Low-Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

### 34.3.15 USCI Interrupts

The USCI has only one interrupt vector that is shared for transmission and for reception. USCI\_Ax and USC\_Bx do not share the same interrupt vector.

#### 34.3.15.1 USCI Transmit Interrupt Operation

The UCTXIFG interrupt flag is set by the transmitter to indicate that UCAxTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCAxTXBUF.

UCTXIFG is set after a PUC or when UCSWRST = 1. UCTXIE is reset after a PUC or when UCSWRST = 1.

#### 34.3.15.2 USCI Receive Interrupt Operation

The UCRXIFG interrupt flag is set each time a character is received and loaded into UCAxRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCRXIFG is automatically reset when UCAxRXBUF is read.

Additional interrupt control features include:

- When UCAxRXIE = 0, erroneous characters do not set UCRXIFG.
- When UCDORM = 1, nonaddress characters do not set UCRXIFG in multiprocessor modes. In plain UART mode, no characters are set UCRXIFG.
- When UCBRKIE = 1, a break condition sets the UCBRK bit and the UCRXIFG flag.

#### 34.3.15.3 UCAxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCAxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCAxIV register that can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCAxIV value.

Any access, read or write, of the UCAxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

**UCAxIV Software Example**

The following software example shows the recommended use of UCAxIV. The UCAxIV value is added to the PC to automatically jump to the appropriate routine. The following example is given for USCI\_A0.

```
USCI_UART_ISR
    ADD    &UCA0IV, PC      ; Add offset to jump table
    RETI
    JMP    RXIFG_ISR      ; Vector 0: No interrupt
    TXIFG_ISR             ; Vector 2: RXIFG
    ...
    RETI                  ; Task starts here
    RXIFG_ISR             ; Vector 4: TXIFG
    ...
    RETI                  ; Task starts here
    RETI                  ; Return
```

### 34.4 USCI\_A UART Mode Registers

The USCI registers applicable in UART mode listed in [Table 34-6](#). The base address can be found in the device-specific data sheet. The address offsets are listed in [Table 34-6](#).

**Table 34-6. USCI\_A UART Mode Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	USCI_Ax Control Word 0	Read/write	Word	0001h	
00h	UCAxCTL1	USCI_Ax Control 1	Read/write	Byte	01h	<a href="#">Section 34.4.2</a>
01h	UCAxCTL0	USCI_Ax Control 0	Read/write	Byte	00h	<a href="#">Section 34.4.1</a>
06h	UCAxBRW	USCI_Ax Baud Rate Control Word	Read/write	Word	0000h	
06h	UCAxBR0	USCI_Ax Baud Rate Control 0	Read/write	Byte	00h	<a href="#">Section 34.4.3</a>
07h	UCAxBR1	USCI_Ax Baud Rate Control 1	Read/write	Byte	00h	<a href="#">Section 34.4.4</a>
08h	UCAxMCTL	USCI_Ax Modulation Control	Read/write	Byte	00h	<a href="#">Section 34.4.5</a>
09h		Reserved - reads zero	Read	Byte	00h	
0Ah	UCAxSTAT	USCI_Ax Status	Read/write	Byte	00h	<a href="#">Section 34.4.6</a>
0Bh		Reserved - reads zero	Read	Byte	00h	
0Ch	UCAxRXBUF	USCI_Ax Receive Buffer	Read/write	Byte	00h	<a href="#">Section 34.4.7</a>
0Dh		Reserved - reads zero	Read	Byte	00h	
0Eh	UCAxTXBUF	USCI_Ax Transmit Buffer	Read/write	Byte	00h	<a href="#">Section 34.4.8</a>
0Fh		Reserved - reads zero	Read	Byte	00h	
10h	UCAxABCTL	USCI_Ax Auto Baud Rate Control	Read/write	Byte	00h	<a href="#">Section 34.4.11</a>
11h		Reserved - reads zero	Read	Byte	00h	
12h	UCAxIRCTL	USCI_Ax IrDA Control	Read/write	Word	0000h	
12h	UCAxIRTCTL	USCI_Ax IrDA Transmit Control	Read/write	Byte	00h	<a href="#">Section 34.4.9</a>
13h	UCAxIRRCTL	USCI_Ax IrDA Receive Control	Read/write	Byte	00h	<a href="#">Section 34.4.10</a>
1Ch	UCAxICTL	USCI_Ax Interrupt Control	Read/write	Word	0000h	
1Ch	UCAxIE	USCI_Ax Interrupt Enable	Read/write	Byte	00h	<a href="#">Section 34.4.12</a>
1Dh	UCAxIFG	USCI_Ax Interrupt Flag	Read/write	Byte	00h	<a href="#">Section 34.4.13</a>
1Eh	UCAxIV	USCI_Ax Interrupt Vector	Read	Word	0000h	<a href="#">Section 34.4.14</a>

**34.4.1 UCAXCTL0 Register**

USCI\_Ax Control Register 0

**Figure 34-12. UCAXCTL0 Register**

7	6	5	4	3	2	1	0
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx	UCSYNC	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

**Table 34-7. UCAXCTL0 Register Description**

Bit	Field	Type	Reset	Description
7	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.
6	UCPAR	RW	0h	Parity select. UCPAR is not used when parity is disabled. 0b = Odd parity 1b = Even parity
5	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
4	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
3	UCSPB	RW	0h	Stop bit select. Number of stop bits. 0b = One stop bit 1b = Two stop bits
2-1	UCMODEx	RW	0h	USCI 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
0	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode

### 34.4.2 UCAXCTL1 Register

USCI\_Ax Control Register 1

**Figure 34-13. UCAXCTL1 Register**

7	6	5	4	3	2	1	0
UCSSELx	UCRXIE	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 34-8. UCAXCTL1 Register Description**

Bit	Field	Type	Reset	Description
7-6	UCSSELx	RW	0h	USCI clock source select. These bits select the BRCLK source clock. 00b = UCAXCLK (external USCI clock) 01b = ACLK 10b = SMCLK 11b = SMCLK
5	UCRXIE	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.
3	UCDORM	RW	0h	Dormant. Puts USCI 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. USCI reset released for operation. 1b = Enabled. USCI logic held in reset state.

### 34.4.3 UCAXBR0 Register

USCI\_Ax Baud Rate Control Register 0

**Figure 34-14. UCAXBR0 Register**

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

Modify only when UCSWRST = 1.

**Table 34-9. UCAXBR0 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Low byte of clock prescaler setting of the baud-rate generator. The 16-bit value of (UCAXBR0 + UCAXBR1 × 256) forms the prescaler value UCBRx.

### 34.4.4 UCAXBR1 Register

USCI\_Ax Baud Rate Control Register 1

**Figure 34-15. UCAXBR1 Register**

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

Modify only when UCSWRST = 1.

**Table 34-10. UCAXBR1 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	High byte of clock prescaler setting of the baud-rate generator. The 16-bit value of (UCAXBR0 + UCAXBR1 × 256) forms the prescaler value UCBRx.

### 34.4.5 UCAXMCTL Register

USCI\_Ax Modulation Control Register

**Figure 34-16. UCAXMCTL Register**

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

Modify only when UCSWRST = 1.

**Table 34-11. UCAXMCTL Register Description**

Bit	Field	Type	Reset	Description
7-4	UCBRFx	RW	0h	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. <a href="#">Table 34-2</a> shows the modulation pattern.
3-1	UCBRSx	RW	0h	Second modulation stage select. These bits determine the modulation pattern for BITCLK. <a href="#">Table 34-2</a> shows the modulation pattern.
0	UCOS16	RW	0h	Oversampling mode enabled 0b = Disabled 1b = Enabled

### 34.4.6 UCAXSTAT Register

USCI\_Ax Status Register

**Figure 34-17. UCAXSTAT Register**

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 34-12. UCAXSTAT Register Description**

Bit	Field	Type	Reset	Description
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 UCPE = 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 error(s). When UCRXERR = 1, one or more error flags, UCFE, UCPE, or UCOE are 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. 0b = Received character is data. 1b = Received character is an address. UCIDLE: Idle line detected in idle-line multiprocessor mode. UCIDLE is cleared when UCAXRXBUF is read. 0b = No idle line detected 1b = Idle line detected
0	UCBUSY	R	0h	USCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = USCI inactive 1b = USCI transmitting or receiving

**34.4.7 UCAXRXBUF Register**

USCI\_Ax Receive Buffer Register

**Figure 34-18. UCAXRXBUF Register**

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

**Table 34-13. UCAXRXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCRXBUFx	R	undefined	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.

**34.4.8 UCAXTXBUF Register**

USCI\_Ax Transmit Buffer Register

**Figure 34-19. UCAXTXBUF Register**

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

**Table 34-14. UCAXTXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCTXBUFx	RW	undefined	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.

### 34.4.9 UCAXIRTCTL Register

USCI\_Ax IrDA Transmit Control Register

**Figure 34-20. UCAXIRTCTL Register**

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 34-15. UCAXIRTCTL Register Description**

Bit	Field	Type	Reset	Description
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 and decoder enable 0b = IrDA encoder and decoder disabled 1b = IrDA encoder and decoder enabled

### 34.4.10 UCAXIRRCTL Register

USCI\_Ax IrDA Receive Control Register

**Figure 34-21. UCAXIRRCTL Register**

7	6	5	4	3	2	1	0
UCIRRXFLx						UCIRRXPL	UCIRRXFE
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

**Table 34-16. UCAXIRRCTL Register Description**

Bit	Field	Type	Reset	Description
7-2	UCIRRXFLx	RW	0h	Receive filter length. The minimum pulse length for receive is given by: t(MIN) = (UCIRRXFLx + 4) / (2 × f(BRCLK))
1	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.
0	UCIRRXFE	RW	0h	IrDA receive filter enabled 0b = Receive filter disabled 1b = Receive filter enabled

**34.4.11 UCAXABCTL Register**

USCI\_Ax Auto Baud Rate Control Register

**Figure 34-22. UCAXABCTL Register**

7	6	5	4	3	2	1	0
Reserved		UCDELMx		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 34-17. UCAXABCTL Register Description**

Bit	Field	Type	Reset	Description
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	UCDELMx	RW	0h	Break and 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. Always reads as 0.
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.

### **34.4.12 UCAXIE Register**

USCI\_Ax Interrupt Enable Register

**Figure 34-23. UCAXIE Register**

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 34-18. UCAXIE Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
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

### **34.4.13 UCAXIFG Register**

USCI\_Ax Interrupt Flag Register

**Figure 34-24. UCAXIFG Register**

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 34-19. UCAXIFG Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
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

**34.4.14 UCAXIV Register**

USCI\_Ax Interrupt Vector Register

**Figure 34-25. 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 34-20. UCAXIV Register Description**

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	USCI interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest

## ***Universal Serial Communication Interface – SPI Mode***

---

---

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface (SPI) mode.

Topic	Page
35.1 Universal Serial Communication Interface (USCI) Overview .....	924
35.2 USCI Introduction – SPI Mode .....	925
35.3 USCI Operation – SPI Mode .....	927
35.4 USCI_A SPI Mode Registers .....	932
35.5 USCI_B SPI Mode Registers .....	940

### 35.1 Universal Serial Communication Interface (USCI) Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI\_A is different from USCI\_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI\_A modules, they are named USCI\_A0 and USCI\_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on which devices.

USCI\_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud-rate detection for LIN communications
- SPI mode

USCI\_Bx modules support:

- I<sup>2</sup>C mode
- SPI mode

## 35.2 USCI Introduction – SPI Mode

In synchronous mode, the USCI connects the device to an external system via three or four pins: UCxSIMO, UCxSOMI, UCxCLK, and UCxSTE. SPI mode is selected when the UCSYNC bit is set, and SPI mode (3-pin or 4-pin) is selected with the UCMODE<sub>x</sub> bits.

SPI mode features include:

- 7-bit or 8-bit data length
- LSB-first or MSB-first data transmit and receive
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Continuous transmit and receive operation
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

[Figure 35-1](#) shows the USCI when configured for SPI mode.

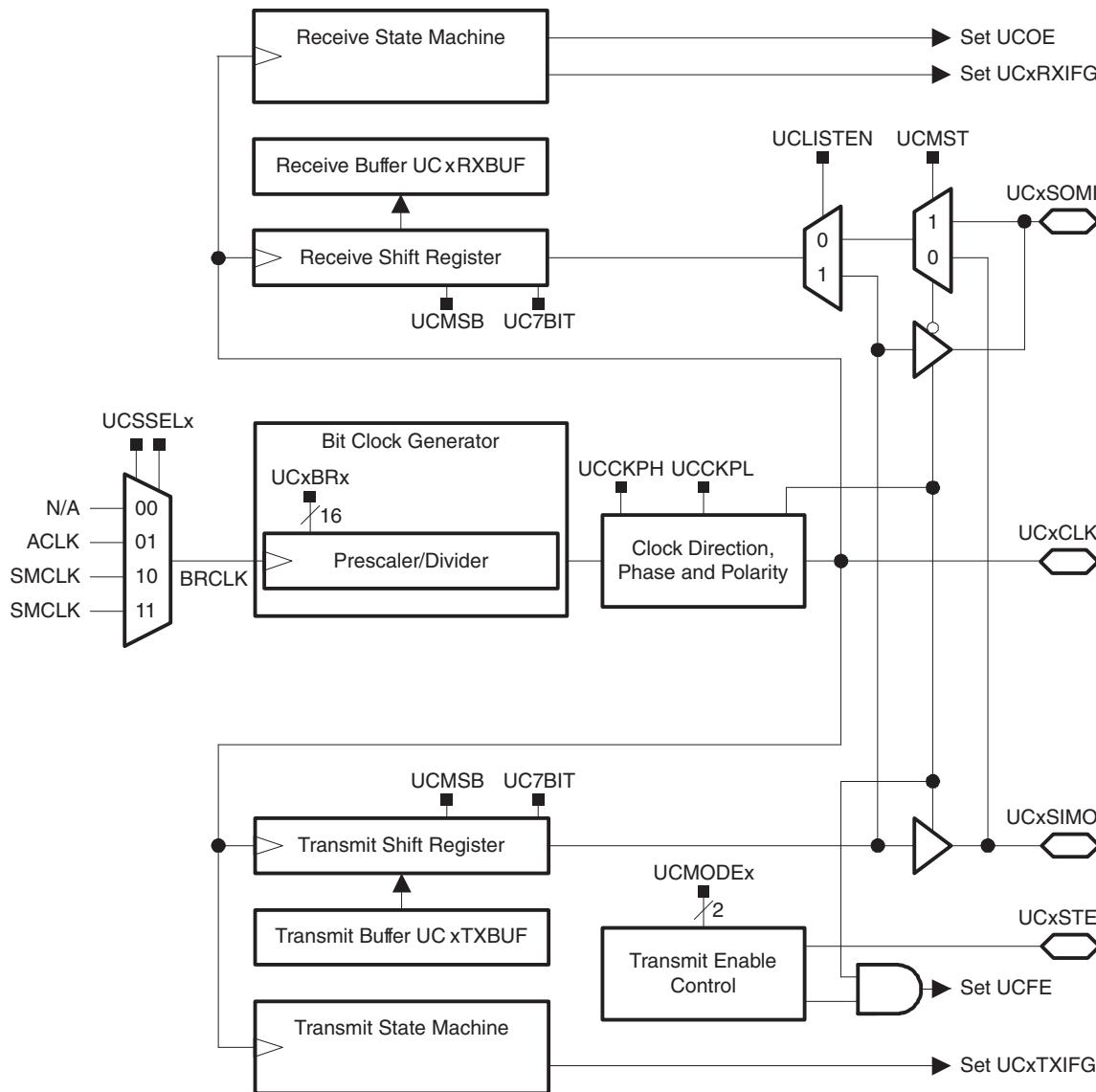


Figure 35-1. USCI Block Diagram – SPI Mode

### 35.3 USCI Operation – SPI Mode

In SPI mode, serial data is transmitted and received by multiple devices using a shared clock provided by the master. An additional pin, UCxSTE, is provided to enable a device to receive and transmit data and is controlled by the master.

Three or four signals are used for SPI data exchange:

- UCxSIMO – slave in, master out

Master mode: UCxSIMO is the data output line.

Slave mode: UCxSIMO is the data input line.

- UCxSOMI – slave out, master in

Master mode: UCxSOMI is the data input line.

Slave mode: UCxSOMI is the data output line.

- UCxCLK – USCI SPI clock

Master mode: UCxCLK is an output.

Slave mode: UCxCLK is an input.

- UCxSTE – slave transmit enable

Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode.

Table 35-1 describes the UCxSTE operation.

**Table 35-1. 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

#### 35.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. When set, the UCSWRST bit resets the UCRXIE, UCTXIE, UCRXIFG, UCOE, and UCFE bits, and sets the UCTXIFG flag. Clearing UCSWRST releases the USCI for operation.

To avoid unpredictable behavior, configure or reconfigure the USCI module only when UCSWRST is set.

---

**NOTE: Initializing or reconfiguring the USCI module**

The recommended USCI initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST, &UCxCTL1`).
  2. Initialize all USCI registers with UCSWRST = 1 (including UCxCTL1).
  3. Configure ports.
  4. Clear UCSWRST via software (`BIC.B #UCSWRST, &UCxCTL1`).
  5. Enable interrupts (optional) via UCRXIE and/or UCTXIE.
- 

#### 35.3.2 Character Format

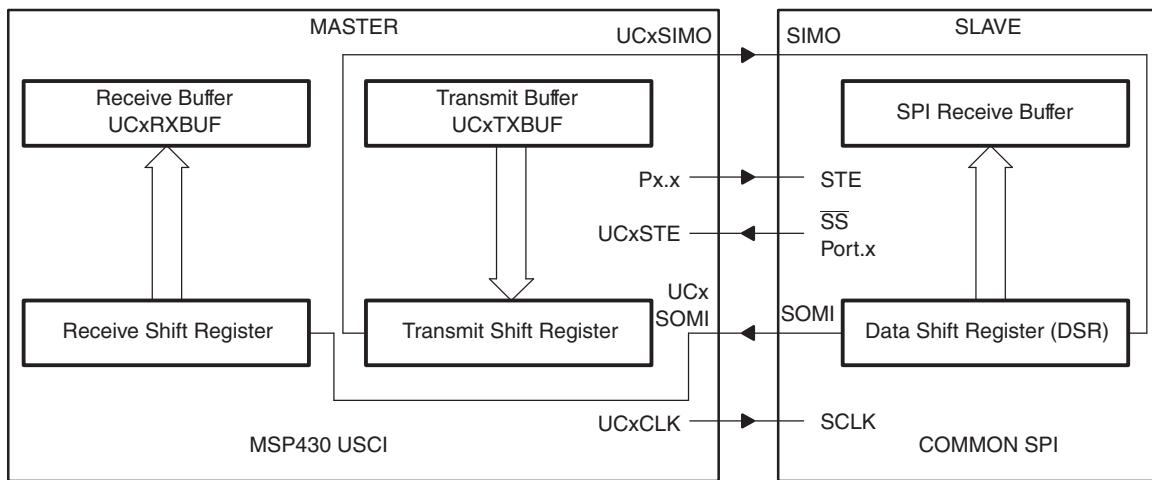
The USCI module in SPI mode supports 7-bit and 8-bit character lengths selected by the UC7BIT bit. In 7-bit data mode, UCxRXBUF is LSB justified and the MSB is always reset. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first.

**NOTE: Default character format**

The default SPI character transmission is LSB first. For communication with other SPI interfaces, MSB-first mode may be required.

**NOTE: Character format for Figures**

Figures throughout this chapter use MSB-first format.

**35.3.3 Master Mode**

**Figure 35-2. USCI Master and External Slave**

Figure 35-2 shows the USCI as a master in both 3-pin and 4-pin configurations. The USCI initiates data transfer when data is moved to the transmit data buffer UCxTXBUF. The UCxTXBUF data is moved to the transmit (TX) shift register when the TX shift register is empty, initiating data transfer on UCxSIMO starting with either the MSB or LSB, depending on the UCMSB setting. Data on UCxSOMI is shifted into the receive shift register on the opposite clock edge. When the character is received, the receive data is moved from the receive (RX) shift register to the received data buffer UCxRXBUF and the receive interrupt flag UCRXIFG is set, indicating the RX/TX operation is complete.

A set transmit interrupt flag, UCTXIFG, indicates that data has moved from UCxTXBUF to the TX shift register and UCxTXBUF is ready for new data. It does not indicate RX/TX completion.

To receive data into the USCI in master mode, data must be written to UCxTXBUF, because receive and transmit operations operate concurrently.

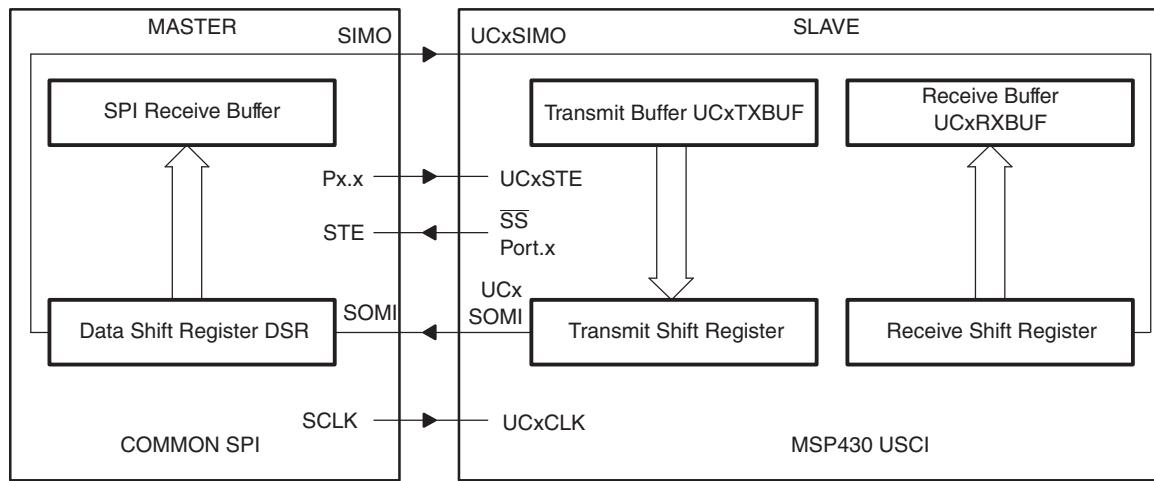
**35.3.3.1 4-Pin SPI Master Mode**

In 4-pin master mode, UCxSTE is used to prevent conflicts with another master and controls the master as described in [Table 35-1](#). When UCxSTE is in the master-inactive state:

- UCxSIMO and UCxCLK are set to inputs and no longer drive the bus.
- The error bit UCFE is set, indicating a communication integrity violation to be handled by the user.
- The internal state machines are reset and the shift operation is aborted.

If data is written into UCxTXBUF while the master is held inactive by UCxSTE, it is transmit as soon as UCxSTE transitions to the master-active state. If an active transfer is aborted by UCxSTE transitioning to the master-inactive state, the data must be rewritten into UCxTXBUF to be transferred when UCxSTE transitions back to the master-active state. The UCxSTE input signal is not used in 3-pin master mode.

### 35.3.4 Slave Mode



**Figure 35-3. USCI Slave and External Master**

Figure 35-3 shows the USCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCxTXBUF and moved to the TX shift register before the start of UCxCLK is transmitted on UCxSOMI. Data on UCxSIMO is shifted into the receive shift register on the opposite edge of UCxCLK and moved to UCxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UCxRXBUF, the UCRXIFG interrupt flag is set, indicating that data has been received. The overrun error bit UCOE is set when the previously received data is not read from UCxRXBUF before new data is moved to UCxRXBUF.

#### 35.3.4.1 4-Pin SPI Slave Mode

In 4-pin slave mode, UCxSTE is used by the slave to enable the transmit and receive operations and is provided by the SPI master. When UCxSTE is in the slave-active state, the slave operates normally. When UCxSTE is in the slave-inactive state:

- Any receive operation in progress on UCxSIMO is halted.
- UCxSOMI is set to the input direction.
- The shift operation is halted until the UCxSTE line transitions into the slave transmit active state.

The UCxSTE input signal is not used in 3-pin slave mode.

#### 35.3.5 SPI Enable

When the USCI module is enabled by clearing the UCSWRST bit, it is ready to receive and transmit. In master mode, the bit clock generator is ready, but is not clocked nor producing any clocks. In slave mode, the bit clock generator is disabled and the clock is provided by the master.

A transmit or receive operation is indicated by UCBUSY = 1.

A PUC or set UCSWRST bit disables the USCI immediately and any active transfer is terminated.

##### 35.3.5.1 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.

### 35.3.5.2 Receive Enable

The SPI receives data when a transmission is active. Receive and transmit operations operate concurrently.

### 35.3.6 Serial Clock Control

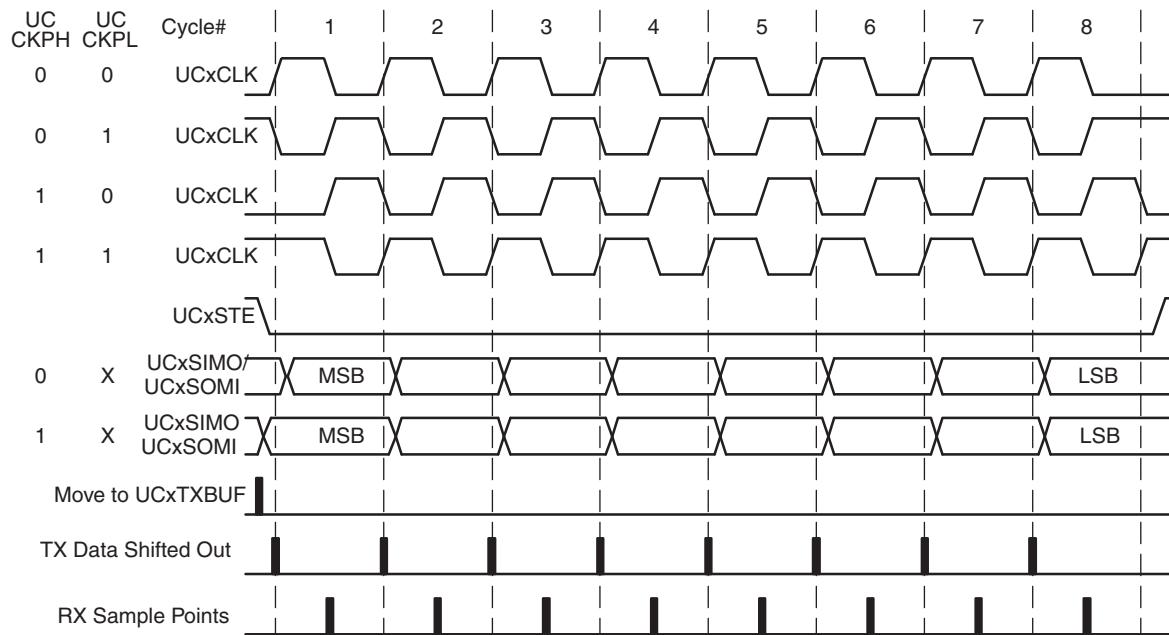
UCxCLK is provided by the master on the SPI bus. When UCMST = 1, the bit clock is provided by the USCI bit clock generator on the UCxCLK pin. The clock used to generate the bit clock is selected with the UCSSELx bits. When UCMST = 0, the USCI clock is provided on the UCxCLK pin by the master, the bit clock generator is not used, and the UCSSELx bits are don't care. The SPI receiver and transmitter operate in parallel and use the same clock source for data transfer.

The 16-bit value of UCBRx in the bit rate control registers (UCxxBR1 and UCxxBR0) is the division factor of the USCI clock source, BRCLK. The maximum bit clock that can be generated in master mode is BRCLK. Modulation is not used in SPI mode, and UCAxMCTL should be cleared when using SPI mode for USCI\_A. The UCAxCLK/UCBxCLK frequency is given by:

$$f_{\text{BitClock}} = f_{\text{BRCLK}} / \text{UCBRx}$$

#### 35.3.6.1 Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the USCI. Timing for each case is shown in [Figure 35-4](#).



**Figure 35-4. USCI SPI Timing With UCMSB = 1**

### 35.3.7 Using the SPI Mode With Low-Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

In SPI slave mode, no internal clock source is required because the clock is provided by the external master. It is possible to operate the USCI in SPI slave mode while the device is in LPM4 and all clock sources are disabled. The receive or transmit interrupt can wake up the CPU from any low-power mode.

### 35.3.8 SPI Interrupts

The USCI has only one interrupt vector that is shared for transmission and for reception. USCI\_Ax and USC\_Bx do not share the same interrupt vector.

#### 35.3.8.1 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.

---

#### 35.3.8.2 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.

#### 35.3.8.3 UCxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCxIV register that can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCxIV value.

Any access, read or write, of the UCxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

#### UCxIV Software Example

The following software example shows the recommended use of UCxIV. The UCxIV value is added to the PC to automatically jump to the appropriate routine. The following example is given for USCI\_B0.

```
USCI_SPI_ISR
    ADD    &UCB0IV, PC ; Add offset to jump table
    RETI          ; Vector 0: No interrupt
    JMP    RXIFG_ISR   ; Vector 2: RXIFG
TXIFG_ISR
    ...
    RETI          ; Task starts here
    ; Return
    ; Vector 4: TXIFG
RXIFG_ISR
    ...
    RETI          ; Task starts here
    ; Return
```

### 35.4 USCI\_A SPI Mode Registers

The USCI\_A registers that are applicable in SPI mode are listed in [Table 35-2](#). The base addresses can be found in the device-specific data sheet. The address offsets are listed in [Table 35-2](#).

**Table 35-2. USCI\_A SPI Mode Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	USCI_Ax Control Word 0	Read/write	Word	0001h	
00h	UCAxCTL1	USCI_Ax Control 1	Read/write	Byte	01h	<a href="#">Section 35.4.2</a>
01h	UCAxCTL0	USCI_Ax Control 0	Read/write	Byte	00h	<a href="#">Section 35.4.1</a>
06h	UCAxBRW	USCI_Ax Bit Rate Control Word	Read/write	Word	0000h	
06h	UCAxBR0	USCI_Ax Bit Rate Control 0	Read/write	Byte	00h	<a href="#">Section 35.4.3</a>
07h	UCAxBR1	USCI_Ax Bit Rate Control 1	Read/write	Byte	00h	<a href="#">Section 35.4.4</a>
08h	UCAxMCTL	USCI_Ax Modulation Control	Read/write	Byte	00h	<a href="#">Section 35.4.5</a>
0Ah	UCAxSTAT	USCI_Ax Status	Read/write	Byte	00h	<a href="#">Section 35.4.6</a>
0Bh		Reserved - reads zero	Read	Byte	00h	
0Ch	UCAxRXBUF	USCI_Ax Receive Buffer	Read/write	Byte	00h	<a href="#">Section 35.4.7</a>
0Dh		Reserved - reads zero	Read	Byte	00h	
0Eh	UCAxTXBUF	USCI_Ax Transmit Buffer	Read/write	Byte	00h	<a href="#">Section 35.4.8</a>
0Fh		Reserved - reads zero	Read	Byte	00h	
1Ch	UCAxICTL	USCI_Ax Interrupt Control	Read/write	Word	0200h	
1Ch	UCAxIE	USCI_Ax Interrupt Enable	Read/write	Byte	00h	<a href="#">Section 35.4.9</a>
1Dh	UCAxIFG	USCI_Ax Interrupt Flag	Read/write	Byte	02h	<a href="#">Section 35.4.10</a>
1Eh	UCAxIV	USCI_Ax Interrupt Vector	Read	Word	0000h	<a href="#">Section 35.4.11</a>

### 35.4.1 UCAXCTL0 Register

USCI\_Ax Control Register 0

**Figure 35-5. UCAXCTL0 Register**

7	6	5	4	3	2	1	0
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx	UCSYNC	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

**Table 35-3. UCAXCTL0 Register Description**

Bit	Field	Type	Reset	Description
7	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
6	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
5	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
4	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
3	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
2-1	UCMODEx	RW	0h	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = I2C mode
0	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode

### 35.4.2 UCAXCTL1 Register

USCI\_Ax Control Register 1

**Figure 35-6. UCAXCTL1 Register**

7	6	5	4	3	2	1	0
UCSSELx		Reserved					
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when UCSWRST = 1.

**Table 35-4. UCAXCTL1 Register Description**

Bit	Field	Type	Reset	Description
7-6	UCSSELx	RW	0h	USCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK
5-1	Reserved	RW	0h	Reserved. Always write as 0.
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. USCI reset released for operation. 1b = Enabled. USCI logic held in reset state.

### 35.4.3 UCAXBR0 Register

USCI\_Ax Bit Rate Control Register 0

**Figure 35-7. UCAXBR0 Register**

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

Modify only when UCSWRST = 1.

**Table 35-5. UCAXBR0 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Bit clock prescaler low byte. The 16-bit value of (UCAXBR0 + UCAXBR1 × 256) forms the prescaler value UCBRx.

### 35.4.4 UCAXBR1 Register

USCI\_Ax Bit Rate Control Register 1

**Figure 35-8. UCAXBR1 Register**

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

Modify only when UCSWRST = 1.

**Table 35-6. UCAXBR1 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Bit clock prescaler high byte. The 16-bit value of (UCAXBR0 + UCAXBR1 × 256) forms the prescaler value UCBRx.

### 35.4.5 UCAXMCTL Register

USCI\_Ax Modulation Control Register

**Figure 35-9. UCAXMCTL Register**

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

**Table 35-7. UCAXMCTL Register Description**

Bit	Field	Type	Reset	Description
7-0	Reserved	R	0h	Reserved. Always write as 0.

**35.4.6 UCAXSTAT Register**

USCI\_Ax Status Register

**Figure 35-10. UCAXSTAT Register**

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 35-8. UCAXSTAT Register Description**

Bit	Field	Type	Reset	Description
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. The transmitter output is internally fed back to the receiver.
6	UCFE	RW	0h	Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode. 0b = No error 1b = Bus conflict occurred.
5	UCOE	RW	0h	Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly. 0b = No error 1b = Overrun error occurred
4-1	Reserved	R	0h	Reserved. Always reads as 0.
0	UCBUSY	R	0h	USCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = USCI inactive 1b = USCI transmitting or receiving

### **35.4.7 UCAXRXBUF Register**

USCI\_Ax Receive Buffer Register

**Figure 35-11. UCAXRXBUF Register**

7	6	5	4	3	2	1	0
UCRXBUF <sub>x</sub>							
r	r	r	r	r	r	r	r

**Table 35-9. UCAXRXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCRXBUF <sub>x</sub>	R	undefined	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCRXBUF resets the receive-error bits and UCRXIFG. In 7-bit data mode, UCRXBUF is LSB justified and the MSB is always reset.

### **35.4.8 UCAXTXBUF Register**

USCI\_Ax Transmit Buffer Register

**Figure 35-12. UCAXTXBUF Register**

7	6	5	4	3	2	1	0
UCTXBUF <sub>x</sub>							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 35-10. UCAXTXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCTXBUF <sub>x</sub>	RW	undefined	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 UCAXTXBUF is not used for 7-bit data and is reset.

**35.4.9 UCAXIE Register**

USCI\_Ax Interrupt Enable Register

**Figure 35-13. UCAXIE Register**

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 35-11. UCAXIE Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
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

**35.4.10 UCAXIFG Register**

USCI\_Ax Interrupt Flag Register

**Figure 35-14. UCAXIFG Register**

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 35-12. UCAXIFG Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
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

### **35.4.11 UCAXIV Register**

USCI\_Ax Interrupt Vector Register

**Figure 35-15. 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 35-13. UCAXIV Register Description**

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	USCI interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest

### 35.5 USCI\_B SPI Mode Registers

The USCI\_B registers applicable in SPI mode are listed in [Table 35-14](#). The base addresses can be found in the device-specific data sheet. The address offsets are listed in [Table 35-14](#).

**Table 35-14. USCI\_B SPI Mode Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	USCI_Bx Control Word 0	Read/write	Word	0101h	
00h	UCBxCTL1	USCI_Bx Control 1	Read/write	Byte	01h	<a href="#">Section 35.5.2</a>
01h	UCBxCTL0	USCI_Bx Control 0	Read/write	Byte	01h	<a href="#">Section 35.5.1</a>
06h	UCBxBRW	USCI_Bx Bit Rate Control Word	Read/write	Word	0000h	
06h	UCBxBR0	USCI_Bx Bit Rate Control 0	Read/write	Byte	00h	<a href="#">Section 35.5.3</a>
07h	UCBxBR1	USCI_Bx Bit Rate Control 1	Read/write	Byte	00h	<a href="#">Section 35.5.4</a>
08h	UCBxMCTL	USCI_Bx Modulation Control	Read/write	Byte	00h	<a href="#">Section 35.5.5</a>
0Ah	UCBxSTAT	USCI_Bx Status	Read/write	Byte	00h	<a href="#">Section 35.5.6</a>
0Bh		Reserved - reads zero	Read	Byte	00h	
0Ch	UCBxRXBUF	USCI_Bx Receive Buffer	Read/write	Byte	00h	<a href="#">Section 35.5.7</a>
0Dh		Reserved - reads zero	Read	Byte	00h	
0Eh	UCBxTXBUF	USCI_Bx Transmit Buffer	Read/write	Byte	00h	<a href="#">Section 35.5.8</a>
0Fh		Reserved - reads zero	Read	Byte	00h	
1Ch	UCBxICTL	USCI_Bx Interrupt Control	Read/write	Word	0200h	
1Ch	UCBxIE	USCI_Bx Interrupt Enable	Read/write	Byte	00h	<a href="#">Section 35.5.9</a>
1Dh	UCBxIFG	USCI_Bx Interrupt Flag	Read/write	Byte	02h	<a href="#">Section 35.5.10</a>
1Eh	UCBxIV	USCI_Bx Interrupt Vector	Read	Word	0000h	<a href="#">Section 35.5.11</a>

### **35.5.1 UCBxCTL0 Register**

USCI\_Bx Control Register 0

**Figure 35-16. UCBxCTL0 Register**

7	6	5	4	3	2	1	0
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx	UCSYNC	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when UCSWRST = 1.

**Table 35-15. UCBxCTL0 Register Description**

Bit	Field	Type	Reset	Description
7	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
6	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
5	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
4	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
3	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
2-1	UCMODEx	RW	0h	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = I2C mode
0	UCSYNC	RW	1h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode

**35.5.2 UCBxCTL1 Register**

USCI\_Bx Control Register 1

**Figure 35-17. UCBxCTL1 Register**

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

Modify only when UCSWRST = 1.

**Table 35-16. UCBxCTL1 Register Description**

Bit	Field	Type	Reset	Description
7-6	UCSSELx	RW	0h	USCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK
5-1	Reserved	RW	0h	Reserved. Always write as 0.
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. USCI reset released for operation. 1b = Enabled. USCI logic held in reset state.

### 35.5.3 UCBxBR0 Register

USCI\_Bx Bit Rate Control Register 0

**Figure 35-18. UCBxBR0 Register**

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

Modify only when UCSWRST = 1.

**Table 35-17. UCBxBR0 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Bit clock prescaler low byte. The 16-bit value of (UCBxBR0 + UCBxBR1 × 256) forms the prescaler value UCBRx.

### 35.5.4 UCBxBR1 Register

USCI\_Bx Bit Rate Control Register 1

**Figure 35-19. UCBxBR1 Register**

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

Modify only when UCSWRST = 1.

**Table 35-18. UCBxBR1 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Bit clock prescaler high byte. The 16-bit value of (UCBxBR0 + UCBxBR1 × 256) forms the prescaler value UCBRx.

### 35.5.5 UCBxMCTL Register

USCI\_Bx Modulation Control Register

**Figure 35-20. UCBxMCTL Register**

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

**Table 35-19. UCBxMCTL Register Description**

Bit	Field	Type	Reset	Description
7-0	Reserved	R	0h	Reserved. Always write as 0.

**35.5.6 UCBxSTAT Register**

USCI\_Bx Status Register

**Figure 35-21. UCBxSTAT Register**

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 35-20. UCBxSTAT Register Description**

Bit	Field	Type	Reset	Description
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. The transmitter output is internally fed back to the receiver.
6	UCFE	RW	0h	Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode. 0b = No error 1b = Bus conflict occurred.
5	UCOE	RW	0h	Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly. 0b = No error 1b = Overrun error occurred
4-1	Reserved	R	0h	Reserved. Always reads as 0.
0	UCBUSY	R	0h	USCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = USCI inactive 1b = USCI transmitting or receiving

### 35.5.7 UCBxRXBUF Register

USCI\_Bx Receive Buffer Register

**Figure 35-22. UCBxRXBUF Register**

7	6	5	4	3	2	1	0
UCRXBUF <sub>x</sub>							
r	r	r	r	r	r	r	r

**Table 35-21. UCBxRXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCRXBUF <sub>x</sub>	R	undefined	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCRXBUF resets the receive-error bits and UCRXIFG. In 7-bit data mode, UCRXBUF is LSB justified and the MSB is always reset.

### 35.5.8 UCBxTXBUF Register

USCI\_Bx Transmit Buffer Register

**Figure 35-23. UCBxTXBUF Register**

7	6	5	4	3	2	1	0
UCTXBUF <sub>x</sub>							
rw	rw	rw	rw	rw	rw	rw	rw

**Table 35-22. UCBxTXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCTXBUF <sub>x</sub>	RW	undefined	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 UCBxTXBUF is not used for 7-bit data and is reset.

**35.5.9 UCBxIE Register**

USCI\_Bx Interrupt Enable Register

**Figure 35-24. UCBxIE Register**

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 35-23. UCBxIE Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
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

**35.5.10 UCBxIFG Register**

USCI\_Bx Interrupt Flag Register

**Figure 35-25. UCBxIFG Register**

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 35-24. UCBxIFG Register Description**

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
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

### **35.5.11 UCBxIV Register**

USCI\_Bx Interrupt Vector Register

**Figure 35-26. 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 35-25. UCBxIV Register Description**

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	USCI interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest

## **Universal Serial Communication Interface – I<sup>2</sup>C Mode**

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the I<sup>2</sup>C mode.

Topic	Page
36.1 Universal Serial Communication Interface (USCI) Overview .....	949
36.2 USCI Introduction – I <sup>2</sup> C Mode .....	950
36.3 USCI Operation – I <sup>2</sup> C Mode .....	951
36.4 USCI_B I2C Mode Registers .....	970

### 36.1 Universal Serial Communication Interface (USCI) Overview

The USCI modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI\_A is different from USCI\_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI\_A modules, they are named USCI\_A0 and USCI\_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on each device.

USCI\_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud-rate detection for LIN communications
- SPI mode

USCI\_Bx modules support:

- I<sup>2</sup>C mode
- SPI mode

## 36.2 USCI Introduction – I<sup>2</sup>C Mode

In I<sup>2</sup>C mode, the USCI module provides an interface between the device and I<sup>2</sup>C-compatible devices connected by the two-wire I<sup>2</sup>C serial bus. External components attached to the I<sup>2</sup>C bus serially transmit and/or receive serial data to/from the USCI module through the 2-wire I<sup>2</sup>C interface.

The I<sup>2</sup>C mode features include:

- Compliance to the Philips Semiconductor I<sup>2</sup>C specification v2.1
- 7-bit and 10-bit device addressing modes
- General call
- START/RESTART/STOP
- Multi-master transmitter/receiver mode
- Slave receiver/transmitter mode
- Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- Slave receiver START detection for auto wake up from LPMx modes (wake up from LPMx.5 is not supported)
- Slave operation in LPM4

[Figure 36-1](#) shows the USCI when configured in I<sup>2</sup>C mode.

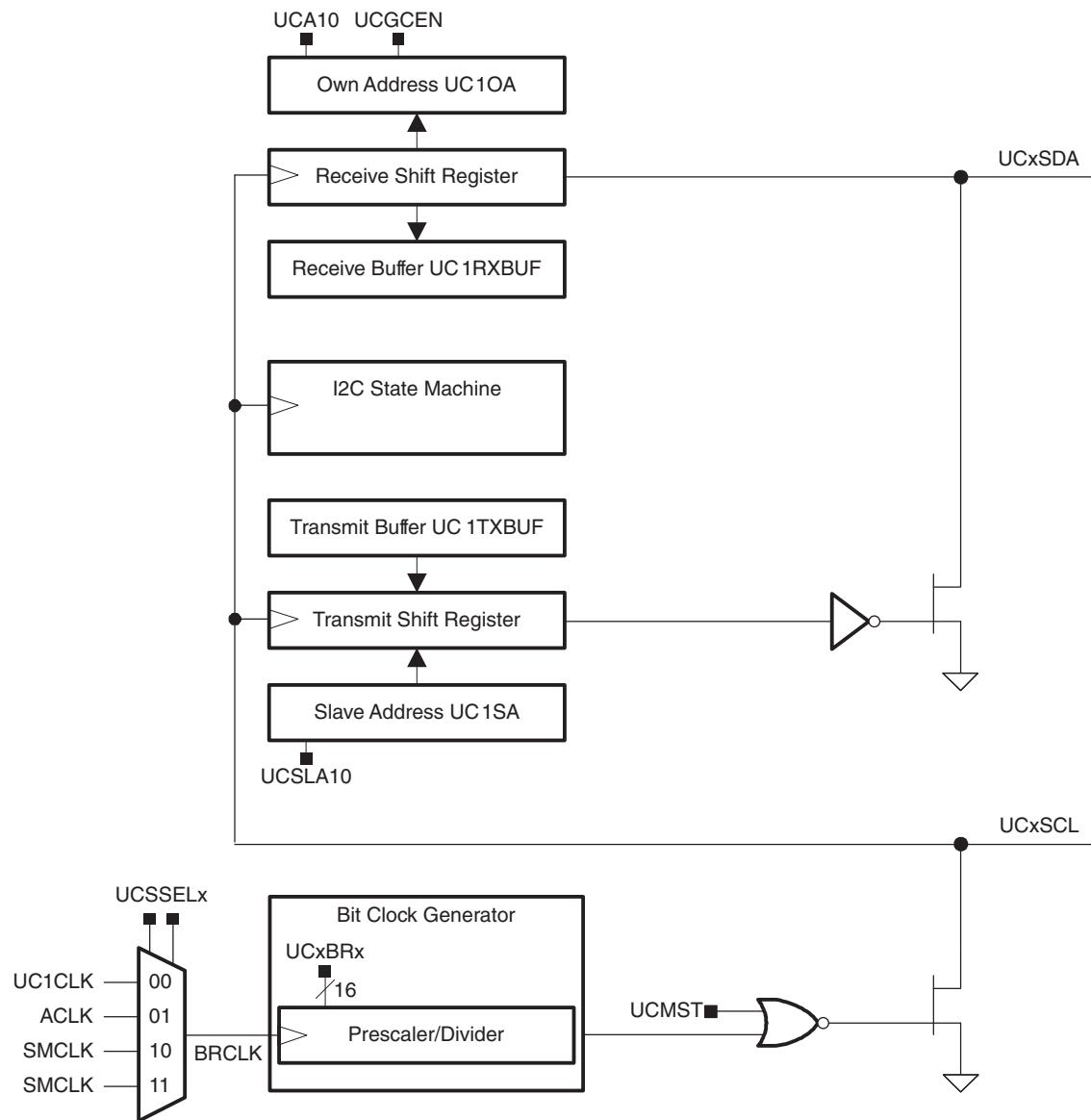
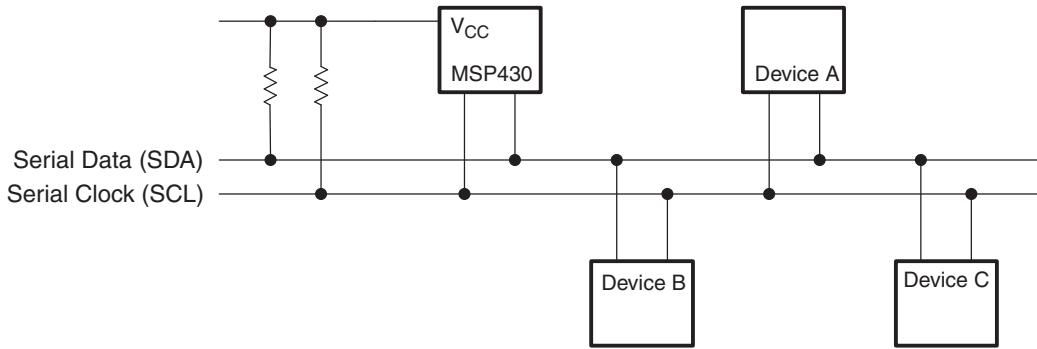


Figure 36-1. USCI Block Diagram – I<sup>2</sup>C Mode

### 36.3 USCI Operation – I<sup>2</sup>C Mode

The I<sup>2</sup>C mode supports any slave or master I<sup>2</sup>C-compatible device. Figure 36-2 shows an example of an I<sup>2</sup>C bus. Each I<sup>2</sup>C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I<sup>2</sup>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<sup>2</sup>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 36-2. I<sup>2</sup>C Bus Connection Diagram**NOTE: SDA and SCL levels**

The SDA and SCL pins must not be pulled up above the device V<sub>cc</sub> level.

**36.3.1 USCI Initialization and Reset**

The USCI is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. To select I<sup>2</sup>C operation, the UCMODEX bits must be set to 11. After module initialization, it is ready for transmit or receive operation. Clearing UCSWRST releases the USCI for operation.

To avoid unpredictable behavior, configure or reconfigure the USCI module only when UCSWRST is set.

Setting UCSWRST in I<sup>2</sup>C mode has the following effects:

- I<sup>2</sup>C communication stops.
- SDA and SCL are high impedance.
- UCBxI2CSTAT, bits 6–0 are cleared.
- Registers UCBxIE and UCBxIFG are cleared.
- All other bits and register remain unchanged.

**NOTE: Initializing or re-configuring the USCI module**

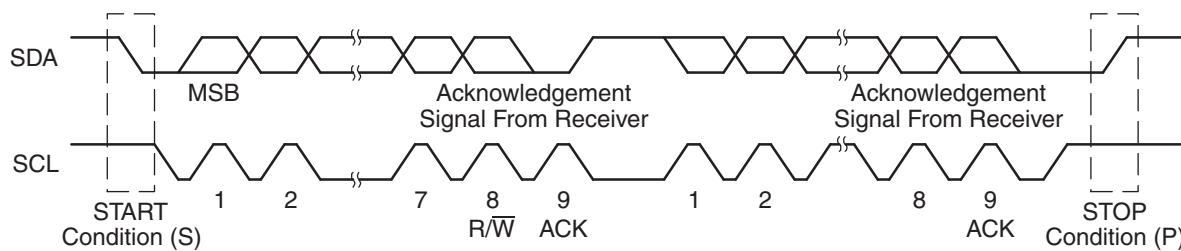
The recommended USCI initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST, &UCxCTL1`).
2. Initialize all USCI registers with UCSWRST = 1.
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST, &UCxCTL1`).
5. Enable interrupts (optional).

**36.3.2 I<sup>2</sup>C Serial Data**

One clock pulse is generated by the master device for each data bit transferred. The I<sup>2</sup>C mode operates with byte data. Data is transferred MSB first as shown in Figure 36-3.

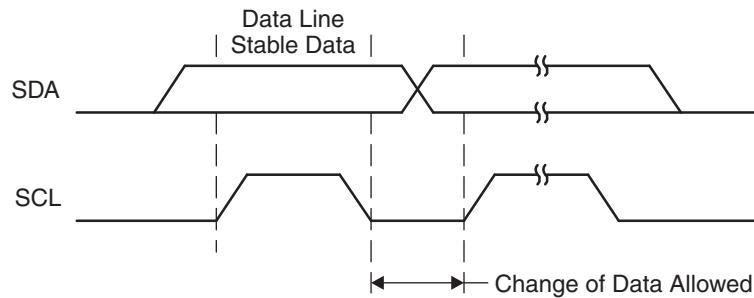
The first byte after a START condition consists of a 7-bit slave address and the R/W bit. When R/W = 0, the master transmits data to a slave. When R/W = 1, the master receives data from a slave. The ACK bit is sent from the receiver after each byte on the ninth SCL clock.



**Figure 36-3. I<sup>2</sup>C Module Data Transfer**

START and STOP conditions are generated by the master and are shown in [Figure 36-3](#). 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 36-4](#)). The high and low state of SDA can only change when SCL is low, otherwise START or STOP conditions are generated.



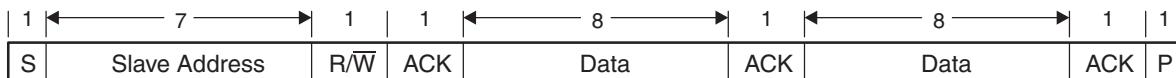
**Figure 36-4. Bit Transfer on I<sup>2</sup>C Bus**

### 36.3.3 I<sup>2</sup>C Addressing Modes

The I<sup>2</sup>C mode supports 7-bit and 10-bit addressing modes.

#### 36.3.3.1 7-Bit Addressing

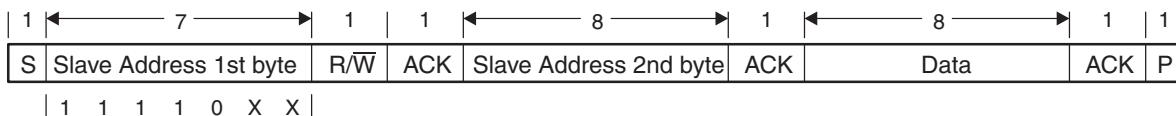
In the 7-bit addressing format (see [Figure 36-5](#)), the first byte is the 7-bit slave address and the R/W bit. The ACK bit is sent from the receiver after each byte.



**Figure 36-5. I<sup>2</sup>C Module 7-Bit Addressing Format**

#### 36.3.3.2 10-Bit Addressing

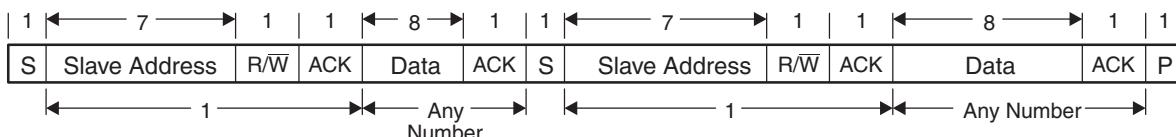
In the 10-bit addressing format (see [Figure 36-6](#)), the first byte is made up of 11110b plus the two MSBs of the 10-bit slave address and the R/W bit. The ACK bit is sent from the receiver after each byte. The next byte is the remaining eight bits of the 10-bit slave address, followed by the ACK bit and the 8-bit data. See [I<sup>2</sup>C Slave 10-bit Addressing Mode](#) and [I<sup>2</sup>C Master 10-bit Addressing Mode](#) for details how to use the 10-bit addressing mode with the USCI module.



**Figure 36-6. I<sup>2</sup>C Module 10-Bit Addressing Format**

#### 36.3.3.3 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/W bit. The RESTART condition is shown in [Figure 36-7](#).



**Figure 36-7. I<sup>2</sup>C Module Addressing Format With Repeated START Condition**

### 36.3.4 I<sup>2</sup>C Module Operating Modes

In I<sup>2</sup>C mode, the USCI module can operate in master transmitter, master receiver, slave transmitter, or slave receiver mode. The modes are discussed in the following sections. Time lines are used to illustrate the modes.

Figure 36-8 shows how to interpret the time-line figures. Data transmitted by the master is represented by grey rectangles; data transmitted by the slave is represented by white rectangles. Data transmitted by the USCI module, either as master or slave, is shown by rectangles that are taller than the others.

Actions taken by the USCI module are shown in grey rectangles with an arrow indicating where in the 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.

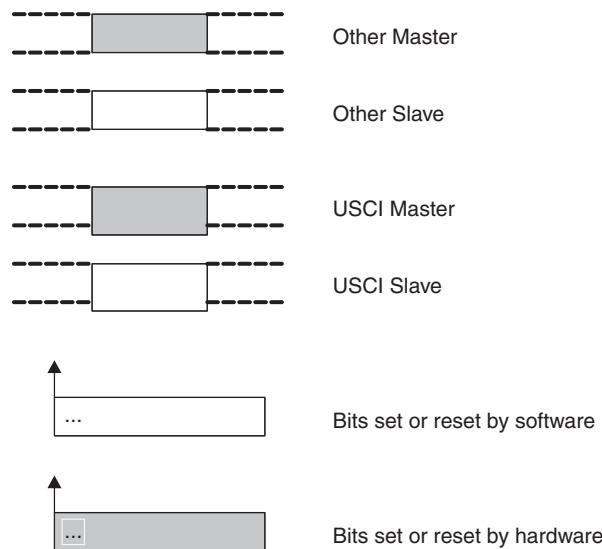


Figure 36-8. I<sup>2</sup>C Time-Line Legend

#### 36.3.4.1 Slave Mode

The USCI module is configured as an I<sup>2</sup>C slave by selecting the I<sup>2</sup>C mode with UCMODE<sub>X</sub> = 11 and UCSYNC = 1 and clearing the UCMST bit.

Initially, the USCI module must be configured in receiver mode by clearing the UCTR bit to receive the I<sup>2</sup>C address. Afterwards, transmit and receive operations are controlled automatically, depending on the R/W bit received together with the slave address.

The USCI slave address is programmed with the UCBxI2COA register. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the slave responds to a general call.

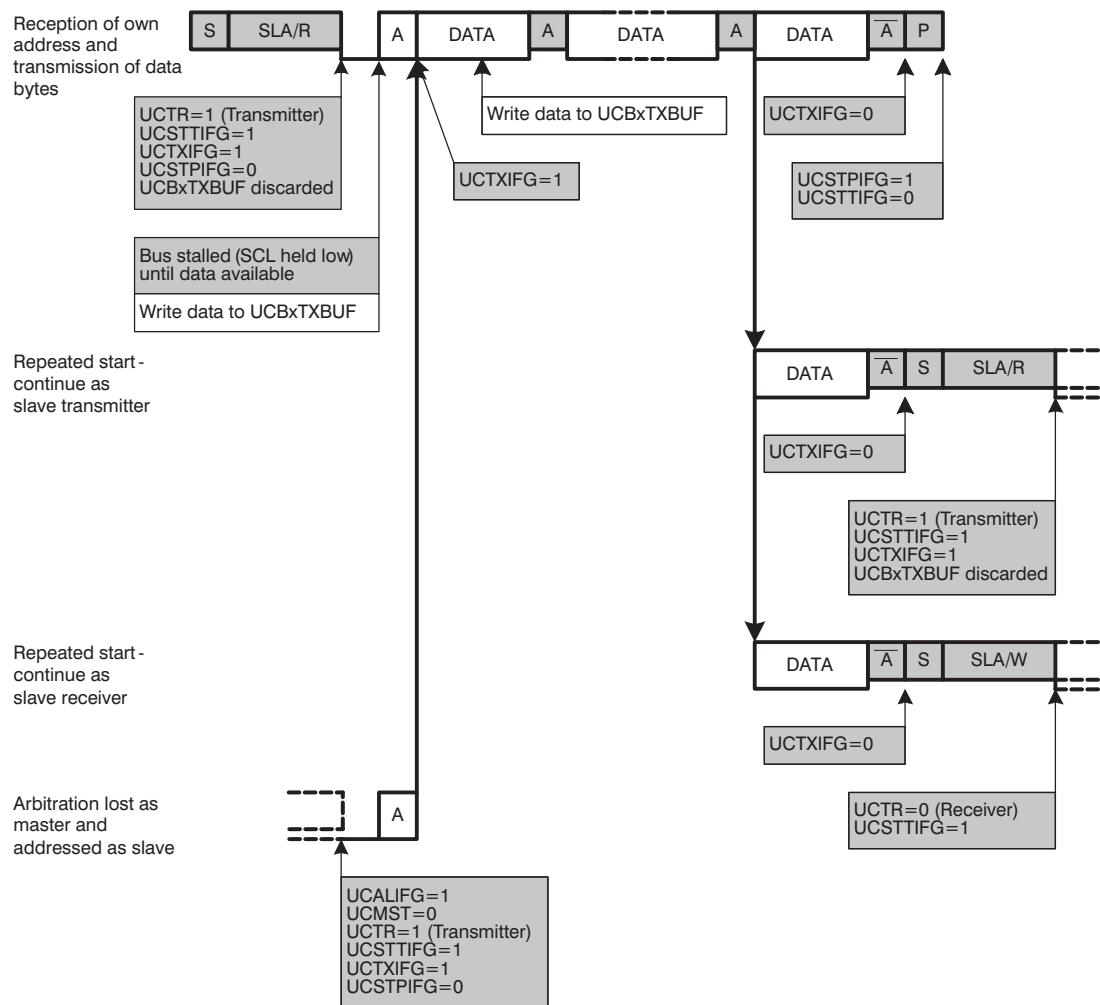
When a START condition is detected on the bus, the USCI module receives the transmitted address and compare it against its own address stored in UCBxI2COA. The UCSTTIFG flag is set when address received matches the USCI slave address.

**I<sup>2</sup>C 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/W bit. The slave transmitter shifts the serial data out on SDA with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it does hold SCL low while intervention of the CPU is required after a byte has been transmitted.

If the master requests data from the slave, the USCI module is automatically configured as a transmitter and UCTR and UCTXIFG become set. The SCL line is held low until the first data to be sent is written into the transmit buffer UCBxTXBUF. Then the address is acknowledged, the UCSTTIFG flag is cleared, and the data is transmitted. As soon as the data is transferred into the shift register, the UCTXIFG is set again. After the data is acknowledged by the master, the next data byte written into UCBxTXBUF is transmitted or, if the buffer is empty, the bus is stalled during the acknowledge cycle by holding SCL low until new data is written into UCBxTXBUF. If the master sends a NACK succeeded by a STOP condition, the UCSTPIFG flag is set. If the NACK is succeeded by a repeated START condition, the USCI I<sup>2</sup>C state machine returns to its address-reception state.

**Figure 36-9** shows the slave transmitter operation.



**Figure 36-9. I<sup>2</sup>C Slave Transmitter Mode**

### I<sup>2</sup>C 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/W bit is received. In slave receiver mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low if intervention of the CPU is required after a byte has been received.

If the slave should receive data from the master, the USCI module is automatically configured as a receiver and UCTR is cleared. After the first data byte is received, the receive interrupt flag UCRXIFG is set. The USCI module automatically acknowledges the received data and can receive the next data byte.

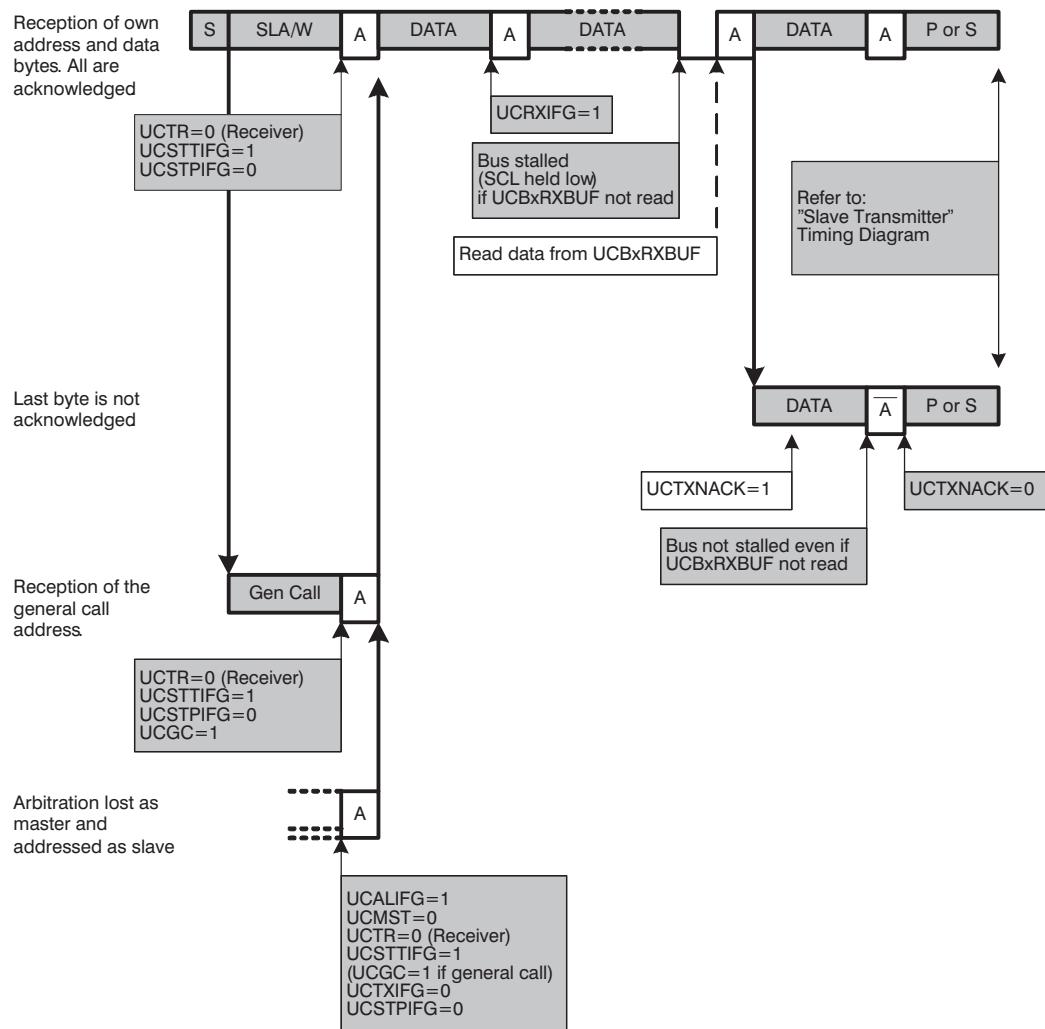
If the previous data was not read from the receive buffer UCBxRXBUF at the end of a reception, the bus is stalled by holding SCL low. As soon as UCBxRXBUF is read, the new data is transferred into UCBxRXBUF, an acknowledge is sent to the master, and the next data can be received.

Setting the UCTXNACK bit causes a NACK to be transmitted to the master during the next acknowledgment cycle. A NACK is sent even if UCBxRXBUF is not ready to receive the latest data. If the UCTXNACK bit is set while SCL is held low, the bus is released, a NACK is transmitted immediately, and UCBxRXBUF is loaded with the last received data. Because the previous data was not read, that data is lost. To avoid loss of data, the UCBxRXBUF must be read before UCTXNACK is set.

When the master generates a STOP condition, the UCSTPIFG flag is set.

If the master generates a repeated START condition, the USCI I<sup>2</sup>C state machine returns to its address reception state.

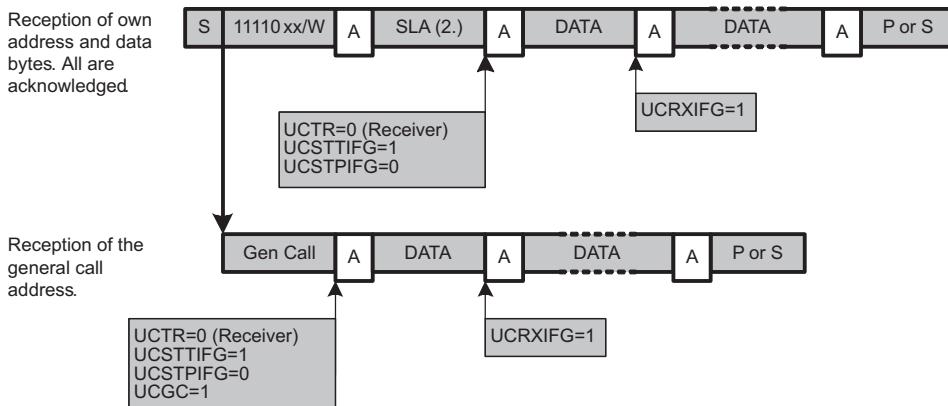
Figure 36-10 shows the I<sup>2</sup>C slave receiver operation.

Figure 36-10. I<sup>2</sup>C Slave Receiver Mode

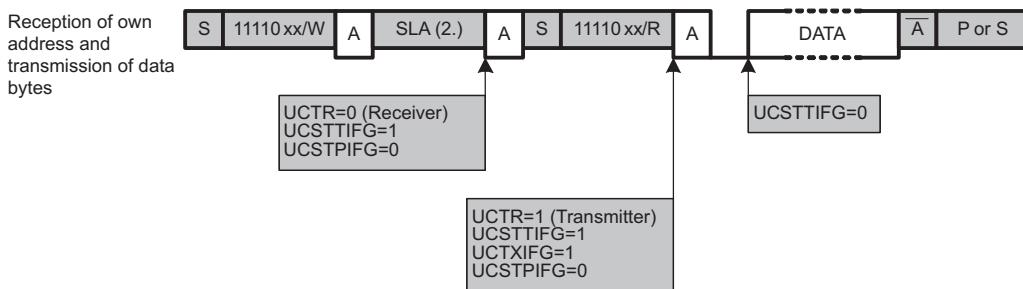
### I<sup>2</sup>C Slave 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in [Figure 36-11](#). In 10-bit addressing mode, the slave is in receive mode after the full address is received. The USCI module indicates this by setting the UCSTTIFG flag while the UCTR bit is cleared. To switch the slave into transmitter mode, the master sends a repeated START condition together with the first byte of the address but with the R/W bit set. This sets the UCSTTIFG flag if it was previously cleared by software, and the USCI modules switches to transmitter mode with UCTR = 1.

#### Slave Receiver



#### Slave Transmitter



**Figure 36-11. I<sup>2</sup>C Slave 10-Bit Addressing Mode**

#### 36.3.4.2 Master Mode

The USCI module is configured as an I<sup>2</sup>C master by selecting the I<sup>2</sup>C mode with UCMODEx = 11 and UCSYNC = 1 and setting the UCMST bit. When the master is part of a multi-master system, UCMM must be set and its own address must be programmed into the UCBxI2COA register. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the USCI module responds to a general call.

## I<sup>2</sup>C Master Transmitter Mode

After initialization, master transmitter mode is initiated by writing the desired slave address to the UCBxI2CSA register, selecting the size of the slave address with the UCSLA10 bit, setting UCTR for transmitter mode, and setting UCTXSTT to generate a START condition.

The USCI module checks if the bus is available, generates the START condition, and transmits the slave address. The UCTXIFG bit is set when the START condition is generated and the first data to be transmitted can be written into UCBxTXBUF. As soon as the slave acknowledges the address, the UCTXSTT bit is cleared.

---

### NOTE: Handling of TXIFG in a multi-master system

In a multi-master system (UCMM =1), if the bus is unavailable, the USCI module waits and checks for bus release. Bus unavailability can occur even after the UCTXSTT bit has been set. While waiting for the bus to become available, the USCI may update the TXIFG based on SCL clock line activity. Checking the UCTXSTT bit to verify if the START condition has been sent ensures that the TXIFG is being serviced correctly.

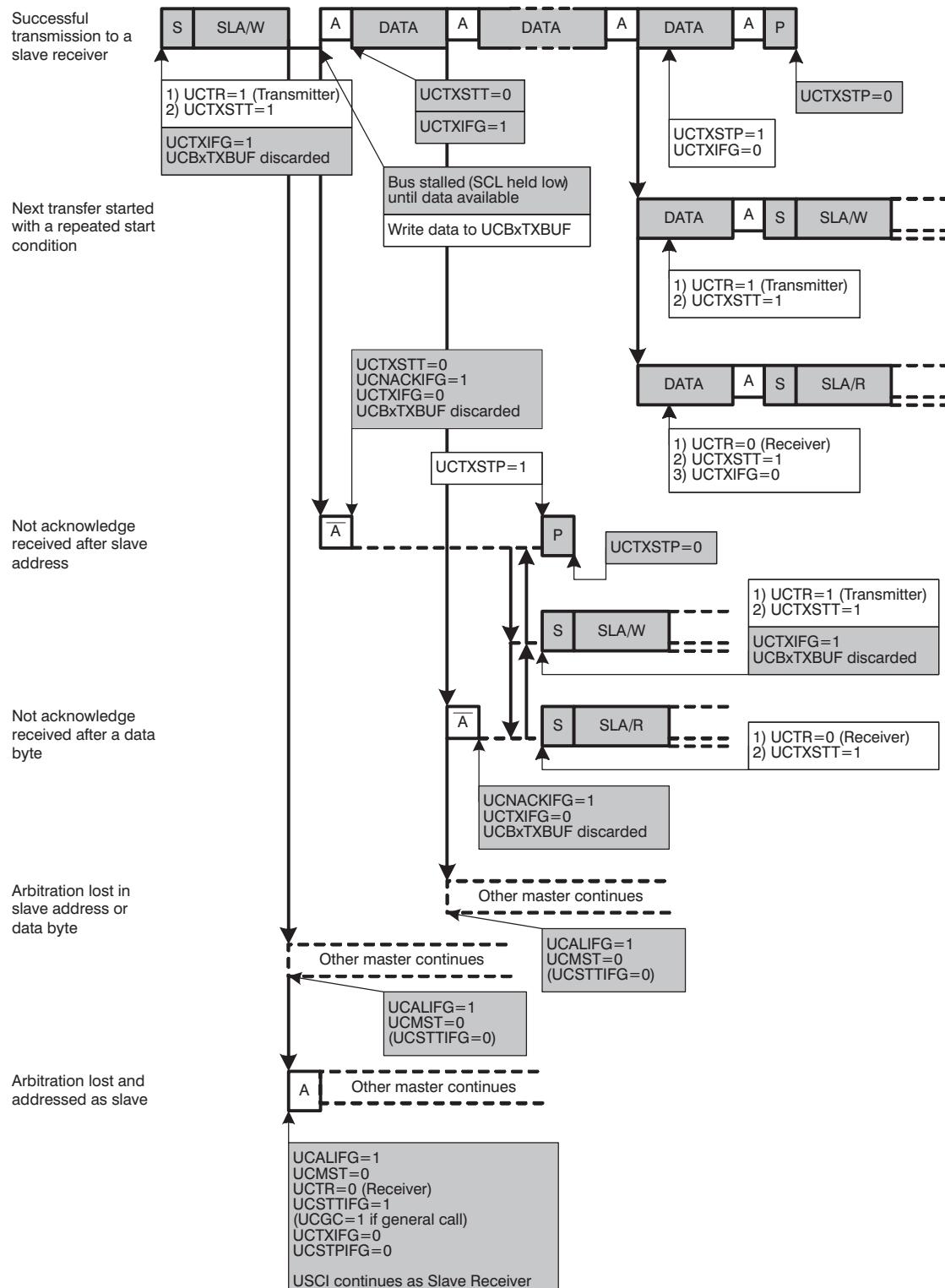
The data written into UCBxTXBUF is transmitted if arbitration is not lost during transmission of the slave address. UCTXIFG is set again as soon as the data is transferred from the buffer into the shift register. If there is no data loaded to UCBxTXBUF before the acknowledge cycle, the bus is held during the acknowledge cycle with SCL low until data is written into UCBxTXBUF. Data is transmitted or the bus is held, as long as the UCTXSTP bit or UCTXSTT bit is not set.

Setting UCTXSTP generates a STOP condition after the next acknowledge from the slave. If UCTXSTP is set during the transmission of the slave's address or while the USCI module waits for data to be written into UCBxTXBUF, a STOP condition is generated, even if no data was transmitted to the slave. When transmitting a single byte of data, the UCTXSTP bit must be set while the byte is being transmitted or anytime after transmission begins, without writing new data into UCBxTXBUF. Otherwise, only the address is transmitted. When the data is transferred from the buffer to the shift register, UCTXIFG is set, indicating data transmission has begun, and the UCTXSTP bit may be set.

Setting UCTXSTT generates a repeated START condition. In this case, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

If the slave does not acknowledge the transmitted data, the not-acknowledge interrupt flag UCNACKIFG is set. The master must react with either a STOP condition or a repeated START condition. If data was already written into UCBxTXBUF, it is discarded. If this data should be transmitted after a repeated START, it must be written into UCBxTXBUF again. Any set UCTXSTT is also discarded. To trigger a repeated START, UCTXSTT must be set again.

Figure 36-12 shows the I<sup>2</sup>C master transmitter operation.


**Figure 36-12. I<sup>2</sup>C Master Transmitter Mode**

**I<sup>2</sup>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 USCI module checks if the bus is available, generates the START condition, and transmits the slave address. As soon as the slave acknowledges the address, the UCTXSTT bit is cleared.

After the acknowledge of the address from the slave, the first data byte from the slave is received and acknowledged and the UCRXIFG flag is set. Data is received from the slave, as long as UCTXSTP or UCTXSTT is not set. If UCBxRXBUF is not read, the master holds the bus during reception of the last data bit and until the UCBxRXBUF is read.

If the slave does not acknowledge the transmitted address, the not-acknowledge interrupt flag UCNACKIFG is set. The master must react with either a STOP condition or a repeated START condition.

Setting the UCTXSTP bit generates a STOP condition. After setting UCTXSTP, a NACK followed by a STOP condition is generated after reception of the data from the slave, or immediately if the USCI module is currently waiting for UCBxRXBUF to be read.

If a master wants to receive a single byte only, the UCTXSTP bit must be set while the byte is being received. For this case, the UCTXSTT may be polled to determine when it is cleared:

```
BIS.B    #UCTXSTT, &UCB0CTL1 ;Transmit START cond.
POLL_STT BIT.B    #UCTXSTT, &UCB0CTL1 ;Poll UCTXSTT bit
        JC      POLL_STT          ;When cleared,
        BIS.B    #UCTXSTP, &UCB0CTL1 ;transmit STOP cond.
```

Setting UCTXSTT generates a repeated START condition. In this case, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

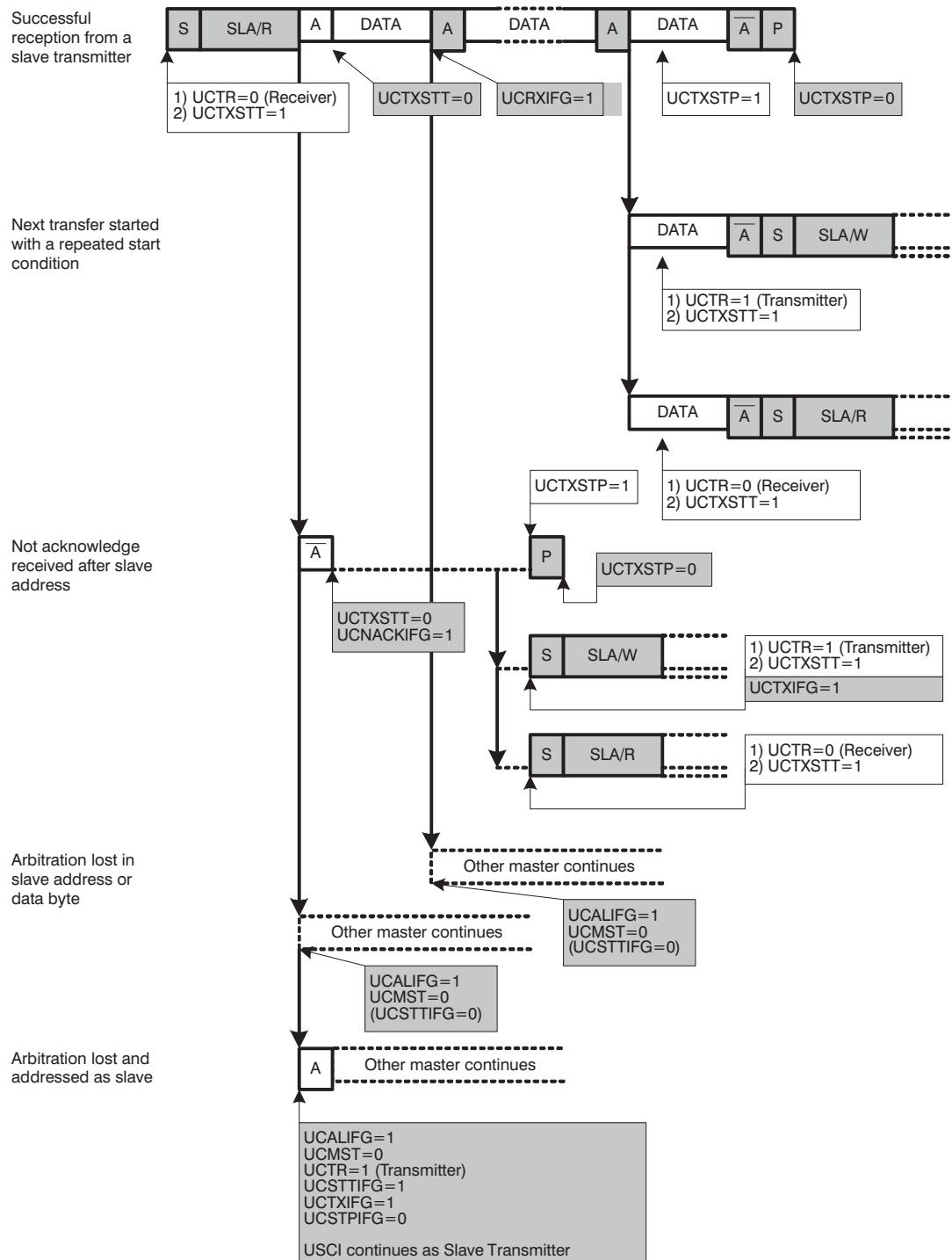
**NOTE: Repeated START**

The UCTXSTT bit must be set before the last data byte is received; that is, immediately after the UCRXIFG is set and the UCRXBUF with the second to last byte is read, the UCTXSTT bit should be set.

**NOTE: Consecutive master transactions without repeated START**

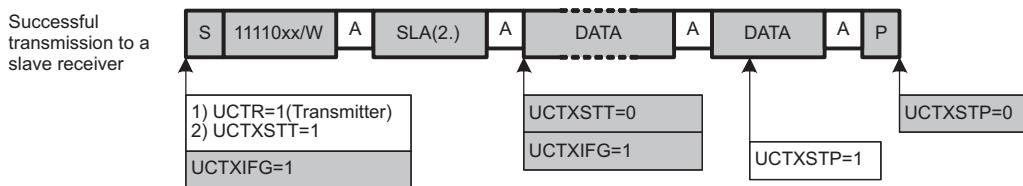
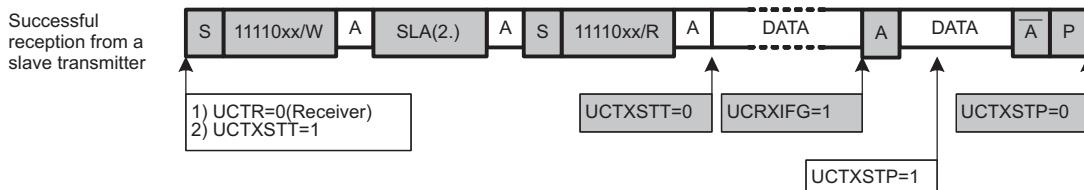
When performing multiple consecutive I<sup>2</sup>C master transactions without the repeated START feature, the current transaction must be completed before the next one is initiated. This can be done by ensuring that the transmit STOP condition flag UCTXSTP is cleared before the next I<sup>2</sup>C transaction is initiated with setting UCTXSTT = 1. Otherwise, the current transaction might be affected.

Figure 36-13 shows the I<sup>2</sup>C master receiver operation.


**Figure 36-13. I<sup>2</sup>C Master Receiver Mode**

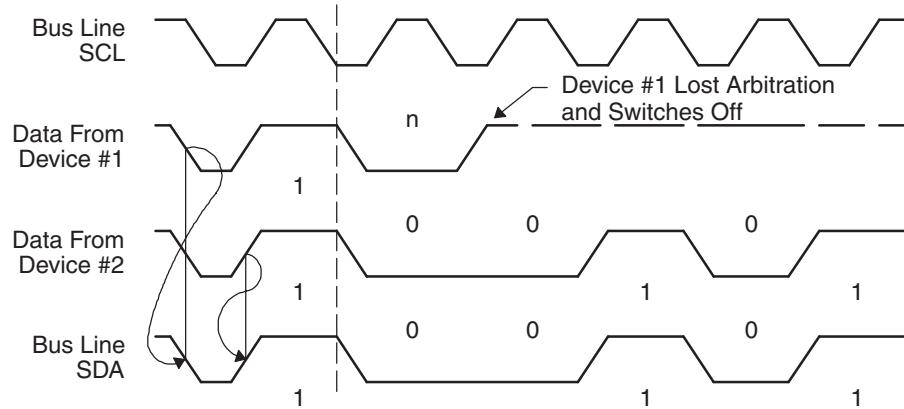
### I<sup>2</sup>C Master 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in [Figure 36-14](#).

**Master Transmitter****Master Receiver****Figure 36-14. I<sup>2</sup>C Master 10-Bit Addressing Mode**

### 36.3.4.3 Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. [Figure 36-15](#) 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 36-15. Arbitration Procedure Between Two Master Transmitters**

If the arbitration procedure is in progress when a repeated START condition or STOP condition is transmitted on SDA, the master transmitters involved in arbitration must send the repeated START condition or STOP condition at the same position in the format frame. Arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

### 36.3.5 I<sup>2</sup>C Clock Generation and Synchronization

The I<sup>2</sup>C clock SCL is provided by the master on the I<sup>2</sup>C bus. When the USCI is in master mode, BITCLK is provided by the USCI bit clock generator and the clock source is selected with the UCSSELx bits. In slave mode, the bit clock generator is not used and the UCSSELx bits are don't care.

The 16-bit value of UCBRx in registers UCBxBR1 and UCBxBR0 is the division factor of the USCI clock source, BRCLK. The maximum bit clock that can be used in single master mode is  $f_{BRCLK}/4$ . In multi-master mode, the maximum bit clock is  $f_{BRCLK}/8$ . The BITCLK frequency is given by:

$$f_{BitClock} = f_{BRCLK}/UCBRx$$

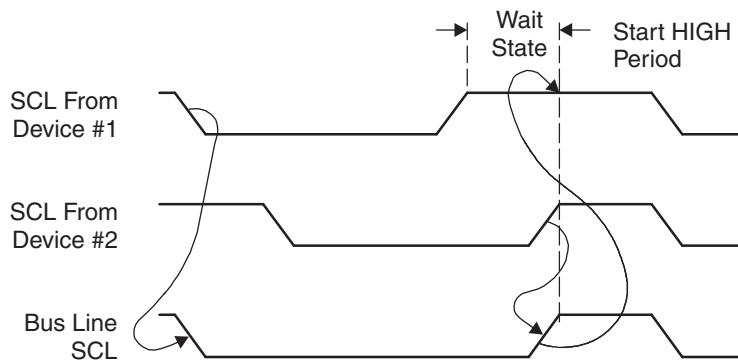
The minimum high and low periods of the generated SCL are:

$$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx/2)/f_{BRCLK} \text{ when UCBRx is even}$$

$$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx - 1/2)/f_{BRCLK} \text{ when UCBRx is odd}$$

The USCI clock source frequency and the prescaler setting UCBRx must be chosen such that the minimum low and high period times of the I<sup>2</sup>C 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 36-16](#) shows the clock synchronization. This allows a slow slave to slow down a fast master.



**Figure 36-16. Synchronization of Two I<sup>2</sup>C Clock Generators During Arbitration**

### 36.3.5.1 Clock Stretching

The USCI module supports clock stretching and also makes use of this feature as described in the Operation Mode sections.

The UCSCLLOW bit can be used to observe if another device pulls SCL low while the USCI module already released SCL due to the following conditions:

- USCI is acting as master and a connected slave drives SCL low.
- USCI is acting as master and another master drives SCL low during arbitration.

The UCSCLLOW bit is also active if the USCI holds SCL low because it is waiting as transmitter for data being written into UCBxTXBUF or as receiver for the data being read from UCBxRXBUF.

The UCSCLLOW bit might get set for a short time with each rising SCL edge because the logic observes the external SCL and compares it to the internally generated SCL.

### 36.3.6 Using the USCI Module in I<sup>2</sup>C Mode With Low-Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

In I<sup>2</sup>C slave mode, no internal clock source is required because the clock is provided by the external master. It is possible to operate the USCI in I<sup>2</sup>C 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.

### 36.3.7 USCI Interrupts in I<sup>2</sup>C Mode

The USCI has only one interrupt vector that is shared for transmission, reception, and the state change. USCI\_Ax and USC\_Bx do not share the same interrupt vector.

Each interrupt flag has its own interrupt enable bit. When an interrupt is enabled and the GIE bit is set, the interrupt flag generates an interrupt request. DMA transfers are controlled by the UCTXIFG and UCRXIFG flags on devices with a DMA controller.

#### 36.3.7.1 I<sup>2</sup>C Transmit Interrupt Operation

The UCTXIFG interrupt flag is set by the transmitter to indicate that UCBxTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCBxTXBUF or if a NACK is received. UCTXIFG is set when UCSWRST = 1 and the I<sup>2</sup>C mode is selected. UCTXIE is reset after a PUC or when UCSWRST = 1.

#### 36.3.7.2 I<sup>2</sup>C Receive Interrupt Operation

The UCRXIFG interrupt flag is set when a character is received and loaded into UCBxRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset after a PUC signal or when UCSWRST = 1. UCRXIFG is automatically reset when UCxRXBUF is read.

### 36.3.7.3 I<sup>2</sup>C State Change Interrupt Operation

[Table 36-1](#) describes the I<sup>2</sup>C state change interrupt flags.

**Table 36-1. I<sup>2</sup>C State Change Interrupt Flags**

Interrupt Flag	Interrupt Condition
UCALIFG	Arbitration-lost. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the USCI operates as master but is addressed as a slave by another master in the system. The UCALIFG flag is set when arbitration is lost. When UCALIFG is set, the UCMST bit is cleared and the I <sup>2</sup> C controller becomes a slave.
UCNACKIFG	Not-acknowledge interrupt. This flag is set when an acknowledge is expected but is not received. UCNACKIFG is automatically cleared when a START condition is received.
UCSTTIIFG	START condition detected interrupt. This flag is set when the I <sup>2</sup> C module detects a START condition together with its own address while in slave mode. UCSTTIIFG is used in slave mode only and is automatically cleared when a STOP condition is received.
UCSTPIFG	STOP condition detected interrupt. This flag is set when the I <sup>2</sup> C module detects a STOP condition while in slave mode. UCSTPIFG is used in slave mode only and is automatically cleared when a START condition is received.

### 36.3.7.4 UCBxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCBxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCBxIV register that can be evaluated or added to the PC to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCBxIV value.

Any access, read or write, of the UCBxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

#### UCBxIV Software Example

The following software example shows the recommended use of UCBxIV. The UCBxIV value is added to the PC to automatically jump to the appropriate routine. The example is given for USCI\_B0.

```
USCI_I2C_ISR
    ADD      &UCB0IV, PC ; Add offset to jump table
    RETI     ; Vector 0: No interrupt
    JMP     ALIFG_ISR   ; Vector 2: ALIFG
    JMP     NACKIFG_ISR ; Vector 4: NACKIFG
    JMP     STTIFG_ISR   ; Vector 6: STTIFG
    JMP     STPIFG_ISR   ; Vector 8: STPIFG
    JMP     RXIFG_ISR   ; Vector 10: RXIFG
    TXIFG_ISR
    ...
    RETI     ; Task starts here
    ; Return
ALIFG_ISR
    ...
    RETI     ; Vector 2
    ; Task starts here
    ; Return
NACKIFG_ISR
    ...
    RETI     ; Vector 4
    ; Task starts here
    ; Return
STTIFG_ISR
    ...
    RETI     ; Vector 6
    ; Task starts here
    ; Return
STPIFG_ISR
    ...
    RETI     ; Vector 8
    ; Task starts here
    ; Return
RXIFG_ISR
    ...
    RETI     ; Vector 10
    ; Task starts here
    ; Return
```

### 36.4 USCI\_B I2C Mode Registers

The USCI registers applicable in I2C mode are listed in [Table 36-2](#). The base address can be found in the device-specific data sheet. The address offsets are listed in [Table 36-2](#).

**Table 36-2. USCI\_B Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	USCI_Bx Control Word 0	Read/write	Word	0101h	
00h	UCBxCTL1	USCI_Bx Control 1	Read/write	Byte	01h	<a href="#">Section 36.4.2</a>
01h	UCBxCTL0	USCI_Bx Control 0	Read/write	Byte	01h	<a href="#">Section 36.4.1</a>
06h	UCBxBRW	USCI_Bx Bit Rate Control Word	Read/write	Word	0000h	
06h	UCBxBR0	USCI_Bx Bit Rate Control 0	Read/write	Byte	00h	<a href="#">Section 36.4.3</a>
07h	UCBxBR1	USCI_Bx Bit Rate Control 1	Read/write	Byte	00h	<a href="#">Section 36.4.4</a>
0Ah	UCBxSTAT	USCI_Bx Status	Read/write	Byte	00h	<a href="#">Section 36.4.5</a>
0Bh		Reserved - reads zero	Read	Byte	00h	
0Ch	UCBxRXBUF	USCI_Bx Receive Buffer	Read/write	Byte	00h	<a href="#">Section 36.4.6</a>
0Dh		Reserved - reads zero	Read	Byte	00h	
0Eh	UCBxTXBUF	USCI_Bx Transmit Buffer	Read/write	Byte	00h	<a href="#">Section 36.4.7</a>
0Fh		Reserved - reads zero	Read	Byte	00h	
10h	UCBxI2COA	USCI_Bx I2C Own Address	Read/write	Word	0000h	<a href="#">Section 36.4.8</a>
12h	UCBxI2CSA	USCI_Bx I2C Slave Address	Read/write	Word	0000h	<a href="#">Section 36.4.9</a>
1Ch	UCBxICTL	USCI_Bx Interrupt Control	Read/write	Word	0200h	
1Ch	UCBxIE	USCI_Bx Interrupt Enable	Read/write	Byte	00h	<a href="#">Section 36.4.10</a>
1Dh	UCBxIFG	USCI_Bx Interrupt Flag	Read/write	Byte	02h	<a href="#">Section 36.4.11</a>
1Eh	UCBxIV	USCI_Bx Interrupt Vector	Read	Word	0000h	<a href="#">Section 36.4.12</a>

### 36.4.1 UCBxCTL0 Register

USCI\_Bx Control Register 0

**Figure 36-17. UCBxCTL0 Register**

7	6	5	4	3	2	1	0
UCA10	UCSLA10	UCMM	Reserved	UCMST	UCMODEx	UCSYNC	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-1

Modify only when UCSWRST = 1.

**Table 36-3. UCBxCTL0 Register Description**

Bit	Field	Type	Reset	Description
7	UCA10	RW	0h	Own addressing mode select 0b = Own address is a 7-bit address 1b = Own address is a 10-bit address
6	UCSLA10	RW	0h	Slave addressing mode select 0b = Address slave with 7-bit address 1b = Address slave with 10-bit address
5	UCMM	RW	0h	Multi-master environment select 0b = Single master environment. There is no other master in the system. The address compare unit is disabled. 1b = Multi-master environment
4	Reserved	R	0h	Reserved. Always reads as 0.
3	UCMST	RW	0h	Master mode select. When a master loses arbitration in a multi-master environment (UCMM = 1), the UCMST bit is automatically cleared and the module acts as slave. 0b = Slave mode 1b = Master mode
2-1	UCMODEx	RW	0h	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI (master/slave enabled if STE = 1) 10b = 4-pin SPI (master/slave enabled if STE = 0) 11b = I <sup>2</sup> C mode
0	UCSYNC	R	1h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode

**36.4.2 UCBxCTL1 Register**

USCI\_Bx Control Register 1

**Figure 36-18. UCBxCTL1 Register**

7	6	5	4	3	2	1	0
UCSSELx	Reserved	UCTR	UCTXNACK	UCTXSTP	UCTXSTT	UCSWRST	
rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when UCSWRST = 1.

**Table 36-4. UCBxCTL1 Register Description**

Bit	Field	Type	Reset	Description
7-6	UCSSELx	RW	0h	USCI clock source select. These bits select the BRCLK source clock. 00b = UCLKI 01b = ACLK 10b = SMCLK 11b = SMCLK
5	Reserved	RW	0h	Reserved. Always reads as 0.
4	UCTR	RW	0h	Transmitter or receiver 0b = Receiver 1b = Transmitter
3	UCTXNACK	RW	0h	Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted. 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. 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 0b = Disabled. USCI reset released for operation. 1b = Enabled. USCI logic held in reset state.

### 36.4.3 UCBxBR0 Register

USCI\_Bx Baud Rate Control Register 0

**Figure 36-19. UCBxBR0 Register**

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw
Modify only when UCSWRST = 1.							

**Table 36-5. UCBxBR0 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Bit clock prescaler low byte. The 16-bit value of (UCxxBR0 + UCxxBR1 × 256) forms the prescaler value UCBRx.

### 36.4.4 UCBxBR1 Register

USCI\_Bx Baud Rate Control Register 1

**Figure 36-20. UCBxBR1 Register**

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw
Modify only when UCSWRST = 1.							

**Table 36-6. UCBxBR1 Register Description**

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Bit clock prescaler high byte. The 16-bit value of (UCxxBR0 + UCxxBR1 × 256) forms the prescaler value UCBRx.

**36.4.5 UCBxSTAT Register**

USCI\_Bx Status Register

**Figure 36-21. UCBxSTAT Register**

7	6	5	4	3	2	1	0
Reserved	UCSCLLOW	UCGC	UCBBUSY		Reserved		
rw-0	r-0	rw-0	r-0	r0	r0	r0	r0

**Table 36-7. UCBxSTAT Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	RW	0h	Reserved. Always reads as 0.
6	UCSCLLOW	R	0h	SCL low 0b = SCL is not held low. 1b = SCL is held low.
5	UCGC	RW	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. Always reads as 0.

### 36.4.6 UCBxRXBUF Register

USCI\_Bx Receive Buffer Register

**Figure 36-22. UCBxRXBUF Register**

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

**Table 36-8. UCBxRXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCRXBUFx	R	undefined	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCBxRXBUF resets UCRXIFG.

### 36.4.7 UCBxTXBUF Register

USCI\_Bx Transmit Buffer Register

**Figure 36-23. UCBxTXBUF Register**

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

**Table 36-9. UCBxTXBUF Register Description**

Bit	Field	Type	Reset	Description
7-0	UCTXBUFx	RW	undefined	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.

### 36.4.8 UCBxI2COA Register

USCIBx I2C Own Address Register

**Figure 36-24. UCBxI2COA Register**

15	14	13	12	11	10	9	8
UCGCEN		Reserved				I2COAx	
rw-0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2COAx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
Modify only when UCSWRST = 1.							

**Table 36-10. UCBxI2COA Register Description**

Bit	Field	Type	Reset	Description
15	UCGCEN	RW	0h	General call response enable 0b = Do not respond to a general call 1b = Respond to a general call
14-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	I2COAx	RW	0h	I2C own address. The I2COAx bits contain the local address of the USCI_Bx I2C controller. The address is right justified. In 7-bit addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit addressing mode, bit 9 is the MSB.

### 36.4.9 UCBxI2CSA Register

USCI\_Bx I2C Slave Address Register

**Figure 36-25. UCBxI2CSA Register**

15	14	13	12	11	10	9	8
		Reserved				I2CSAx	
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2CSAx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 36-11. UCBxI2CSA Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	I2CSAx	RW	0h	I2C slave address. The I2CSAx bits contain the slave address of the external device to be addressed by the USCI_Bx module. It is only used in master mode. The address is right justified. In 7-bit slave addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit slave addressing mode, bit 9 is the MSB.

### **36.4.10 UCBxIE Register**

USCI\_Bx I2C Interrupt Enable Register

**Figure 36-26. UCBxIE Register**

7	6	5	4	3	2	1	0
Reserved		UCNACKIE	UCALIE	UCSTPIE	UCSTTIE	UCTXIE	UCRXIE
r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

**Table 36-12. UCBxIE Register Description**

Bit	Field	Type	Reset	Description
7-6	Reserved	R	0h	Reserved. Always reads as 0.
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
2	UCSTTIE	RW	0h	START condition 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

**36.4.11 UCBxIFG Register**

USCI\_Bx I2C Interrupt Flag Register

**Figure 36-27. UCBxIFG Register**

7	6	5	4	3	2	1	0
Reserved		UCNACKIFG	UCALIFG	UCSTPIFG	UCSTTIFG	UCTXIFG	UCRXIFG
r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

**Table 36-13. UCBxIFG Register Description**

Bit	Field	Type	Reset	Description
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5	UCNACKIFG	RW	0h	Not-acknowledge received interrupt flag. UCNACKIFG is automatically cleared when a START condition is received. 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. UCSTPIFG is automatically cleared when a START condition is received. 0b = No interrupt pending 1b = Interrupt pending
2	UCSTTIFG	RW	0h	START condition interrupt flag. UCSTTIFG is automatically cleared if a STOP condition is received. 0b = No interrupt pending 1b = Interrupt pending
1	UCTXIFG	RW	0h	USCI transmit interrupt flag. UCTXIFG is set when UCBxTXBUF is empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	USCI receive interrupt flag. UCRXIFG is set when UCBxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

### 36.4.12 UCBxIV Register

USCI\_Bx Interrupt Vector Register

**Figure 36-28. 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 36-14. UCBxIV Register Description**

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	USCI interrupt vector value 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: Data received; Interrupt Flag: UCRXIFG 0Ch = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest

## ***Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode***

---



---



---

The enhanced universal serial communication interface A (eUSCI\_A) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode.

Topic	Page
37.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview .....	981
37.2 eUSCI_A Introduction – UART Mode .....	981
37.3 eUSCI_A Operation – UART Mode .....	983
37.4 eUSCI_A UART Registers .....	998

## 37.1 Enhanced Universal Serial Communication Interface A (eUSCI\_A) Overview

The eUSCI\_A module supports two serial communication modes:

- UART mode
- SPI mode

## 37.2 eUSCI\_A Introduction – UART Mode

In asynchronous mode, the eUSCI\_Ax modules connect the device to an external system via two external pins, UCAxRXD and UCAxTXD. UART mode is selected when the UCSYNC bit is cleared.

UART mode features include:

- 7-bit or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first or MSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto wake up from LPMx modes (wake up from LPMx.5 is not supported)
- Programmable baud rate with modulation for fractional baud-rate support
- Status flags for error detection and suppression
- Status flags for address detection
- Independent interrupt capability for receive, transmit, start bit received, and transmit complete

Figure 37-1 shows the eUSCI\_Ax when configured for UART mode.

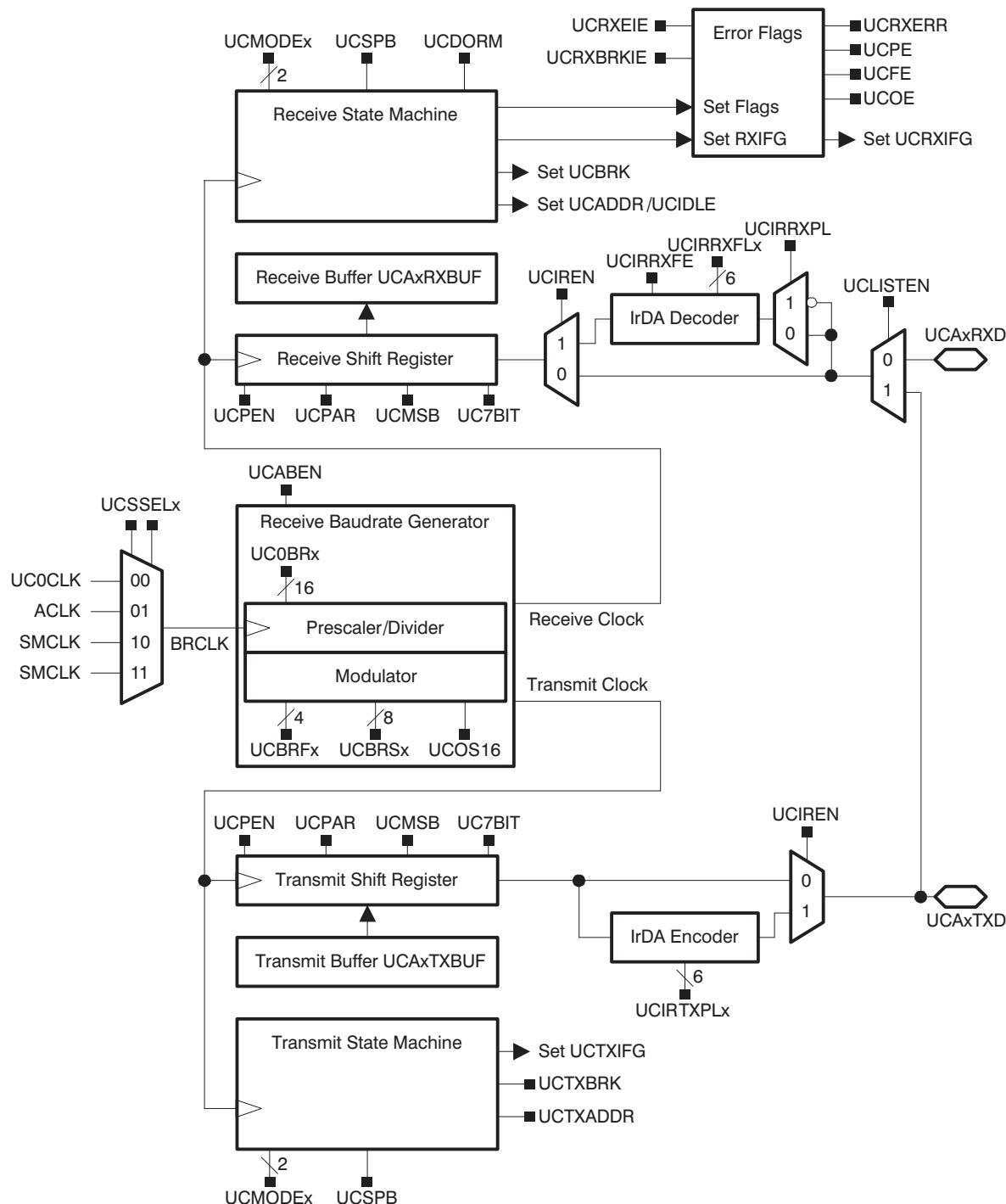


Figure 37-1. eUSCI\_Ax Block Diagram – UART Mode (UCSYNC = 0)

### 37.3 eUSCI\_A Operation – UART Mode

In UART mode, the eUSCI\_A transmits and receives characters at a bit rate asynchronous to another device. Timing for each character is based on the selected baud rate of the eUSCI\_A. The transmit and receive functions use the same baud-rate frequency.

#### 37.3.1 eUSCI\_A Initialization and Reset

The eUSCI\_A is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI\_A in a reset condition. When set, the UCSWRST bit sets the UCTXIFG bit and resets the UCRXIE, UCTXIE, UCRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE, and UCBTOE bits. Clearing UCSWRST releases the eUSCI\_A for operation.

Configuring and reconfiguring the eUSCI\_A module should be done when UCSWRST is set to avoid unpredictable behavior.

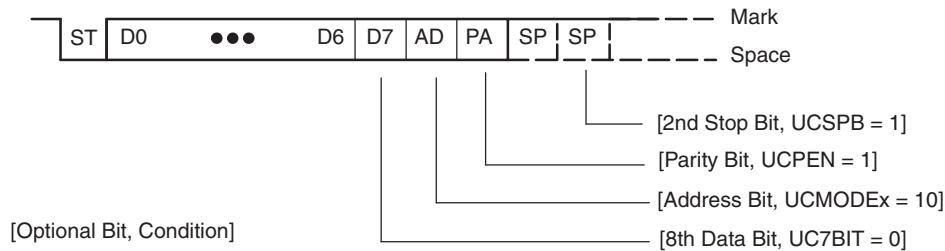
**NOTE: Initializing or reconfiguring the eUSCI\_A module**

The recommended eUSCI\_A initialization/reconfiguration process is:

1. Set UCSWRST (**BIS.B**  
#UCSWRST, &UCAxCTL1).
2. Initialize all eUSCI\_A registers with UCSWRST = 1 (including UCAxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (**BIC.B**  
#UCSWRST, &UCAxCTL1).
5. Enable interrupts (optional) via UCRXIE or UCTXIE.

#### 37.3.2 Character Format

The UART character format (see [Figure 37-2](#)) consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first. LSB first is typically required for UART communication.



**Figure 37-2. Character Format**

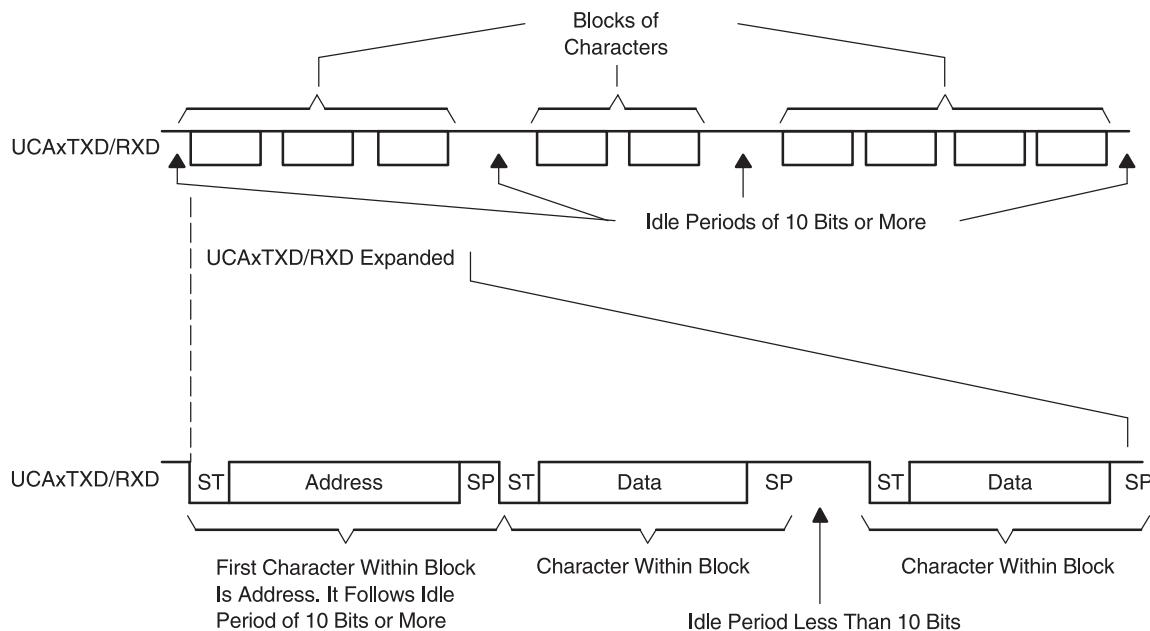
#### 37.3.3 Asynchronous Communication Format

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.

##### 37.3.3.1 Idle-Line Multiprocessor Format

When UCMODEEx = 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 37-3](#)). An idle receive line is detected when ten or more continuous ones (marks) are received after the one or two stop bits of a character. The baud-rate generator is switched off after reception of an idle line until the next start edge is detected. When an idle line is detected, the UCIDLE bit is set.

The first character received after an idle period is an address character. The UCIDLE bit is used as an address tag for each block of characters. In idle-line multiprocessor format, this bit is set when a received character is an address.



**Figure 37-3. Idle-Line Format**

The UCDORM bit is used to control data reception in the idle-line multiprocessor format. When UCDORM = 1, all non-address characters are assembled but not transferred into the UCAxRXBUF, and interrupts are not generated. When an address character is received, the character is transferred into UCAxRXBUF, UCRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and an address character is received but has a framing error or parity error, the character is not transferred into UCAxRXBUF and UCRXIFG is not set.

If an address is received, user software can validate the address and must reset UCDORM to continue receiving data. If UCDORM remains set, only address characters are received. When UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception completed. The UCDORM bit is not modified automatically by the eUSCI\_A hardware.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the eUSCI\_A to generate address character identifiers on UCAxTXD. The double-buffered UCTXADDR flag indicates if the next character loaded into UCAxTXBUF is preceded by an idle line of 11 bits. UCTXADDR is automatically cleared when the start bit is generated.

### 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.

### 37.3.3.2 Address-Bit Multiprocessor Format

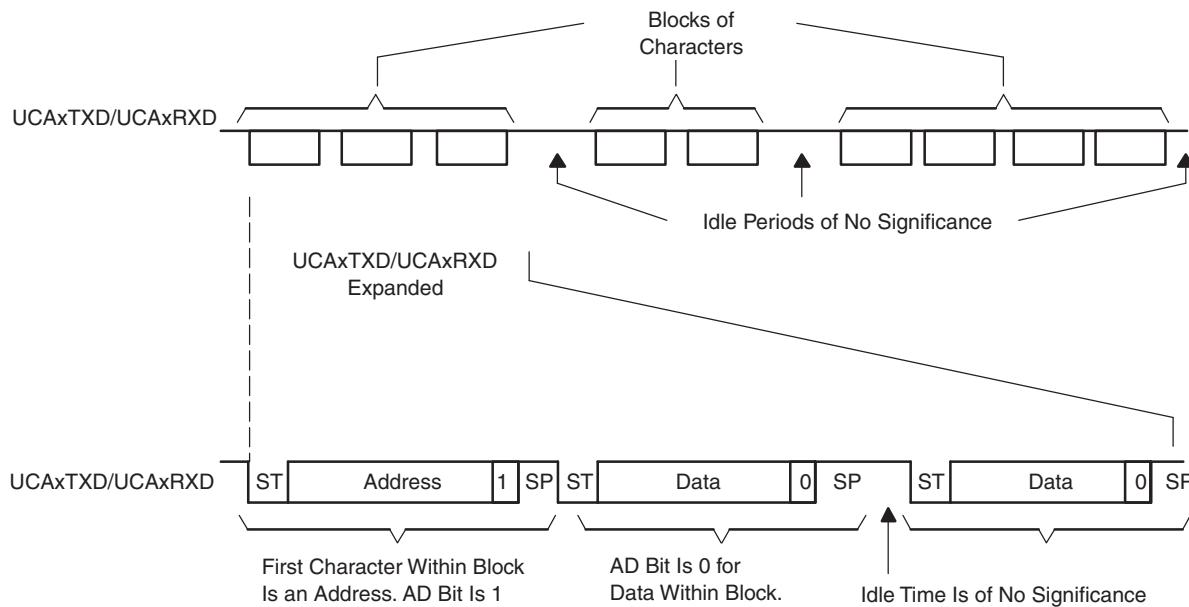
When UCMODEx = 10, the address-bit multiprocessor format is selected. Each processed character contains an extra bit used as an address indicator (see [Figure 37-4](#)). The first character in a block of characters carries a set address bit that indicates that the character is an address. The eUSCI\_A 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 37-4. Address-Bit Multiprocessor Format**

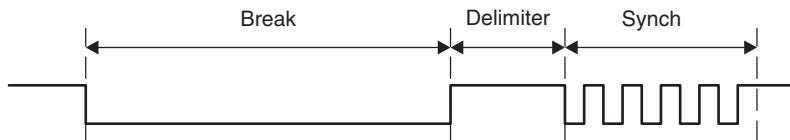
### Break Reception and Generation

When UCMODEx = 00, 01, or 10, the receiver detects a break when all data, parity, and stop bits are low, regardless of the parity, address mode, or other character settings. When a break is detected, the UCBRK bit is set. If the break interrupt enable bit (UCBRKIE) is set, the receive interrupt flag UCRXIFG is also set. In this case, the value in UCAXRXBUF is 0h, because all data bits were zero.

To transmit a break, set the UCTXBRK bit, then write 0h to UCAXTXBUF. UCAXTXBUF must be ready for new data (UCTXIFG = 1). This generates a break with all bits low. UCTXBRK is automatically cleared when the start bit is generated.

### 37.3.4 Automatic Baud-Rate Detection

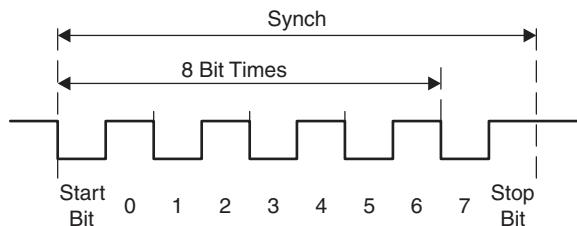
When UCMODE<sub>x</sub> = 11, UART mode with automatic baud-rate detection is selected. For automatic baud-rate detection, a data frame is preceded by a synchronization sequence that consists of a break and a synch field. A break is detected when 11 or more continuous zeros (spaces) are received. If 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 37-5](#).



**Figure 37-5. 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 37-6](#)). 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 37-6. Auto Baud-Rate Detection – Synch Field**

The UCDORM bit is used to control data reception in this mode. When UCDORM is set, all characters are received but not transferred into the UCAxRXBUF, and interrupts are not generated. When a break/synch field is detected, the UCBRK flag is set. The character following the break/synch field is transferred into UCAxRXBUF and the UCRXIFG interrupt flag is set. Any applicable error flag is also set. If the UCBRKIE bit is set, reception of the break/synch sets the UCRXIFG. The UCBRK bit is reset by user software or by reading the receive buffer UCAxRXBUF.

When a break/synch field is received, user software must reset UCDORM to continue receiving data. If UCDORM remains set, only the character after the next reception of a break/synch field is received. The UCDORM bit is not modified by the eUSCI\_A hardware automatically.

When UCDORM = 0, all received characters set the receive interrupt flag UCRXIFG. If UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception is complete.

The counter used to detect the baud rate is limited to 0FFFFh ( $2^{16}$ ) counts. This means the minimum baud rate detectable is 244 baud in oversampling mode and 15 baud in low-frequency mode. The highest detectable baudrate is 1 Mbaud.

The automatic baud-rate detection mode can be used in a full-duplex communication system with some restrictions. The eUSCI\_A cannot transmit data while receiving the break/sync field and, if a 0h byte with framing error is received, any data transmitted during this time is corrupted. The latter case can be discovered by checking the received data and the UCPE bit.

### 37.3.4.1 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.

### 37.3.5 IrDA Encoding and Decoding

When UCIREN is set, the IrDA encoder and decoder are enabled and provide hardware bit shaping for IrDA communication.

#### 37.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART (see Figure 37-7). The pulse duration is defined by UCIRTXPLx bits specifying the number of one-half clock periods of the clock selected by UCIRTXCLK.

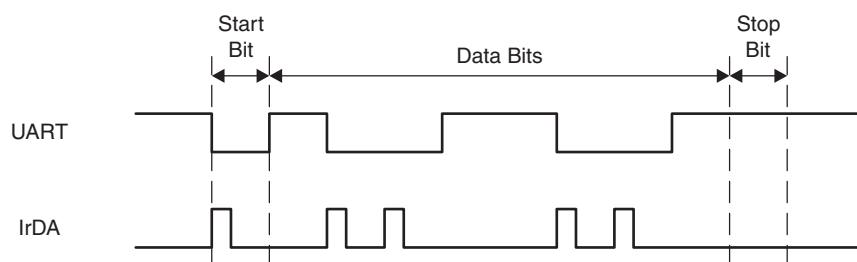


Figure 37-7. UART vs IrDA Data Format

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:

$$\text{UCIRTXPLx} = t_{PULSE} \times 2 \times f_{BRCLK} - 1$$

When UCIRTXCLK = 0, the prescaler UCBRx must be set to a value greater or equal to 5.

#### 37.3.5.2 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:

$$\text{UCIRRXFLx} = (t_{PULSE} - t_{WAKE}) \times 2 \times f_{BRCLK} - 4$$

Where:

$t_{PULSE}$  = Minimum receive pulse width

$t_{WAKE}$  = Wake time from any low-power mode. Zero when the device is in active mode.

### 37.3.6 Automatic Error Detection

Glitch suppression prevents the eUSCI\_A from being accidentally started. Any pulse on UC<sub>A</sub>xRXD shorter than the deglitch time  $t_g$  (selected by UCGLITx) is ignored (see the device-specific data sheet for parameters).

When a low period on UC<sub>A</sub>xRXD exceeds  $t_g$ , 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 UC<sub>A</sub>xRXD. 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 37-1](#).

**Table 37-1. 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 UC <sub>A</sub> xRXBUF 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 UC<sub>A</sub>xRXBUF. When UCRXEIE = 1, characters are received into UC<sub>A</sub>xRXBUF 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 UC<sub>A</sub>xRXBUF is read. UCOE must be reset by reading UC<sub>A</sub>xRXBUF. Otherwise, it does not function properly. To detect overflows reliably, the following flow is recommended. After a character is received and UC<sub>A</sub>xRXIFG is set, first read UC<sub>A</sub>xSTATW to check the error flags including the overflow flag UCOE. Read UC<sub>A</sub>xRXBUF next. This clears all error flags except UCOE, if UC<sub>A</sub>xRXBUF was overwritten between the read access to UC<sub>A</sub>xSTATW and to UC<sub>A</sub>xRXBUF. Therefore, the UCOE flag should be checked after reading UC<sub>A</sub>xRXBUF to detect this condition. Note that, in this case, the UCRXERR flag is not set.

### 37.3.7 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.

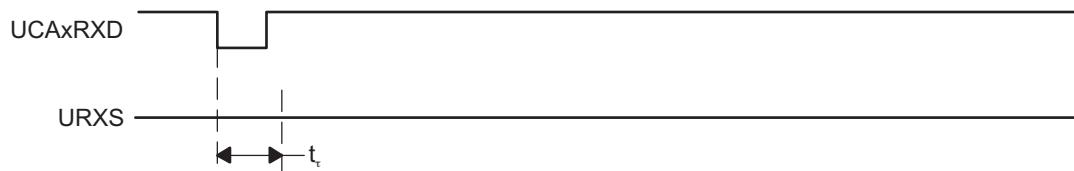
The falling edge of the start bit enables the baud rate generator and the UART state machine checks for a valid start bit. If no valid start bit is detected the UART state machine returns to its idle state and the baud rate generator is turned off again. If a valid start bit is detected, a character 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.

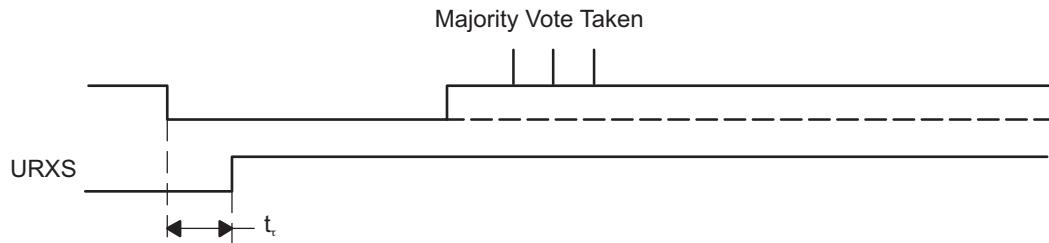
#### 37.3.7.1 Receive Data Glitch Suppression

Glitch suppression prevents the eUSCI\_A from being accidentally started. Any glitch on UCAXRXD shorter than the deglitch time  $t_g$  is ignored by the eUSCI\_A, and further action is initiated as shown in [Figure 37-8](#) (see the device-specific data sheet for parameters). The deglitch time  $t_g$  can be set to four different values using the UCGLITx bits.



**Figure 37-8. Glitch Suppression, eUSCI\_A Receive Not Started**

When a glitch is longer than  $t_g$  or a valid start bit occurs on UCAXRXD, the eUSCI\_A receive operation is started and a majority vote is taken (see [Figure 37-9](#)). If the majority vote fails to detect a start bit, the eUSCI\_A halts character reception.



**Figure 37-9. Glitch Suppression, eUSCI\_A Activated**

### 37.3.8 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.

### 37.3.9 UART Baud-Rate Generation

The eUSCI\_A baud-rate generator is capable of producing standard baud rates from nonstandard source frequencies. It provides two modes of operation selected by the UCOS16 bit.

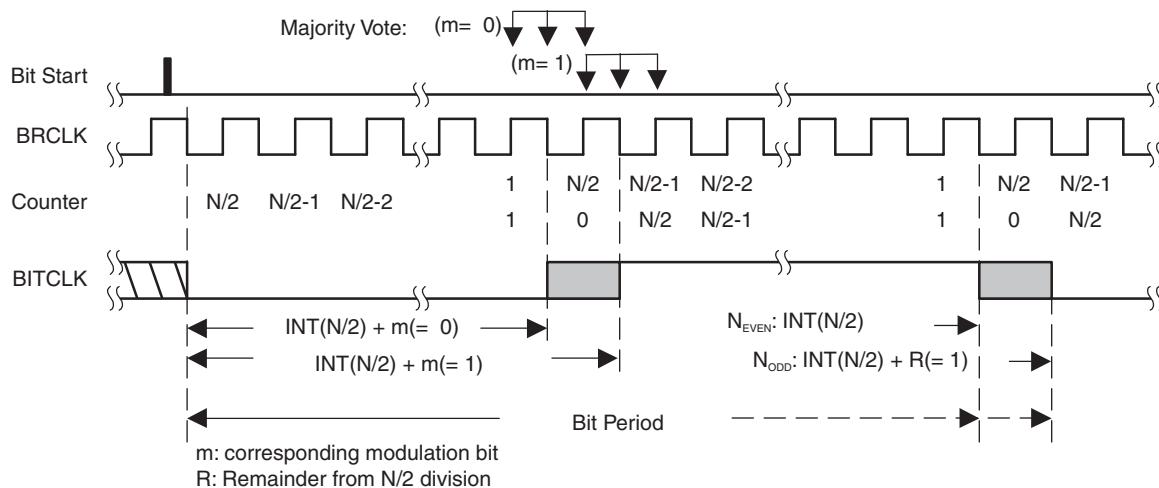
A quick setup for finding the correct baudrate settings for the eUSCI\_A can be found in [Section 37.3.10](#).

#### 37.3.9.1 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 37-10](#). 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 37-10. BITCLK Baud-Rate Timing With UCOS16 = 0**

Modulation is based on the UCBRSx setting as shown in [Table 37-2](#). 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 37-2. 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 37.3.10](#).

### 37.3.9.2 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 37-3](#)). 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 37-3. 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	0	1	1
07h	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1
08h	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
09h	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1
0Ah	0	1	1	1	1	1	0	0	0	0	0	0	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

### 37.3.10 Setting a Baud Rate

For a given BRCLK clock source, the baud rate used determines the required division factor N:

$$N = f_{BRCLK}/\text{Baudrate}$$

The division factor N is often a noninteger value, thus, at least one divider and one modulator stage is used to meet the factor as closely as possible.

If N is equal or greater than 16, it is recommended to use the oversampling baud-rate generation mode by setting UCOS16.

**NOTE: Baudrate settings quick set up**

To calculate the correct settings for the baudrate generation, perform these steps:

1. Calculate  $N = f_{BRCLK}/\text{Baudrate}$  [if  $N > 16$  continue with step 3, otherwise with step 2]
2.  $OS16 = 0$ ,  $UCBRx = \text{INT}(N)$  [continue with step 4]
3.  $OS16 = 1$ ,  $UCBRx = \text{INT}(N/16)$ ,  $UCBRFx = \text{INT}([(N/16) - \text{INT}(N/16)] \times 16)$
4.  $UCBRSx$  can be found by looking up the fractional part of N ( $= N - \text{INT}(N)$ ) in table [Table 37-4](#)
5. If  $OS16 = 0$  was chosen, a detailed error calculation is recommended to be performed

[Table 37-4](#) 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 37-4. UCBRSx Settings for Fractional Portion of  $N = f_{BRCLK}/\text{Baudrate}$**

Fractional Portion of N	UCBRSx <sup>(1)</sup>	Fractional Portion of N	UCBRSx <sup>(1)</sup>
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

<sup>(1)</sup> The UCBRSx setting in one row is valid from the fractional portion given in that row until the one in the next row

#### 37.3.10.1 Low-Frequency Baud-Rate Mode Setting

In low-frequency mode, the integer portion of the divisor is realized by the prescaler:

$$UCBRx = \text{INT}(N)$$

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 37-5](#)).

### **37.3.10.2 Oversampling Baud-Rate Mode Setting**

In the oversampling mode, the prescaler is set to:

$$\text{UCBRx} = \text{INT}(N/16)$$

and the first stage modulator is set to:

$$\text{UCBRFx} = \text{INT}([(N/16) - \text{INT}(N/16)] \times 16)$$

The second modulation stage setting (UCBRSx) can be found by performing a detailed error calculation or by using [Table 37-4](#) and the fractional part of  $N = f_{\text{BRCLK}}/\text{Baudrate}$ .

### **37.3.11 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.

#### **37.3.11.1 Low-Frequency Baud-Rate Mode Bit Timing**

In low-frequency mode, calculation of the length of bit  $i$   $T_{\text{bit,TX}}[i]$  is based on the UCBRx and UCBRSx settings:

$$T_{\text{bit,TX}}[i] = (1/f_{\text{BRCLK}})(\text{UCBRx} + m_{\text{UCBRSx}}[i])$$

Where:

$m_{\text{UCBRSx}}[i]$  = Modulation of bit  $i$  of UCBRSx

#### **37.3.11.2 Oversampling Baud-Rate Mode Bit Timing**

In oversampling baud-rate mode, calculation of the length of bit  $i$   $T_{\text{bit,TX}}[i]$  is based on the baud-rate generator UCBRx, UCBRFx and UCBRSx settings:

$$T_{\text{bit,TX}}[i] = \frac{1}{f_{\text{BRCLK}}} \left( (16 * \text{UCBRx}) + \sum_{j=0}^{15} m_{\text{UCBRFx}}[j] + m_{\text{UCBRSx}}[i] \right)$$

Where:

$$\sum_{j=0}^{15} m_{\text{UCBRFx}}[j] = \text{Sum of ones from the corresponding row in } \text{Table 37-3}$$

$m_{\text{UCBRSx}}[i]$  = Modulation of bit  $i$  of UCBRSx

This results in an end-of-bit time  $t_{\text{bit,TX}}[i]$  equal to the sum of all previous and the current bit times:

$$T_{\text{bit,TX}}[i] = \sum_{j=0}^i T_{\text{bit,TX}}[j]$$

To calculate bit error, this time is compared to the ideal bit time  $t_{\text{bit,ideal,TX}}[i]$ :

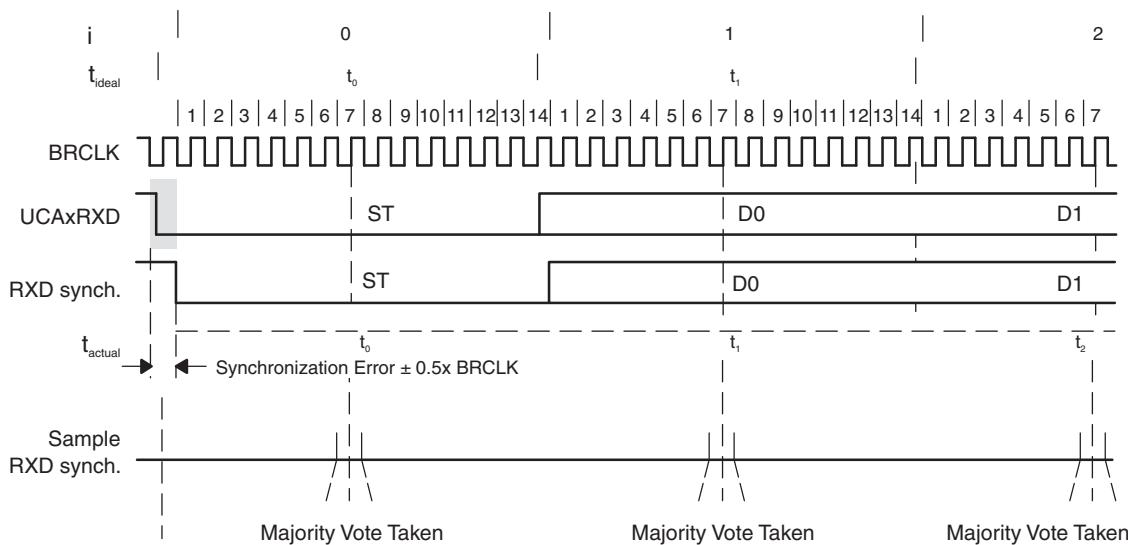
$$t_{\text{bit,ideal,TX}}[i] = (1/\text{Baudrate})(i + 1)$$

This results in an error normalized to one ideal bit time (1/baudrate):

$$\text{Error}_{\text{TX}}[i] = (t_{\text{bit,TX}}[i] - t_{\text{bit,ideal,TX}}[i]) \times \text{Baudrate} \times 100\%$$

### **37.3.12 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 37-11](#) 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  $t_{\text{SYNC}}$  is between  $-0.5$  BRCLKs and  $+0.5$  RCLKs, independent of the selected baud-rate generation mode.

**Figure 37-11. Receive Error**

The ideal sampling time  $t_{bit,ideal,RX}[i]$  is in the middle of a bit period:

$$t_{bit,ideal,RX}[i] = (1/\text{Baudrate})(i + 0.5)$$

The real sampling time,  $t_{bit,RX}[i]$ , is equal to the sum of all previous bits according to the formulas shown in the transmit timing section, plus one-half BITCLK for the current bit  $i$ , plus the synchronization error  $t_{SYNC}$ .

This results in the following  $t_{bit,RX}[i]$  for the low-frequency baud-rate mode:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left( \text{INT}(\frac{1}{2}\text{UCBRx}) + m_{UCBRSx}[i] \right)$$

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:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left( (8 * \text{UCBRx}) + \sum_{j=0}^7 m_{UCBRFx}[j] + m_{UCBRSx}[i] \right)$$

Where:

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}} \left( (16 * \text{UCBRx}) + \sum_{j=0}^{15} m_{UCBRFx}[j] + m_{UCBRSx}[i] \right)$$

$\sum_{j=0}^{7 + m_{UCBRSx}[i]} m_{UCBRFx}[j]$  = Sum of ones from columns 0 to  $(7 + m_{UCBRSx}[i])$  from the corresponding row in [Table 37-3](#).

$m_{UCBRSx}[i]$  = Modulation of bit  $i$  of UCBRSx

This results in an error normalized to one ideal bit time (1/baudrate) according to the following formula:

$$\text{Error}_{RX}[i] = (t_{bit,RX}[i] - t_{bit,ideal,RX}[i]) \times \text{Baudrate} \times 100\%$$

### 37.3.13 Typical Baud Rates and Errors

Standard baud-rate data for UCBRx, UCBSRx, and UCBRFx are listed in [Table 37-5](#) 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 37-5. Recommended Settings for Typical Crystals and Baudrates**

BRCLK	Baudrate	UCOS16	UCBRx	UCFx	UCSx	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 37-5. Recommended Settings for Typical Crystals and Baudrates (continued)**

BRCLK	Baudrate	UCOS16	UCBRx	UCFx	UCSx	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

### 37.3.14 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.

### 37.3.15 eUSCI\_A Interrupts

The eUSCI\_A has only one interrupt vector that is shared for transmission and for reception.

#### 37.3.15.1 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.

### 37.3.15.2 eUSCI\_A Receive Interrupt Operation

The UCRXIFG interrupt flag is set each time a character is received and loaded into UCAXRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCRXIFG is automatically reset when UCAXRXBUF is read.

Additional interrupt control features include:

- When UCAXRXEIE = 0, erroneous characters do not set UCRXIFG.
- When UCDORM = 1, nonaddress characters do not set UCRXIFG in multiprocessor modes. In plain UART mode, no characters are set UCRXIFG.
- When UCBRKIE = 1, a break condition sets the UCBRK bit and the UCRXIFG flag.

### 37.3.15.3 eUSCI\_A Receive Interrupt Operation

Table 37-6 describes the I<sup>2</sup>C state change interrupt flags.

**Table 37-6. 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.

### 37.3.15.4 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 37-1 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 37-1. UCAXIV Software Example

```
#pragma vector = USCI_A0_VECTOR __interrupt void USCI_A0_ISR(void) {
    switch(__even_in_range(UCAXIV,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;
    }
}
```

### 37.4 eUSCI\_A UART Registers

The eUSCI\_A registers applicable in UART mode and their address offsets are listed in [Table 37-7](#). The base address can be found in the device-specific data sheet.

**Table 37-7. eUSCI\_A UART Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	eUSCI_Ax Control Word 0	Read/write	Word	0001h	<a href="#">Section 37.4.1</a>
01h	UCAxCTL0 <sup>(1)</sup>	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	<a href="#">Section 37.4.2</a>
06h	UCAxBRW	eUSCI_Ax Baud Rate Control Word	Read/write	Word	0000h	<a href="#">Section 37.4.3</a>
06h	UCAxBR0 <sup>(1)</sup>	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	<a href="#">Section 37.4.4</a>
0Ah	UCAxSTATW	eUSCI_Ax Status	Read/write	Word	00h	<a href="#">Section 37.4.5</a>
0Ch	UCAxRXBUF	eUSCI_Ax Receive Buffer	Read/write	Word	00h	<a href="#">Section 37.4.6</a>
0Eh	UCAxTXBUF	eUSCI_Ax Transmit Buffer	Read/write	Word	00h	<a href="#">Section 37.4.7</a>
10h	UCAxABCTL	eUSCI_Ax Auto Baud Rate Control	Read/write	Word	00h	<a href="#">Section 37.4.8</a>
12h	UCAxIRCTL	eUSCI_Ax IrDA Control	Read/write	Word	0000h	<a href="#">Section 37.4.9</a>
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	<a href="#">Section 37.4.10</a>
1Ch	UCAxIFG	eUSCI_Ax Interrupt Flag	Read/write	Word	02h	<a href="#">Section 37.4.11</a>
1Eh	UCAxIV	eUSCI_Ax Interrupt Vector	Read	Word	0000h	<a href="#">Section 37.4.12</a>

<sup>(1)</sup> 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".

### 37.4.1 UCAXCTLW0 Register

eUSCI\_Ax Control Word Register 0

**Figure 37-12. 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
UCSELx	UCRXIE	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 37-8. UCAXCTLW0 Register Description**

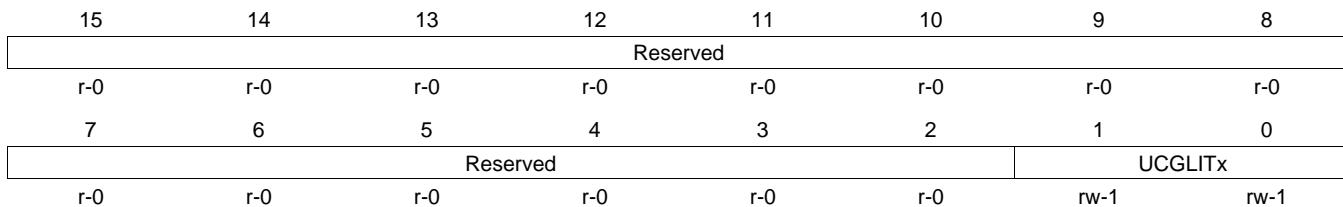
Bit	Field	Type	Reset	Description
15	UCPEN	RW	0h	Parity enable 0b = Parity disabled 1b = Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.
14	UCPAR	RW	0h	Parity select. UCPAR is not used 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	UCSELx	RW	0h	eUSCI_A clock source select. These bits select the BRCLK source clock. 00b = UCLK 01b = ACLK 10b = SMCLK 11b = SMCLK
5	UCRXIE	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 37-8. 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.

### 37.4.2 UCAXCTLW1 Register

eUSCI\_Ax Control Word Register 1

**Figure 37-13. UCAXCTLW1 Register****Table 37-9. 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

### 37.4.3 UCAXBRW Register

eUSCI\_Ax Baud Rate Control Word Register

**Figure 37-14. 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 37-10. UCAXBRW Register Description**

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Clock prescaler setting of the Baud rate generator

### 37.4.4 UCAXMCTLW Register

eUSCI\_Ax Modulation Control Word Register

**Figure 37-15. 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 37-11. 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

**37.4.5 UCAXSTATW Register**

eUSCI\_Ax Status Register

**Figure 37-16. 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 37-12. 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 UCAXRXBUF 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 UCPE = 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, one 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

### 37.4.6 UCAXRXBUF Register

eUSCI\_Ax Receive Buffer Register

**Figure 37-17. 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 37-13. 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.

### 37.4.7 UCAXTXBUF Register

eUSCI\_Ax Transmit Buffer Register

**Figure 37-18. 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 37-14. 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.

**37.4.8 UCAXABCTL Register**

eUSCI\_Ax Auto Baud Rate Control Register

**Figure 37-19. 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	UCDELMx	UCSTOE	UCBTOE	Reserved	UCABDEN		
r-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0	rw-0

Modify only when UCSWRST = 1

**Table 37-15. UCAXABCTL Register Description**

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved
5-4	UCDELMx	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.

### 37.4.9 UCAXIRCTL Register

eUSCI\_Ax IrDA Control Word Register

**Figure 37-20. 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 37-16. 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: $t(MIN) = (UCIRRXFLx + 4) / [2 \times f(IRTXCLK)]$
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 $t(PULSE) = (UCIRTXPLx + 1) / [2 \times f(IRTXCLK)]$
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

**37.4.10 UCAXIE Register**

eUSCI\_Ax Interrupt Enable Register

**Figure 37-21. 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 37-17. 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

### 37.4.11 UCAXIFG Register

eUSCI\_Ax Interrupt Flag Register

**Figure 37-22. 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 37-18. 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

**37.4.12 UCAXIV Register**

eUSCI\_Ax Interrupt Vector Register

**Figure 37-23. 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 37-19. UCAXIV Register Description**

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	eUSCI_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Receive buffer full; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG 06h = Interrupt Source: Start bit received; Interrupt Flag: UCSTTIFG 08h = Interrupt Source: Transmit complete; Interrupt Flag: UCTXCPTIFG; Interrupt Priority: Lowest

## ***Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode***

The enhanced universal serial communication interfaces, eUSCI\_A and eUSCI\_B, support multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface (SPI) mode.

Topic	Page
38.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview .....	1010
38.2 eUSCI Introduction – SPI Mode .....	1010
38.3 eUSCI Operation – SPI Mode .....	1012
38.4 eUSCI_A SPI Registers .....	1018
38.5 eUSCI_B SPI Registers .....	1028

### 38.1 Enhanced Universal Serial Communication Interfaces (eUSCI\_A, eUSCI\_B) Overview

Both the eUSCI\_A and the eUSCI\_B support serial communication in SPI mode.

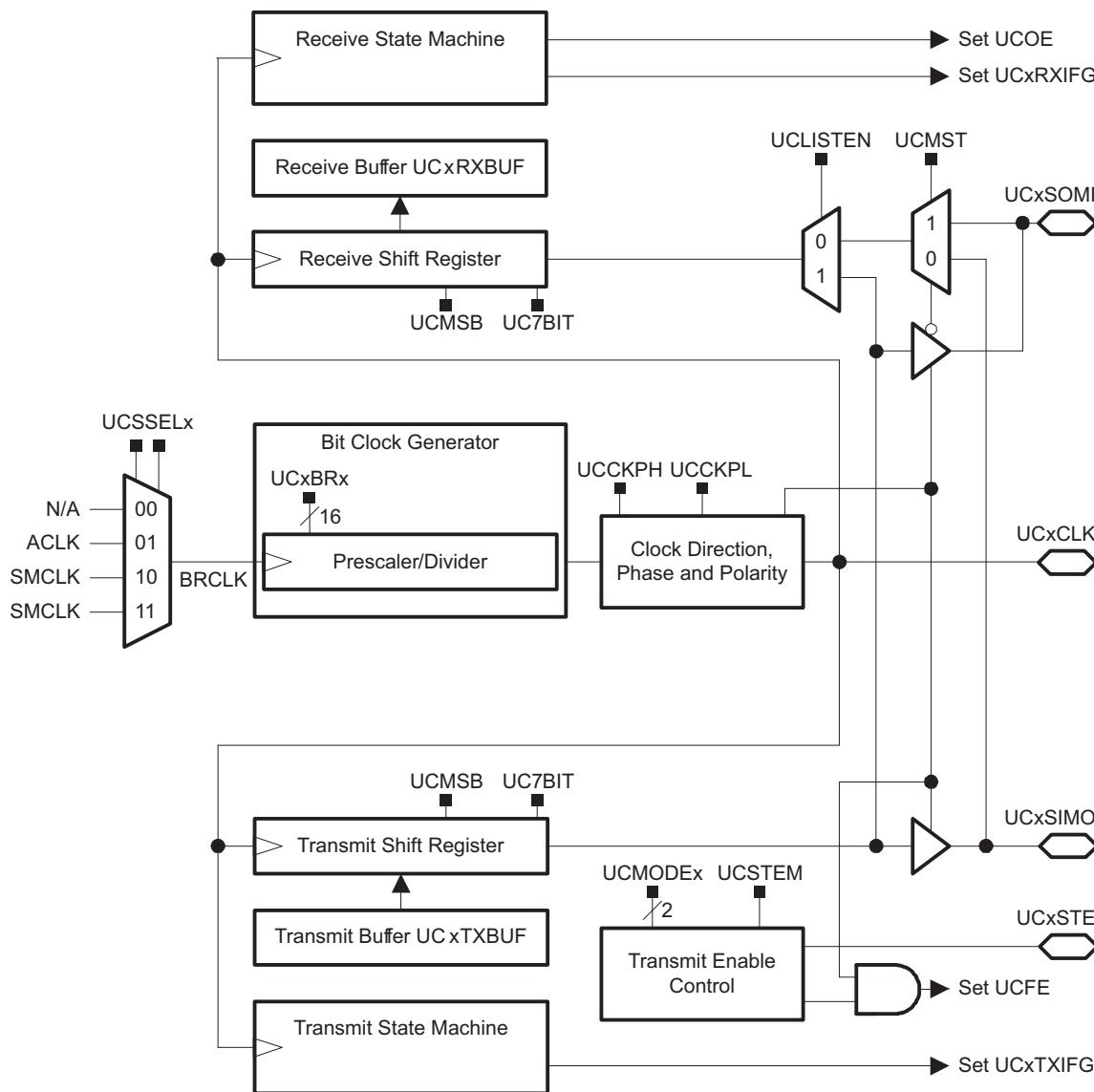
### 38.2 eUSCI Introduction – SPI Mode

In synchronous mode, the eUSCI connects the device to an external system via three or four pins: UCxSIMO, UCxSOMI, UCxCLK, and UCxSTE. SPI mode is selected when the UCSYNC bit is set, and SPI mode (3-pin or 4-pin) is selected with the UCMODEEx bits.

SPI mode features include:

- 7-bit or 8-bit data length
- LSB-first or MSB-first data transmit and receive
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Continuous transmit and receive operation
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

[Figure 38-1](#) shows the eUSCI when configured for SPI mode.



**Figure 38-1. eUSCI Block Diagram – SPI Mode**

### 38.3 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 38-1](#) describes the UCxSTE operation.

**Table 38-1. 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

#### 38.3.1 eUSCI Initialization and Reset

The eUSCI is reset by a PUC or by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI in a reset condition. When set, the UCSWRST bit resets the UCRXIE, UCTXIE, UCRXIFG, UCOE, and UCFE bits, and sets the UCTXIFG flag. Clearing UCSWRST releases the eUSCI for operation.

Configuring and reconfiguring the eUSCI module should be done when UCSWRST is set to avoid unpredictable behavior.

**NOTE: Initializing or reconfiguring the eUSCI module**

The recommended eUSCI initialization/reconfiguration process is:

1. Set UCSWRST.  
`BIS.B #UCSWRST,&UCxCTL1`
2. Initialize all eUSCI registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software.  
`BIC.B #UCSWRST,&UCxCTL1`
5. Enable interrupts (optional) via UCRXIE or UCTXIE.

### 38.3.2 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.

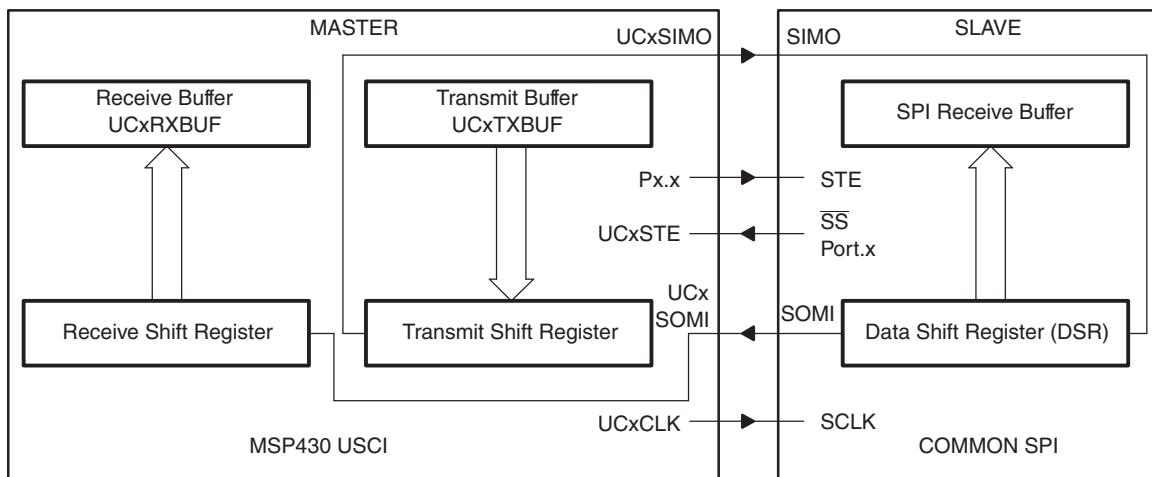
---

**NOTE: Character format for figures**

Figures throughout this chapter use MSB-first format.

---

### 38.3.3 Master Mode



**Figure 38-2. eUSCI Master and External Slave (UCSTEM = 0)**

Figure 38-2 shows the eUSCI as a master in both 3-pin and 4-pin configurations. The eUSCI initiates data transfer when data is moved to the transmit data buffer UCxTXBUF. The UCxTXBUF data is moved to the transmit (TX) shift register when the TX shift register is empty, initiating data transfer on UCxSIMO starting with either the MSB or LSB, depending on the UCMSB setting. Data on UCxSOMI is shifted into the receive shift register on the opposite clock edge. When the character is received, the receive data is moved from the receive (RX) shift register to the received data buffer UCxRXBUF and the receive interrupt flag UCRXIFG is set, indicating the RX/TX operation is complete.

A set transmit interrupt flag, UCTXIFG, indicates that data has moved from UCxTXBUF to the TX shift register and UCxTXBUF is ready for new data. It does not indicate RX/TX completion.

To receive data into the eUSCI in master mode, data must be written to UCxTXBUF, because receive and transmit operations operate concurrently.

There are two different options for configuring the eUSCI as a 4-pin master, which are described in the next sections:

- The fourth pin is used as input to prevent conflicts with other masters (UCSTEM = 0).
- The fourth pin is used as output to generate a slave enable signal (UCSTEM = 1).

The bit UCSTEM is used to select the corresponding mode.

### 38.3.3.1 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 38-1](#). 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.

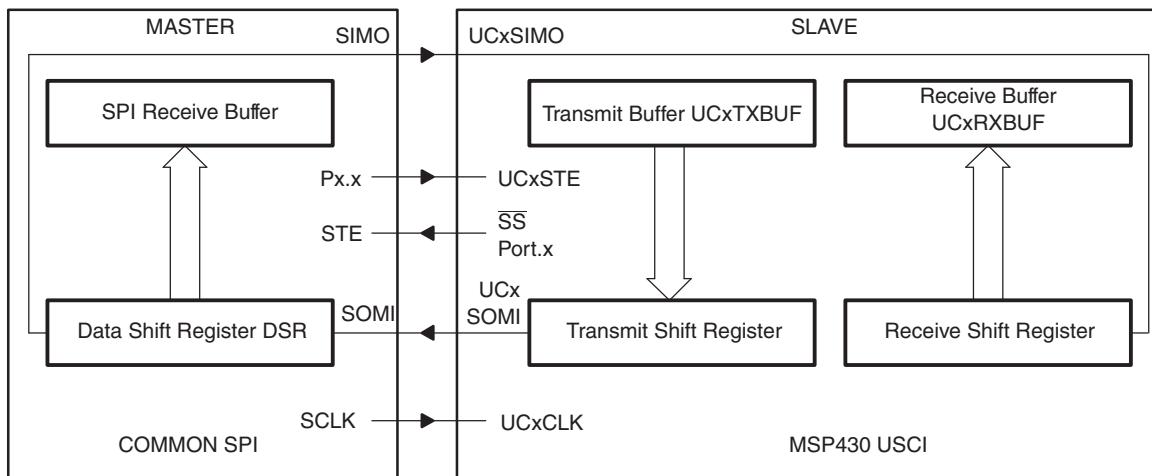
If data is written into UCxTXBUF while the master is held inactive by UCxSTE, it is transmit as soon as UCxSTE transitions to the master-active state. If an active transfer is aborted by UCxSTE transitioning to the master-inactive state, the data must be rewritten into UCxTXBUF to be transferred when UCxSTE transitions back to the master-active state. The UCxSTE input signal is not used in 3-pin master mode.

### 38.3.3.2 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 38-4](#).

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.

### 38.3.4 Slave Mode



**Figure 38-3. eUSCI Slave and External Master**

[Figure 38-3](#) shows the eUSCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCxTXBUF and moved to the TX shift register before the start of UCxCLK is transmitted on UCxSOMI. Data on UCxSIMO is shifted into the receive shift register on the opposite edge of UCxCLK and moved to UCxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UCxRXBUF, the UCRXIFG interrupt flag is set, indicating that data has been received. The overrun error bit UCOE is set when the previously received data is not read from UCxRXBUF before new data is moved to UCxRXBUF.

### 38.3.4.1 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.

### 38.3.5 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.

#### 38.3.5.1 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.

#### 38.3.5.2 Receive Enable

The SPI receives data when a transmission is active. Receive and transmit operations operate concurrently.

### 38.3.6 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 UCSELx 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 UCSELx bits are don't care. The SPI receiver and transmitter operate in parallel and use the same clock source for data transfer.

The 16-bit value of UCBRx in the bit rate control registers (UCxxBR1 and UCxxBR0) is the division factor of the eUSCI clock source, BRCLK. The maximum bit clock that can be generated in master mode is BRCLK. Modulation is not used in SPI mode, and UCAxMCTL should be cleared when using SPI mode for eUSCI\_A. The UCAxCLK/UCBxCLK frequency is given by:

$$f_{\text{BitClock}} = f_{\text{BRCLK}} / UCBRx$$

#### 38.3.6.1 Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the eUSCI. Timing for each case is shown in [Figure 38-4](#).

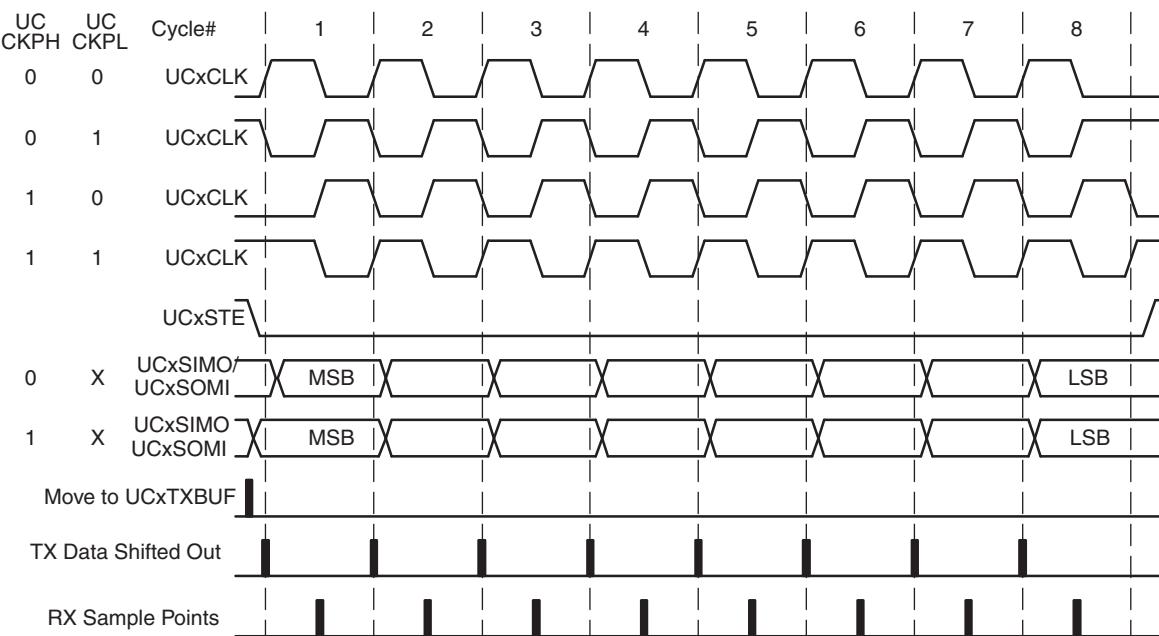


Figure 38-4. eUSCI SPI Timing With UCMSB = 1

### 38.3.7 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.

### 38.3.8 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.

#### 38.3.8.1 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.

#### 38.3.8.2 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.

### 38.3.8.3 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.

#### 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
    JMP      RXIFG_ISR ; Vector 0: No interrupt
    TXIFG_ISR ; Vector 2: RXIFG
    ...       ; Vector 4: TXIFG
    ...       ; Task starts here
    RETI
    RXIFG_ISR ; Return
    ...       ; Vector 2
    ...       ; Task starts here
    RETI
```

### 38.4 eUSCI\_A SPI Registers

The eUSCI\_A registers applicable in SPI mode and their address offsets are listed in [Table 38-2](#). The base addresses can be found in the device-specific data sheet.

**Table 38-2. eUSCI\_A SPI Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	eUSCI_Ax Control Word 0	Read/write	Word	0001h	<a href="#">Section 38.4.1</a>
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	<a href="#">Section 38.4.2</a>
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	<a href="#">Section 38.4.3</a>
0Ch	UCAxRXBUF	eUSCI_Ax Receive Buffer	Read/write	Word	00h	<a href="#">Section 38.4.4</a>
0Eh	UCAxTXBUF	eUSCI_Ax Transmit Buffer	Read/write	Word	00h	<a href="#">Section 38.4.5</a>
1Ah	UCAxIE	eUSCI_Ax Interrupt Enable	Read/write	Word	00h	<a href="#">Section 38.4.6</a>
1Ch	UCAxIFG	eUSCI_Ax Interrupt Flag	Read/write	Word	02h	<a href="#">Section 38.4.7</a>
1Eh	UCAxIV	eUSCI_Ax Interrupt Vector	Read	Word	0000h	<a href="#">Section 38.4.8</a>

### **38.4.1 UCAXCTLW0 Register**

eUSCI\_Ax Control Register 0

**Figure 38-5. 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
UCSELx		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 38-3. UCAXCTLW0 Register Description**

Bit	Field	Type	Reset	Description
15	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
14	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = I2C mode
8	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSELx	RW	0h	eUSCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK
5-2	Reserved	R	0h	Reserved
1	UCSTEM	RW	0h	STE mode select in master mode. This byte is ignored in slave or 3-wire mode. 0b = STE pin is used to prevent conflicts with other masters 1b = STE pin is used to generate the enable signal for a 4-wire slave

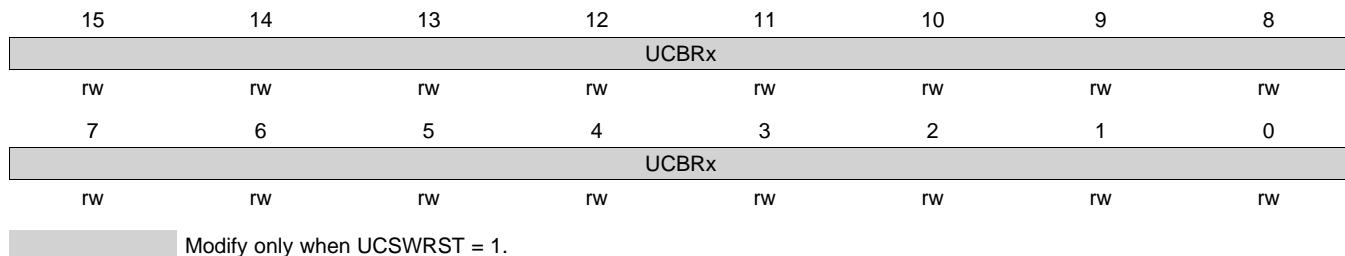
**Table 38-3. UCAXCTLW0 Register Description (continued)**

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.

### **38.4.2 UCAXBRW Register**

eUSCI\_Ax Bit Rate Control Register 1

**Figure 38-6. UCAXBRW Register**



Modify only when UCSWRST = 1.

**Table 38-4. UCAXBRW Register Description**

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler setting.

**38.4.3 UCAXSTATW Register**

eUSCI\_Ax Status Register

**Figure 38-7. 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 38-5. UCAXSTATW Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. The transmitter output is internally fed back to the receiver.
6	UCFE	RW	0h	Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode. 0b = No error 1b = Bus conflict occurred
5	UCOE	RW	0h	Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly. 0b = No error 1b = Overrun error occurred
4-1	Reserved	RW	0h	Reserved
0	UCBUSY	R	0h	eUSCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = eUSCI inactive 1b = eUSCI transmitting or receiving

### **38.4.4 UCAXRXBUF Register**

eUSCI\_Ax Receive Buffer Register

**Figure 38-8. 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 38-6. 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.

**38.4.5 UCAXTXBUF Register**

eUSCI\_Ax Transmit Buffer Register

**Figure 38-9. 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
UCTXBUF <sub>x</sub>							
rW	rW	rW	rW	rW	rW	rW	rW

**Table 38-7. UCAXTXBUF Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCTXBUF <sub>x</sub>	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.

### **38.4.6 UCAXIE Register**

eUSCI\_Ax Interrupt Enable Register

**Figure 38-10. UCAXIE Register**

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

**Table 38-8. 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

**38.4.7 UCAXIFG Register**

eUSCI\_Ax Interrupt Flag Register

**Figure 38-11. 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 38-9. UCAXIFG Register Description**

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

### **38.4.8 UCAXIV Register**

eUSCI\_Ax Interrupt Vector Register

**Figure 38-12. 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 38-10. 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

## 38.5 eUSCI\_B SPI Registers

The eUSCI\_B registers applicable in SPI mode and their address offsets are listed in [Table 38-11](#). The base addresses can be found in the device-specific data sheet.

**Table 38-11. eUSCI\_B SPI Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	eUSCI_Bx Control Word 0	Read/write	Word	01C1h	<a href="#">Section 38.5.1</a>
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	<a href="#">Section 38.5.2</a>
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	<a href="#">Section 38.5.3</a>
0Ch	UCBxRXBUF	eUSCI_Bx Receive Buffer	Read/write	Word	00h	<a href="#">Section 38.5.4</a>
0Eh	UCBxTXBUF	eUSCI_Bx Transmit Buffer	Read/write	Word	00h	<a href="#">Section 38.5.5</a>
2Ah	UCBxIE	eUSCI_Bx Interrupt Enable	Read/write	Word	00h	<a href="#">Section 38.5.6</a>
2Ch	UCBxIFG	eUSCI_Bx Interrupt Flag	Read/write	Word	02h	<a href="#">Section 38.5.7</a>
2Eh	UCBxIV	eUSCI_Bx Interrupt Vector	Read	Word	0000h	<a href="#">Section 38.5.8</a>

### **38.5.1 UCBxCTLW0 Register**

eUSCI\_Bx Control Register 0

**Figure 38-13. 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
UCSELx		Reserved					
rw-1	rw-1	r0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when UCSWRST = 1.

**Table 38-12. UCBxCTLW0 Register Description**

Bit	Field	Type	Reset	Description
15	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
14	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = I2C mode
8	UCSYNC	RW	1h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSELx	RW	3h	eUSCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK
5-2	Reserved	R	0h	Reserved
1	UCSTEM	RW	0h	STE mode select in master mode. This byte is ignored in slave or 3-wire mode. 0b = STE pin is used to prevent conflicts with other masters 1b = STE pin is used to generate the enable signal for a 4-wire slave

**Table 38-12. 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.

### 38.5.2 UCBxBRW Register

eUSCI\_Bx Bit Rate Control Register 1

**Figure 38-14. 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 38-13. UCBxBRW Register Description**

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler setting.

### 38.5.3 UCBxSTATW Register

eUSCI\_Bx Status Register

**Figure 38-15. 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 38-14. UCBxSTATW Register Description**

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. The transmitter output is internally fed back to the receiver.
6	UCFE	RW	0h	Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode. 0b = No error 1b = Bus conflict occurred
5	UCOE	RW	0h	Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly. 0b = No error 1b = Overrun error occurred
4-1	Reserved	R	0h	Reserved
0	UCBUSY	R	0h	eUSCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = eUSCI inactive 1b = eUSCI transmitting or receiving

**38.5.4 UCBxRXBUF Register**

eUSCI\_Bx Receive Buffer Register

**Figure 38-16. UCBxRXBUF Register**

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

**Table 38-15. 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.

**38.5.5 UCBxTXBUF Register**

eUSCI\_Bx Transmit Buffer Register

**Figure 38-17. UCBxTXBUF Register**

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

**Table 38-16. 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.

### **38.5.6 UCBxIE Register**

eUSCI\_Bx Interrupt Enable Register

**Figure 38-18. 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 38-17. 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

### **38.5.7 UCBxIFG Register**

eUSCI\_Bx Interrupt Flag Register

**Figure 38-19. 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							
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

**Table 38-18. UCBxIFG Register Description**

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

**38.5.8 UCBxIV Register**

eUSCI\_Bx Interrupt Vector Register

**Figure 38-20. 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 38-19. 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<sup>2</sup>C Mode***

The enhanced universal serial communication interface B (eUSCI\_B) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the I<sup>2</sup>C mode.

Topic	Page
39.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview .....	1036
39.2 eUSCI_B Introduction – I <sup>2</sup> C Mode .....	1036
39.3 eUSCI_B Operation – I <sup>2</sup> C Mode .....	1037
39.4 eUSCI_B I2C Registers .....	1057

### 39.1 Enhanced Universal Serial Communication Interface B (eUSCI\_B) Overview

The eUSCI\_B module supports two serial communication modes:

- I<sup>2</sup>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.

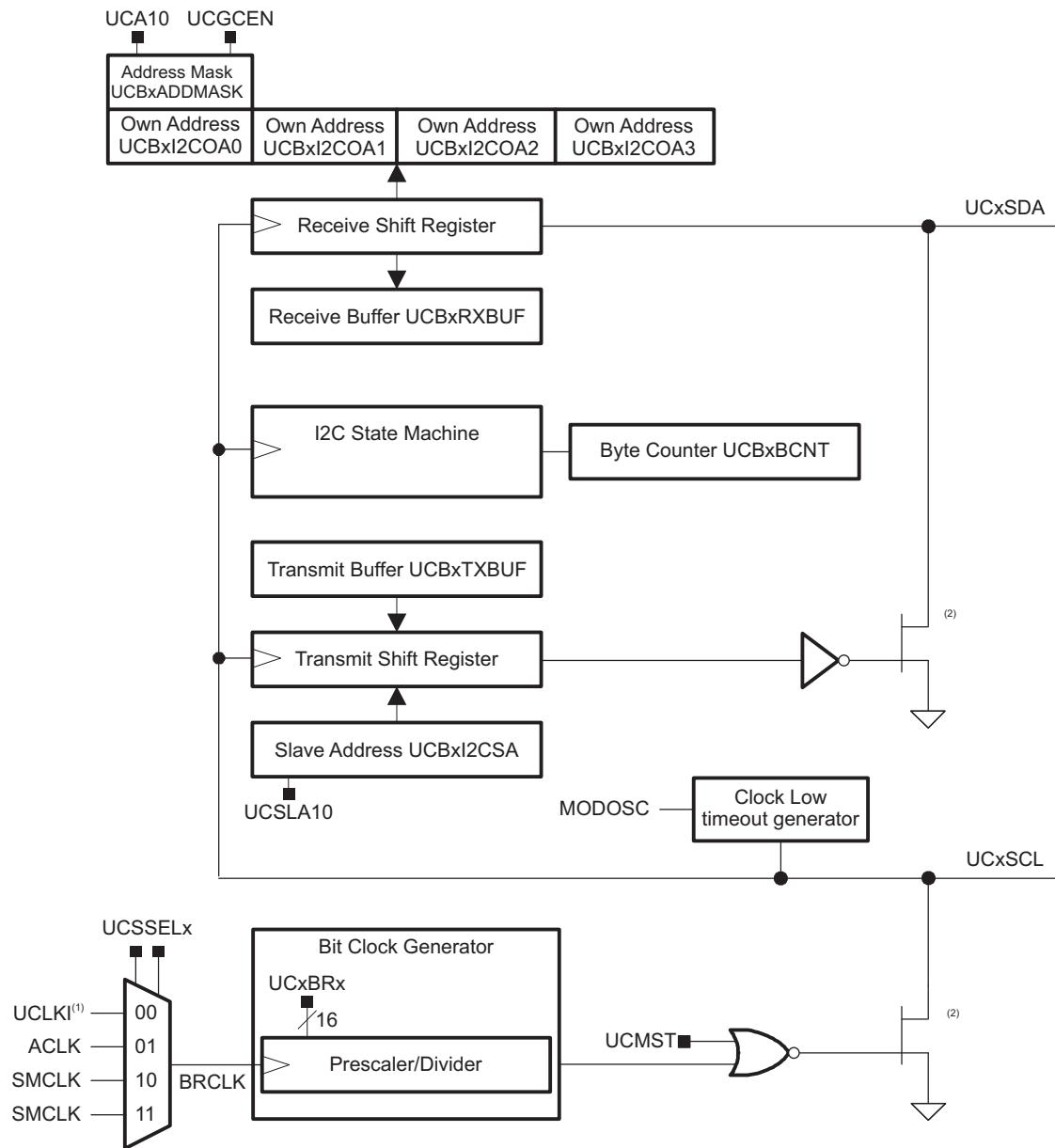
### 39.2 eUSCI\_B Introduction – I<sup>2</sup>C Mode

In I<sup>2</sup>C mode, the eUSCI\_B module provides an interface between the device and I<sup>2</sup>C-compatible devices connected by the two-wire I<sup>2</sup>C serial bus. External components attached to the I<sup>2</sup>C bus serially transmit or receive serial data to or from the eUSCI\_B module through the 2-wire I<sup>2</sup>C interface.

The eUSCI\_B I<sup>2</sup>C mode features include:

- 7-bit and 10-bit device addressing modes
- General call
- START/RESTART/STOP
- Multi-master transmitter/receiver mode
- Slave receiver/transmitter mode
- Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- 8-bit byte counter with interrupt capability and automatic STOP assertion
- Up to four hardware slave addresses, each having its own interrupt and DMA trigger
- Mask register for slave address and address received interrupt
- Clock low timeout interrupt to avoid bus stalls
- Slave operation in LPM4
- Slave receiver START detection for auto wake-up from LPMx modes (not LPM3.5 and LPM4.5)

[Figure 39-1](#) shows the eUSCI\_B when configured in I<sup>2</sup>C mode.

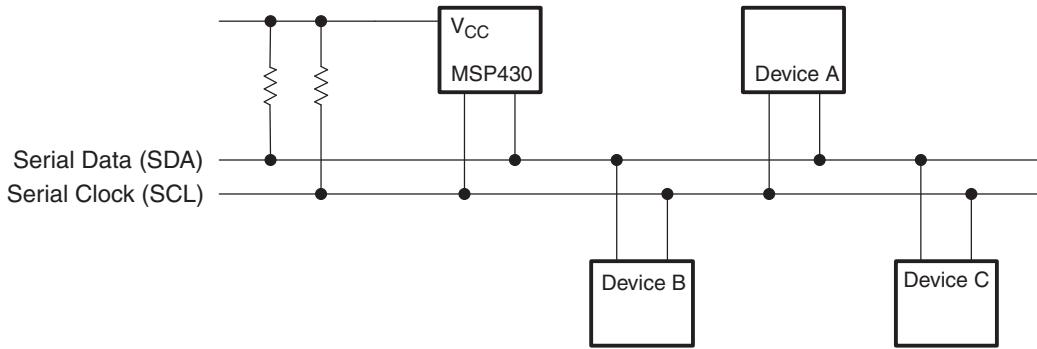


**Figure 39-1. eUSCI\_B Block Diagram – I<sup>2</sup>C Mode**

### 39.3 eUSCI\_B Operation – I<sup>2</sup>C Mode

The I<sup>2</sup>C mode supports any slave or master I<sup>2</sup>C-compatible device. Figure 39-2 shows an example of an I<sup>2</sup>C bus. Each I<sup>2</sup>C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I<sup>2</sup>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<sup>2</sup>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 39-2. I<sup>2</sup>C Bus Connection Diagram****NOTE: SDA and SCL levels**

The SDA and SCL pins must not be pulled up above the device V<sub>cc</sub> level.

**39.3.1 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 I<sup>2</sup>C 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 I<sup>2</sup>C mode has the following effects:

- I<sup>2</sup>C communication stops.
- SDA and SCL are high impedance.
- UCBxSTAT, bits 15-9 and 6-4 are cleared.
- Registers UCBxIE and UCBxIFG are cleared.
- All other bits and registers remain unchanged.

**NOTE: Initializing or re-configuring the eUSCI\_B module**

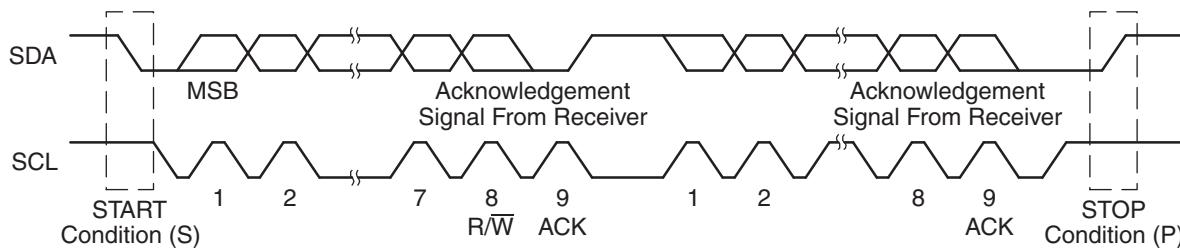
The recommended eUSCI\_B initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST, &UCxCTL1`).
2. Initialize all eUSCI\_B registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST, &UCxCTL1`).
5. Enable interrupts (optional).

**39.3.2 I<sup>2</sup>C Serial Data**

One clock pulse is generated by the master device for each data bit transferred. The I<sup>2</sup>C mode operates with byte data. Data is transferred MSB first as shown in [Figure 39-3](#).

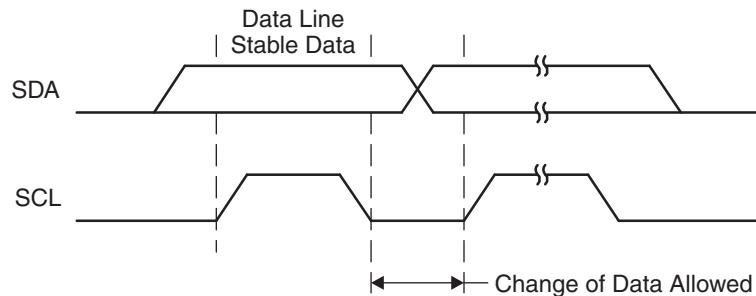
The first byte after a START condition consists of a 7-bit slave address and the R/W bit. When R/W = 0, the master transmits data to a slave. When R/W = 1, the master receives data from a slave. The ACK bit is sent from the receiver after each byte on the ninth SCL clock.



**Figure 39-3. I<sup>2</sup>C Module Data Transfer**

START and STOP conditions are generated by the master and are shown in Figure 39-3. 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 39-4). The high and low state of SDA can change only when SCL is low, otherwise START or STOP conditions are generated.



**Figure 39-4. Bit Transfer on I<sup>2</sup>C Bus**

### 39.3.3 I<sup>2</sup>C Addressing Modes

The I<sup>2</sup>C mode supports 7-bit and 10-bit addressing modes.

#### 39.3.3.1 7-Bit Addressing

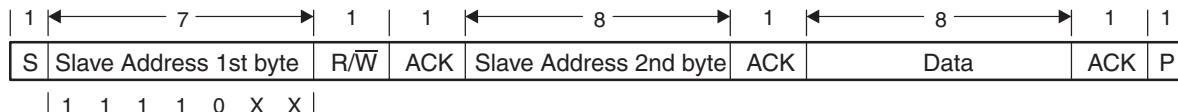
In the 7-bit addressing format (see Figure 39-5), the first byte is the 7-bit slave address and the R/W bit. The ACK bit is sent from the receiver after each byte.

1	7	1	1	8	1	8	1	1
S	Slave Address	R/W	ACK	Data	ACK	Data	ACK	P

**Figure 39-5. I<sup>2</sup>C Module 7-Bit Addressing Format**

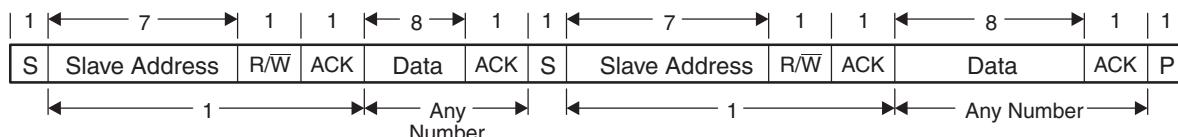
#### 39.3.3.2 10-Bit Addressing

In the 10-bit addressing format (see Figure 39-6), the first byte is made up of 11110b plus the two MSBs of the 10-bit slave address and the R/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 [I<sup>2</sup>C Slave 10-bit Addressing Mode](#) and [I<sup>2</sup>C Master 10-bit Addressing Mode](#) for details how to use the 10-bit addressing mode with the eUSCI\_B module.

Figure 39-6. I<sup>C</sup> Module 10-Bit Addressing Format

### 39.3.3.3 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/W bit. The RESTART condition is shown in Figure 39-7.

Figure 39-7. I<sup>C</sup> Module Addressing Format With Repeated START Condition

### 39.3.4 I<sup>C</sup> Quick Setup

This section gives a quick introduction into the operation of the eUSCI\_B in I<sup>C</sup> 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 39.3.5](#).

The latest code examples can be found on the MSP430 web under "Code Examples".

To set up the eUSCI\_B as a master transmitter that transmits to a slave with the address 0x12h, only a few steps are needed (see [Example 39-1](#)).

#### Example 39-1. Master TX With 7-Bit Address

```

UCBxCTL1 |= UCSWRST;           // put eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3 + UCMST; // I2C master mode
UCBxBRW = 0x0008;              // baudrate = SMCLK / 8
UCBxCTLW1 = UCASLP_2;          // autom. STOP assertion
UCBxTBCNT = 0x07;              // TX 7 bytes of data
UCBxI2CSA = 0x0012;             // address slave is 12hex
P2SEL |= 0x03;                 // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;          // eUSCI_B in operational state
UCBxIE |= UCTXIE;               // enable TX-interrupt
GIE;                           // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;              // fill TX buffer

```

As shown in the code example, all configurations must be done while UCSWRST is set. To select the I<sup>C</sup> operation of the eUSCI\_B, UCMODE must be set accordingly. The baudrate of the transmission is set by writing the correct divider in the UCBxBRW register. The default clock selected is SMCLK. How many bytes are transmitted in one frame is controlled by the byte counter threshold register UCBxTBCNT together with the UCASLPx bits.

The slave address to send to is specified in the UCBxI2CSA register. Finally, the ports must be configured. This step is device dependent; see the data sheet for the pins that must be used.

Each byte that is to be transmitted must be written to the UCBxTXBUF inside the interrupt service routine. The recommended structure of the interrupt service routine can be found in [Example 39-3](#).

[Example 39-2](#) 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 39-2. 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 39-2](#), all configurations must be done while UCSWRST is set. For the slave, I<sup>2</sup>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.

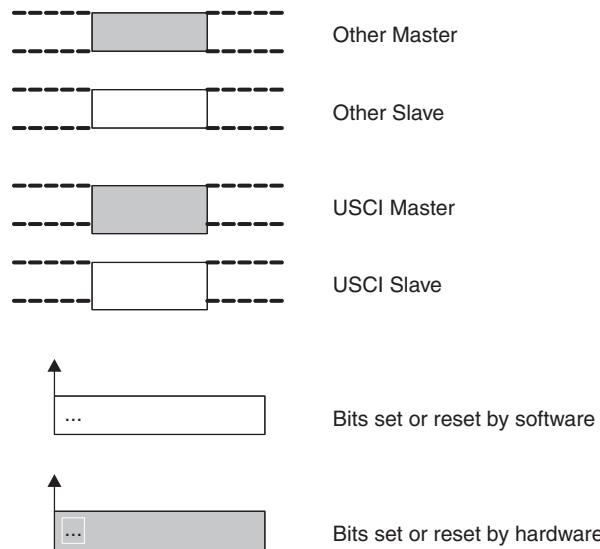
The RX interrupt service routine is called for every byte received by a master device. The TX interrupt service routine is executed each time the master requests a byte. The recommended structure of the interrupt service routine can be found in [Example 39-3](#).

#### 39.3.5 I<sup>2</sup>C Module Operating Modes

In I<sup>2</sup>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 39-8](#) 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 39-8. I<sup>2</sup>C Time-Line Legend**

### 39.3.5.1 Slave Mode

The eUSCI\_B module is configured as an I<sup>2</sup>C slave by selecting the I<sup>2</sup>C mode with UCMODEEx = 11 and UCSYNC = 1 and clearing the UCMST bit.

Initially, the eUSCI\_B module must be configured in receiver mode by clearing the UCTR bit to receive the I<sup>2</sup>C address. Afterwards, transmit and receive operations are controlled automatically, depending on the R/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 39.3.9](#). 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.

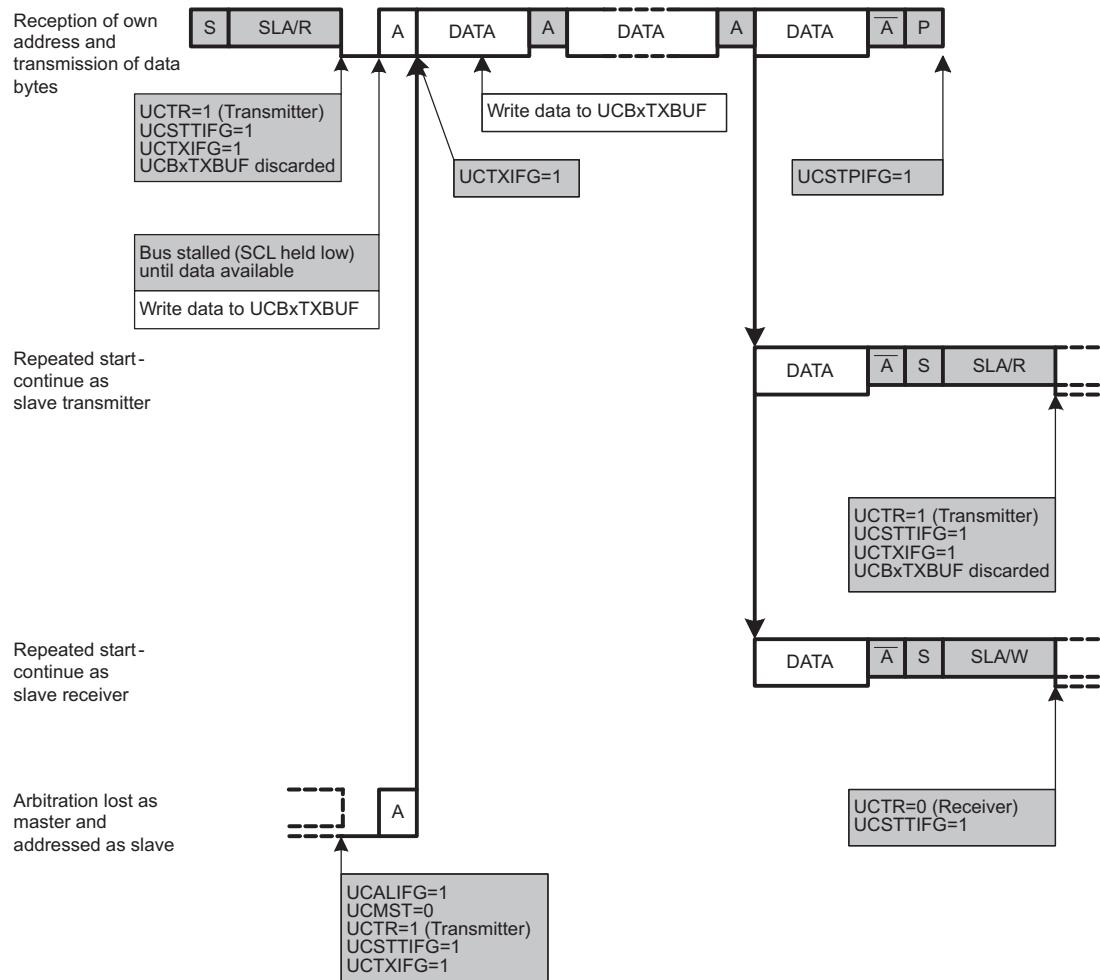
When a START condition is detected on the bus, the eUSCI\_B module receives the transmitted address and compares it against its own address stored in UCBxI2COA0. The UCSTTIFG flag is set when address received matches the eUSCI\_B slave address.

### I<sup>2</sup>C 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/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 I<sup>2</sup>C state machine returns to its address-reception state.

[Figure 39-9](#) shows the slave transmitter operation.



**Figure 39-9. I<sup>2</sup>C Slave Transmitter Mode**

### I<sup>2</sup>C 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/W bit is received. In slave receiver mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low if intervention of the CPU is required after a byte has been received.

If the slave receives data from the master, the eUSCI\_B module is automatically configured as a receiver and UCTR is cleared. After the first data byte is received, the receive interrupt flag UCRXIFG0 is set. The eUSCI\_B module automatically acknowledges the received data and can receive the next data byte.

If the previous data was not read from the receive buffer UCBxRXBUF at the end of a reception, the bus is stalled by holding SCL low. As soon as UCBxRXBUF is read, the new data is transferred into UCBxRXBUF, an acknowledge is sent to the master, and the next data can be received.

Setting the UCTXNACK bit causes a NACK to be transmitted to the master during the next acknowledgment cycle. A NACK is sent even if UCBxRXBUF is not ready to receive the latest data. If the UCTXNACK bit is set while SCL is held low, the bus is released, a NACK is transmitted immediately, and UCBxRXBUF is loaded with the last received data. Because the previous data was not read, that data is lost. To avoid loss of data, the UCBxRXBUF must be read before UCTXNACK is set.

When the master generates a STOP condition, the UCSTPIFG flag is set.

If the master generates a repeated START condition, the eUSCI\_B I<sup>2</sup>C state machine returns to its address-reception state.

Figure 39-10 shows the I<sup>2</sup>C slave receiver operation.

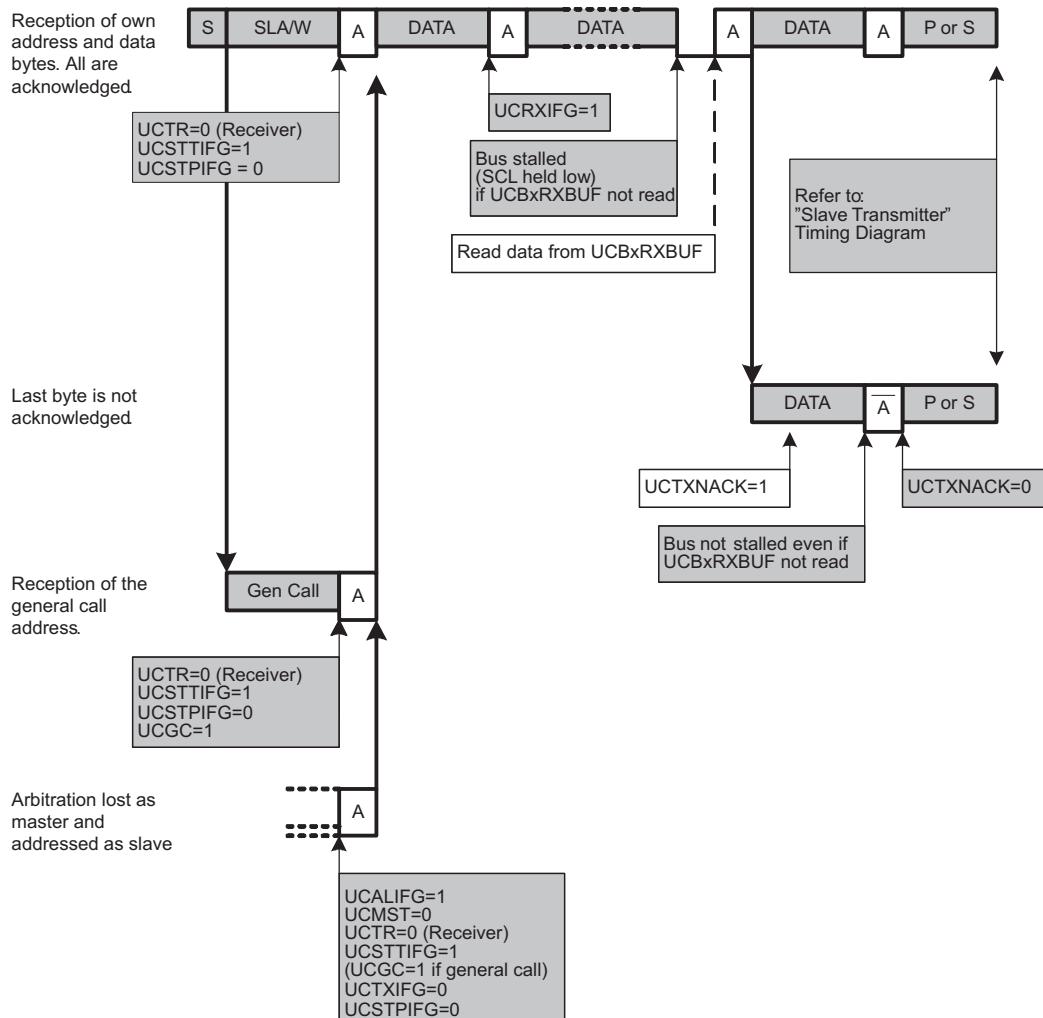
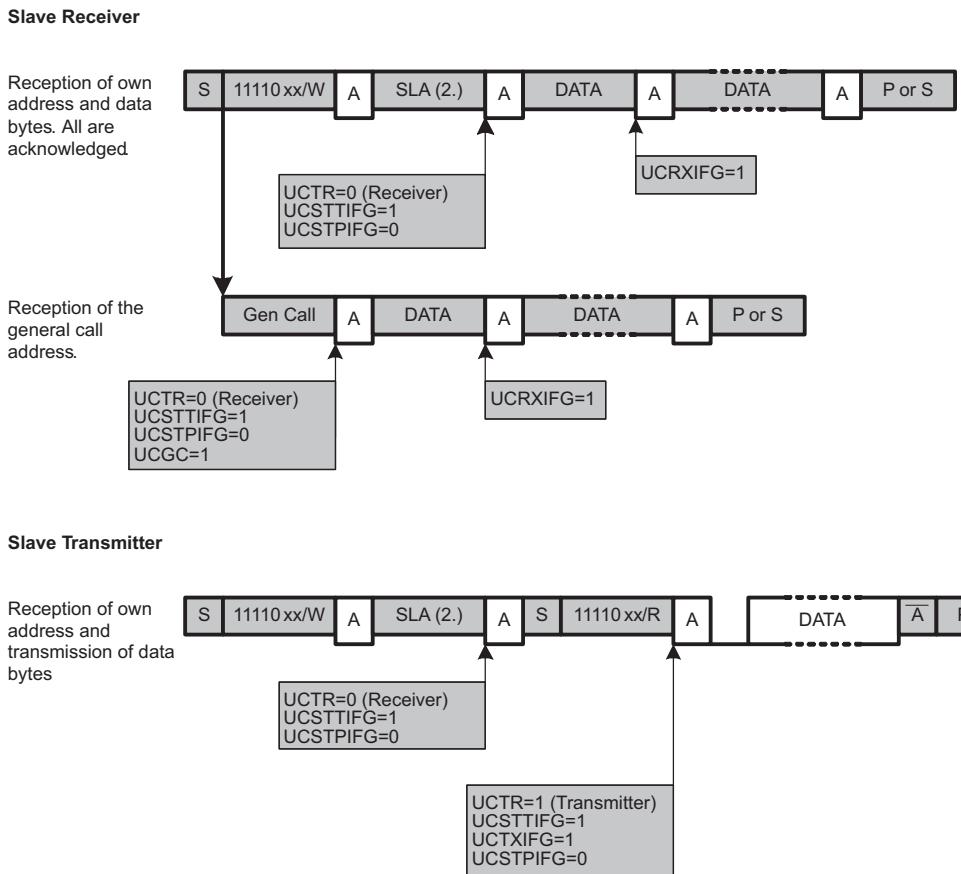


Figure 39-10. I<sup>2</sup>C Slave Receiver Mode

### I<sup>2</sup>C Slave 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 39-11. 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/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 39-11. I<sup>2</sup>C Slave 10-Bit Addressing Mode**

### 39.3.5.2 Master Mode

The eUSCI\_B module is configured as an I<sup>2</sup>C master by selecting the I<sup>2</sup>C mode with UCMODEx = 11 and UCSYNC = 1 and setting the UCMST bit. When the master is part of a multi-master system, UCMM must be set and its own address must be programmed into the UCBxI2COA0 register. Support for multiple slave addresses is explained in [Section 39.3.9](#). When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the eUSCI\_B module responds to a general call.

---

**NOTE: Addresses and multi-master systems**

In master mode with own-address detection enabled (UCOAEN = 1)—especially in multi-master systems—it is not allowed to specify the same address in the own address and slave address register (UCBxI2CSA = UCBxI2COAx). This would mean that the eUSCI\_B addresses itself.

The user software must ensure that this situation does not occur. There is no hardware detection for this case, and the consequence is unpredictable behavior of the eUSCI\_B.

---

### I<sup>2</sup>C 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:

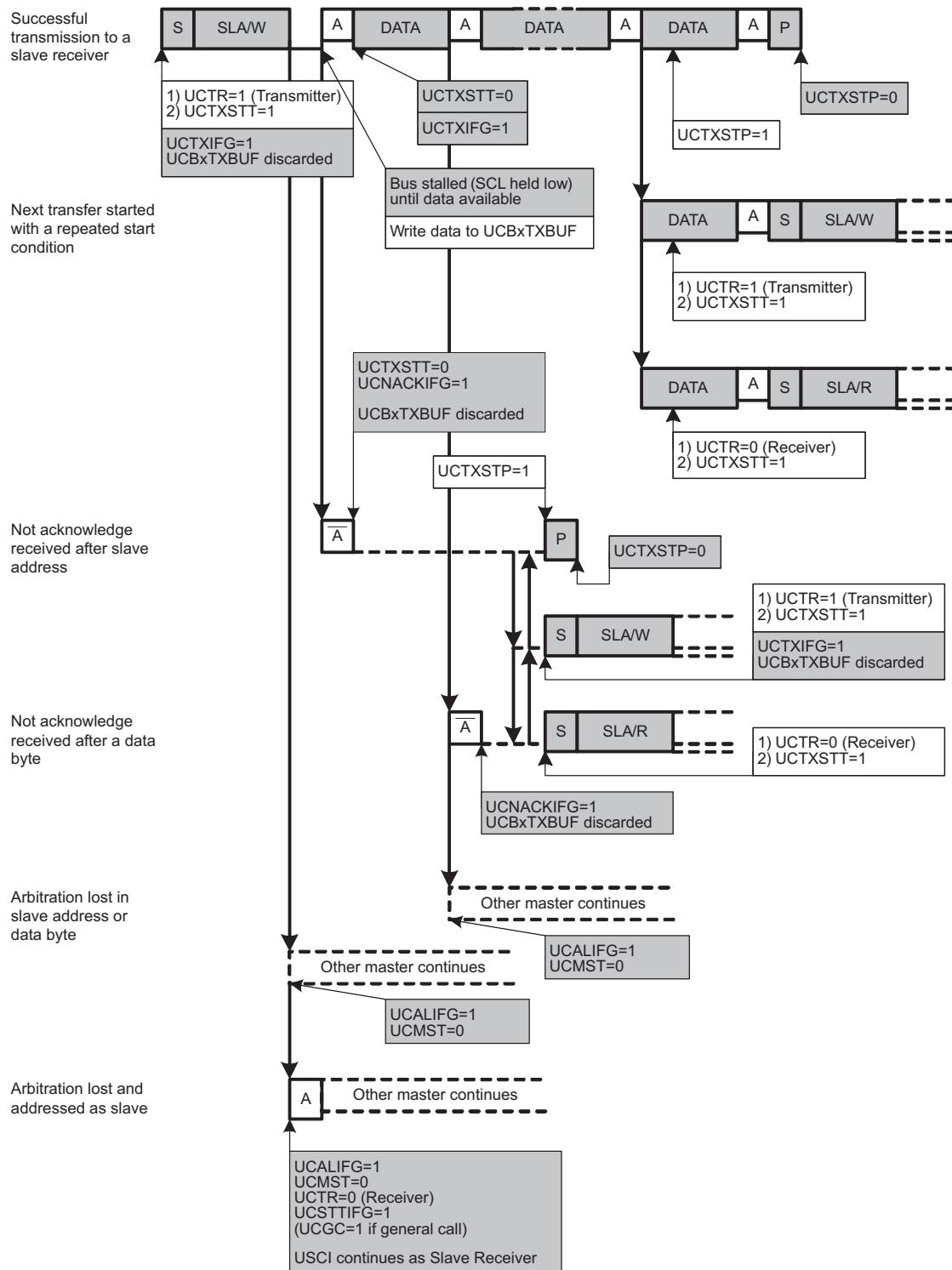
- No automatic STOP is generated
- The UCTXSTP bit is not set
- The UCTXSTT bit is not set

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 39-12 shows the I<sup>2</sup>C master transmitter operation.


**Figure 39-12. I<sup>2</sup>C Master Transmitter Mode**

### I<sup>2</sup>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.

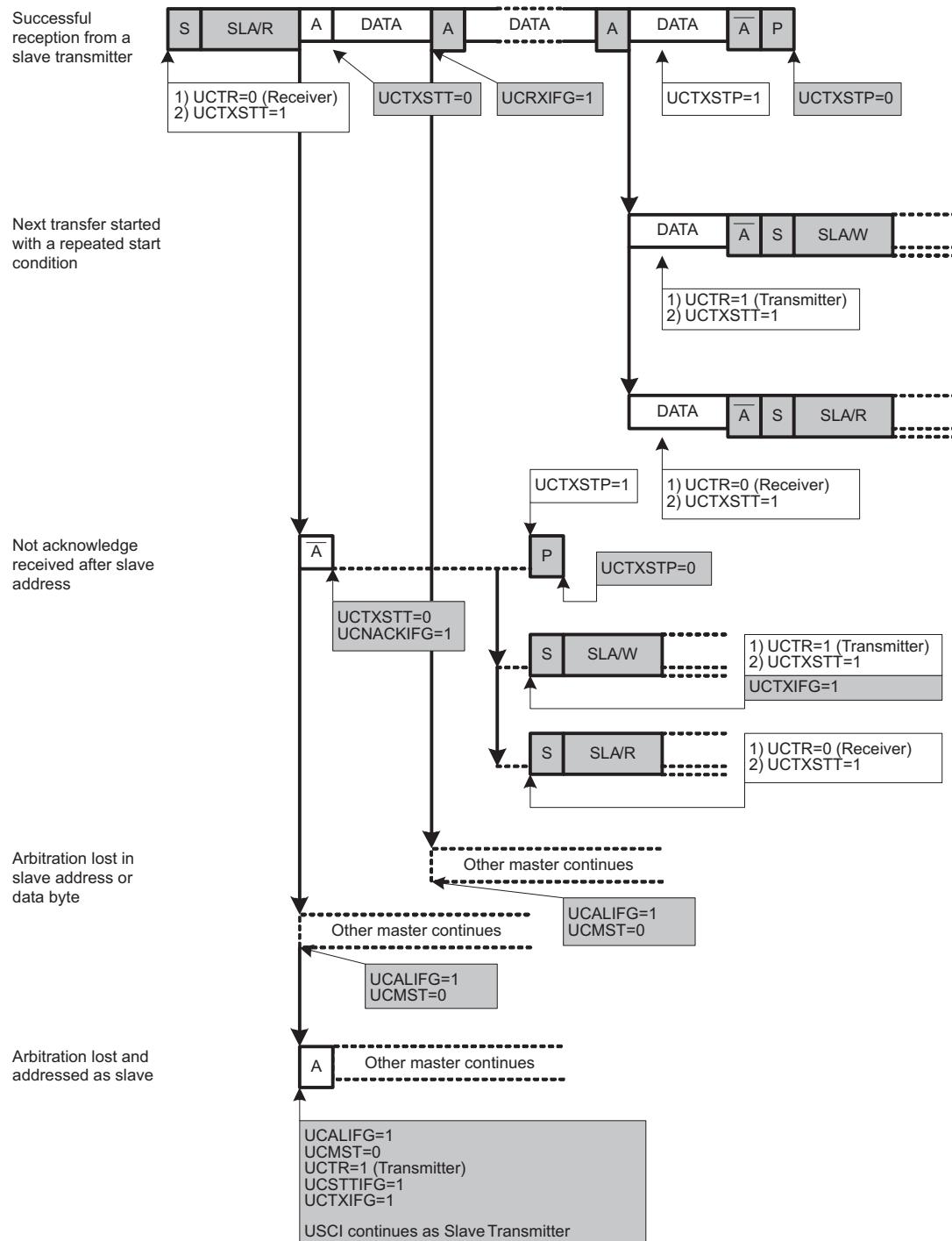
A STOP condition is either generated by the automatic STOP generation or by setting the UCTXSTP bit. The next byte received from the slave is followed by a NACK and a STOP condition. This NACK occurs immediately if the eUSCI\_B module is currently waiting for UCBxRXBUF to be read.

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 39-13](#) shows the I<sup>2</sup>C master receiver operation.

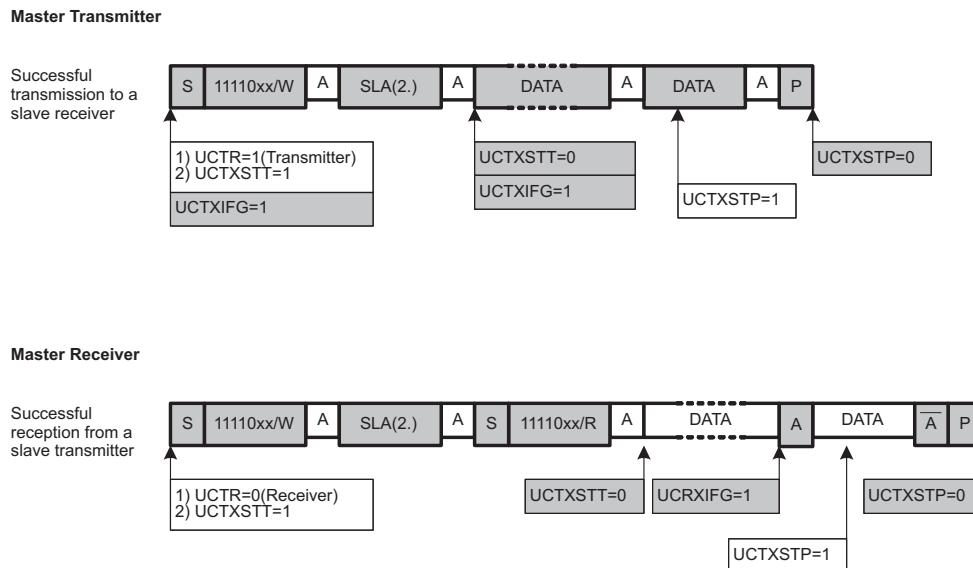
**NOTE: Consecutive master transactions without repeated START**

When performing multiple consecutive I<sup>2</sup>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<sup>2</sup>C transaction is initiated with setting UCTXSTT = 1. Otherwise, the current transaction might be affected.


**Figure 39-13. I<sup>2</sup>C Master Receiver Mode**

**I<sup>2</sup>C Master 10-Bit Addressing Mode**

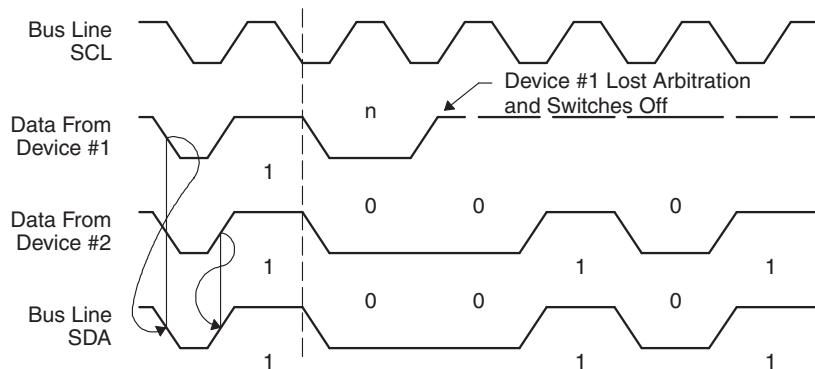
The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in [Figure 39-14](#).



**Figure 39-14. I<sup>2</sup>C Master 10-Bit Addressing Mode**

**39.3.5.3 Arbitration**

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. [Figure 39-15](#) 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 39-15. 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.

### 39.3.6 Glitch Filtering

According to the I<sup>2</sup>C 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 39-1. Glitch Filter Length Selection Bits**

UCGLITx	Corresponding Glitch Filter Length on SDA and SCL	According to I <sup>2</sup> C 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

### 39.3.7 I<sup>2</sup>C Clock Generation and Synchronization

The I<sup>2</sup>C clock SCL is provided by the master on the I<sup>2</sup>C bus. When the eUSCI\_B is in master mode, BITCLK is provided by the eUSCI\_B bit clock generator and the clock source is selected with the UCSSELx bits. In slave mode, the bit clock generator is not used and the UCSSELx bits are don't care.

The 16-bit value of UCBRx in registers UCBxBR1 and UCBxBR0 is the division factor of the eUSCI\_B clock source, BRCLK. The maximum bit clock that can be used in single master mode is  $f_{BRCLK}/4$ . In multi-master mode, the maximum bit clock is  $f_{BRCLK}/8$ . The BITCLK frequency is given by:

$$f_{BitClock} = f_{BRCLK}/UCBRx$$

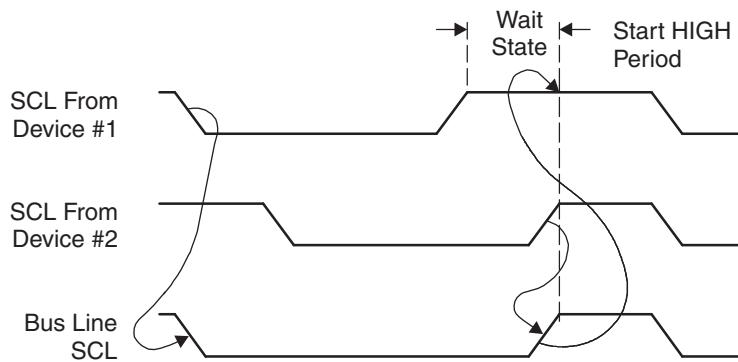
The minimum high and low periods of the generated SCL are:

$$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx/2)/f_{BRCLK} \text{ when UCBRx is even}$$

$$t_{LOW,MIN} = t_{HIGH,MIN} = ((UCBRx - 1)/2)/f_{BRCLK} \text{ when UCBRx is odd}$$

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 I<sup>2</sup>C 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 39-16](#) shows the clock synchronization. This allows a slow slave to slow down a fast master.



**Figure 39-16. Synchronization of Two I<sup>2</sup>C Clock Generators During Arbitration**

#### 39.3.7.1 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.

### 39.3.7.2 Avoiding Clock Stretching

Even though clock stretching is part of the I<sup>C</sup> specification, there are applications in which clock stretching should be avoided.

The clock is stretched by the eUSCI\_B under the following conditions:

- The internal shift register is expecting data, but the TXIFG is still pending
- The internal shift register is full, but the RXIFG is still pending
- The arbitration lost interrupt is pending
- UCSWACK is selected and UCBxI2COA0 did cause a match

To avoid clock stretching, all of these situations for clock stretch either need to be avoided or the corresponding interrupt flags need to be processed before the actual clock stretch can occur.

Using the DMA (on devices that contain a DMA) is the most secure way to avoid clock stretching. If no DMA is available, the software must ensure that the corresponding interrupts are serviced in time before the clock is stretched.

In slave transmitter mode, the TXIFG is set only after the reception of the direction bit; therefore, there is only a short amount of time for the software to write the TXBUF before a clock stretch occurs. This situation can be remedied by using the early Transmit Interrupt (see [Section 39.3.11.2](#)).

### 39.3.7.3 Clock Low Timeout

The UCCLTOIFG interrupt allows the software to react if the clock is low longer than a defined time. It is possible to detect the situation, when a clock is stretched by a master or slave for a too long time. The user can then, for example, reset the eUSCI\_B module by using the UCSWRST bit.

The clock low timeout feature is enabled using the UCCLTO bits. It is possible to select one of three predefined times for the clock low timeout. If the clock has been low longer than the time defined with the UCCLTO bits and the eUSCI\_B was actively receiving or transmitting, the UCCLTOIFG is set and an interrupt request is generated if UCCLTOIE and GIE are set as well. The UCCLTOIFG is set only once, even if the clock is stretched a multiple of the time defined in UCCLTO.

### 39.3.8 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 byte position, if an arbitration lost occurs during the first bit of data.

#### 39.3.8.1 Byte Counter Interrupt

If UCASTPx = 01 or 10 the UCBCNTIFG is set when the byte counter threshold value UCBxTBCNT is reached in both master- and slave-mode. Writing zero to UCBxTBCNT does not generate an interrupt.

Because the UCBCNTIFG has a lower interrupt priority than the UCBTXIFG and UCBRXIFG, it is recommended to only use it for protocol control together with the DMA handling the received and transmitted bytes. Otherwise the application must have enough processor bandwidth to ensure that the UCBCNT interrupt routine is executed in time to generate for example a RESTART.

### 39.3.8.2 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.

### 39.3.9 Multiple Slave Addresses

The eUSCI\_B module supports two different ways of implementing multiple slave addresses at the same time:

- Hardware support for up to 4 different slave addresses, each with its own interrupt flag and DMA trigger
- Software support for up to  $2^{10}$  different slave addresses all sharing one interrupt

#### 39.3.9.1 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.

#### 39.3.9.2 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 not masked by UCBxADDMASK the eUSCI\_B considers the seen address as its own address and sends an acknowledge. The user has the choice to either automatically acknowledge the address seen on the bus or to evaluate this address and send the acknowledge in software using UCTXACK. The selection between these options is done using the UCSWACK bit. If the software is used for generation of the ACK of the slave address, it is recommended to use the UCSTTIIFG. The received address can be found in the UCBxADDRX register.

A slave address seen on the bus is automatically acknowledged by the eUSCI\_B module, if it matches any of the slave addresses defined in UCBxI2COA1 to UCBxI2COA3.

---

**NOTE: UCSWACK and slave-transmitter**

If the user selects manual acknowledge of slave addresses, the TXIFG is set if the slave is addressed as a transmitter. If the user decides not to acknowledge the address, the TXIFG0 also must be reset.

---

### 39.3.10 Using the eUSCI\_B Module in I<sup>2</sup>C 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 I<sup>2</sup>C 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 I<sup>2</sup>C 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.

### 39.3.11 eUSCI\_B Interrupts in I<sup>2</sup>C Mode

The eUSCI\_B has only one interrupt vector that is shared for transmission, reception, and the state change.

Each interrupt flag has its own interrupt enable bit. When an interrupt is enabled and the GIE bit is set, the interrupt flag generates an interrupt request. DMA transfers are controlled by the UCTXIFGx and UCRXIFGx flags on devices with a DMA controller. It is possible to react on each slave address with an individual DMA channel.

All interrupt flags are not cleared automatically, but they need to be cleared together by user interactions (for example, reading the UCRXBUFFL 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.

#### 39.3.11.1 I<sup>2</sup>C 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 UCTXIE 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)

UCTXIE is reset after a PUC or when UCSWRST = 1.

#### 39.3.11.2 Early I<sup>2</sup>C 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.

#### 39.3.11.3 I<sup>2</sup>C 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 UCRXIE and GIE are also set. UCRXIFGx and UCRXIE are reset after a PUC signal or when UCSWRST = 1. UCRXIFGx is automatically reset when UCxRXBUF is read.

### 39.3.11.4 I<sup>2</sup>C State Change Interrupt Operation

[Table 39-2](#) describes the I<sup>2</sup>C state change interrupt flags.

**Table 39-2. I<sup>2</sup>C State Change Interrupt Flags**

Interrupt Flag	Interrupt Condition
UCALIFG	Arbitration-lost. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the 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 I <sup>2</sup> C controller becomes a slave.
UCNACKIFG	Not-acknowledge interrupt. This flag is set when an acknowledge is expected but is not received. UCNACKIFG is used in master mode only.
UCCLTOIFG	Clock low timeout. This interrupt flag is set, if the clock is held low longer than defined by the UCCLTO bits.
UCBIT9IFG	This interrupt flag is generated each time the eUSCI_B is transferring 9th clock cycle of a byte of data. This gives the user the possibility to follow the I <sup>2</sup> C communication in software if wanted. The 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 UCASPx = 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 I <sup>2</sup> C module detects a START condition together with its own address <sup>(1)</sup> . UCSTTIFG is used in slave mode only.
UCSTPIFG	STOP condition detected interrupt. This flag is set when the I <sup>2</sup> C module detects a STOP condition on the bus. UCSTPIFG is used in slave and master mode.

<sup>(1)</sup> The address evaluation includes the address mask register if it is used.

### 39.3.11.5 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 39-3](#) 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 39-3. 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 0xa: ... // Vector 10: RXIFG3
            break;
        case 0xc: ... // Vector 14: TXIFG3
            break;
        case 0xe: ... // Vector 16: RXIFG2
            break;
        case 0x10: ... // Vector 18: TXIFG2
            break;
        case 0x12: ... // Vector 20: RXIFG1
            break;
        case 0x14: ... // Vector 22: TXIFG1
            break;
        case 0x16: ... // Vector 24: RXIFG0
            break;
        case 0x18: ... // Vector 26: TXIFG0
            break;
        case 0x1a: ... // Vector 28: BCNTIFG
            break;
        case 0x1c: ... // Vector 30: clock low timeout
            break;
        case 0x1e: ... // Vector 32: 9th bit
            break;
        default: break;
    }
}
```

### 39.4 eUSCI\_B I2C Registers

The eUSCI\_B registers applicable in I2C mode and their address offsets are listed in [Table 39-3](#). The base address can be found in the device-specific data sheet.

**Table 39-3. eUSCI\_B Registers**

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	eUSCI_Bx Control Word 0	Read/write	Word	01C1h	<a href="#">Section 39.4.1</a>
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	<a href="#">Section 39.4.2</a>
06h	UCBxBRW	eUSCI_Bx Bit Rate Control Word	Read/write	Word	0000h	<a href="#">Section 39.4.3</a>
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	<a href="#">Section 39.4.4</a>
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	<a href="#">Section 39.4.5</a>
0Ch	UCBxRXBUF	eUSCI_Bx Receive Buffer	Read/write	Word	00h	<a href="#">Section 39.4.6</a>
0Eh	UCBxTXBUF	eUSCI_Bx Transmit Buffer	Read/write	Word	00h	<a href="#">Section 39.4.7</a>
14h	UCBxI2COA0	eUSCI_Bx I2C Own Address 0	Read/write	Word	0000h	<a href="#">Section 39.4.8</a>
16h	UCBxI2COA1	eUSCI_Bx I2C Own Address 1	Read/write	Word	0000h	<a href="#">Section 39.4.9</a>
18h	UCBxI2COA2	eUSCI_Bx I2C Own Address 2	Read/write	Word	0000h	<a href="#">Section 39.4.10</a>
1Ah	UCBxI2COA3	eUSCI_Bx I2C Own Address 3	Read/write	Word	0000h	<a href="#">Section 39.4.11</a>
1Ch	UCBxADDRX	eUSCI_Bx Received Address Register	Read	Word		<a href="#">Section 39.4.12</a>
1Eh	UCBxADDMASK	eUSCI_Bx Address Mask Register	Read/write	Word	03FFh	<a href="#">Section 39.4.13</a>
20h	UCBxI2CSA	eUSCI_Bx I2C Slave Address	Read/write	Word	0000h	<a href="#">Section 39.4.14</a>
2Ah	UCBxIE	eUSCI_Bx Interrupt Enable	Read/write	Word	0000h	<a href="#">Section 39.4.15</a>
2Ch	UCBxIFG	eUSCI_Bx Interrupt Flag	Read/write	Word	2A02h	<a href="#">Section 39.4.16</a>
2Eh	UCBxIV	eUSCI_Bx Interrupt Vector	Read	Word	0000h	<a href="#">Section 39.4.17</a>

**39.4.1 UCBxCTLW0 Register**

eUSCI\_Bx Control Word Register 0

**Figure 39-17. 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
UCSELx	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 39-4. UCBxCTLW0 Register Description**

Bit	Field	Type	Reset	Description
15	UCA10	RW	0h	Own addressing mode select. Modify only when UCSWRST = 1. 0b = Own address is a 7-bit address. 1b = Own address is a 10-bit address.
14	UCSLA10	RW	0h	Slave addressing mode select 0b = Address slave with 7-bit address 1b = Address slave with 10-bit address
13	UCMM	RW	0h	Multi-master environment select. Modify only when UCSWRST = 1. 0b = Single master environment. There is no other master in the system. The address compare unit is disabled. 1b = Multi-master environment
12	Reserved	R	0h	Reserved
11	UCMST	RW	0h	Master mode select. When a master loses arbitration in a multi-master environment (UCMM = 1), the UCMST bit is automatically cleared and the module acts as slave. 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI_B mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. Modify only when UCSWRST = 1. 00b = 3-pin SPI 01b = 4-pin SPI (master or slave enabled if STE = 1) 10b = 4-pin SPI (master or slave enabled if STE = 0) 11b = I2C mode
8	UCSYNC	RW	1h	Synchronous mode enable. For eUSCI_B always read and write as 1.
7-6	UCSELx	RW	3h	eUSCI_B clock source select. These bits select the BRCLK source clock. These bits are ignored in slave mode. Modify only when UCSWRST = 1. 00b = UCLKI 01b = ACLK 10b = SMCLK 11b = SMCLK

**Table 39-4. UCBxCTLW0 Register Description (continued)**

Bit	Field	Type	Reset	Description
5	UCTXACK	RW	0h	<p>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.</p> <p>0b = Do not acknowledge the slave address 1b = Acknowledge the slave address</p>
4	UCTR	RW	0h	<p>Transmitter/receiver</p> <p>0b = Receiver 1b = Transmitter</p>
3	UCTXNACK	RW	0h	<p>Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted. Only for slave receiver mode.</p> <p>0b = Acknowledge normally 1b = Generate NACK</p>
2	UCTXSTP	RW	0h	<p>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.</p> <p>0b = No STOP generated 1b = Generate STOP</p>
1	UCTXSTT	RW	0h	<p>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.</p> <p>0b = Do not generate START condition 1b = Generate START condition</p>
0	UCSWRST	RW	1h	<p>Software reset enable.</p> <p>Modify only when UCSWRST = 1.</p> <p>0b = Disabled. eUSCI_B released for operation. 1b = Enabled. eUSCI_B logic held in reset state.</p>

**39.4.2 UCBxCTLW1 Register**

eUSCI\_Bx Control Word Register 1

**Figure 39-18. 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	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
Modify only when UCSWRST = 1.							

**Table 39-5. UCBxCTLW1 Register Description**

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved
8	UCETXINT	RW	0h	<p>Early UCTXIFG0. Only in slave mode. When this bit is set, the slave addresses defined in UCxI2COA1 to UCxI2COA3 must be disabled.</p> <p>Modify only when UCSWRST = 1.</p> <p>0b = UCTXIFGx is set after an address match with UCxI2COAx and the direction bit indicating slave transmit</p> <p>1b = UCTXIFG0 is set for each START condition</p>
7-6	UCCLTO	RW	0h	<p>Clock low timeout select.</p> <p>Modify only when UCSWRST = 1.</p> <p>00b = Disable clock low timeout counter</p> <p>01b = 135 000 MODCLK cycles (approximately 28 ms)</p> <p>10b = 150 000 MODCLK cycles (approximately 31 ms)</p> <p>11b = 165 000 MODCLK cycles (approximately 34 ms)</p>
5	UCSTPNACK	RW	0h	<p>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.</p> <p>Modify only when UCSWRST = 1.</p> <p>0b = Send a non-acknowledge before the STOP condition as a master receiver (conform to I2C standard)</p> <p>1b = All bytes are acknowledged by the eUSCI_B when configured as master receiver</p>
4	UCSWACK	RW	0h	<p>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.</p> <p>0b = The address acknowledge of the slave is controlled by the eUSCI_B module</p> <p>1b = The user needs to trigger the sending of the address ACK by issuing UCTXACK</p>
3-2	UCASTPx	RW	0h	<p>Automatic STOP condition generation. In slave mode only UCBCNTIFG is available.</p> <p>Modify only when UCSWRST = 1.</p> <p>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.</p> <p>01b = UCBCNTIFG is set with the byte counter reaches the threshold defined in UCBxTBCNT</p> <p>10b = A STOP condition is generated automatically after the byte counter value reached UCBxTBCNT. UCBCNTIFG is set with the byte counter reaching the threshold.</p> <p>11b = Reserved</p>

**Table 39-5. 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

### 39.4.3 UCBxBRW Register

eUSCI\_Bx Bit Rate Control Word Register

**Figure 39-19. 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 39-6. UCBxBRW Register Description**

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler. Modify only when UCSWRST = 1.

### 39.4.4 UCBxSTATW

eUSCI\_Bx Status Word Register

**Figure 39-20. 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 39-7. 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

### **39.4.5 UCBxTBCNT Register**

eUSCI\_Bx Byte Counter Threshold Register

**Figure 39-21. 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 39-8. 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 I <sup>2</sup> C data bytes after which the automatic STOP or the UCSTPIFG should occur. This value is evaluated only if UCSTPx is different from 00.  Modify only when UCSWRST = 1.

### 39.4.6 UCBxRXBUF Register

eUSCI\_Bx Receive Buffer Register

**Figure 39-22. 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 39-9. 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.

### 39.4.7 UCBxTXBUF

eUSCI\_Bx Transmit Buffer Register

**Figure 39-23. 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 39-10. 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.

### 39.4.8 UCBxI2COA0 Register

eUSCI\_Bx I2C Own Address 0 Register

**Figure 39-24. 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 39-11. 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 eUSCI_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.

### 39.4.9 UCBxI2COA1 Register

eUSCI\_Bx I2C Own Address 1 Register

**Figure 39-25. 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 39-12. 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.

### 39.4.10 UCBxI2COA2 Register

eUSCI\_Bx I2C Own Address 2 Register

**Figure 39-26. 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 39-13. 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.

### 39.4.11 UCBxI2COA3 Register

eUSCI\_Bx I2C Own Address 3 Register

**Figure 39-27. 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 39-14. 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.

### 39.4.12 UCBxADDRX Register

eUSCI\_Bx I2C Received Address Register

**Figure 39-28. 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 39-15. 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.

**39.4.13 UCBxADDMASK Register**

eUSCI\_Bx I2C Address Mask Register

**Figure 39-29. 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 39-16. 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.

**39.4.14 UCBxI2CSA Register**

eUSCI\_Bx I2C Slave Address Register

**Figure 39-30. 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 39-17. 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 eUSCI_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.

### **39.4.15 UCBxIE Register**

eUSCI\_Bx I2C Interrupt Enable Register

**Figure 39-31. 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 39-18. UCBxIE Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved
14	UCBIT9IE	RW	0h	Bit position 9 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
13	UCTXIE3	RW	0h	Transmit interrupt enable 3 0b = Interrupt disabled 1b = Interrupt enabled
12	UCRXIE3	RW	0h	Receive interrupt enable 3 0b = Interrupt disabled 1b = Interrupt enabled
11	UCTXIE2	RW	0h	Transmit interrupt enable 2 0b = Interrupt disabled 1b = Interrupt enabled
10	UCRXIE2	RW	0h	Receive interrupt enable 2 0b = Interrupt disabled 1b = Interrupt enabled
9	UCTXIE1	RW	0h	Transmit interrupt enable 1 0b = Interrupt disabled 1b = Interrupt enabled
8	UCRXIE1	RW	0h	Receive interrupt enable 1 0b = Interrupt disabled 1b = Interrupt enabled
7	UCCLTOIE	RW	0h	Clock low timeout interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
6	UCBCNTIE	RW	0h	Byte counter interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
5	UCNACKIE	RW	0h	Not-acknowledge interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
4	UCALIE	RW	0h	Arbitration lost interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
3	UCSTPIE	RW	0h	STOP condition interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

**Table 39-18. UCBxE 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

### 39.4.16 UCBxIFG Register

eUSCI\_Bx I2C Interrupt Flag Register

**Figure 39-32. 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 39-19. UCBxIFG Register Description**

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved
14	UCBIT9IFG	RW	0h	Bit position 9 interrupt flag 0b = No interrupt pending 1b = Interrupt pending
13	UCTXIFG3	RW	1h	eUSCI_B transmit interrupt flag 3. UCTXIFG3 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA3 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
12	UCRXIFG3	RW	0h	Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
11	UCTXIFG2	RW	0h	eUSCI_B transmit interrupt flag 2. UCTXIFG2 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA2 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
10	UCRXIFG2	RW	0h	Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
9	UCTXIFG1	RW	1h	eUSCI_B transmit interrupt flag 1. UCTXIFG1 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA1 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
8	UCRXIFG1	RW	0h	Receive interrupt flag 1. UCRXIFG1 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA1 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
7	UCCLTOIFG	RW	0h	Clock low timeout interrupt flag 0b = No interrupt pending 1b = Interrupt pending
6	UCBCNTIFG	RW	0h	Byte counter interrupt flag. When using this interrupt the user needs to ensure enough processing bandwidth (see the Byte Counter Interrupt section). 0b = No interrupt pending 1b = Interrupt pending

**Table 39-19. 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

### 39.4.17 UCBxIV Register

eUSCI\_Bx Interrupt Vector Register

**Figure 39-33. 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 39-20. UCBxIV Register Description**

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	<p>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.</p> <p>00h = No interrupt pending</p> <p>02h = Interrupt Source: Arbitration lost; Interrupt Flag: UCALIFG; Interrupt Priority: Highest</p> <p>04h = Interrupt Source: Not acknowledgment; Interrupt Flag: UCNACKIFG</p> <p>06h = Interrupt Source: Start condition received; Interrupt Flag: UCSTTIFG</p> <p>08h = Interrupt Source: Stop condition received; Interrupt Flag: UCSTPIFG</p> <p>0Ah = Interrupt Source: Slave 3 Data received; Interrupt Flag: UCRXIFG3</p> <p>0Ch = Interrupt Source: Slave 3 Transmit buffer empty; Interrupt Flag: UCTXIFG3</p> <p>0Eh = Interrupt Source: Slave 2 Data received; Interrupt Flag: UCRXIFG2</p> <p>10h = Interrupt Source: Slave 2 Transmit buffer empty; Interrupt Flag: UCTXIFG2</p> <p>12h = Interrupt Source: Slave 1 Data received; Interrupt Flag: UCRXIFG1</p> <p>14h = Interrupt Source: Slave 1 Transmit buffer empty; Interrupt Flag: UCTXIFG1</p> <p>16h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG0</p> <p>18h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG0</p> <p>1Ah = Interrupt Source: Byte counter zero; Interrupt Flag: UCBCNTIFG</p> <p>1Ch = Interrupt Source: Clock low timeout; Interrupt Flag: UCCLTOIFG</p> <p>1Eh = Interrupt Source: Nineth bit position; Interrupt Flag: UCBIT9IFG; Priority: Lowest</p>

## **USB Module**

This chapter describes the USB module that is available in some devices.

<b>Topic</b>	<b>Page</b>
<b>40.1 USB Introduction .....</b>	<b>1075</b>
<b>40.2 USB Operation .....</b>	<b>1077</b>
<b>40.3 USB Transfers .....</b>	<b>1088</b>
<b>40.4 USB Registers .....</b>	<b>1095</b>

## 40.1 USB Introduction

The features of the USB module include:

- Fully compliant with the USB 2.0 Full-speed specification
  - Full-speed device (12 Mbps) with integrated USB transceiver (PHY)
  - Up to eight input and eight output endpoints
  - Supports control, interrupt, and bulk transfers
  - Supports USB suspend, resume, and remote wakeup
- A power supply system independent from the PMM system
  - Integrated 3.3-V LDO regulator with sufficient output to power entire MSP430 and system circuitry from 5-V VBUS
  - Integrated 1.8-V LDO regulator for PHY and PLL
  - Easily used in either bus-powered or self-powered operation
  - Current-limiting capability on 3.3-V LDO output
  - Autonomous power-up of MSP430 upon arrival of USB power possible (low or no battery condition)
- Internal 48-MHz USB clock
  - Integrated programmable PLL
  - Highly-flexible input clock frequencies for use with lowest-cost crystals
- 1904 bytes of dedicated USB buffer space for endpoints, with fully configurable size to a granularity of eight bytes
- Timestamp generator with 62.5-ns resolution
- When USB is disabled
  - Buffer space is mapped into general RAM, providing additional 2KB to the system
  - USB interface pins become high-current general purpose I/O pins

---

**NOTE: Use of the word *device***

The word *device* is used throughout the chapter. This word can mean one of two things, depending on the context. In a USB context, it means what the USB specification refers to as a device, function, or peripheral; that is, a piece of equipment that can be attached to a USB host or hub. In a semiconductor context, it refers to an integrated circuit such as the MSP430.

To avoid confusion, the term *USB device* in this document refers to the USB-context meaning of the word. The word *device* by itself refers to silicon devices such as the MSP430.

---

Figure 40-1 shows a block diagram of the USB module.

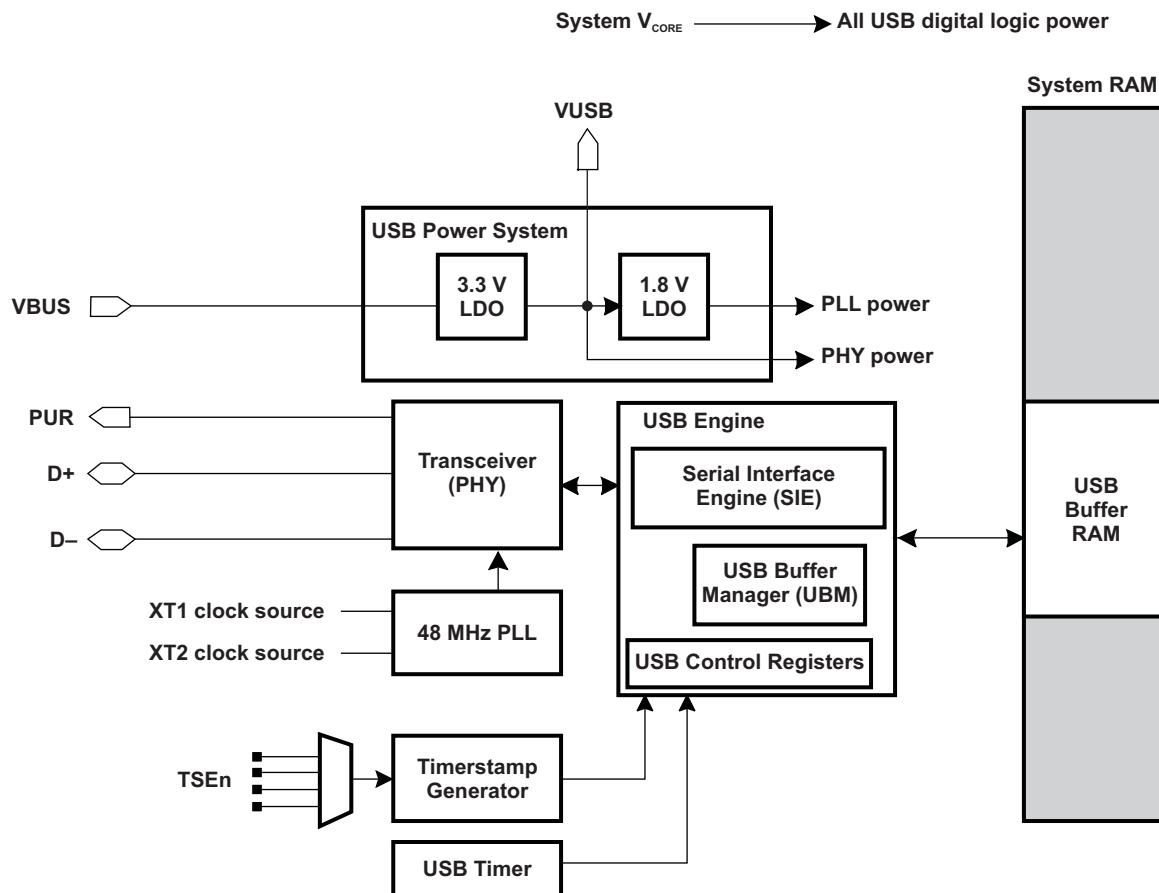


Figure 40-1. USB Block Diagram

## 40.2 USB Operation

The USB module is a comprehensive full-speed USB device compliant with the USB 2.0 specification.

The USB engine coordinates all USB-related traffic. It consists of the USB SIE (serial interface engine) and USB Buffer Manager (UBM). All traffic received on the USB receive path is de-serialized and placed into receive buffers in the USB buffer RAM. Data in the buffer RAM marked 'ready to be sent' are serialized into packets and sent to the USB host.

The USB engine requires an accurate 48-MHz clock to sample the incoming data stream. This is generated by a PLL that is fed from one of the system oscillators (XT1 or XT2). A crystal of 4 MHz or greater is required. In addition to crystal operation, the crystal bypass mode can also be used to supply the clock required by the PLL. The PLL is very flexible and can adapt to a wide range of crystal and input frequencies, allowing for cost-effective clock designs.

---

**NOTE:** The reference clock to the PLL depends on the device configuration. On devices that contain the optional XT2, the reference clock to the PLL is XT2CLK, regardless of whether or not XT1 is available. If the device has only XT1, then the reference is XT1CLK. See the device-specific data sheet for clock sources available.

---

**NOTE:** The USB module only supports active operation during power modes AM through LPM1.

---

The USB buffer memory is where data is exchanged between the USB interface and the application software. It is also where the usage of endpoints 1 to 7 are defined. This buffer memory is implemented such that it can be easily accessed like RAM by the CPU or DMA while USB module is not in suspend condition.

### 40.2.1 USB Transceiver (PHY)

The physical layer interface (USB transceiver) is a differential line driver directly powered from VUSB (3.3 V). The line driver is connected to the DP and DM pins, which form the signaling mechanism of the USB interface.

When the PUSEL bit is set, DP and DM are configured to function as USB drivers controlled by the USB core logic. When the bit is cleared, these two pins become "Port U", which is a pair of high-current general purpose I/O pins. In this case, the pins are controlled by the Port U control registers. Port U is powered from the VUSB rail, separate from the main device DVCC. If these pins are to be used, whether for USB or general purpose use, it is necessary that VUSB be properly powered from either the internal regulators or an external source.

#### 40.2.1.1 D+ Pullup Via PUR Pin

When a full-speed USB device is attached to a USB host, it must pull up the D+ line (DP pin) for the host to recognize its presence. The MSP430 USB module implements this with a software-controlled pin that activates a pullup resistor. The bit that controls this function is PUR\_EN. If software control is not desired, the pullup can be connected directly to VUSB.

#### 40.2.1.2 Shorts on Damaged Cables and Clamping

USB devices must tolerate connection to a cable that is damaged, such that it has developed shorts on either ground or VBUS. The device should not become damaged by this event, either electrically or physically. To this end, the MSP430 USB power system features a current limitation mechanism that limits the available transceiver current in the event of a short to ground. The transceiver interface itself therefore does not need a current limiting function.

Note that if VUSB is to be powered from a source other than the integrated regulator, the absence of current-limiting in the transceiver means that the external power source must itself be tolerant of this same shorting event, through its own means of current limiting.

#### 40.2.1.3 Port U Control

When PUSEL is cleared, the Port U pins (PU.0 and PU.1) function as general-purpose, high-current I/O pins. These pins can only be configured together as either both inputs or both outputs. Port U is supplied by the VUSB rail. If the 3.3-V LDO is not being used in the system (disabled), the VUSB pin can be supplied externally.

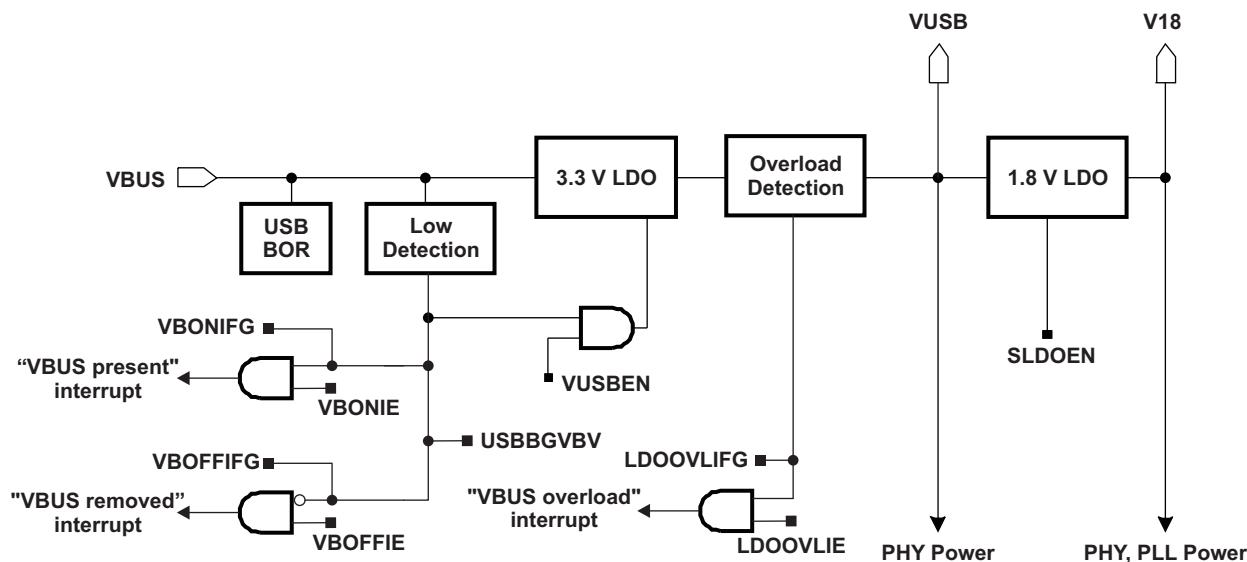
PUOPE controls the enable of both outputs residing on the Port U pins. Setting PUIPE = 1 causes both input buffers to be enabled. When Port U outputs are enabled (PUOPE = 1), the PUIN0 and PUIN1 pins mirror what is present on the outputs assuming PUIPE = 1. To use the Port U pins as inputs, the outputs should be disabled by setting PUOPE = 0, and enabling the input buffers by setting PUIPE = 1. Once configured as inputs (PUIPE = 1), the PUIN0 and PUIN1 bits can be read to determine the respective input values.

When PUOPE is set, both Port U pins function as outputs, controlled by PUOUT0 and PUOUT1. When driven high, they use the VUSB rail, and they are capable of a drive current higher than other I/O pins on the device. See the device-specific datasheet for parameters.

By default, PUOPE and PUIPE are cleared. PU.0 and PU.1 are high-impedance (input buffers are disabled and outputs are disabled).

#### 40.2.2 USB Power System

The USB power system incorporates dual LDO regulators (3.3 V and 1.8 V) that allow the entire MSP430 device to be powered from 5-V VBUS when it is made available from the USB host. Alternatively, the power system can supply power only to the USB module, or it can be unused altogether, as in a fully self-powered device. The block diagram is shown in [Figure 40-2](#).



**Figure 40-2. USB Power System**

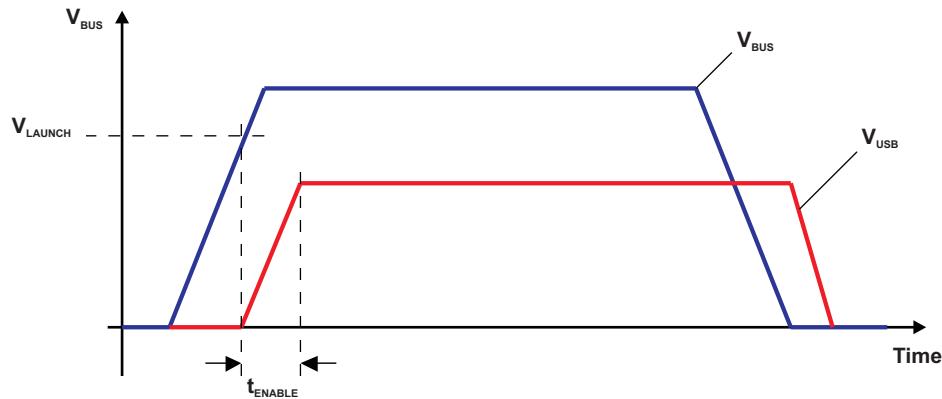
The 3.3-V LDO receives 5 V from VBUS and provides power to the transceiver, as well as the VUSB pin. Using this setup prevents the relatively high load of the transceiver and PLL from loading a local system power supply, if used. Thus it is very useful in battery-powered devices.

The 1.8-V LDO receives power from the VUSB pin – which is to be sourced either from the internal 3.3-V LDO or externally – and provides power to the USB PLL and transceiver. The 1.8-V LDO in the USB module is not related to the LDO that resides in the MSP430 Power Management Module (PMM).

The inputs and outputs of the LDOs are shown in [Figure 40-2](#). VBUS, VUSB, and V18 need to be connected to external capacitors. The V18 pin is not intended to source other components in the system, rather it exists solely for the attachment of a load capacitor.

#### 40.2.2.1 Enabling and Disabling

The 3.3-V LDO is enabled or disabled by setting or clearing VUSBEN, respectively. Even if enabled, if the voltage on VBUS is detected to be low or nonexistent, the LDO is suspended. No additional current is consumed while the LDO is suspended. When VBUS rises above the USB power brownout level, the LDO reference and low voltage detection become enabled. When VBUS rises further above the launch voltage  $V_{LAUNCH}$ , the LDO module becomes enabled (see [Figure 40-3](#)). See device specific datasheet for value of  $V_{LAUNCH}$ .



**Figure 40-3. USB Power Up and Down Profile**

The 1.8-V LDO can be enabled or disabled by setting SLDOEN accordingly. By default, the 1.8-V LDO is controlled automatically according to whether power is available on VBUS. This auto-enable feature is controlled by SLDOAON. Please note, in this case, that the SLDOEN bit does not reflect the state of the 1.8-V LDO. If the user wishes to know the state while using the auto-enable feature, the USBBGVBV bit in USBPWRCTL can be read. In addition, to disable the 1.8-V LDO, SLDOAON must be cleared along with SLDOEN. If providing VUSB from an external source, rather than through the integrated 3.3-V LDO, keep in mind that if 5 V is not present on VBUS, the 1.8-V LDO is not automatically enabled. In this situation, either VBUS must be attached to USB bus power, or the SLDOAON bit must be cleared and SLDOEN set.

It is required that power from the USB cable's VBUS be directed through a Schottky diode prior to entering the VBUS terminal. This prevents current from draining into the cable's VBUS from the LDO input, allowing the MSP430 to tolerate a suspended or unpowered USB cable that remains electrically connected.

The VBONIFG flag can be used to indicate that the voltage on VBUS has risen above the launch voltage. In addition to the VBONIFG being set, an interrupt is also generated when VBONIE = 1. Similarly, the VBOFFIFG flag can be used to indicate that the voltage on VBUS has fallen below the launch voltage. In addition to the VBOFFIFG being set, an interrupt is also generated when VBOFFIE = 1. The USBBGVBV bit can also be polled to indicate the level of VBUS; that is, above or below the launch voltage.

#### 40.2.2.2 Powering the Rest of the MSP430 From USB Bus Power Via VUSB

The output of the 3.3-V LDO can be used to power the entire MSP430 device, sourcing the DVCC rail. If this is desired, the VUSB and DVCC should be connected externally. Power from the 3.3-V LDO is sourced into DVCC (see [Figure 40-4](#)).

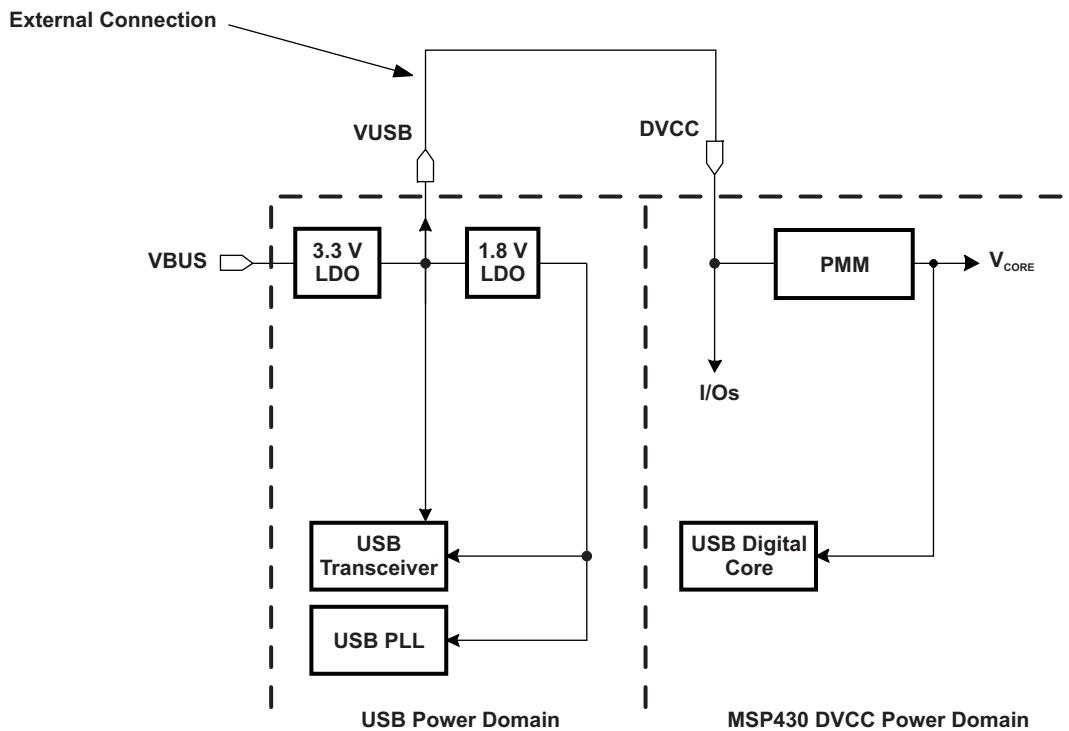


Figure 40-4. Powering Entire MSP430 From VBUS

With this connection made, the MSP430 allows for autonomous power up of the device when VBUS rises above  $V_{LAUNCH}$ . If no voltage is present on  $V_{CORE}$  – meaning the device is unpowered (or, in LPMx.5 mode) – then both the 3.3-V and 1.8-V LDOs automatically turn on when VBUS rises above  $V_{LAUNCH}$ .

Note that if DVCC is being driven from VUSB in this manner, and if power is available from VUSB, attempting to place the device into LPMx.5 results in the device immediately re-powering. This is because it re-creates the conditions of the autonomous feature described above (no  $V_{CORE}$  but power available on VBUS). The resulting drop of  $V_{CORE}$  would cause the system to immediately power up again.

When DVCC is being powered from VUSB, it is up to the user to ensure that the total current being drawn from VBUS stays below  $I_{DET}$ .

#### 40.2.2.3 Powering Other Components in the System from VUSB

There is sufficient current capacity available from the 3.3-V LDO to power not only the entire MSP430 but also other components in the system, via the VUSB pin.

If the device is to always be connected to USB, then perhaps no other power system is needed. If it only occasionally connects to USB and is battery-powered otherwise, then sourcing system power via the 3.3-V LDO takes power burden away from the battery. Alternatively, if the battery is rechargeable, the recharging can be driven from VUSB.

#### 40.2.2.4 Self-Powered Devices

Some applications may be self-powered, in that the VUSB power is supplied externally. In these cases, the 3.3-V LDO would be disabled ( $VUSBEN = 0$ ). For proper USB operation, the voltage on VBUS can still be detected, even while the 3.3-V LDO disabled, by setting  $USBDETEEN = 1$ . When VBUS rises above the USB power brownout level, low voltage detection becomes enabled. When VBUS rises further above the launch voltage  $V_{LAUNCH}$ , the voltage on VBUS is detected.

#### 40.2.2.5 Current Limitation and Overload Protection

The 3.3-V LDO features current limitation to protect the transceiver during shorted-cable conditions. A short or overload condition – that is, when the output of the LDO becomes current-limited to  $I_{DET}$ . This is reported to software via the VUOVLIFG flag. See device specific datasheet for value of  $I_{DET}$ .

If this event occurs, it means USB operation may become unreliable, due to insufficient power supply. As a result, software may wish to cease USB operation. If the OVLAOFF bit is set, USB operation is automatically terminated by clearing VUSBEN.

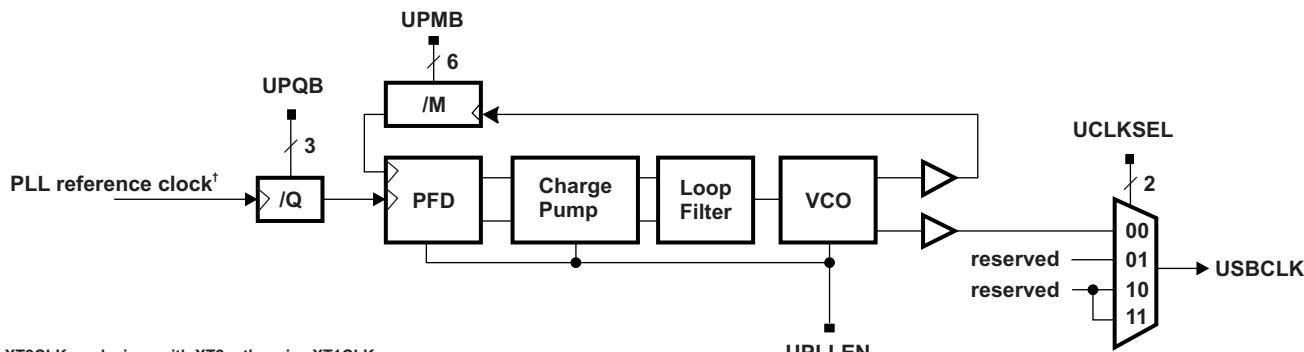
During overload conditions, VUSB and V18 drop below their nominal output voltage. In power scenarios where DVCC is exclusively supplied from VUSB, repetitive system restarts may be triggered as long the short or overload condition exists. For this reason, firmware should avoid re-enabling USB after detection of an overload on the previous power session, until the cause of failure can be identified. Ultimately, it is the user's responsibility to ensure that the current drawn from VBUS does not exceed  $I_{DET}$ .

The VUOVLIFG flag can be used to indicate an overcurrent condition on the 3.3-V LDO. When an overcurrent condition is detected, VUOVLIFG = 1. In addition to the VUOVLIFG being set, an interrupt is also generated when VUOVLIE = 1.

The USB power system brownout circuit is supplied from VBUS or DVCC, whichever carries the higher voltage.

#### 40.2.3 USB Phase-Locked Loop (PLL)

The PLL provides the low-jitter high-accuracy clock needed for USB operation (see [Figure 40-5](#)).



<sup>†</sup> XT2CLK on devices with XT2, otherwise XT1CLK.

**Figure 40-5. USB-PLL Analog Block Diagram**

The reference clock to the PLL depends on the device configuration. On devices that contain the optional XT2, the reference clock to the PLL is XT2CLK, regardless if XT1 is available. If the device has only XT1, then the reference is XT1CLK. A four-bit prescale counter controlled by the UPQB bits allows division of the reference to generate the PLL update clock. The UPMB bits control the divider in the feedback path and define the multiplication rate of the PLL (see [Equation 16](#)).

$$f_{OUT} = CLK_{SEL} \times \frac{DIVM}{DIVQ} \quad \text{with} \quad \frac{CLK_{SEL}}{DIVQ} = f_{UPD} \geq 1.5 \text{ MHz} \quad (16)$$

Where

$CLK_{SEL}$  is the PLL reference clock frequency

$DIVQ$  is derived from [Table 40-1](#)

$DIVM$  represents the value of UPMB field

[Table 40-2](#) lists some common clock input frequencies for  $CLK_{SEL}$ , along with the appropriate register settings for generating the nominal 48-MHz clock required by the USB serial engine. For crystal operation, a 4 MHz or higher crystal is required. For crystal bypass mode of operation, 1.5 MHz is the lowest external clock input possible for  $CLK_{SEL}$ .

If USB operation is used in a bus-powered configuration, disabling the PLL is necessary to pass the USB requirement of not consuming more than 500  $\mu$ A. The UPLLEN bit enables or disables the PLL. The PFDEN bit must be set to enable the phase and frequency discriminator. Out-of-lock, loss-of-signal, and out-of-range are indicated and flagged in the interrupt flags OOLIFG, LOSIFG, OORIFG, respectively.

**NOTE:** UCLKSEL bits should always be cleared, which is the default operation. All other combinations are reserved for future usages.

**Table 40-1. USB-PLL Pre-Scale Divider**

UPQB	DIVQ
000	1
001	2
010	3
011	4
100	6
101	8
110	13
111	16

**Table 40-2. Register Settings to Generate 48 MHz Using Common Clock Input Frequencies**

CLK <sub>SEL</sub> (MHz)	UPQB	UPMB	DIVQ	DIVM	CLKLOOP (MHz)	UPLLCLK (MHz)	ACCURACY (ppm)
1.5	000	011111	1	32	1.5	48	0
1.6	000	011101	1	30	1.6	48	0
1.7778	000	011010	1	27	1.7778	48	0
1.8432	000	011001	1	26	1.8432	47.92	-1570
1.8461	000	011001	1	26	1.8461	48	0
1.92	000	011000	1	25	1.92	48	0
2	000	010111	1	24	2	48	0
2.4	000	010011	1	20	2.4	48	0
2.6667	000	010001	1	18	2.6667	48	0
3	000	001111	1	16	3	48	0
3.2	001	011110	2	30	1.6	48	0
3.5556	001	011010	2	27	1.7778	48	0
3.84	001	011001	2	25	1.92	48	0
4 <sup>(1)</sup>	001	010111	2	24	2	48	0
4.1739	001	010110	2	23	2.086	48	0
4.3636	001	010101	2	22	2.1818	48	0
4.5	010	011111	3	32	1.5	48	0
4.8	001	010011	2	20	2.4	48	0
5.33 $\pm$ (16/3)	001	010001	2	18	2.6667	48	0
5.76	010	011000	3	25	1.92	48	0
6	010	010111	3	24	2	48	0
6.4	011	011101	4	30	1.6	48	0
7.2	010	010011	3	20	2.4	48	0
7.68	011	011000	4	25	1.92	48	0
8 <sup>(1)</sup>	010	010001	3	18	2.6667	48	0
9	010	001111	3	16	3	48	0

<sup>(1)</sup> This frequency can be automatically detected by the factory-supplied BSL, for use in production programming of the MSP430 via USB. See the *MSP430 Programming Via the Bootstrap Loader User's Guide* ([SLAU319](#)) for details.

**Table 40-2. Register Settings to Generate 48 MHz Using Common Clock Input Frequencies (continued)**

<b>CLK<sub>SEL</sub> (MHz)</b>	<b>UPQB</b>	<b>UPMB</b>	<b>DIVQ</b>	<b>DIVM</b>	<b>CLKLOOP (MHz)</b>	<b>UPLLCLK (MHz)</b>	<b>ACCURACY (ppm)</b>
9.6	011	010011	4	20	2.4	48	0
10.66 $\neq$ (32/3)	011	010001	4	18	2.6667	48	0
12 <sup>(1)</sup>	011	001111	4	16	3	48	0
12.8	101	011101	8	30	1.6	48	0
14.4	100	010011	6	20	2.4	48	0
16	100	010001	6	18	2.6667	48	0
16.9344	100	010000	6	17	2.8224	47.98	-400
16.94118	100	010000	6	17	2.8235	48	0
18	100	001111	6	16	3	48	0
19.2	101	010011	8	20	2.4	48	0
24 <sup>(1)</sup>	101	001111	8	16	3	48	0
25.6	111	011101	16	30	1.6	48	0
26.0	110	010111	13	24	2	48	0
32	111	010111	16	24	2.6667	48	0

#### 40.2.3.1 Modifying the Divider Values

Updating the values of UPQB (DIVQ) and UPMB (DIVM) to select the desired PLL frequency must occur simultaneously to avoid spurious frequency artifacts. The values of UPQB and UPMB can be calculated and written to their buffer registers; the final update of UPQB and UPMB occurs when the upper byte of UPLL DIVB (UPQB) is written.

#### 40.2.3.2 PLL Error Indicators

The PLL can detect three kinds of errors. Out-of-lock (OOL) is indicated if a frequency correction is performed in the same direction (that is, up or down) for four consecutive update periods. Loss-of-signal (LOS) is indicated if a frequency correction is performed in the same direction (that is, up or down) for 16 consecutive update periods. Out-of-range (OOR) is indicated if PLL was unable to lock for more than 32 update periods.

OOL, LOS, and OOR trigger their respective interrupt flags (USBOOLIFG, USBLOSIFG, USBOORIFG) if errors occur, and interrupts are generated if enabled by their enable bits (USBOOLIE, USBLOSIE, USBOORIE).

#### 40.2.3.3 PLL Startup Sequence

To achieve the fastest startup of the PLL, the following sequence is recommended.

1. Enable VUSB and V18.
2. Wait 2 ms for external capacitors to charge, so that proper VUSB is in place. (During this time, the USB registers and buffers can be initialized.)
3. Activate the PLL, using the required divider values.
4. Wait 2 ms and check PLL. If it stays locked, it is ready to be used.

#### 40.2.4 USB Controller Engine

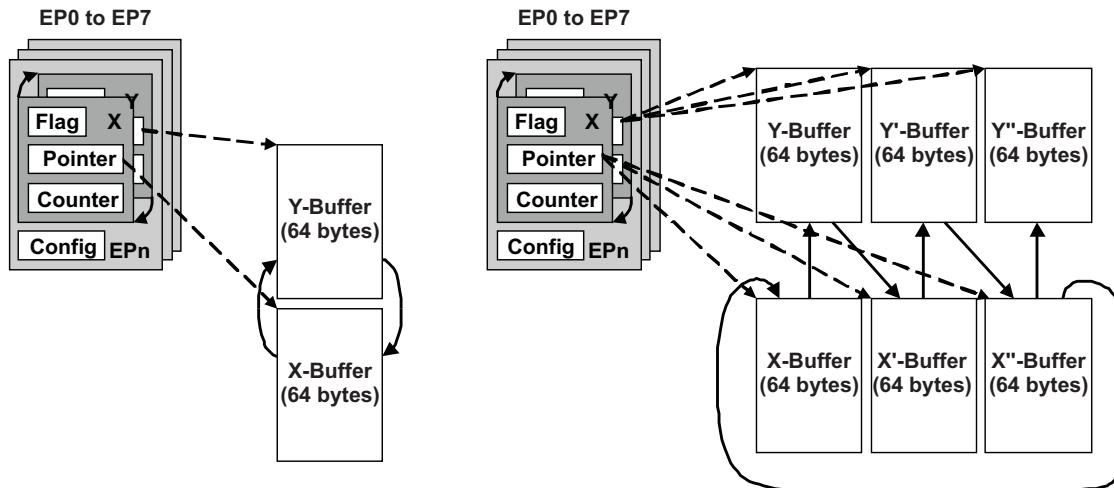
The USB controller engine transfers data packets arriving from the USB host into the USB buffers, and also transmits valid data from the buffers to the USB host. The controller engine has dedicated, fixed buffer space for input endpoint 0 and output endpoint 0, which are the default USB endpoints for control transfers.

The 14 remaining endpoints (seven input and seven output) may have one or more USB buffers assigned to them. All the buffers are located in the USB buffer memory. This memory is implemented as "multiport" memory, in that it can be accessed both by the USB buffer manager and also by the CPU and DMA.

Each endpoint has a dedicated set of descriptor registers that describe the use of that endpoint (see [Figure 40-6](#)). Configuration of each endpoint is performed by setting its descriptor registers. These data structures are located in the USB buffer memory and contain address pointers to the next memory buffer for receive or transmit.

Assigning one or two data buffers to an endpoint, of up to 64 bytes, requires no further software involvement after configuration. If more than two buffers per endpoint are desired, however, software must change the address pointers on the fly during a receive or transmit process.

Synchronization of empty and full buffers is done using validation flags. All events are indicated by flags and fire a vector interrupt when enabled. Transfer event indication can be enabled separately.



**Figure 40-6. Data Buffers and Descriptors**

#### 40.2.4.1 USB Serial Interface Engine (SIE)

The SIE logic manages the USB packet protocol requirements for the packets being received and transmitted on the bus. For packets being received, the SIE decodes the packet identifier field (packet ID) to determine the type of packet being received and to ensure the packet ID is valid. For token and data packets being received, the SIE calculates the packet cycle redundancy check (CRC) and compares the value to the CRC contained in the packet to verify that the packet was not corrupted during transmission.

For token and data packets being transmitted, the SIE generates the CRC that is transmitted with the packet. For packets being transmitted, the SIE also generates the synchronization field (SYNC), which is an eight-bit field at the beginning of each packet. In addition, the SIE generates the correct packet ID for all packets being transmitted.

Another major function of the SIE is the overall serial-to-parallel conversion of the data packets being received or transmitted.

#### 40.2.4.2 USB Buffer Manager (UBM)

The USB buffer manager provides the control logic that interfaces the SIE to the USB endpoint buffers.

One of the major functions of the UBM is to decode the USB device address to determine if the USB host is addressing this particular USB device. In addition, the endpoint address field and direction signal are decoded to determine which particular USB endpoint is being addressed. Based on the direction of the USB transaction and the endpoint number, the UBM either writes or reads the data packet to or from the appropriate USB endpoint data buffer.

The TOGGLE bit for each output endpoint configuration register is used by the UBM to track successful output data transactions. If a valid data packet is received and the data packet ID matches the expected packet ID, the TOGGLE bit is toggled. Similarly, the TOGGLE bit for each input endpoint configuration is used by the UBM to track successful input data transactions. If a valid data packet is transmitted, the TOGGLE bit is toggled. If the TOGGLE bit is cleared, a DATA0 packet ID is transmitted in the data packet to the host. If the TOGGLE bit is set, a DATA1 packet ID is transmitted in the data packet to the host. See [Section 40.3](#) regarding details of USB transfers.

#### 40.2.4.3 USB Buffer Memory

The USB buffer memory contains the data buffers for all endpoints and for SETUP packets. In that the buffers for endpoints 1 to 7 are flexible, there are USB buffer configuration registers that define them, and these too are in the USB buffer memory. (Endpoint 0 is defined with a set of registers in the USB control register space.) Storing these in open memory allows for efficient, flexible use, which is advantageous because use of these endpoints is very application-specific.

This memory is implemented as "multiport" memory, in that it can be accessed both by the USB buffer manager and also by the CPU and DMA. The SIE allows CPU or DMA access, but reserves priority. As a result, CPU or DMA access is delayed using wait states if a conflict arises with an SIE access.

When the USB module is disabled (USBEN = 0), the buffer memory behaves like regular RAM. When changing the state of the USBEN bit (enabling or disabling the USB module), the USB buffer memory should not be accessed within four clocks before and eight clocks after changing this bit, as doing so reconfigures the access method to the USB memory.

Accessing of the USB buffer memory by CPU or DMA is only possible if the USB PLL is active. When a host requests suspend condition the application software (for example, USB stack) of client has to switch off the PLL within 10 ms. Note that the MSP430 USB suspend interrupt occurs around 5 ms after the host request.

Each endpoint is defined by a block of six configuration "registers" (based in RAM, they are not true registers in the strict sense of the word). These registers specify the endpoint type, buffer address, buffer size and data packet byte count. They define an endpoint buffer space that is 1904 bytes in size. An additional 24 bytes are allotted to three remaining blocks – the EP0\_IN buffer, the EP0\_OUT buffer, and the SETUP packet buffer (see [Table 40-3](#)).

**Table 40-3. USB Buffer Memory Map**

Memory	Short Form	Access Type	Address Offset
Start of buffer space	STABUFF	Read/Write	0000h
1904 bytes of configurable buffer space	:	Read/Write	:
End of buffer space	TOPBUFF	Read/Write	076Fh
Output endpoint_0 buffer	USBOEP0BUF	Read/Write	0770h
		Read/Write	:
		Read/Write	0777h
Input endpoint_0 buffer	USBIEP0BUF	Read/Write	0778h
		Read/Write	:
		Read/Write	077Fh
Setup Packet Block	USBSUBLK	Read/Write	0780h
		Read/Write	:
		Read/Write	0787h

Software can configure each buffer according to the total number of endpoints needed. Single or double buffering of each endpoint is possible.

Unlike the descriptor registers for endpoints 1 to 7, which are defined as memory entries in USB RAM, endpoint 0 is described by a set of four registers (two for output and two for input) in the USB control register set. Endpoint 0 has no base-address register, since these addresses are hardwired. The bit positions have been preserved to provide consistency with endpoint\_n (n = 1 to 7).

#### 40.2.4.4 USB Fine Timestamp

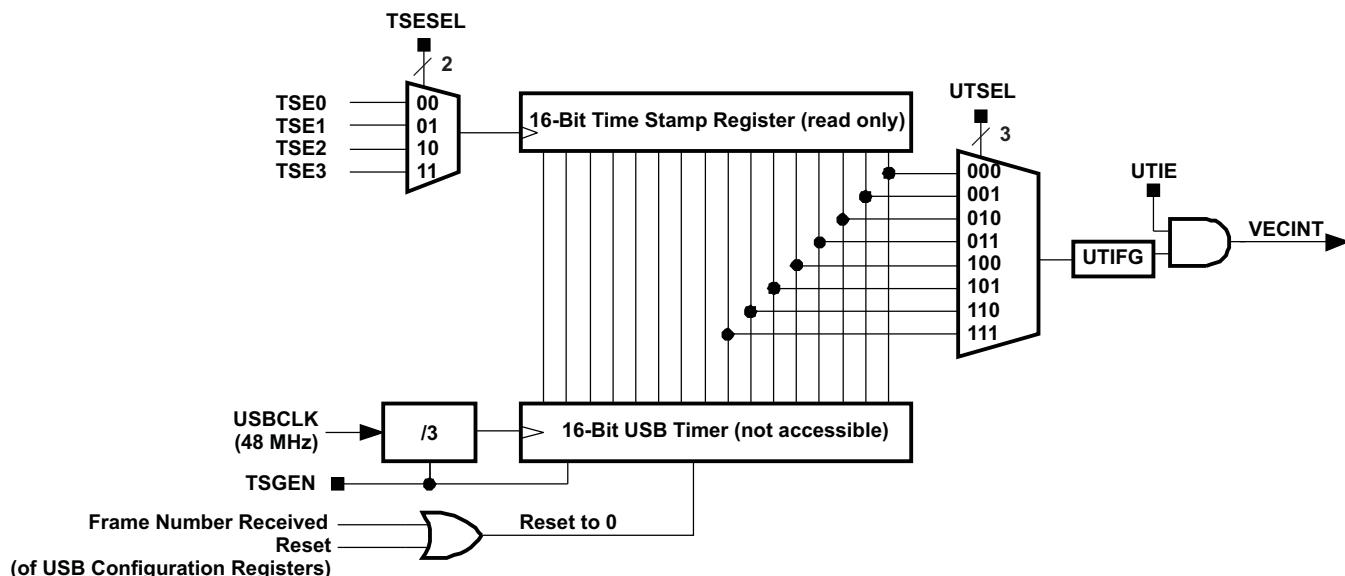
The USB module is capable of saving a timestamp associated with particular USB events (see [Figure 40-7](#)). This can be useful in compensating for delays in software response. The timestamp values are based on the USB module's internal timer, driven by USBCLK.

Up to four events can be selected to generate the timestamp, selected with the TSESEL bits. When they occur, the value of the USB timer is transferred to the timestamp register USBSREG, and thus the exact moment of the event is recorded. The trigger options include one of three DMA channels, or a software-driven event. The USB timer cannot be directly accessed by reading.

Furthermore, the value of the USB timer can be used to generate periodic interrupts. Since the USBCLK can have a frequency different from the other system clocks, this gives another option for periodic system interrupts. The UTSEL bits select the divider from the USB clock. UTIE must be set for an interrupt vector to get triggered.

The timestamp register is set to zero on a frame-number-receive event and pseudo-start-of-frame.

TSGEN enables or disables the time stamp generator.



**Figure 40-7. USB Timer and Time Stamp Generation**

#### 40.2.4.5 Suspend and Resume Logic

The USB suspend and resume logic detects suspend and resume conditions on the USB bus. These events are flagged in SUSRIFG and RESRIFG, respectively, and they fire dedicated interrupts, if the interrupts are enabled (SUSRIE and RESRIE).

The remote wakeup mechanism, in which a USB device can cause the USB host to awaken and resume the device, is triggered by setting the RWUP bit of the USBCTL register.

See [Section 40.2.6](#) for more information.

#### 40.2.4.6 Reset Logic

A PUC resets the USB module logic. When FRSTE = 1, the logic is also reset when a USB reset event occurs on the bus, triggered from the USB host. (A USB reset also sets the RSTRIFG flag.) USB buffer memory is not reset by a USB reset.

#### 40.2.5 USB Vector Interrupts

The USB module uses a single interrupt vector generator register to handle multiple USB interrupts. All USB-related interrupt sources trigger the USBVECINT (also called USBIV) vector, which then contains a 6-bit vector value that identifies the interrupt source. Each of the interrupt sources results in a different offset value read. The interrupt vector returns zero when no interrupt is pending.

Reading the interrupt vector register clears the corresponding interrupt flag and updates its value. The interrupt with highest priority returns the value 0002h; the interrupt with lowest priority returns the value 003Eh when reading the interrupt vector register. Writing to this register clears all interrupt flags.

For each input and output endpoints resides an USB transaction interrupt indication enable. Software may set this bit to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt enable and flag must be set.

**Table 40-4. USB Interrupt Vector Generation**

USBVECINT Value	Interrupt Source	Interrupt Flag Bit	Interrupt Enable Bit	Indication Enable Bit
0000h	no interrupt	–	–	–
0002h	USB-PWR drop ind.	USBPWRCTL.VUOVLIFG	USBPWRCTL.VUOVLIE	–
0004h	USB-PLL lock error	USBPLLIR.USBPLLOOLIFG	USBPLLIR.USBPLLOOLIE	–
0006h	USB-PLL signal error	USBPLLIR.USBPLLOSIFG	USBPLLIR.USBPLLOSIE	–
0008h	USB-PLL range error	USBPLLIR.USBPLLOORIFG	USBPLLIR.USBPLLOORIE	–
000Ah	USB-PWR VBUS-on	USBPWRCTL.VBONIFG	USBPWRCTL.VBONIE	–
000Ch	USB-PWR VBUS-off	USBPWRCTL.VBOFFIFG	USBPWRCTL.VBOFFIE	–
000Eh	reserved	–	–	–
0010h	USB timestamp event	USBMAINTL.UTIFG	USBMAINTL.UTIE	–
0012h	Input Endpoint-0	USBIEPIFG.EP0	USBIEPIE.EP0	USBOEPCNFG_0.USBIIE
0014h	Output Endpoint-0	USBOEPIFG.EP0	USBOEPIE.EP0	USBOEPCNFG_0.USBIIE
0016h	RSTR interrupt	USBIFG.RSTRIFG	USBIE.RSTRIE	–
0018h	SUSR interrupt	USBIFG.SUSRIFG	USBIE.SUSRIE	–
001Ah	RESR interrupt	USBIFG.RESRIFG	USBIE.RESRIE	–
001Ch	reserved	–	–	–
001Eh	reserved	–	–	–
0020h	Setup packet received	USBIFG.SETUPIFG	USBIE.SETUPIE	–
0022h	Setup packet overwrite	USBIFG.STPOWIFG	USBIE.STPOWIE	–
0024h	Input Endpoint-1	USBIEPIFG.EP1	USBIEPIE.EP1	USIEPCNF_1.USBIIE
0026h	Input Endpoint-2	USBIEPIFG.EP2	USBIEPIE.EP2	USIEPCNF_2.USBIIE
0028h	Input Endpoint-3	USBIEPIFG.EP3	USBIEPIE.EP3	USIEPCNF_3.USBIIE
002Ah	Input Endpoint-4	USBIEPIFG.EP4	USBIEPIE.EP4	USIEPCNF_4.USBIIE
002Ch	Input Endpoint-5	USBIEPIFG.EP5	USBIEPIE.EP5	USIEPCNF_5.USBIIE
002Eh	Input Endpoint-6	USBIEPIFG.EP6	USBIEPIE.EP6	USIEPCNF_6.USBIIE
0030h	Input Endpoint-7	USBIEPIFG.EP7	USBIEPIE.EP7	USIEPCNF_7.USBIIE
0032h	Output Endpoint-1	USBOEPIFG.EP1	USBOEPIE.EP1	USBOEPCNFG_1.USBIIE
0034h	Output Endpoint-2	USBOEPIFG.EP2	USBOEPIE.EP2	USBOEPCNFG_2.USBIIE
0036h	Output Endpoint-3	USBOEPIFG.EP3	USBOEPIE.EP3	USBOEPCNFG_3.USBIIE
0038h	Output Endpoint-4	USBOEPIFG.EP4	USBOEPIE.EP4	USBOEPCNFG_4.USBIIE
003Ah	Output Endpoint-5	USBOEPIFG.EP5	USBOEPIE.EP5	USBOEPCNFG_5.USBIIE
003Ch	Output Endpoint-6	USBOEPIFG.EP6	USBOEPIE.EP6	USBOEPCNFG_6.USBIIE
003Eh	Output Endpoint-7	USBOEPIFG.EP7	USBOEPIE.EP7	USBOEPCNFG_7.USBIIE

#### 40.2.6 Power Consumption

USB functionality consumes more power than is typically drawn in the MSP430. Since most MSP430 applications are power sensitive, the MSP430 USB module has been designed to protect the battery by ensuring that significant power load only occurs when attached to the bus, allowing power to be drawn from VBUS.

The two components of the USB module that draw the most current are the transceiver and the PLL. The transceiver can consume large amounts of power while transmitting, but in its quiescent state – that is, when not transmitting data – the transceiver actually consumes very little power. This is the amount specified as  $I_{IDLE}$ . This amount is so little that the transceiver can be kept active during suspend mode without presenting a problem for bus-powered applications. Fortunately the transceiver always has access to VBUS power when drawing the level of current required for transmitting.

The PLL consumes a larger amount of current. However, it need only be active while connected to the host, and the host can supply the power. When the PLL is disabled (for example, during USB suspend), USBCLK automatically is sourced from the VLO.

#### 40.2.7 Suspend and Resume

All USB devices must support the ability to be suspended into a no-activity state, and later resumed. When suspended, a device is not allowed to consume more than 500 $\mu$ A from the USB's VBUS power rail, if the device is drawing any power from that source. A suspended device must also monitor for a resume event on the bus.

The host initiates a suspend condition by creating a constant idle state on the bus for more than 3.0 ms. It is the responsibility of the software to ensure the device enters its low power suspend state within 10 ms of the suspend condition. The USB specification requires that a suspended bus-powered USB device not draw in excess of 500  $\mu$ A from the bus.

##### 40.2.7.1 Entering Suspend

When the host suspends the USB device, a suspend interrupt is generated (SUSRIFFG). From this point, the software has 10 ms to ensure that no more than 500 $\mu$ A is being drawn from the host via VBUS.

For most applications, the integrated 3.3-V LDO is being used. In this case, the following actions should be taken:

- Disable the PLL by clearing UPLLEN (UPLLEN = 0)
- Limit all current sourced from VBUS that causes the total current sourced from VBUS equal to 500  $\mu$ A minus the suspend current,  $I_{SUSPEND}$  (see the device-specific data sheet).

Disabling the PLL eliminates the largest on-chip draw of power from VBUS. During suspend, the USBCLK is automatically sourced by the VLO (VLOCLK), allowing the USB module to detect resume when it occurs. It is a good idea to also then ensure that the RESRIE bit is also set, so that an interrupt is generated when the host resumes the device. If desired, the high frequency crystal can also be disabled to save additional system power, however it does not contribute to the power from VBUS since it draws power from the DVCC supply.

##### 40.2.7.2 Entering Resume Mode

When the USB device is in a suspended condition, any non-idle signaling, including reset signaling, on the host side is detected by the suspend and resume logic and device operation is resumed. RESRIFFG is set, causing an USB interrupt. The interrupt service routine can be used to resume USB operation.

### 40.3 USB Transfers

The USB module supports control, bulk, and interrupt data transfer types. In accordance with the USB specification, endpoint 0 is reserved for the control endpoint and is bidirectional. In addition to the control endpoint, the USB module is capable of supporting up to 7 input endpoints and 7 output endpoints. These additional endpoints can be configured either as bulk or interrupt endpoints. The software handles all control, bulk, and interrupt endpoint transactions.

#### 40.3.1 Control Transfers

Control transfers are used for configuration, command, and status communication between the host and the USB device. Control transfers to the USB device use input endpoint 0 and output endpoint 0. The three types of control transfers are control write, control write with no data stage, and control read. Note that the control endpoint must be initialized before connecting the USB device to the USB.

#### 40.3.1.1 Control Write Transfer

The host uses a control write transfer to write data to the USB device. A control write transfer consists of a setup stage transaction, at least one output data stage transaction, and an input status stage transaction.

The stage transactions for a control write transfer are:

- Setup stage transaction:
  1. Input endpoint 0 and output endpoint 0 are initialized by programming the appropriate USB endpoint configuration blocks. This entails enabling the endpoint interrupt (USBIIE = 1) and enabling the endpoint (UBME = 1). The NAK bit for both input endpoint 0 and output endpoint 0 must be cleared.
  2. The host sends a setup token packet followed by the setup data packet addressed to output endpoint 0. If the data is received without an error, then the UBM writes the data to the setup data packet buffer, sets the setup stage transaction bit (SETUPIFG = 1) in the USB Interrupt Flag register (USBIFG), returns an ACK handshake to the host, and asserts the setup stage transaction interrupt. Note that as long as SETUPIFG = 1, the UBM returns a NAK handshake for any data stage or status stage transactions regardless of the endpoint 0 NAK or STALL bit values.
  3. The software services the interrupt, reads the setup data packet from the buffer, and then decodes the command. If the command is not supported or invalid, the software should set the STALL bit in the output endpoint 0 configuration register (USBOEPCNFG\_0) and the input endpoint 0 configuration register (USBIEPCNFG\_0). This causes the device to return a STALL handshake for any data or status stage transaction. For control write transfers, the packet ID used by the host for the first data packet output is a DATA1 packet ID and the TOGGLE bit must match.

---

**NOTE:** When using USBIV, SETUPIFG is cleared upon reading USBIV. In addition, the NAK on input endpoint 0 and output endpoint 0 are also cleared. In this case, the host may send or receive the next setup packet even if MSP430 did not perform the first setup packet. To prevent this, first read the SETUPIFG directly, perform the required setup, and then use the USBIV for further processing.

---

---

**NOTE:** The priority of input endpoint 0 is higher than the setup flag inside USBIV (SETUPIFG). Therefore, if both the USBIEPIFG.EP0 and SETUPIFG are pending, reading USBIV gives the higher priority interrupt (EP0) as opposed to SETUPIFG. Therefore, read SETUPIFG directly, process the pending setup packet, then proceed to read the USBIV.

---

- Data stage transaction:
  1. The host sends an OUT token packet followed by a data packet addressed to output endpoint 0. If the data is received without an error, the UBM writes the data to the output endpoint buffer (USBOEP0BUF), updates the data count value, toggles the TOGGLE bit, sets the NAK bit, returns an ACK handshake to the host, and asserts the output endpoint interrupt 0 (OEPIFG0).
  2. The software services the interrupt and reads the data packet from the output endpoint buffer. To read the data packet, the software first needs to obtain the data count value inside the USBOEPBCNT\_0 register. After reading the data packet, the software should clear the NAK bit to allow the reception of the next data packet from the host.
  3. If the NAK bit is set when the data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host.
- Status stage transaction:
  1. For input endpoint 0, the software updates the data count value to zero, sets the TOGGLE bit, then clears the NAK bit to enable the data packet to be sent to the host. Note that for a status stage transaction, a null data packet with a DATA1 packet ID is sent to the host.
  2. The host sends an IN token packet addressed to input endpoint 0. After receiving the IN token, the UBM transmits a null data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM then toggles the TOGGLE bit and sets the NAK bit.

3. If the NAK bit is set when the IN token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

#### 40.3.1.2 Control Write Transfer with No Data Stage Transfer

The host uses a control write transfer to write data to the USB device. A control write with no data stage transfer consists of a setup stage transaction and an input status stage transaction. For this type of transfer, the data to be written to the USB device is contained in the two byte value field of the setup stage transaction data packet.

The stage transactions for a control write transfer with no data stage transfer are:

- Setup stage transaction:
  1. Input endpoint 0 and output endpoint 0 are initialized by programming the appropriate USB endpoint configuration blocks. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt (USBIIE = 1), initializing the TOGGLE bit, enabling the endpoint (UBME = 1). The NAK bit for both input endpoint 0 and output endpoint 0 must be cleared.
  2. The host sends a setup token packet followed by the setup data packet addressed to output endpoint 0. If the data is received without an error then the UBM writes the data to the setup data packet buffer, sets the setup stage transaction (SETUP) bit in the USB status register, returns an ACK handshake to the host, and asserts the setup stage transaction interrupt. Note that as long as the setup transaction (SETUP) bit is set, the UBM returns a NAK handshake for any data stage or status stage transaction regardless of the endpoint 0 NAK or STALL bit values.
  3. The software services the interrupt and reads the setup data packet from the buffer then decodes the command. If the command is not supported or invalid, the software should set the STALL bits in the output endpoint 0 and the input endpoint 0 configuration registers before clearing the setup stage transaction (SETUP) bit. This causes the device to return a STALL handshake for data or status stage transactions. After reading the data packet and decoding the command, the software should clear the interrupt, which automatically clears the setup stage transaction status bit.

---

**NOTE:** When using USBIV, the SETUPIFG is cleared upon reading USBIV. In addition, the NAK on input endpoint 0 and output endpoint 0 is also cleared. In this case, the host may send or receive the next setup packet even if MSP430 did not perform the first setup packet. To prevent this, first read the SETUPIFG directly, perform the required setup, and then use the USBIV for further processing.

---



---

**NOTE:** The priority of input endpoint 0 is higher than setup flag inside USBIV. Therefore, if both the USBIEPIFG.EP0 and SETUPIFG are pending, reading the USBIV gives the higher priority interrupt (EP0) as opposed to the SETUPIFG. Therefore, read SETUPIFG directly, process the pending setup packet, then proceed to read the USBIV.

---

- Status stage transaction:
  1. For input endpoint 0, the software updates the data count value to zero, sets the TOGGLE bit, then clears the NAK bit to enable the data packet to be sent to the host. Note that for a status stage transaction a null data packet with a DATA1 packet ID is sent to the host.
  2. The host sends an IN token packet addressed to input endpoint 0. After receiving the IN token, the UBM transmits a null data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM then toggles the TOGGLE bit, sets the NAK bit, and asserts the endpoint interrupt.
  3. If the NAK bit is set when the IN token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

#### 40.3.1.3 Control Read Transfer

The host uses a control read transfer to read data from the USB device. A control read transfer consists of a setup stage transaction, at least one input data stage transaction and an output status stage transaction.

The stage transactions for a control read transfer are:

- Setup stage transaction:
  1. Input endpoint 0 and output endpoint 0 are initialized by programming the appropriate USB endpoint configuration blocks. This entails enabling the endpoint interrupt (USBIIE = 1) and enabling the endpoint (UBME = 1). The NAK bit for both input endpoint 0 and output endpoint 0 must be cleared.
  2. The host sends a setup token packet followed by the setup data packet addressed to output endpoint 0. If the data is received without an error, then the UBM writes the data to the setup buffer, sets the setup stage transaction (SETUP) bit in the USB status register, returns an ACK handshake to the host, and asserts the setup stage transaction interrupt. Note that as long as the setup transaction (SETUP) bit is set, the UBM returns a NAK handshake for any data stage or status stage transactions regardless of the endpoint 0 NAK or STALL bit values.
  3. The software services the interrupt and reads the setup data packet from the buffer then decodes the command. If the command is not supported or invalid, the software should set the STALL bits in the output endpoint 0 and the input endpoint 0 configuration registers before clearing the setup stage transaction (SETUP) bit. This causes the device to return a STALL handshake for a data stage or status stage transactions. After reading the data packet and decoding the command, the software should clear the interrupt, which automatically clears the setup stage transaction status bit. The software should also set the TOGGLE bit in the input endpoint 0 configuration register. For control read transfers, the packet ID used by the host for the first input data packet is a DATA1 packet ID.

---

**NOTE:** When using USBIV, the SETUPIFG is cleared upon reading USBIV. In addition, it also clears NAK on input endpoint 0 and output endpoint 0. In this case, the host may send or receive the next setup packet even if MSP430 did not perform the first setup packet. To prevent this, first read the SETUPIFG directly, perform the required setup, and then use the USBIV for further processing.

---

---

**NOTE:** The priority of input endpoint 0 is higher than the setup flag inside USBIV. Therefore, if both the USBIPIFG.EP0 and SETUPIFG are pending, reading the USBIV gives the higher priority interrupt (EP0) as opposed to the SETUPIFG. Therefore, read SETUPIFG directly, process the pending setup packet, then proceed to read the USBIV.

---

- Data stage transaction:
  1. The data packet to be sent to the host is written to the input endpoint 0 buffer by the software. The software also updates the data count value then clears the input endpoint 0 NAK bit to enable the data packet to be sent to the host.
  2. The host sends an IN token packet addressed to input endpoint 0. After receiving the IN token, the UBM transmits the data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM sets the NAK bit and asserts the endpoint interrupt.
  3. The software services the interrupt and prepares to send the next data packet to the host.
  4. If the NAK bit is set when the IN token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.
  5. The software continues to send data packets until all data has been sent to the host.
- Status stage transaction:
  1. For output endpoint 0, the software sets the TOGGLE bit, then clears the NAK bit to enable the data packet to be sent to the host. Note that for a status stage transaction a null data packet with a DATA1 packet ID is sent to the host.

2. The host sends an OUT token packet addressed to output endpoint 0. If the data packet is received without an error then the UBM updates the data count value, toggles the TOGGLE bit, sets the NAK bit, returns an ACK handshake to the host, and asserts the endpoint interrupt.
3. The software services the interrupt. If the status stage transaction completed successfully, then the software should clear the interrupt and clear the NAK bit.
4. If the NAK bit is set when the input data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the in data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host.

#### **40.3.2 Interrupt Transfers**

The USB module supports interrupt data transfers both to and from the host. Devices that need to send or receive a small amount of data with a specified service period are best served by the interrupt transfer type. Input endpoints 1 through 7 and output endpoints 1 through 7 can be configured as interrupt endpoints.

##### **40.3.2.1 Interrupt OUT Transfer**

The steps for an interrupt OUT transfer are:

1. The software initializes one of the output endpoints as an output interrupt endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and clearing the NAK bit.
2. The host sends an OUT token packet followed by a data packet addressed to the output endpoint. If the data is received without an error then the UBM writes the data to the endpoint buffer, updates the data count value, toggles the toggle bit, sets the NAK bit, returns an ACK handshake to the host, and asserts the endpoint interrupt.
3. The software services the interrupt and reads the data packet from the buffer. To read the data packet, the software first needs to obtain the data count value. After reading the data packet, the software should clear the interrupt and clear the NAK bit to allow the reception of the next data packet from the host.
4. If the NAK bit is set when the data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host device.

In double buffer mode, the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM writes the data packet to the X buffer. If the toggle bit is a 1, the UBM writes the data packet to the Y buffer. When a data packet is received, the software could determine which buffer contains the data packet by reading the toggle bit. However, when using double buffer mode, the possibility exists for data packets to be received and written to both the X and Y buffer before the software responds to the endpoint interrupt. In this case, simply using the toggle bit to determine which buffer contains the data packet would not work. Hence, in double buffer mode, the software should read the X buffer NAK bit, the Y buffer NAK bit, and the toggle bits to determine the status of the buffers.

##### **40.3.2.2 Interrupt IN Transfer**

The steps for an interrupt IN transfer are:

1. The software initializes one of the input endpoints as an input interrupt endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and setting the NAK bit.
2. The data packet to be sent to the host is written to the buffer by the software. The software also updates the data count value then clears the NAK bit to enable the data packet to be sent to the host.
3. The host sends an IN token packet addressed to the input endpoint. After receiving the IN token, the UBM transmits the data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM then toggles the toggle bit, sets the NAK bit, and

asserts the endpoint interrupt.

4. The software services the interrupt and prepares to send the next data packet to the host.
5. If the NAK bit is set when the in token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

In double buffer mode, the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM reads the data packet from the X buffer. If the toggle bit is a 1, the UBM reads the data packet from the Y buffer.

#### 40.3.3 Bulk Transfers

The USB module supports bulk data transfers both to and from the host. Devices that need to send or receive a large amount of data without a suitable bandwidth are best served by the bulk transfer type. In endpoints 1 through 7 and out endpoints 1 through 7 can all be configured as bulk endpoints.

##### 40.3.3.1 Bulk OUT Transfer

The steps for a bulk OUT transfer are:

1. The software initializes one of the output endpoints as an output bulk endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and clearing the NAK bit.
2. The host sends an out token packet followed by a data packet addressed to the output endpoint. If the data is received without an error then the UBM writes the data to the endpoint buffer, updates the data count value, toggles the toggle bit, sets the NAK bit, returns an ACK handshake to the host, and asserts the endpoint interrupt.
3. The software services the interrupt and reads the data packet from the buffer. To read the data packet, the software first needs to obtain the data count value. After reading the data packet, the software should clear the interrupt and clear the NAK bit to allow the reception of the next data packet from the host.
4. If the NAK bit is set when the data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host.

In double buffer mode, the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM writes the data packet to the X buffer. If the toggle bit is a 1, the UBM writes the data packet to the Y buffer. When a data packet is received, the software could determine which buffer contains the data packet by reading the toggle bit. However, when using double buffer mode, the possibility exists for data packets to be received and written to both the X and Y buffer before the software responds to the endpoint interrupt. In this case, simply using the toggle bit to determine which buffer contains the data packet would not work. Hence, in double buffer mode, the software should read the X buffer NAK bit, the Y buffer NAK bit, and the toggle bits to determine the status of the buffers.

##### 40.3.3.2 Bulk IN Transfer

The steps for a bulk IN transfer are:

1. The software initializes one of the input endpoints as an input bulk endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and setting the NAK bit.
2. The data packet to be sent to the host is written to the buffer by the software. The software also updates the data count value then clears the NAK bit to enable the data packet to be sent to the host.
3. The host sends an IN token packet addressed to the input endpoint. After receiving the IN token, the UBM transmits the data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM then toggles the toggle bit, sets the NAK bit, and

asserts the endpoint interrupt.

4. The software services the interrupt and prepares to send the next data packet to the host.
5. If the NAK bit is set when the in token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the In token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

In double buffer mode , the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM reads the data packet from the X buffer. If the toggle bit is a 1, the UBM reads the data packet from the Y buffer.

## 40.4 USB Registers

The USB register space is subdivided into configuration registers, control registers, and USB buffer memory.

The configuration and control registers are physical registers located in peripheral memory, while the buffer memory is implemented in RAM. See the device-specific data sheet for base addresses of these register groupings.

The USB control registers can be written only while the USB module is enabled.

When the USB module is disabled, it no longer uses the RAM buffer memory. This memory then behaves as a 2KB RAM block and can be used by the CPU or DMA without any limitation.

### 40.4.1 USB Configuration Registers

The configuration registers control the hardware functions needed to make a USB connection, including the PHY, PLL, and LDOs.

Access to the configuration registers is allowed or disallowed using the USBKEYPID register. Writing the proper value (9628h) unlocks the configuration registers and enables access. Writing any other value disables access while leaving the values of the registers intact. Locking should be done intentionally after the configuration is finished. Read access is available without the need to write to the USBKEYPID register.

The configuration registers are listed in [Table 40-5](#). All addresses are expressed as offsets; the base address can be found in the device-specific data sheet.

All registers are byte and word accessible.

**Table 40-5. USB Configuration Registers**

Offset	Acronym	Register Name	Type	Reset	Section
00h	USBKEYPID	USB controller key and ID register	Read/Write	0000h	<a href="#">Section 40.4.1.1</a>
02h	USBCNF	USB controller configuration register	Read/Write	0000h	<a href="#">Section 40.4.1.2</a>
04h	USBPHYCTL	USB-PHY control register	Read/Write	0000h	<a href="#">Section 40.4.1.3</a>
08h	USBPWRCTL	USB-PWR control register	Read/Write	1850h	<a href="#">Section 40.4.1.4</a>
10h	USBPLLCTL	USB-PLL control register	Read/Write	0000h	<a href="#">Section 40.4.1.5</a>
12h	USBPLLDIVB	USB-PLL divider buffer register	Read/Write	0000h	<a href="#">Section 40.4.1.6</a>
14h	USBPLLIR	USB-PLL interrupt register	Read/Write	0000h	<a href="#">Section 40.4.1.7</a>

**40.4.1.1 USBKEYPID Register**

USB Key Register

**Figure 40-8. USBKEYPID Register**

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

**Table 40-6. USBKEYPID Register Description**

Bit	Field	Type	Reset	Description
15-0	USBKEY	RW	00h	Key register. Must be written with a value of 9628h to be recognized as a valid key. This "unlocks" the configuration registers. If written with any other value, the registers become "locked". Reads back as A528h if the registers are unlocked.

**40.4.1.2 USBCNF Register**

USB Module Configuration Register

This register can be modified only when USBKEYPID is unlocked.

**Figure 40-9. USBCNF 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							
r0	r0	r0	rw-0	rw-0	r	rw-0	rw-0
FNTEN BLKRDY PUR_IN PUR_EN USB_EN							

Can be modified only when USBKEYPID is unlocked

**Table 40-7. USBCNF Register Description**

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always reads as 0.
4	FNTEN	RW	0h	Frame number receive trigger enable for DMA transfers 0b = Frame number receive trigger is blocked. 1b = Frame number receive trigger is gated through to DMA.
3	BLKRDY	RW	0h	Block transfer ready signaling for DMA transfers 0b = DMA triggering is disabled. 1b = DMA is triggered whenever the USB bus interface can accept new write transfers.
2	PUR_IN	R	0h	PUR input value. This bit reflects the input value present on PUR. This bit may be used as an indication to start a USB based boot loading program (USB-BSL). The PUR input logic is powered by VUSB. PUR_IN returns zero when VUSB is zero
1	PUR_EN	RW	0h	PUR pin enable 0b = PUR pin is in high-impedance state 1b = PUR pin is driven high
0	USB_EN	RW	0h	USB module enable 0b = USB module is disabled 1b = USB module is enabled

#### 40.4.1.3 USBPHYCTL Register

USB-PHY Control Register

This register can be modified only when USBKEYPID is unlocked.

**Figure 40-10. USBPHYCTL Register**

15	14	13	12	11	10	9	8
Reserved							Reserved
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
PUSEL	Reserved	PUOPE	Reserved	PUIN1	PUIN0	PUOUT1	PUOUT0
rw-0	r	rw-0	rw-0	r	r	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked.

**Table 40-8. USBPHYCTL Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9	Reserved	RW	0h	Reserved. Always write as 0.
8	PUIPE	RW	0h	PU input enable. This bit is valid only when PUSEL = 0. 0b = PU.0 and PU.1 inputs are disabled 1b = PU.0 and PU.1 inputs are enabled.
7	PUSEL	RW	0h	USB port function select. This bit selects the function of the PU.0/DP and PU.1/DM pins. 0b = PU.0 and PU.1 function selected (general purpose I/O) 1b = DP and DM function selected (USB terminals)
6	Reserved	R	0h	Reserved. Always reads as 0.
5	PUOPE	RW	0h	PU output enable. This bit is valid only when PUSEL = 0. 0b = PU.0 and PU.1 outputs are disabled 1b = PU.0 and PU.1 outputs are enabled.
4	Reserved	RW	0h	Reserved. Always write as 0.
3	PUIN1	R	0h	PU.1 input data, This bit reflects the logic value on the PU.1 terminal when PUIPE = 1.
2	PUIN0	R	0h	PU.0 input data, This bit reflects the logic value on the PU.0 terminal when PUIPE = 1.
1	PUOUT1	RW	0h	PU.1 output data. This bits defines the value of the PU.1 pin when PUOPE = 1.
0	PUOUT0	RW	0h	PU.0 output data. This bits defines the value of the PU.0 pin when PUOPE = 1.

**40.4.1.4 USBPWRCTL Register****USB-Power Control Register**

This register can be modified only when USBKEYPID is unlocked.

**Figure 40-11. USBPWRCTL Register**

15	14	13	12	11	10	9	8
Reserved			SLDOEN	VUSBEN	VBOFFIE	VBONIE	VUOVLIE
r0	r0	r0	rw-1	rw-1	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved	SLDOAON	OVLAOFF	USBDETEN	USBBGVBV	VBOFFIFG	VBONIFG	VUOVLIFG
r0	rw-1	rw-0	rw-1	r	rw-0	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked

**Table 40-9. USBPWRCTL Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12	SLDOEN	RW	1h	1.8-V (secondary) LDO enable. When set, the LDO is enabled. Can be cleared only if SLDOAON = 0. 0b = 1.8-V LDO is disabled 1b = 1.8-V LDO is enabled
11	VUSBEN	RW	1h	3.3-V LDO enable. When set, the LDO is enabled. 0b = 3.3-V LDO is disabled 1b = 3.3-V LDO is enabled
10	VBOFFIE	RW	0h	VBUS "going OFF" interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
9	VBONIE	RW	0h	VBUS "coming ON" interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
8	VUOVLIE	RW	0h	VUSB overload indication interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
7	Reserved	R	0h	Reserved. Always reads as 0.
6	SLDOAON	RW	1h	1.8-V LDO auto-on enable 0b = LDO must be turned on manually using SLDOEN 1b = A "VBUS coming on" enables the secondary LDO, but SLDOEN is not set automatically.
5	OVLAOFF	RW	0h	LDO overload auto-off enable 0b = During an overload on the 3.3-V LDO, the LDO automatically enters current-limiting mode and stays there until the condition stops. 1b = An overload indication clears the VUSBEN bit.
4	USBDETEN	RW	1h	Enable bit for VBUS on and off events. 0b = USB module does not detect USB-PWR VBUS on and off events 1b = USB module does detect USB-PWR VBUS on and off events
3	USBBGVBV	R	0h	VBUS valid 0b = VBUS is not valid yet 1b = VBUS is valid and within bounds

**Table 40-9. USBPWRCTL Register Description (continued)**

Bit	Field	Type	Reset	Description
2	VBOFFIFG	RW	0h	<p>VBUS "going OFF" interrupt flag. This bit indicates that VBUS fell below the launch voltage. It is automatically cleared when the corresponding vector of the USB interrupt vector register is read, or if a value is written to the interrupt vector register.</p> <p>0b = VBUS did not fall below the launch voltage.            1b = VBUS fell below the launch voltage.</p>
1	VBONIFG	RW	0h	<p>VBUS "coming ON" interrupt flag. This bit indicates that VBUS rose above the launch voltage. This bit is automatically cleared when the corresponding vector of the USB interrupt vector register is read, or if a value is written to the interrupt vector register.</p> <p>0b = VUSB did not rise above the launch voltage.            1b = VUSB rose above the launch voltage.</p>
0	VUOVLIFG	RW	0h	<p>VUSB overload interrupt flag. This bit indicates that the 3.3-V LDO entered an overload condition.</p> <p>0b = No overload condition detected.            1b = Overload condition detected.</p>

**40.4.1.5 USBPLLCTL Register**

USB-PLL Control Register

This register can be modified only when USBKEYPID is unlocked.

**Figure 40-12. USBPLLCTL Register**

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

Can be modified only when USBKEYPID is unlocked

**Table 40-10. USBPLLCTL Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12	Reserved	RW	0h	Reserved. Always write as 0.
11-10	Reserved	R	0h	Reserved. Always reads as 0.
9	UPFDEN	RW	0h	Phase frequency discriminator (PFD) enable 0b = PFD is disabled 1b = PFD is enabled
8	UPLLEN	RW	0h	PLL enable 0b = PLL is disabled 1b = PLL is enabled
7-6	UCLKSEL	RW	0h	USB module clock select. Must always be written with 00. 00b = PLLCLK (default) 01b = Reserved 10b = Reserved 11b = Reserved
5-0	Reserved	R	0h	Reserved. Always reads as 0.

#### 40.4.1.6 USBPLL DIVB Register

USB-PLL Clock Divider Buffer Register

This register can be modified only when USBKEYPID is unlocked.

**Figure 40-13. USBPLL DIVB Register**

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

Can be modified only when USBKEYPID is unlocked

**Table 40-11. USBPLL DIVB Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	UPQB	RW	0h	PLL pre-scale divider buffer register. These bits select the pre-scale division value. The value of this register is transferred to UPQB as soon it is written. 000b = f(UPD) = f(REF) 001b = f(UPD) = f(REF) / 2 010b = f(UPD) = f(REF) / 3 011b = f(UPD) = f(REF) / 4 100b = f(UPD) = f(REF) / 6 101b = f(UPD) = f(REF) / 8 110b = f(UPD) = f(REF) / 13 111b = f(UPD) = f(REF) / 16
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5-0	UPMB	RW	0h	USB PLL feedback divider buffer register. These bits select the value of the feedback divider. The value of this register is transferred to UPMB automatically when UPQB is written. 000000b = Feedback division rate: 1 000001b = Feedback division rate: 2 ... 111111b = Feedback division rate: 64

**40.4.1.7 USBPLLIR Register**

## USB-PLL Interrupt Register

This register can be modified only when USBKEYPID is unlocked.

**Figure 40-14. USBPLLIR Register**

15	14	13	12	11	10	9	8
		Reserved			USBOORIE	USBLOSIE	USBOOLIE
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
		Reserved			USBOORIFG	USBLOSIFG	USBOOLIFG
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked.

**Table 40-12. USBPLLIR Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10	USBOORIE	RW	0h	PLL out-of-range interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
9	USBLOSIE	RW	0h	PLL loss-of-signal interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
8	USBOOLIE	RW	0h	PLL out-of-lock interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
7-3	Reserved	R	0h	Reserved. Always reads as 0.
2	USBOORIFG	RW	0h	PLL out-of-range interrupt flag 0b = No interrupt pending 1b = Interrupt pending
1	USBLOSIFG	RW	0h	PLL loss-of-signal interrupt flag 0b = No interrupt pending 1b = Interrupt pending
0	USBOOLIFG	RW	0h	PLL out-of-lock interrupt flag 0b = No interrupt pending 1b = Interrupt pending

#### 40.4.2 USB Control Registers

The control registers affect core USB operations that are fundamental for any USB connection. This includes control endpoint 0, interrupts, bus address and frame, and timestamps. Control of endpoints other than zero are found in the operation registers. Unlike the operation registers, the control registers are actual physical registers, whereas the operation registers exist in RAM, which can be re-allocated to general-purpose use.

The control registers are listed in [Table 40-13](#). All addresses are expressed as offsets; the base address can be found in the device-specific data sheet.

All registers are byte and word accessible.

**Table 40-13. USB Control Registers**

Offset	Acronym	Register Name	Type	Reset	Section
00h	USBIEPCNF_0	Input endpoint_0: Configuration	Read/Write	00h	<a href="#">Section 40.4.2.1</a>
01h	USBIEPBCNT_0	Input endpoint_0: Byte Count	Read/Write	80h	<a href="#">Section 40.4.2.2</a>
02h	USBOEPCNFG_0	Output endpoint_0: Configuration	Read/Write	00h	<a href="#">Section 40.4.2.3</a>
03h	USBOEPBCNT_0	Output endpoint_0: Byte count	Read/Write	00h	<a href="#">Section 40.4.2.4</a>
0Eh	USBIEPIE	Input endpoint interrupt enables	Read/Write	00h	<a href="#">Section 40.4.2.5</a>
0Fh	USBOEPIE	Output endpoint interrupt enables	Read/Write	00h	<a href="#">Section 40.4.2.6</a>
10h	USBIEPIFG	Input endpoint interrupt flags	Read/Write	00h	<a href="#">Section 40.4.2.7</a>
11h	USBOEPIFG	Output endpoint interrupt flags	Read/Write	00h	<a href="#">Section 40.4.2.8</a>
12h	USBVECINT or USBIV	Vector interrupt register	Read/Write	0000h	<a href="#">Section 40.4.2.9</a>
16h	USBMAINT	Timestamp maintenance register	Read/Write	0000h	<a href="#">Section 40.4.2.10</a>
18h	USBTSREG	Timestamp register	Read/Write	0000h	<a href="#">Section 40.4.2.11</a>
1Ah	USBFN	USB frame number	Read only	0000h	<a href="#">Section 40.4.2.12</a>
1Ch	USBCTL	USB control register	Read/Write	00h	<a href="#">Section 40.4.2.13</a>
1Dh	USBIE	USB interrupt enable register	Read/Write	00h	<a href="#">Section 40.4.2.14</a>
1Eh	USBIFG	USB interrupt flag register	Read/Write	00h	<a href="#">Section 40.4.2.15</a>
1Fh	USBFUNADR	Function address register	Read/Write	00h	<a href="#">Section 40.4.2.16</a>

**40.4.2.1 USBIIEPCNF\_0 Register**

USB Input Endpoint-0 Configuration Register

**Figure 40-15. USBIIEPCNF\_0 Register**

7	6	5	4	3	2	1	0
UBME	Reserved	TOGGLE	Reserved	STALL	USBIIE	Reserved	
rw-0	r0	r-0	r0	rw-0	rw-0	r0	r0

Can be modified only when USBEN = 1

**Table 40-14. USBIIEPCNF\_0 Register Description**

Bit	Field	Type	Reset	Description
7	UBME	RW	0h	UBM in endpoint-0 enable 0b = UBM cannot use this endpoint 1b = UBM can use this endpoint
6	Reserved	R	0h	Reserved. Always reads as 0.
5	TOGGLE	R	0h	Toggle bit. Reads as 0, because the configuration endpoint does not need to toggle.
4	Reserved	R	0h	Reserved. Always reads as 0.
3	STALL	RW	0h	USB stall condition. When set, hardware automatically returns a stall handshake to the USB host for any transaction transmitted from endpoint-0. The stall bit is cleared automatically by the next setup transaction. 0b = Indicates no stall 1b = Indicates stall
2	USBIIE	RW	0h	USB transaction interrupt indication enable. Software may set this bit to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt flag must be set (IEPIE). 0b = Corresponding interrupt flag is not set 1b = Corresponding interrupt flag is set
1-0	Reserved	R	0h	Reserved. Always reads as 0.

#### 40.4.2.2 USBIEPBCNT\_0 Register

USB Input Endpoint-0 Byte Count Register

**Figure 40-16. USBIEPBCNT\_0 Register**

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

Can be modified only when USBEN = 1

**Table 40-15. USBIEPBCNT\_0 Register Description**

Bit	Field	Type	Reset	Description
7	NAK	RW	0h	No acknowledge status bit. This bit is set by the UBM at the end of a successful USB IN transaction from endpoint-0, to indicate that the EP-0 IN buffer is empty. When this bit is set, all subsequent transactions from endpoint-0 result in a NAK handshake response to the USB host. To re-enable this endpoint to transmit another data packet to the host, this bit must be cleared by software. 0b = Buffer contains a valid data packet for host device 1b = Buffer is empty (Host-In request receives a NAK)
6-4	Reserved	R	0h	Reserved. Always reads as 0.
3-0	CNT	RW	0h	Byte count. The In_EP-0 buffer data count value should be set by software when a new data packet is written to the buffer. This four-bit value contains the number of bytes in the data packet. 0000b to 1000b are valid numbers for 0 to 8 bytes to be sent. 1001b to 1111b are reserved values (if used, defaults to 8).

**40.4.2.3 USBOEPCNFG\_0 Register**

USB Output Endpoint-0 Configuration Register

**Figure 40-17. USBOEPCNFG\_0 Register**

7	6	5	4	3	2	1	0
UBME	Reserved	TOGGLE	Reserved	STALL	USBIIE	Reserved	
rw-0	r0	r-0	r0	rw-0	rw-0	r0	r0

Can be modified only when USBEN = 1

**Table 40-16. USBOEPCNFG\_0 Register Description**

Bit	Field	Type	Reset	Description
7	UBME	RW	0h	UBM out Endpoint-0 enable 0b = UBM cannot use this endpoint 1b = UBM can use this endpoint
6	Reserved	R	0h	Reserved. Always reads as 0.
5	TOGGLE	RW	0h	Toggle bit. Reads as 0, because the configuration endpoint does not need to toggle.
4	Reserved	R	0h	Reserved. Always reads as 0.
3	STALL	RW	0h	USB stall condition. When set, hardware automatically returns a stall handshake to the USB host for any transaction transmitted into endpoint-0. The stall bit is cleared automatically by the next setup transaction. 0b = Indicates no stall 1b = Indicates stall
2	USBIIE	RW	0h	USB transaction interrupt indication enable. Software may set this bit to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt flag must be set (OEPIE). 0b = Corresponding interrupt flag will not be set 1b = Corresponding interrupt flag will be set
1-0	Reserved	R	0h	Reserved. Always reads as 0.

#### 40.4.2.4 USBOEPBCNT\_0 Register

USB Output Endpoint-0 Byte Count Register

**Figure 40-18. USBOEPBCNT\_0 Register**

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

Can be modified only when USBEN = 1

**Table 40-17. USBOEPBCNT\_0 Register Description**

Bit	Field	Type	Reset	Description
7	NAK	RW	0h	No acknowledge status bit. This bit is set by the UBM at the end of a successful USB out transaction into endpoint-0, to indicate that the EP-0 buffer contains a valid data packet and that the buffer data count value is valid. When this bit is set, all subsequent transactions to endpoint-0 result in a NAK handshake response to the USB host. To re-enable this endpoint to receive another data packet from the host, this bit must be cleared by software. 0b = No valid data in the buffer. The buffer is ready to receive a host OUT transaction 1b = The buffer contains a valid packet from the host that has not been picked up. (Any subsequent Host-Out requests receive a NAK.)
6-4	Reserved	R	0h	Reserved. Always reads as 0.
3-0		RW	0h	Byte count. This data count value is set by the UBM when a new data packet is received by the buffer for the out endpoint-0. The four-bit value contains the number of bytes received in the data buffer. 0000b to 1000b are valid numbers for 0 to 8 received bytes 1001b to 1111b are reserved values

**40.4.2.5 USBIEPIE Register**

USB Input Endpoint Interrupt Enable Register

**Figure 40-19. USBIEPIE Register**

7	6	5	4	3	2	1	0
IEPIE7	IEPIE6	IEPIE5	IEPIE4	IEPIE3	IEPIE2	IEPIE1	IEPIE0
rw-0							

Can be modified only when USBEN = 1

**Table 40-18. USBIEPIE Register Description**

Bit	Field	Type	Reset	Description
7	IEPIE7	RW	0h	<p>Input endpoint interrupt enable 7. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt 1b = Event does generate an interrupt</p>
6	IEPIE6	RW	0h	<p>Input endpoint interrupt enable 6. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt 1b = Event does generate an interrupt</p>
5	IEPIE5	RW	0h	<p>Input endpoint interrupt enable 5. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt 1b = Event does generate an interrupt</p>
4	IEPIE4	RW	0h	<p>Input endpoint interrupt enable 4. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt 1b = Event does generate an interrupt</p>
3	IEPIE3	RW	0h	<p>Input endpoint interrupt enable 3. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt 1b = Event does generate an interrupt</p>
2	IEPIE2	RW	0h	<p>Input endpoint interrupt enable 2. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt 1b = Event does generate an interrupt</p>
1	IEPIE1	RW	0h	<p>Input endpoint interrupt enable 1. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt 1b = Event does generate an interrupt</p>

**Table 40-18. USBIEPIE Register Description (continued)**

Bit	Field	Type	Reset	Description
0	IEPIE0	RW	0h	<p>Input endpoint interrupt enable 0. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt            1b = Event does generate an interrupt</p>

**40.4.2.6 USBOEPIE Register**

USB Output Endpoint Interrupt Enable Register

**Figure 40-20. USBOEPIE Register**

7	6	5	4	3	2	1	0
OEPIE7	OEPIE6	OEPIE5	OEPIE4	OEPIE3	OEPIE2	OEPIE1	OEPIE0
rw-0							

Can be modified only when USBEN = 1

**Table 40-19. USBOEPIE Register Description**

Bit	Field	Type	Reset	Description
7	OEPIE7	RW	0h	Output endpoint interrupt enable 7. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors. 0b = Event does not generate an interrupt 1b = Event does generate an interrupt
6	OEPIE6	RW	0h	Output endpoint interrupt enable 6. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors. 0b = Event does not generate an interrupt 1b = Event does generate an interrupt
5	OEPIE5	RW	0h	Output endpoint interrupt enable 5. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors. 0b = Event does not generate an interrupt 1b = Event does generate an interrupt
4	OEPIE4	RW	0h	Output endpoint interrupt enable 4. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors. 0b = Event does not generate an interrupt 1b = Event does generate an interrupt
3	OEPIE3	RW	0h	Output endpoint interrupt enable 3. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors. 0b = Event does not generate an interrupt 1b = Event does generate an interrupt
2	OEPIE2	RW	0h	Output endpoint interrupt enable 2. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors. 0b = Event does not generate an interrupt 1b = Event does generate an interrupt
1	OEPIE1	RW	0h	Output endpoint interrupt enable 1. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors. 0b = Event does not generate an interrupt 1b = Event does generate an interrupt

**Table 40-19. USBOEPIE Register Description (continued)**

Bit	Field	Type	Reset	Description
0	OEPIE0	RW	0h	<p>Output endpoint interrupt enable 0. This bit enables or disables whether an event can trigger an interrupt. It does not influence whether the event is flagged; the flag is enabled or disabled with the interrupt indication enable bit in the Endpoint descriptors.</p> <p>0b = Event does not generate an interrupt            1b = Event does generate an interrupt</p>

**40.4.2.7 USBIEPIFG Register**

USB Input Endpoint Interrupt Flag Register

**Figure 40-21. USBIEPIFG Register**

7	6	5	4	3	2	1	0
IEPIFG7	IEPIFG6	IEPIFG5	IEPIFG4	IEPIFG3	IEPIFG2	IEPIFG1	IEPIFG0
rw-0							

Can be modified only when USBEN = 1

**Table 40-20. USBIEPIFG Register Description**

Bit	Field	Type	Reset	Description
7	IEPIFG7	RW	0h	Input endpoint interrupt flag 7. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.
6	IEPIFG6	RW	0h	Input endpoint interrupt flag 6. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.
5	IEPIFG5	RW	0h	Input endpoint interrupt flag 5. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.
4	IEPIFG4	RW	0h	Input endpoint interrupt flag 4. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.
3	IEPIFG3	RW	0h	Input endpoint interrupt flag 3. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.
2	IEPIFG2	RW	0h	Input endpoint interrupt flag 2. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.
1	IEPIFG1	RW	0h	Input endpoint interrupt flag 1. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.
0	IEPIFG0	RW	0h	Input endpoint interrupt flag 0. This bit is set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.

#### 40.4.2.8 USBOEPIFG Register

USB Output Endpoint Interrupt Flag Register

**Figure 40-22. USBOEPIFG Register**

7	6	5	4	3	2	1	0
OEPIFG7	OEPIFG6	OEPIFG5	OEPIFG4	OEPIFG3	OEPIFG2	OEPIFG1	OEPIFG0
rw-0							

Can be modified only when USBEN = 1

**Table 40-21. USBOEPIFG Register Description**

Bit	Field	Type	Reset	Description
7	OEPIFG7	RW	0h	Output endpoint interrupt flag 7. The output endpoint interrupt flag is set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.
6	OEPIFG6	RW	0h	Output endpoint interrupt flag 6. The output endpoint interrupt flag is set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.
5	OEPIFG5	RW	0h	Output endpoint interrupt flag 5. The output endpoint interrupt flag is set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.
4	OEPIFG4	RW	0h	Output endpoint interrupt flag 4. The output endpoint interrupt flag is set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.
3	OEPIFG3	RW	0h	Output endpoint interrupt flag 3. The output endpoint interrupt flag set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.
2	OEPIFG2	RW	0h	Output endpoint interrupt flag 2. The output endpoint interrupt flag set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.
1	OEPIFG1	RW	0h	Output endpoint interrupt flag 1. The output endpoint interrupt flag set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.

**Table 40-21. USBOEPIFG Register Description (continued)**

Bit	Field	Type	Reset	Description
0	OEPIFG0	RW	0h	Output endpoint interrupt flag 0. The output endpoint interrupt flag set to 1 by the UBM when a successful completion of a transaction occurs to that out endpoint. When the bit is set, a USB interrupt is generated. The interrupt flag is cleared when the MCU reads the value from the USBVECINT (USBIV) register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to this bit location.

**40.4.2.9 USBVECINT Register**

USB Interrupt Vector Register

This register is also referred to as USBIV.

**Figure 40-23. USBVECINT Register**

15	14	13	12	11	10	9	8
USBIV							
r0	r0	r0	r0	r0	r0	r0	r0
USBIV							
r0	r0	r-0	r-0	r-0	r-0	r-0	r0

**Table 40-22. USBVECINT Register Description**

Bit	Field	Type	Reset	Description
15-0	USBIV	R	0h	<p>USB interrupt vector value. This register is to be accessed as a whole word only. When an interrupt is pending, reading this register results in a value that can be added to the program counter to handle the corresponding event. Writing to this register clears all pending USB interrupt flags independent of the status of USBEN.</p> <p>00h = No interrupt pending            02h = See <a href="#">Section 40.2.5.</a>; Interrupt Priority: Highest            3Eh = Interrupt Priority: Lowest</p>

#### 40.4.2.10 USBMAINT Register

Timestamp Maintenance Register

**Figure 40-24. USBMAINT Register**

15	14	13	12	11	10	9	8
	UTSEL		Reserved	TSE3		TSESEL	TSGEN
rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved							UTIE
r0	r0	r0	r0	r0	r0	rw-0	rw-0

Can be modified only when USBEN = 1

**Table 40-23. USBMAINT Register Description**

Bit	Field	Type	Reset	Description
15-13	UTSEL	RW	0h	USB timer selection 000b = USB Timer Period: 4096 µs; Approximate Frequency: 250 Hz (244 Hz) 001b = USB Timer Period: 2048 µs; Approximate Frequency: 500 Hz (488 Hz) 010b = USB Timer Period: 1024 µs; Approximate Frequency: 1 kHz (977 Hz) 011b = USB Timer Period: 512 µs; Approximate Frequency: 2 kHz (1953 Hz) 100b = USB Timer Period: 256 µs; Approximate Frequency: 4 kHz (3906 Hz) 101b = USB Timer Period: 128 µs; Approximate Frequency: 8 kHz (7812 Hz) 110b = USB Timer Period: 64 µs; Approximate Frequency: 16 kHz (15625 Hz) 111b = USB Timer Period: 32 µs; Approximate Frequency: 31 kHz (31250 Hz)
12	Reserved	R	0h	Reserved. Always reads as 0.
11	TSE3	RW	0h	Timestamp Event 3 bit. This bit allows the triggering of a software-driven timestamp event (when TSESEL = 11b). 0b = No TSE3 event signaled 1b = TSE3 event signaled
10-9	TSESEL	RW	0h	Timestamp Event Selection. TSE[2:0] are connected to the event multiplexer of the three DMA channels of the DMA controller if not otherwise noted in data sheet 00b = TSE0 (DMA0) signal is qualified timestamp event 01b = TSE1 (DMA1) signal is qualified timestamp event 10b = TSE2 (DMA2) signal is qualified timestamp event 11b = Software-driven timestamp event
8	TSGEN	RW	0h	Timestamp generator enable 0b = Timestamp mechanism disabled 1b = Timestamp mechanism enabled
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1	UTIE	RW	0h	USB timer interrupt enable bit 0b = USB timer interrupt disabled 1b = USB timer interrupt enabled
0	UTIFG	RW	0h	USB timer interrupt flag 0b = No interrupt pending 1b = Interrupt pending

**40.4.2.11 USBTSREG Register**

USB Timestamp Register

**Figure 40-25. USBTSREG Register**

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

Can be modified only when USBEN = 1

**Table 40-24. USBTSREG Register Description**

Bit	Field	Type	Reset	Description
15-0	TVAL	R	0h	Timestamp high register. The timestamp value is updated by hardware from the USB timer. A qualified timestamp trigger signal causes the current timer value to be latched into this register.

**40.4.2.12 USBFN Register**

USB Frame Number Register

**Figure 40-26. USBFN Register**

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

**Table 40-25. USBFN Register Description**

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-0	USBFN	R	0h	USB Frame Number register. The frame number bit values are updated by hardware; each USB frame with the frame number field value received in the USB start-of-frame packet. The frame number can be used as a timestamp. If the local (MSP430's) frame timer is not locked to the USB host's frame timer, then the frame number is automatically incremented from the previous value when a pseudo start-of-frame occurs.

#### 40.4.2.13 USBCTL Register

USB Control Register

**Figure 40-27. USBCTL Register**

7	6	5	4	3	2	1	0
Reserved	FEN	RWUP	FRSTE		Reserved		DIR
r0	rw-0	rw-0	rw-0	r0	r0	r0	rw-0

Can be modified only when USBEN = 1

**Table 40-26. USBCTL Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	R	0h	Reserved. Always reads as 0.
6	FEN	RW	0h	Function Enable Bit. This bit needs to be set to enable the USB device to respond to USB transactions. If this bit is not set, the UBM ignores all USB transactions. It is cleared by a USB reset. (This bit is primarily intended for debugging.) 0b = Function is disabled 1b = Function is enabled
5	RWUP	RW	0h	Device Remote Wakeup request. The remote wake-up bit is set by software to request the suspend/resume logic to generate resume signaling upstream on the USB. This bit is used to exit a USB low-power suspend state when a remote wake-up event occurs. The bit is self-clearing. 0b = Writing 0 has no effect 1b = A Remote-Wakeup pulse is generated
4	FRSTE	RW	0h	Function Reset Connection Enable. This bit selects whether a bus reset on the USB causes an internal reset of the USB module. 0b = Bus reset does not cause a reset of the module 1b = Bus reset does cause a reset of the module
3-1	Reserved	R	0h	Reserved. Always reads as 0.
0	DIR	RW	0h	Data response to setup packet interrupt status bit. Software must decode the request and set/clear this bit to reflect the data transfer direction. 0b = USB data-OUT transaction (from host to device) 1b = USB data-IN transaction (from device to host)

**40.4.2.14 USBIE Register**

USB Interrupt Enable Register

**Figure 40-28. USBIE Register**

7	6	5	4	3	2	1	0
RSTRIE	SUSRIE	RESRIE	Reserved		SETUPIE	Reserved	STPOWIE
rw-0	rw-0	rw-0	r0	r0	rw-0	r0	rw-0

Can be modified only when USBEN = 1

**Table 40-27. USBIE Register Description**

Bit	Field	Type	Reset	Description
7	RSTRIE	RW	0h	USB reset interrupt enable. Causes an interrupt to be generated if the RSTRIFG bit is set. 0b = Function Reset interrupt disabled 1b = Function Reset interrupt enabled
6	SUSRIE	RW	0h	Suspend interrupt enable. Causes an interrupt to be generated if the SUSRIFG bit is set. 0b = Suspend interrupt disabled 1b = Suspend interrupt enabled
5	RESRIE	RW	0h	Resume interrupt enable. Causes an interrupt to be generated if the RESRIFG bit is set. 0b = Resume interrupt disabled 1b = Resume interrupt enabled
4-3	Reserved	R	0h	Reserved. Always reads as 0.
2	SETUPIE	RW	0h	Setup interrupt enable. Causes an interrupt to be generated if the SETUPIFG bit is set. 0b = Setup interrupt disabled 1b = Setup interrupt enabled
1	Reserved	R	0h	Reserved. Always reads as 0.
0	STPOWIE	RW	0h	Setup Overwrite interrupt enable. Causes an interrupt to be generated if the STPOWFIG bit is set. 0b = Setup Overwrite interrupt disabled 1b = Setup Overwrite interrupt enabled

#### 40.4.2.15 USBIFG Register

USB Interrupt Flag Register

**Figure 40-29. USBIFG Register**

7	6	5	4	3	2	1	0
RSTRIFG	SUSRIFG	RESRIFG		Reserved	SETUPIFG	Reserved	STPOWIFG
rw-0	rw-0	rw-0	r0	r0	rw-0	r0	rw-0

Can be modified only when USBEN = 1

**Table 40-28. USBIFG Register Description**

Bit	Field	Type	Reset	Description
7	RSTRIFG	RW	0h	USB reset request bit. This bit is set to one by hardware in response to the host initiating a USB port reset. A USB reset causes a reset of the USB module logic, but this bit is not affected.
6	SUSRIFG	RW	0h	Suspend request bit. This bit is set by hardware in response to the host/hub causing a global or selective suspend condition.
5	RESRIFG	RW	0h	Resume request bit. This bit is set by hardware in response to the host/hub causing a resume event.
4-3	Reserved	R	0h	Reserved. Always reads as 0.
2	SETUPIFG	RW	0h	Setup transaction received bit. This bit is set by hardware when a SETUP transaction is received. As long as this bit is set, transactions on IN and OUT on endpoint-0 receive a NAK, regardless of their corresponding NAK bit value.
1	Reserved	R	0h	Reserved. Always reads as 0.
0	STPOWIFG	RW	0h	Setup overwrite bit. This bit is set by hardware when a setup packet is received while there is already a packet in the setup buffer.

#### 40.4.2.16 USBFUNADR Register

USB Function Address Register

**Figure 40-30. USBFUNADR Register**

7	6	5	4	3	2	1	0
Reserved	FA6	FA5	FA4	FA3	FA2	FA1	FA0
r0	rw-0						

Can be modified only when USBEN = 1

**Table 40-29. USBFUNADR Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	R	0h	Reserved. Always reads as 0.
6-0	FA[6:0]	RW	0h	Function address (USB address 0 to 127). These bits define the current device address assigned to this USB device. Software must write a value from 0 to 127 when a Set-Address command is received from the host.

#### 40.4.3 USB Buffer Registers and Memory

The data buffers for all endpoints, as well as the registers that define endpoints 1 to 7, are stored in the USB RAM buffer memory. Doing so allows for efficient and flexible use of this memory. The memory area is known as the USB buffer memory, and the registers that define its use are the buffer descriptor registers.

The buffer memory blocks are listed in [Table 40-30](#). The registers are listed in [Table 40-31](#). All addresses are expressed as offsets; the base address can be found in the device-specific data sheet.

All memory is byte and word accessible.

**Table 40-30. USB Buffer Memory**

Offset	Acronym	Memory	Type
0000h	USBSTABUFF	Start of buffer space	Read/Write
:	:	1904 bytes of configurable buffer space	:
076Fh	USBTOPBUFF	End of buffer space	Read/Write
0770h			Read/Write
:			:
0777h			Read/Write
0778h			Read/Write
:			:
077Fh			Read/Write
0780h			Read/Write
:			:
0787h			Read/Write

**Table 40-31. USB Buffer Descriptor Registers**

Offset	Acronym	Register Name	Type	Section
0788h	USBOEPCNF_1	Output Endpoint_1 Configuration Register	Read/Write	<a href="#">Section 40.4.3.1</a>
0789h	USBOEPBBAX_1	Output Endpoint_1 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.2</a>
078Ah	USBOEPBCTX_1	Output Endpoint_1 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.3</a>
078Dh	USBOEPBBAY_1	Output Endpoint_1 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.4</a>
078Eh	USBOEPBCTY_1	Output Endpoint_1 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.5</a>
078Fh	USBOEPSIZXY_1	Output Endpoint_1 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.6</a>
0790h	USBOEPCNF_2	Output Endpoint_2 Configuration Register	Read/Write	<a href="#">Section 40.4.3.1</a>
0791h	USBOEPBBAX_2	Output Endpoint_2 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.2</a>
0792h	USBOEPBCTX_2	Output Endpoint_2 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.3</a>
0795h	USBOEPBBAY_2	Output Endpoint_2 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.4</a>
0796h	USBOEPBCTY_2	Output Endpoint_2 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.5</a>
0797h	USBOEPSIZXY_2	Output Endpoint_2 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.6</a>
0798h	USBOEPCNF_3	Output Endpoint_3 Configuration Register	Read/Write	<a href="#">Section 40.4.3.1</a>
0799h	USBOEPBBAX_3	Output Endpoint_3 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.2</a>
079Ah	USBOEPBCTX_3	Output Endpoint_3 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.3</a>
079Dh	USBOEPBBAY_3	Output Endpoint_3 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.4</a>
079Eh	USBOEPBCTY_3	Output Endpoint_3 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.5</a>
079Fh	USBOEPSIZXY_3	Output Endpoint_3 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.6</a>

**Table 40-31. USB Buffer Descriptor Registers (continued)**

<b>Offset</b>	<b>Acronym</b>	<b>Register Name</b>	<b>Type</b>	<b>Section</b>
07A0h	USBOEPCNF_4	Output Endpoint_4 Configuration Register	Read/Write	<a href="#">Section 40.4.3.1</a>
07A1h	USBOEPBBAX_4	Output Endpoint_4 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.2</a>
07A2h	USBOEPBCTX_4	Output Endpoint_4 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.3</a>
07A5h	USBOEPBBAY_4	Output Endpoint_4 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.4</a>
07A6h	USBOEPBCTY_4	Output Endpoint_4 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.5</a>
07A7h	USBOEPSIZXY_4	Output Endpoint_4 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.6</a>
07A8h	USBOEPCNF_5	Output Endpoint_5 Configuration Register	Read/Write	<a href="#">Section 40.4.3.1</a>
07A9h	USBOEPBBAX_5	Output Endpoint_5 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.2</a>
07AAh	USBOEPBCTX_5	Output Endpoint_5 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.3</a>
07ADh	USBOEPBBAY_5	Output Endpoint_5 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.4</a>
07AEh	USBOEPBCTY_5	Output Endpoint_5 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.5</a>
07AFh	USBOEPSIZXY_5	Output Endpoint_5 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.6</a>
07B0h	USBOEPCNF_6	Output Endpoint_6 Configuration Register	Read/Write	<a href="#">Section 40.4.3.1</a>
07B1h	USBOEPBBAX_6	Output Endpoint_6 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.2</a>
07B2h	USBOEPBCTX_6	Output Endpoint_6 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.3</a>
07B5h	USBOEPBBAY_6	Output Endpoint_6 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.4</a>
07B6h	USBOEPBCTY_6	Output Endpoint_6 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.5</a>
07B7h	USBOEPSIZXY_6	Output Endpoint_6 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.6</a>
07B8h	USBOEPCNF_7	Output Endpoint_7 Configuration Register	Read/Write	<a href="#">Section 40.4.3.1</a>
07B9h	USBOEPBBAX_7	Output Endpoint_7 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.2</a>
07BAh	USBOEPBCTX_7	Output Endpoint_7 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.3</a>
07BDh	USBOEPBBAY_7	Output Endpoint_7 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.4</a>
07BEh	USBOEPBCTY_7	Output Endpoint_7 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.5</a>
07BFh	USBOEPSIZXY_7	Output Endpoint_7 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.6</a>
07C8h	USBIEPCNF_1	Input Endpoint_1 Configuration Register	Read/Write	<a href="#">Section 40.4.3.7</a>
07C9h	USBIEPBBAX_1	Input Endpoint_1 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.8</a>
07CAh	USBIEPBCTX_1	Input Endpoint_1 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.9</a>
07CDh	USBIEPBBAY_1	Input Endpoint_1 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.10</a>
07CEh	USBIEPBCTY_1	Input Endpoint_1 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.11</a>
07CFh	USBIEEPSIZXY_1	Input Endpoint_1 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.12</a>
07D0h	USBIEPCNF_2	Input Endpoint_2 Configuration Register	Read/Write	<a href="#">Section 40.4.3.7</a>
07D1h	USBIEPBBAX_2	Input Endpoint_2 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.8</a>
07D2h	USBIEPBCTX_2	Input Endpoint_2 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.9</a>
07D5h	USBIEPBBAY_2	Input Endpoint_2 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.10</a>
07D6h	USBIEPBCTY_2	Input Endpoint_2 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.11</a>
07D7h	USBIEEPSIZXY_2	Input Endpoint_2 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.12</a>
07D8h	USBIEPCNF_3	Input Endpoint_3 Configuration Register	Read/Write	<a href="#">Section 40.4.3.7</a>
07D9h	USBIEPBBAX_3	Input Endpoint_3 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.8</a>
07DAh	USBIEPBCTX_3	Input Endpoint_3 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.9</a>
07DDh	USBIEPBBAY_3	Input Endpoint_3 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.10</a>
07DEh	USBIEPBCTY_3	Input Endpoint_3 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.11</a>
07DFh	USBIEEPSIZXY_3	Input Endpoint_3 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.12</a>

**Table 40-31. USB Buffer Descriptor Registers (continued)**

Offset	Acronym	Register Name	Type	Section
07E0h	USBIEPCNF_4	Input Endpoint_4 Configuration Register	Read/Write	<a href="#">Section 40.4.3.7</a>
07E1h	USBIEPBBAX_4	Input Endpoint_4 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.8</a>
07E2h	USBIEPBCTX_4	Input Endpoint_4 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.9</a>
07E5h	USBIEPBBAY_4	Input Endpoint_4 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.10</a>
07E6h	USBIEPBCTY_4	Input Endpoint_4 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.11</a>
07E7h	USBIEPSIZXY_4	Input Endpoint_4 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.12</a>
07E8h	USBIEPCNF_5	Input Endpoint_5 Configuration Register	Read/Write	<a href="#">Section 40.4.3.7</a>
07E9h	USBIEPBBAX_5	Input Endpoint_5 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.8</a>
07EAh	USBIEPBCTX_5	Input Endpoint_5 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.9</a>
07EDh	USBIEPBBAY_5	Input Endpoint_5 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.10</a>
07EEh	USBIEPBCTY_5	Input Endpoint_5 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.11</a>
07EFh	USBIEPSIZXY_5	Input Endpoint_5 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.12</a>
07F0h	USBIEPCNF_6	Input Endpoint_6 Configuration Register	Read/Write	<a href="#">Section 40.4.3.7</a>
07F1h	USBIEPBBAX_6	Input Endpoint_6 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.8</a>
07F2h	USBIEPBCTX_6	Input Endpoint_6 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.9</a>
07F5h	USBIEPBBAY_6	Input Endpoint_6 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.10</a>
07F6h	USBIEPBCTY_6	Input Endpoint_6 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.11</a>
07F7h	USBIEPSIZXY_6	Input Endpoint_6 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.12</a>
07F8h	USBIEPCNF_7	Input Endpoint_7 Configuration Register	Read/Write	<a href="#">Section 40.4.3.7</a>
07F9h	USBIEPBBAX_7	Input Endpoint_7 X Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.8</a>
07FAh	USBIEPBCTX_7	Input Endpoint_7 X Byte Count Register	Read/Write	<a href="#">Section 40.4.3.9</a>
07FDh	USBIEPBBAY_7	Input Endpoint_7 Y Buffer Base Address Register	Read/Write	<a href="#">Section 40.4.3.10</a>
07FEh	USBIEPBCTY_7	Input Endpoint_7 Y Byte Count Register	Read/Write	<a href="#">Section 40.4.3.11</a>
0FFFh	USBIEPSIZXY_7	Input Endpoint_7 X and Y Buffer Size Register	Read/Write	<a href="#">Section 40.4.3.12</a>

#### 40.4.3.1 USBOEPCNF\_n Register

Output Endpoint-n Configuration Register

This register can be modified only when USBEN = 1.

**Figure 40-31. USBOEPCNF\_n Register**

7	6	5	4	3	2	1	0
UBME	Reserved	TOGGLE	DBUF	STALL	USBIIE	Reserved	
rw	r0	rw	rw	rw	rw	r0	r0

Can be modified only when USBEN = 1

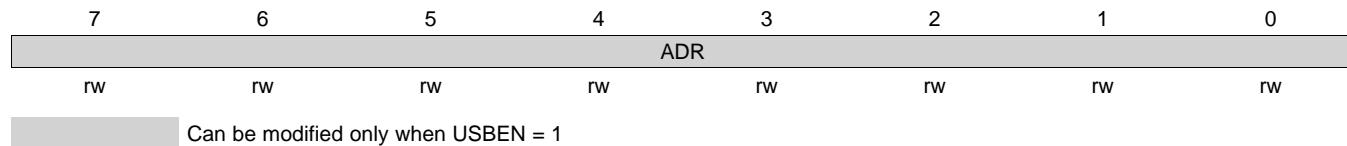
**Table 40-32. USBOEPCNF\_n Register Description**

Bit	Field	Type	Reset	Description
7	UBME	RW	0h	UBM out endpoint-n enable. This bit is to be set or cleared by software. 0b = UBM cannot use this endpoint 1b = UBM can use this endpoint
6	Reserved	R	0h	Reserved. Always reads as 0.
5	TOGGLE	RW	0h	Toggle bit. The toggle bit is controlled by the UBM and is toggled at the end of a successful out data stage transaction, if a valid data packet is received and the data packet's packet ID matches the expected packet ID.
4	DBUF	RW	0h	Double buffer enable. This bit can be set to enable the use of both the X and Y data packet buffers for USB transactions, for a particular out endpoint. Clearing it results in the use of single buffer mode. In this mode, only the X buffer is used. 0b = Primary buffer only (X-buffer only) 1b = Toggle bit selects buffer
3	STALL	RW	0h	USB stall condition. This bit can be set to cause endpoint transactions to be stalled. When set, the hardware automatically returns a stall handshake to the host for any transaction received on endpoint-0. The stall bit is cleared automatically by the next setup transaction. 0b = Indicates no stall 1b = Indicates stall
2	USBIIE	RW	0h	USB transaction interrupt indication enable. Can be set or cleared to define if interrupts are to be flagged in general. To generate an interrupt, the corresponding interrupt flag must be set (OEPIE). 0b = Corresponding interrupt flag will not be set 1b = Corresponding interrupt flag will be set
1-0	Reserved	R	0h	Reserved. Always reads as 0.

**40.4.3.2 USBOEPBBAX\_n Register**

Output Endpoint-n X Buffer Base Address Register

This register can be modified only when USBEN = 1.

**Figure 40-32. USBOEPBBAX\_n Register****Table 40-33. USBOEPBBAX\_n Register Description**

Bit	Field	Type	Reset	Description
7-0	ADR	RW	0h	X-buffer base address. These are the upper eight bits of the X-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

#### 40.4.3.3 USBOEPBCTX\_n Register

Output Endpoint-n X Byte Count Register

This register can be modified only when USBEN = 1.

**Figure 40-33. USBOEPBCTX\_n Register**

7	6	5	4	3	2	1	0
NAK				CNT			
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

**Table 40-34. USBOEPBCTX\_n Register Description**

Bit	Field	Type	Reset	Description
7	NAK	RW	0h	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB out transaction to that endpoint, to indicate that the USB endpoint-"n" buffer contains a valid data packet, and that the buffer data count value is valid. When this bit is set, all subsequent transactions to that endpoint result in a NAK handshake response to the USB host. To re-enable this endpoint to receive another data packet from the host, this bit must be cleared. 0b = No valid data in buffer. The buffer is ready to receive OUT packets from the host. 1b = The buffer contains a valid packet from the host, and it has not been picked up (subsequent host-out requests receive a NAK)
6-0	CNT	RW	0h	X-buffer data count. The Out_EP-n data count value is set by the UBM when a new data packet is written to the X-buffer for that out endpoint. It is set to the number of bytes received in the data buffer.

#### 40.4.3.4 USBOEPBBAY\_n Register

Output Endpoint-n Y Buffer Base Address Register

This register can be modified only when USBEN = 1.

**Figure 40-34. USBOEPBBAY\_n Register**

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

Can be modified only when USBEN = 1

**Table 40-35. USBOEPBBAY\_n Register Description**

Bit	Field	Type	Reset	Description
7-0	ADR	RW	0h	Y-buffer base address. These are the upper eight bits of the Y-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

**40.4.3.5 USBOEPBCTY\_n Register**

Output Endpoint-n X Byte Count Register

This register can be modified only when USBEN = 1.

**Figure 40-35. USBOEPBCTY\_n Register**

7	6	5	4	3	2	1	0
NAK	CNT						
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

**Table 40-36. USBOEPBCTY\_n Register Description**

Bit	Field	Type	Reset	Description
7	NAK	RW	0h	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB out transaction to that endpoint, to indicate that the USB endpoint-"n" buffer contains a valid data packet, and that the buffer data count value is valid. When this bit is set, all subsequent transactions to that endpoint result in a NAK handshake response to the USB host. To re-enable this endpoint to receive another data packet from the host, this bit must be cleared. 0b = No valid data in buffer. The buffer is ready to receive OUT packets from the host. 1b = The buffer contains a valid packet from the host, and it has not been picked up (subsequent host-out requests receive a NAK)
6-0	CNT	RW	0h	Y-buffer data count. The Out_EP-n data count value is set by the UBM when a new data packet is written to the X-buffer for that out endpoint. It is set to the number of bytes received in the data buffer.

**40.4.3.6 USBOEPSIZXY\_n Register**

Output Endpoint-n X and Y Buffer Size Register

This register can be modified only when USBEN = 1.

**Figure 40-36. USBOEPSIZXY\_n Register**

7	6	5	4	3	2	1	0
Reserved	SIZx						
r0	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

**Table 40-37. USBOEPSIZXY\_n Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	R	0h	Reserved. Always reads as 0.
6-0	SIZx	RW	0h	Buffer size count. This value needs to be set by software to configure the size of the X and Y data packet buffers. Both buffers are set to the same size, based on this value. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value greater than or equal to 100:0001b results in unpredictable results.

#### **40.4.3.7 USBIEPCNF\_n Register**

Input Endpoint-n Configuration Register

This register can be modified only when USBEN = 1.

**Figure 40-37. USBIEPCNF\_n Register**

7	6	5	4	3	2	1	0
UBME	Reserved	TOGGLE	DBUF	STALL	USBIIE	Reserved	
rw	r0	rw	rw	rw	rw	r0	r0

Can be modified only when USBEN = 1

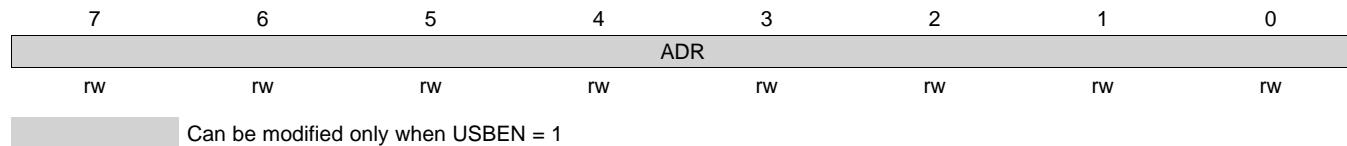
**Table 40-38. USBIEPCNF\_n Register Description**

Bit	Field	Type	Reset	Description
7	UBME	RW	0h	UBM in endpoint-n enable. This value needs to be set or cleared by software. 0b = UBM cannot use this endpoint 1b = UBM can use this endpoint
6	Reserved	R	0h	Reserved. Always reads as 0.
5	TOGGLE	RW	0h	Toggle bit. The toggle bit is controlled by the UBM and is toggled at the end of a successful in data stage transaction, if a valid data packet is transmitted. If this bit is cleared, a DATA0 packet ID is transmitted in the data packet to the host. If this bit is set, a DATA1 packet ID is transmitted in the data packet.
4	DBUF	RW	0h	Double buffer enable. This bit can be set to enable the use of both the X and Y data packet buffers for USB transactions, for a particular out endpoint. Clearing it results in the use of single buffer mode. In this mode, only the X buffer is used. 0b = Primary buffer only (X-buffer only) 1b = Toggle bit selects buffer
3	STALL	RW	0h	USB stall condition. This bit can be set to cause endpoint transactions to be stalled. When set, the hardware automatically returns a stall handshake to the host for any transaction received on endpoint-0. The stall bit is cleared automatically by the next setup transaction. 0b = Indicates no stall 1b = Indicates stall
2	USBIIE	RW	0h	USB transaction interrupt indication enable. Can be set or cleared to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt flag must be set (OEPIE). 0b = Corresponding interrupt flag will not be set 1b = Corresponding interrupt flag will be set
1-0	Reserved	R	0h	Reserved. Always reads as 0.

**40.4.3.8 USBIEPBBAX\_n Register**

Input Endpoint-n X Buffer Base Address Register

This register can be modified only when USBEN = 1.

**Figure 40-38. USBIEPBBAX\_n Register****Table 40-39. USBIEPBBAX\_n Register Description**

Bit	Field	Type	Reset	Description
7-0	ADR	RW	0h	X-buffer base address. These are the upper eight bits of the X-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

#### 40.4.3.9 USBIEPBCTX\_n Register

Input Endpoint-n X Byte Count Register

This register can be modified only when USBEN = 1.

**Figure 40-39. USBIEPBCTX\_n Register**

7	6	5	4	3	2	1	0
NAK				CNT			
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

**Table 40-40. USBIEPBCTX\_n Register Description**

Bit	Field	Type	Reset	Description
7	NAK	RW	0h	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB transaction from that endpoint, to indicate that the EP-n in buffer is empty. For interrupt or bulk endpoints, when this bit is set, all subsequent transactions from that endpoint result in a NAK handshake response to the USB host. To re-enable this endpoint to transmit another data packet to the host, this bit must be cleared. 0b = Buffer contains a valid data packet for the host 1b = Buffer is empty (any host-In requests receive a NAK)
6-0	CNT	RW	0h	X-buffer data count. The In_EP-n X-buffer data count value must be set by software when a new data packet is written to the buffer. It should be the number of bytes in the data packet for interrupt, or bulk endpoint transfers. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value greater than or equal to 100:0001b results in unpredictable results.

#### 40.4.3.10 USBIEPBBAY\_n Register

Input Endpoint-n Y Buffer Base Address Register

This register can be modified only when USBEN = 1.

**Figure 40-40. USBIEPBBAY\_n Register**

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

Can be modified only when USBEN = 1

**Table 40-41. USBIEPBBAY\_n Register Description**

Bit	Field	Type	Reset	Description
7-0	ADR	RW	0h	Y-buffer base address. These are the upper eight bits of the Y-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

**40.4.3.11 USBIEPBCTY\_n Register**

Input Endpoint-n Y Byte Count Register

This register can be modified only when USBEN = 1.

**Figure 40-41. USBIEPBCTY\_n Register**

7	6	5	4	3	2	1	0
NAK	CNT						
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

**Table 40-42. USBIEPBCTY\_n Register Description**

Bit	Field	Type	Reset	Description
7	NAK	RW	0h	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB transaction from that endpoint, to indicate that the EP-n in buffer is empty. For interrupt or bulk endpoints, when this bit is set, all subsequent transactions from that endpoint result in a NAK handshake response to the host. To re-enable this endpoint to transmit another data packet to the host, this bit must be cleared. This bit is set by USB SW-init. 0b = Buffer contains a valid data packet for host device 1b = Buffer is empty (any host-in requests receive a NAK)
6-0	CNT	RW	0h	Y-Buffer data count. The In EP-n Y-buffer data count value needs to be set by software when a new data packet is written to the buffer. It should be the number of bytes in the data packet for interrupt, or bulk endpoint transfers. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value greater than or equal to 100:0001b results in unpredictable results.

**40.4.3.12 USBIEPSIZXY\_n Register**

Input Endpoint-n X and Y Buffer Size Register

This register can be modified only when USBEN = 1.

**Figure 40-42. USBIEPSIZXY\_n Register**

7	6	5	4	3	2	1	0
Reserved	SIZ						
r0	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

**Table 40-43. USBIEPSIZXY\_n Register Description**

Bit	Field	Type	Reset	Description
7	Reserved	R	0h	Reserved. Always reads as 0.
6-0		RW	0h	Buffer size count. This value needs to be set by software to configure the size of the X and Y data packet buffers. Both buffers are set to the same size, based on this value. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value greater than or equal to 100:0001b results in unpredictable results.

## **LDO-PWR Module**

This chapter describes the LDO-PWR module that is available in some devices.

<b>Topic</b>	<b>Page</b>
41.1 LDO-PWR Introduction .....	1132
41.2 LDO-PWR Operation .....	1133
41.3 LDO-PWR Registers .....	1136

## 41.1 LDO-PWR Introduction

The features of the LDO-PWR module include:

- Integrated 3.3-V LDO regulator with sufficient output to power the entire MSP430™ microcontroller and system circuitry from 5-V external supply
- Current-limiting capability on 3.3-V LDO output with detection flag and interrupt generation
- LDO input voltage detection flag and interrupt generation

Figure 41-1 shows a block diagram of the LDO-PWR module.

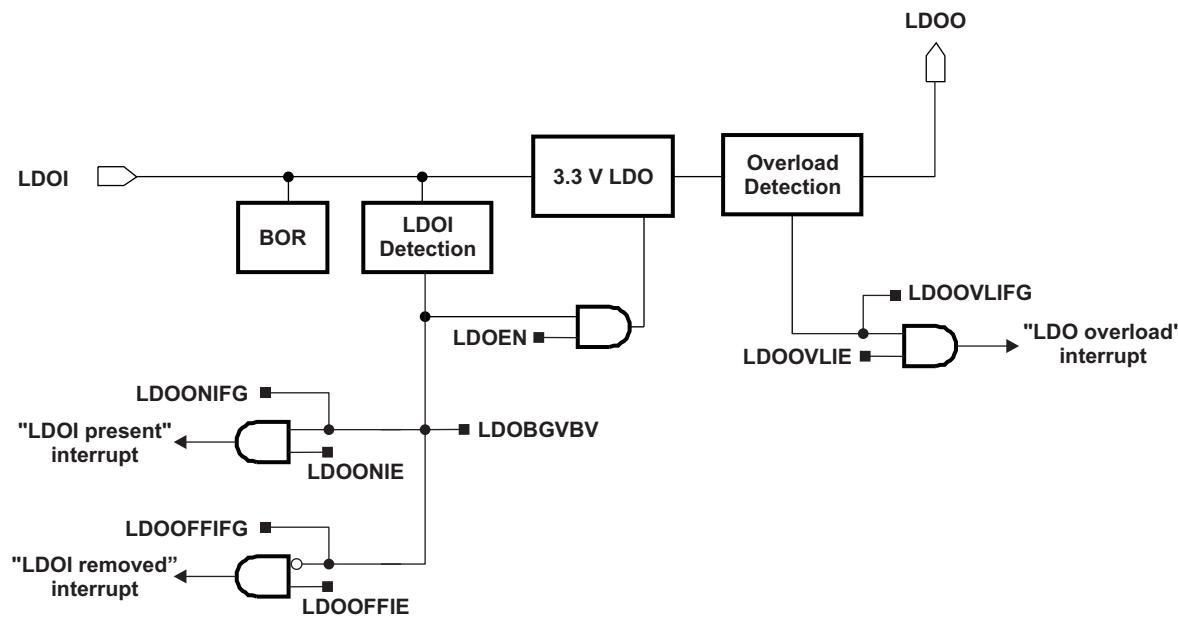


Figure 41-1. LDO Block Diagram

## 41.2 LDO-PWR Operation

The LDO-PWR power system incorporates an integrated 3.3-V LDO regulator that allows the entire MSP430 microcontroller to be powered from nominal 5-V LDOI when it is made available from the system. Alternatively, the power system can supply power only to other components within the system, or it can be unused altogether.

### 41.2.1 Enabling/Disabling

The 3.3-V LDO is enabled/disabled by setting/clearing LDOEN (LDOEN = 1 by default). If the voltage on LDOI is detected to be low or nonexistent, the LDO is suspended even if enabled by LDOEN = 1. No additional current is consumed while the LDO is suspended. When the voltage on LDOI rises above the LDO power brownout level, the LDO reference and low voltage detection become enabled. When the voltage on LDOI rises further above the launch voltage  $V_{LAUNCH}$ , the 3.3-V LDO becomes enabled (see [Figure 41-2](#)). See device-specific data sheet for value of  $V_{LAUNCH}$ .

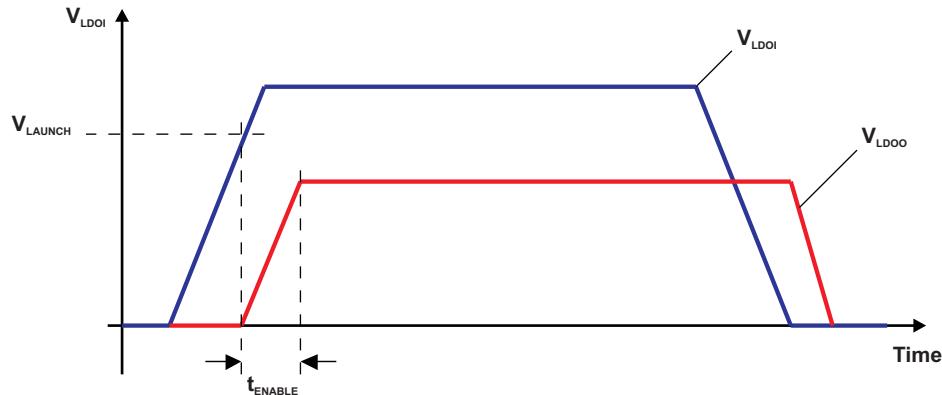


Figure 41-2. 3.3-V LDO Power Up/Down Profile

### 41.2.2 Powering the Rest of the MSP430 from the LDO-PWR

The output of the 3.3-V LDO (LDOO) can be used to power the entire MSP430 device, sourcing the DVCC rail. If this is desired, LDOO and DVCC should be connected externally. Power from the 3.3-V LDO is sourced into DVCC (see [Figure 41-3](#)).

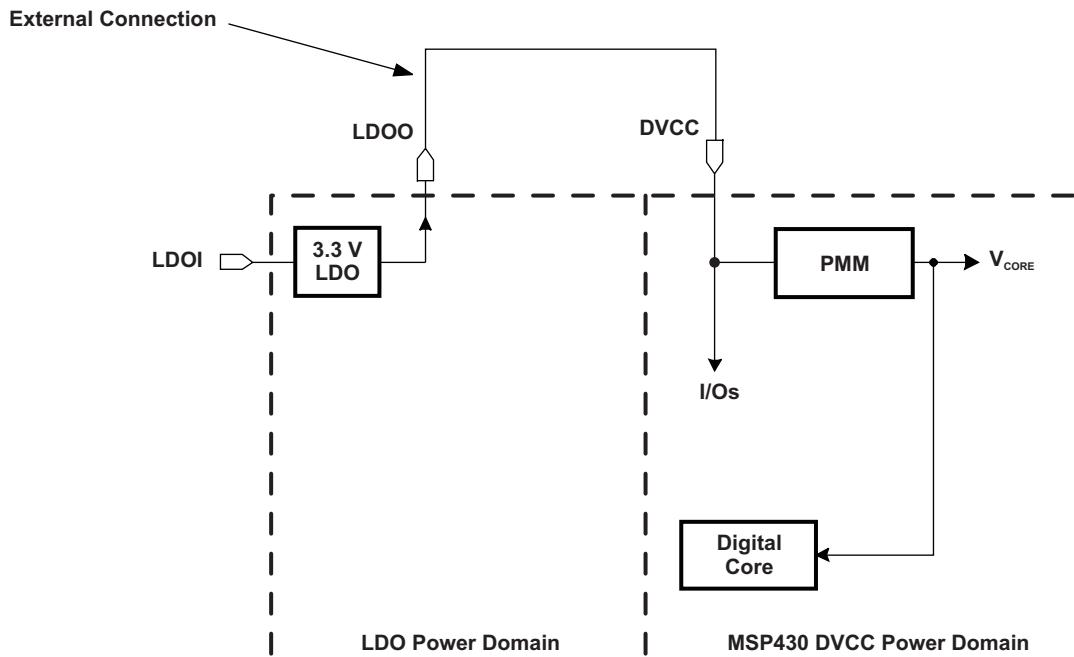


Figure 41-3. Powering Entire MSP430 From LDOI

With this connection made, the MSP430 allows for autonomous power up of the device when the voltage on LDOI rises above  $V_{LAUNCH}$ . If no voltage is present on  $V_{CORE}$  – meaning the device is unpowered (or, in LPMx.5 mode) – then the 3.3-V LDO automatically turns on when the voltage on LDOI rises above  $V_{LAUNCH}$ .

Note that if DVCC is being driven from the 3.3-V LDO in this manner, and if power is available from LDOI, attempting to place the device into LPMx.5 results in the device immediately re-powering. This is because it recreates the conditions of the autonomous feature described above (no  $V_{CORE}$  but power available on LDOI). The resulting drop of  $V_{CORE}$  would cause the system to immediately power up again.

When DVCC is being powered from LDOI, it is up to the user to ensure that the total current being drawn from LDOI stays below the current overload detection,  $I_{DET}$ .

#### 41.2.3 Powering Other Components in the System from LDO-PWR

There is sufficient current capacity available from the 3.3-V LDO to power not only the entire MSP430 but also other components in the system, via the LDOO pin. It is also possible to use the 3.3-V LDO only to power other components in the system.

#### 41.2.4 Applications That Do Not Require LDO-PWR

There may be applications where the 3.3-V LDO is not required. In these cases, keep LDOI tied low to prevent the 3.3-V LDO from turning on and drawing any current even if LDOEN = 1.

#### 41.2.5 Current Limitation and Overload Protection

The 3.3-V LDO features current limitation to protect itself from an overload condition; that is, when the output of the LDO becomes current-limited to  $I_{DET}$ . This is reported to software via the VUOVLIFG flag. See device-specific data sheet for value of  $I_{DET}$ .

During overload conditions, LDOO drops below its nominal output voltage. In power scenarios where DVCC is exclusively supplied from LDOO, repetitive system restarts may be triggered as long the overload condition exists. For this reason, firmware should avoid re-enabling high current draw modules after detection of an overload on the previous power session, until the cause of failure can be identified and corrected. Ultimately, it is the user's responsibility to ensure that the current drawn from LDOI does not exceed  $I_{DET}$ .

The LDO-PWR power system brownout circuit is supplied from LDOI or DVCC, whichever carries the higher voltage.

#### 41.2.6 LDO-PWR Interrupts

The LDO-PWR module shares a single interrupt vector to handle multiple LDO-PWR interrupts.

The LDOONIFG flag can be used to indicate that the voltage on LDOI has risen above the launch voltage. In addition to the LDOONIFG being set, an interrupt is also generated when LDOONIE = 1. Similarly, the LDOOFFIFG flag can be used to indicate that the voltage on LDOI has fallen below the launch voltage. In addition to the LDOOFFIFG being set, an interrupt is also generated when LDOOFFIE = 1. The LDOBGVBV bit can also be polled to indicate the level of LDOI; that is, above or below the launch voltage.

The LDOOVLIFG flag can be used to indicate an overcurrent condition on the 3.3-V LDO. When an overcurrent condition is detected, LDOOVLIFG = 1. In addition to the LDOOVLIFG being set, an interrupt is also generated when LDOOVLIE = 1.

#### 41.2.7 Port U Control

The Port U pins (PU.0/PU.1) function as general-purpose high-current I/O pins. These pins can only be configured together as either both inputs or both outputs. Port U is supplied by the LDOO rail. If the 3.3-V LDO is not being used in the system (disabled), the LDOO pin can be supplied externally.

PUOPE controls the enable of both outputs residing on the Port U pins. Setting PUIPE = 1 causes both input buffers to be enabled. When Port U outputs are enabled (PUOPE = 1), the PUIN0 and PUIN1 pins mirror what is present on the outputs assuming PUIPE = 1. To use the Port U pins as inputs, the outputs should be disabled by setting PUOPE = 0, and enabling the input buffers by setting PUIPE = 1. Once configured as inputs (PUIPE = 1), the PUIN0 and PUIN1 bits can be read to determine the respective input values.

When PUOPE is set, both Port U pins function as outputs, controlled by PUOUT0/PUOUT1. When driven high, they use the LDOO rail, and they are capable of a drive current higher than other I/O pins on the device. See the device-specific datasheet for parameters.

By default, PUOPE and PUIPE are cleared. PU.0/PU.1 are high-impedance (input buffers are disabled and outputs are disabled).

## 41.3 LDO-PWR Registers

Write access to the configuration registers is allowed or disallowed using the LDOKEYPID register. Writing the proper value (9628h) unlocks the configuration registers and enables write access. Writing any other value disables access while leaving the values of the registers intact. Locking should be done intentionally after the configuration is finished. Read access is available without the need to write to the LDOKEYPID register.

The configuration registers are listed in [Table 41-1](#). All addresses are expressed as offsets; the base address can be found in the device-specific data sheet.

All registers are byte and word accessible.

**Table 41-1. LDO-PWR Registers**

Offset	Acronym	Register Name	Type	Reset	Section
00h	LDOKEYPID	LDO key and ID register	Read/Write	0000h	<a href="#">Section 41.3.1</a>
04h	PUCTL	Port U control register	Read/Write	0000h	<a href="#">Section 41.3.2</a>
08h	LDOPWRCTL	LDO-PWR control register	Read/Write	0810h	<a href="#">Section 41.3.3</a>

### 41.3.1 LDOKEYPID Register

LDO Key Register

**Figure 41-4. LDOKEYPID Register**

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

**Table 41-2. LDOKEYPID Register Description**

Bit	Field	Type	Reset	Description
15-0	LDOKEY	RW	0h	Key register. Must be written with a value of 9628h in order to be recognized as a valid key. This "unlocks" the configuration registers. If written with any other value, the registers become "locked". Reads back as A528h if the registers are unlocked.

### 41.3.2 PUCTL Register

Port U Control Register

**Figure 41-5. PUCTL Register**

15	14	13	12	11	10	9	8
Reserved						Reserved	PUIPE
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved	PUOPE	Reserved	PUIN1	PUINO	PUOUT1	PUOUT0	PUOUT0
r0	r0	rw-0	rw-0	r	r	rw-0	rw-0

**Table 41-3. PUCTL Register Description**

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9	Reserved	RW	0h	Reserved. Always write as 0.
8	PUIPE	RW	0h	PU input enable 0b = PU.0 and PU.1 inputs are disabled 1b = PU.0 and PU.1 inputs are enabled
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5	PUOPE	RW	0h	PU output enable 0b = PU.0 and PU.1 outputs are disabled 1b = PU.0 and PU.1 outputs are enabled
4	Reserved	R	0h	Reserved. Always write as 0.
3	PUIN1	R	0h	PU.1 input data, This bit reflects the logic value on the PU.1 terminal when PUIPE = 1.
2	PUINO	R	0h	PU.0 input data, This bit reflects the logic value on the PU.0 terminal when PUIPE = 1.
1	PUOUT1	RW	0h	PU.1 output data. This bits defines the value of the PU.1 pin when PUOPE = 1.
0	PUOUT0	RW	0h	PU.0 output data. This bits defines the value of the PU.0 pin when PUOPE = 1.

**41.3.3 LDOPWRCTL Register**

LDO-Power Control Register

**Figure 41-6. LDOPWRCTL Register**

15	14	13	12	11	10	9	8
Reserved		Reserved	LDOEN	LDOOFFIE	LDOONIE	LDOOVLIE	
r0	r0	r0	rw-0	rw-1	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved		OVLAOFF	Reserved	LDOBGBVB	LDOOFFIFG	LDOONIFG	LDOOVLIFG
r0	r0	rw-0	rw-1	r	rw-0	rw-0	rw-0

Can be modified only when LDOKEYPID is unlocked

**Table 41-4. LDOPWRCTL Register Description**

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12	Reserved	RW	0h	Reserved. Always reads as 0.
11	LDOEN	RW	1h	3.3V LDO enable. When set, the LDO is enabled. 0b = LDO disabled 1b = LDO enabled
10	LDOOFFIE	RW	0h	LDOI voltage "going OFF" interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
9	LDOONIE	RW	0h	LDOI voltage "coming ON" interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
8	LDOOVLIE	RW	0h	LDO overload indication interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5	OVLAOFF	RW	0h	LDO overload auto-off enable 0b = During an overload on the 3.3-V LDO, the LDO automatically enters current-limiting mode and stays there until the condition stops. 1b = An overload indication clears the LDOEN bit.
4	Reserved	RW	1h	Reserved
3	LDOBGBVB	RW	0h	LDOI valid 0b = LDOI is not valid yet 1b = LDOI is valid and within bounds
2	LDOOFFIFG	RW	0h	LDOI "going OFF" interrupt flag. This bit indicates that LDOI fell below the launch voltage. 0b = LDOI did not fall below the launch voltage. 1b = LDOI fell below the launch voltage.
1	LDOONIFG	RW	0h	LDOI "coming ON" interrupt flag. This bit indicates that LDOI rose above the launch voltage. 0b = LDOI did not rise above the launch voltage. 1b = LDOI rose above the launch voltage.
0	LDOOVLIFG	RW	0h	LDO overload interrupt flag. This bit indicates that the 3.3-V LDO entered an overload condition. 0b = No overload condition detected. 1b = Overload condition detected.

## ***Embedded Emulation Module (EEM)***

This chapter describes the embedded emulation module (EEM) that is implemented in all flash devices.

Topic	Page
<b>42.1 Embedded Emulation Module (EEM) Introduction .....</b>	<b>1140</b>
<b>42.2 EEM Building Blocks .....</b>	<b>1142</b>
<b>42.3 EEM Configurations .....</b>	<b>1143</b>

## 42.1 Embedded Emulation Module (EEM) Introduction

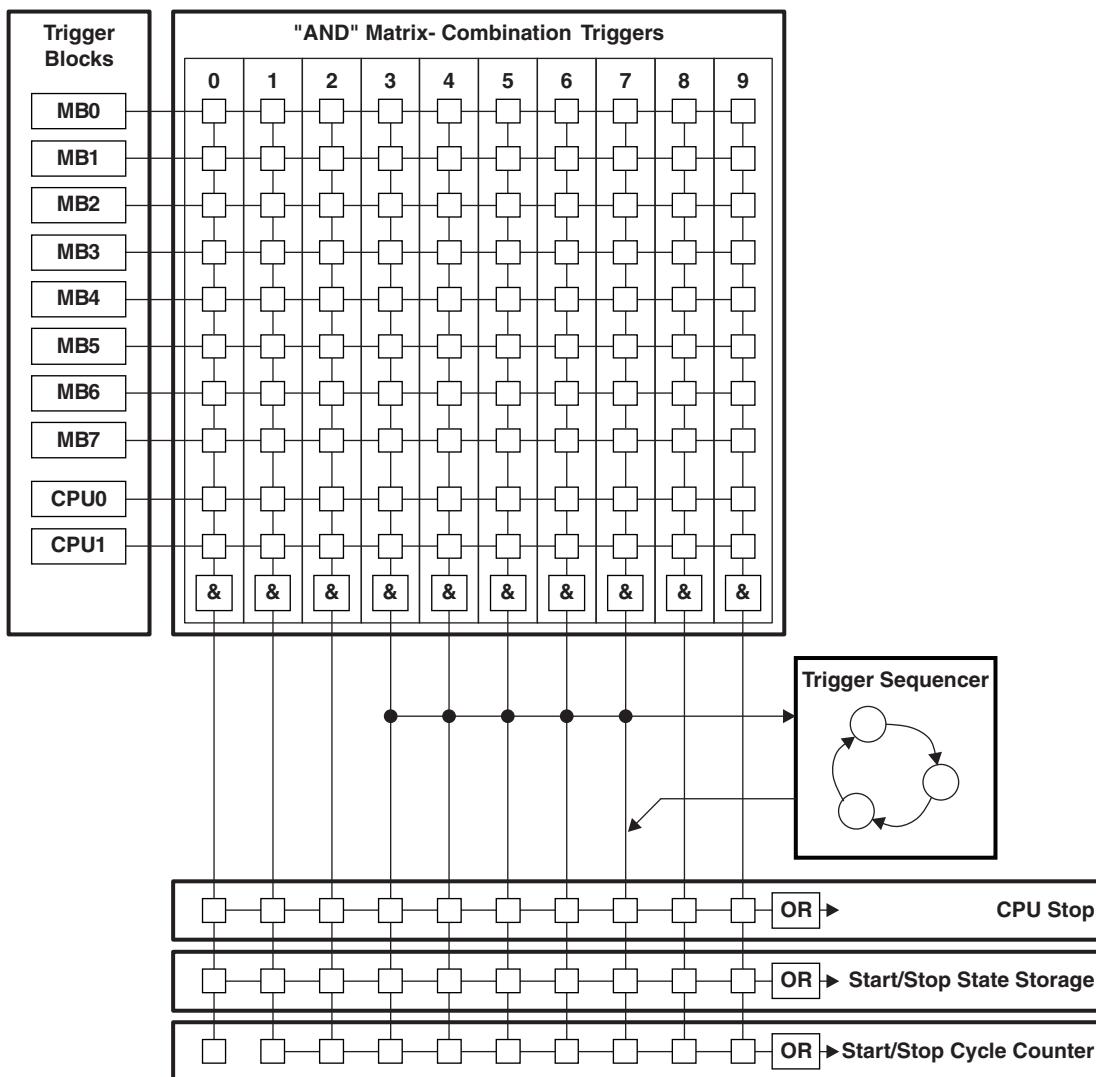
Every MSP430 flash-based microcontroller implements an EEM. It is accessed and controlled through either 4-wire JTAG mode or Spy-Bi-Wire mode. Each implementation is device dependent and is described in [Section 42.3](#), the EEM Configurations section, and the device-specific data sheet.

In general, the following features are available:

- Nonintrusive code execution with real-time breakpoint control
- Single-step, step-into, and step-over functionality
- Full support of all low-power modes
- Support for all system frequencies, for all clock sources
- Up to eight (device-dependent) hardware triggers or breakpoints on memory address bus (MAB) or memory data bus (MDB)
- Up to two (device-dependent) hardware triggers or breakpoints on CPU register write accesses
- MAB, MDB, and CPU register access triggers can be combined to form up to ten (device dependent) complex triggers or breakpoints
- Up to two (device dependent) cycle counters
- Trigger sequencing (device dependent)
- Storage of internal bus and control signals using an integrated trace buffer (device dependent)
- Clock control for timers, communication peripherals, and other modules on a global device level or on a per-module basis during an emulation stop

[Figure 42-1](#) shows a simplified block diagram of the largest currently-available EEM implementation.

For more details on how the features of the EEM can be used together with the IAR Embedded Workbench™ debugger or with Code Composer Studio (CCS), see *Advanced Debugging Using the Enhanced Emulation Module (SLAA393)* at [www.msp430.com](http://www.msp430.com). Most other debuggers supporting the MSP430 devices have the same or a similar feature set. For details, see the user's guide of the applicable debugger.



**Figure 42-1. Large Implementation of EEM**

## 42.2 EEM Building Blocks

### 42.2.1 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.

### 42.2.2 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.

### 42.2.3 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.

### 42.2.4 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.

#### 42.2.5 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.

### 42.3 EEM Configurations

[Table 42-1](#) gives an overview of the EEM configurations. The implemented configuration is device dependent, and details can be found in the device-specific data sheet and these documents:

*Advanced Debugging Using the Enhanced Emulation Module (EEM) With CCS Version 4* ([SLAA393](#))

*IAR Embedded Workbench Version 3+ for MSP430 User's Guide* ([SLAU138](#))

*Code Composer Studio v4.2 for MSP430 User's Guide* ([SLAU157](#))

**Table 42-1. EEM Configurations**

Feature	XS	S	M	L
Memory bus triggers	2 (=, ≠ only)	3	5	8
Memory bus trigger mask for	1) Low byte 2) High byte 3) Four upper addr bits	1) Low byte 2) High byte 3) Four upper addr bits	1) Low byte 2) High byte 3) Four upper addr bits	All 16 or 20 bits
CPU register write triggers	0	1	1	2
Combination triggers	2	4	6	10
Sequencer	No	No	Yes	Yes
State storage	No	No	No	Yes
Cycle counter	1	1	1	2 (including triggered start or stop)

In general, the following features can be found on any device:

- At least two MAB or MDB triggers supporting:
  - Distinction between CPU, DMA, read, and write accesses
  - =, ≠, ≥, or ≤ comparison (in XS, only =, ≠)
- At least two trigger combination registers
- Hardware breakpoints using the CPU stop reaction
- At least one 40-bit cycle counter
- Enhanced clock control with individual control of module clocks

## Revision History

### Changes From Revision L (January 2013) to Revision M (February 2013)

Location	Description
Section 1.6	Clarified port nomenclature.
Section 5.2, Section 5.2.4, Section 5.2.5	Added information regarding XIN and XOUT pins with and without shared I/O.
Section 5.2.6	Clarified FLL frequency calculation with respect to FLLN = 0h settings.
Section 8.6.3	Changed description.
Section 8.7	Changed code example.
Section 21.1	Corrected typo (changed pmm to ppm).
Section 34.2, Section 37.2	Added comment that wake from LPMx.5 is not supported.

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

### Changes From Revision K (August 2012) to Revision L (January 2013)

Location	Description
Section 1.3.4.1, Section 6.6.2.19, Section 6.6.2.20	Added note "Enable and Disable Interrupt".
Section 8.6.1	Fixed typo.
Section 12.4.3	Fixed typo in "Bit" column
Section 17.2.1.1	Added INCLK as a clock source option.
Section 17.3.1	Changed TASSEL bit option 11b to INCLK.
Section 18.2.1.2	Added INCLK as a clock source option.
Section 18.3.1	Changed TBSEL bit option 11b to INCLK.
Section 23.2.4	Added note "Changing RT0IP or RT1IP"
Section 23.2.6	Added to description regarding interrupt flags.
Section 24.2.6	Changed note "Changing RT0IP or RT1IP".
Table 24-2	Corrected some reset values.
Section 28.2.7.2	Changed description of ADC12SC.
Section 28.3.5	Changed description of ADC12INCHx.
Section 37.4.11	Changed UCTXIFG reset to correct value.
Section 37.4.5	Changed descriptions of UCFE, UCPE, and UCBRK.
Table 38-11	Corrected register offsets.
Section 38.3.3.1	Updated UCxSTE description.

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

### Changes From Revision J (December 2011) to Revision K (August 2012)

Location	Description
Throughout	Changed format of all register descriptions
Section 1.2.1	Added note "Some SRAM locations can be modified by the boot code..."
Section 1.4.2	Added to description of clock settings
Section 1.5	Added to description of entering low-power modes
Table 1-3	Added XIN, XOUT, XT2IN, XT2OUT; Added note "The default USB BSL evaluates...".
Section 1.13.4	Added section
Section 2.2.3	Added typical applications and available settings
Section 2.2.9	Added sections regarding control and performance modes
Figure 3-1	Updated figure
Section 3.2.3	Updated section

**Changes From Revision J (December 2011) to Revision K (August 2012) (continued)**

<b>Location</b>	<b>Description</b>
<a href="#">Section 5.2</a>	Added note regarding XT1 enable
<a href="#">Section 5.2.12</a> and following	Corrected DCOFFG to DCOFFG in several places
<a href="#">Table 5-1</a>	Updated description regarding LPM3.5 data retention
<a href="#">Section 5.4.2</a>	Updated description
<a href="#">Section 5.4.7</a>	Updated description regarding LPM3.5 data retention
<a href="#">Section 5.4.10</a>	Added UCSCTL9 register description
<a href="#">Chapter 8</a>	Added Memory Integrity Detection (MID) chapter
<a href="#">Table 10-1</a>	Added byte-access register information
<a href="#">Chapter 15</a>	Added AES Accelerator chapter
<a href="#">Section 16.3.1</a>	Updated WDTIS description
<a href="#">Section 17.2.1.1</a>	Changed description
<a href="#">Section 17.3.4</a>	Added description of TAxCRRn registers
<a href="#">Section 17.3.6</a>	Changed bit name from IDEX to TAIDEX
<a href="#">Section 18.2.1.2</a>	Changed description
<a href="#">Section 18.3.4</a>	Added TBxCCRn register description
<a href="#">Section 19.3.9</a>	Corrected bit field values
<a href="#">Section 22.2.4</a>	Updated description; Added note regarding minimal calibration
<a href="#">Section 22.3.13, Section 22.3.26</a>	Corrected hour range (0 to 23)
<a href="#">Section 23.3.9, Section 23.3.20</a>	Corrected hour range (0 to 23)
<a href="#">Chapter 24</a>	Updated throughout with counter mode information
<a href="#">Section 25.2.4</a>	Updated description to clarify radix point rather than decimal point
<a href="#">Section 26.2.2.1, Section 26.2.3.2</a>	Removed references to ADC10REFBURST (not supported)
<a href="#">Figure 27-2</a>	Updated figure
<a href="#">Section 27.2.5.3</a>	Added "The minimum sample time must not be violated..."
<a href="#">Section 28.2.7.2</a>	Added "The ADC12SC must be cleared by software after each sequence..."
<a href="#">Section 28.3.6</a>	Fixed typo
<a href="#">Table 31-1</a>	Changed CBIV reset
<a href="#">Section 33.3.5</a>	Corrected reset values
<a href="#">Section 34.4, Section 35.5, Section 37.4, Section 38.4, Section 38.5, Section 39.4</a>	Marked bits that are "Modify only when UCSWRST = 1"

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

**Changes From Revision I (September 2011) to Revision J (December 2011)**

<b>Location</b>	<b>Description</b>
	Added the following chapters: <a href="#">Chapter 4 Auxiliary Supply System (AUX)</a> <a href="#">Chapter 24 Real-Time Clock C (RTC_C)</a> <a href="#">Chapter 29 SD24_B</a> <a href="#">Chapter 33 LCD_C Controller</a> <a href="#">Chapter 37 Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode</a> <a href="#">Chapter 38 Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode</a> <a href="#">Chapter 39 Enhanced Universal Serial Communication Interface (eUSCI) – I<sup>2</sup>C Mode</a>
<a href="#">Section 3.2.1</a>	Added description of LOCKBAK reset condition

**Changes From Revision I (September 2011) to Revision J (December 2011) (continued)**

Location	Description
Section 28.2.8	Removed "Selecting the temperature sensor automatically turns on the on-chip reference..."
Section 21.1	Added RTC_C information
Section 28.3.6	Fixed typo in ADC12_A Interrupt Enable Register (ADC12IE)

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

**Changes From Revision H (December 2010) to Revision I (September 2011)**

Location	Description
Section 1.2.1	Added note that a blank device enters LPM4 on system reset.
Section 1.3.4.1	Changed step 7 of interrupt acceptance
Section 1.4.2	Changed code example to show LPMx.5 naming
Section 1.9	Changed boot code description
Section 1.10	Added comment that USB devices use a USB-based BSL, not the standard UART BSL
Section 1.11.2	Add note regarding customer return devices if the device JTAG is protected
Section 1.15.8	SYSUNIV register, Added BUSIFG to register description
Section 1.14.2	SFRIFG1 register, Changed WDTIFG description
Section 2.2.6	Added section
Section 2.2.9	Corrected typos in tables
Section 2.3.1	PMMCTL0 register, Changed PMM password description
Section 5.2.12 and following	Corrected DCOFFG to DCOFG in several places
Section 5.2.5	Corrected typo in note: "XT2OFF = 0. TX1 enabled..." – changed to XT2
Section 5.4.2	UCSCTL1 register, Updated description
Section 5.4.10	UCSCTL9 register, Added register
Section 6.5.1	Changed Tables 5-4, 5-5, 5-13, and 5-14 status flags
Section 6.6.2.9	Changed instruction flow description. Changed offset to ±32KB
Section 6.6.2.13	Changed description offset to ±32KB
Section 6.6.2.17	Changed code examples
Section 6.6.4.5	Changed code examples
Section 7.4.2	FCTL3 register, Removed reserved bit 5 description
Section 9.3.1	RCCTL0 register, Changed descriptions to be more generic on number of sectors available
Section 12.3	Added comment about clearing pending port interrupt flags prior to LPMx.5 entry
Section 17.3.6	TA0EX0 register, Changed bit name from IDEX to TAIDEX
Section 18.3.6	TBxEX0 register, Changed bit name from IDEX to TBIDEX
Section 19.1	Added Timer_D chapter
Section 20.1	Added TEC chapter
Section 22.2.4	Updated description. Added note regarding minimal calibration.
Section 23.2.5	Updated description. Added note regarding minimal calibration.
Section 23.2.6	Added requirement that LOCKLPM5 must be cleared if set already to re-enter LPM5 mode.
Section 23.2.6	Corrected LPMx.5 entry code example
Section 27.2.3	The internal AVCC can also be used as the reference.
Section 27.3.3	ADC10CTL2 register, Removed ADC10REFBURST bit (changed to Reserved)
Section 28.2.3	The internal AVCC can also be used as the reference.
Section 30.3	Changed column names for Table 30-2 to Pm.x and Pn.y to include the possibility that the outputs could be on different ports (m does not have to equal n)
Table 30-3	Changed table names for ports and register description to clarify that selection is for each DAC channel
Section 40.1	Updated Figure 1-1 to remove PHY power from the 1.8-V regulator output
Section 40.2	Added note that USB active operation is possible only in AM through LPM1

**Changes From Revision H (December 2010) to Revision I (September 2011) (continued)**

Location	Description
Section 40.3.1.1	Added note regarding priority inversion of EP0 and SETUPIFG
Section 40.4.1.4	USBPWRCTL register, Changed SLDOEN description
Section 40.4.2.15	USBIFG register, Added SETUPIFG and STPOWIFG flags
Section 41.2.6	Corrected names for interrupt enables and flags

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.

<b>Products</b>	<b>Applications</b>		
Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>	Automotive and Transportation	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>	Communications and Telecom	<a href="http://www.ti.com/communications">www.ti.com/communications</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>	Computers and Peripherals	<a href="http://www.ti.com/computers">www.ti.com/computers</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>	Consumer Electronics	<a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>	Energy and Lighting	<a href="http://www.ti.com/energy">www.ti.com/energy</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>	Industrial	<a href="http://www.ti.com/industrial">www.ti.com/industrial</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>	Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>	Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>	Space, Avionics and Defense	<a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>	Video and Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>	<b>TI E2E Community</b>	
OMAP Applications Processors	<a href="http://www.ti.com/omap">www.ti.com/omap</a>	<a href="http://e2e.ti.com">e2e.ti.com</a>	
Wireless Connectivity	<a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a>		