

# ***TMS320F2802x, TMS320F2802xx Microcontrollers***

*Technical Reference Manual*



Literature Number: SPRUI09A  
DECEMBER 2018 – REVISED JUNE 2022



<b>Read This First</b> .....	21
About This Manual.....	21
Notational Conventions.....	21
Glossary.....	21
Related Documentation From Texas Instruments.....	21
Support Resources.....	21
Trademarks.....	21
<b>1 System Control and Interrupts</b> .....	23
1.1 Flash and OTP Memory.....	24
1.1.1 Flash Memory.....	24
1.1.2 OTP Memory.....	24
1.1.3 Flash and OTP Power Modes.....	25
1.1.4 Flash and OTP Registers.....	30
1.2 Code Security Module (CSM).....	36
1.2.1 Functional Description.....	36
1.2.2 CSM Impact on Other On-Chip Resources.....	39
1.2.3 Incorporating Code Security in User Applications.....	40
1.2.4 Do's and Don'ts to Protect Security Logic.....	44
1.2.5 CSM Features - Summary.....	44
1.2.6 CSM Status and Control Registers.....	45
1.3 Clocking.....	46
1.3.1 Clocking and System Control.....	46
1.3.2 OSC and PLL Block.....	51
1.3.3 Low-Power Modes Block.....	72
1.3.4 CPU Watchdog Block.....	75
1.3.5 32-Bit CPU Timers 0/1/2.....	81
1.4 General-Purpose Input/Output (GPIO).....	86
1.4.1 GPIO Module Overview.....	86
1.4.2 Configuration Overview.....	92
1.4.3 Digital General-Purpose I/O Control.....	94
1.4.4 Input Qualification.....	95
1.4.5 GPIO and Peripheral Multiplexing (MUX).....	99
1.4.6 GPIO Registers.....	103
1.5 Peripheral Frames.....	126
1.5.1 Peripheral Frame Registers.....	126
1.5.2 EALLOW-Protected Registers.....	128
1.5.3 Device Emulation Registers.....	132
1.5.4 Write-Followed-by-Read Protection.....	135
1.6 Peripheral Interrupt Expansion (PIE).....	136
1.6.1 Overview of the PIE Controller.....	136
1.6.2 Vector Table Mapping.....	139
1.6.3 Interrupt Sources.....	141
1.6.4 PIE Configuration Registers.....	150
1.6.5 External Interrupt Configuration Registers.....	161
1.7 VREG/BOR/POR.....	163
1.7.1 On-Chip Voltage Regulator (VREG).....	163
1.7.2 On-chip Power-On Reset (POR) and Brown-Out Reset (BOR) Circuit.....	164
<b>2 Boot ROM</b> .....	165
2.1 Boot ROM Memory Map.....	166
2.1.1 On-Chip Boot ROM IQmath Tables.....	167

2.1.2 On-Chip Boot ROM IQmath Functions.....	168
2.1.3 On-Chip Flash API.....	168
2.1.4 CPU Vector Table.....	169
2.2 Bootloader Features.....	170
2.2.1 Bootloader Functional Operation.....	170
2.2.2 Bootloader Device Configuration.....	172
2.2.3 PLL Multiplier and DIVSEL Selection.....	172
2.2.4 Watchdog Module.....	173
2.2.5 Taking an ITRAP Interrupt.....	173
2.2.6 Internal Pullup Circuit.....	173
2.2.7 PIE Configuration.....	173
2.2.8 Reserved Memory.....	173
2.2.9 Bootloader Modes.....	174
2.2.10 Device_Cal.....	180
2.2.11 Bootloader Data Stream Structure.....	180
2.2.12 Basic Transfer Procedure.....	183
2.2.13 InitBoot Assembly Routine.....	184
2.2.14 SelectBootMode Function.....	184
2.2.15 CopyData Function.....	187
2.2.16 SCI_Boot Function.....	187
2.2.17 Parallel_Boot Function (GPIO).....	189
2.2.18 SPI_Boot Function.....	194
2.2.19 I2C Boot Function.....	197
2.2.20 ExitBoot Assembly Routine.....	200
2.3 Building the Boot Table.....	201
2.3.1 The C2000 Hex Utility.....	201
2.3.2 Example: Preparing a COFF File for SCI Bootloading.....	202
2.4 Bootloader Code Overview.....	205
2.4.1 Boot ROM Version and Checksum Information.....	205
2.4.2 Bootloader Code Revision History.....	205
<b>3 Enhanced Pulse Width Modulator (ePWM) Module.....</b>	<b>207</b>
3.1 Introduction.....	208
3.1.1 EPWM Related Collateral.....	208
3.1.2 Submodule Overview.....	208
3.1.3 Register Mapping.....	213
3.2 ePWM Submodules.....	215
3.2.1 Overview.....	215
3.2.2 Time-Base (TB) Submodule.....	217
3.2.3 Counter-Compare (CC) Submodule.....	226
3.2.4 Action-Qualifier (AQ) Submodule.....	231
3.2.5 Dead-Band Generator (DB) Submodule.....	244
3.2.6 PWM-Chopper (PC) Submodule.....	248
3.2.7 Trip-Zone (TZ) Submodule.....	252
3.2.8 Event-Trigger (ET) Submodule.....	258
3.2.9 Digital Compare (DC) Submodule.....	263
3.3 Applications to Power Topologies.....	268
3.3.1 Overview of Multiple Modules.....	268
3.3.2 Key Configuration Capabilities.....	269
3.3.3 Controlling Multiple Buck Converters With Independent Frequencies.....	270
3.3.4 Controlling Multiple Buck Converters With Same Frequencies.....	273
3.3.5 Controlling Multiple Half H-Bridge (HHB) Converters.....	276
3.3.6 Controlling Dual 3-Phase Inverters for Motors (ACI and PMSM).....	278
3.3.7 Practical Applications Using Phase Control Between PWM Modules.....	282
3.3.8 Controlling a 3-Phase Interleaved DC/DC Converter.....	284
3.3.9 Controlling Zero Voltage Switched Full Bridge (ZVSFB) Converter.....	287
3.3.10 Controlling a Peak Current Mode Controlled Buck Module.....	290
3.3.11 Controlling H-Bridge LLC Resonant Converter.....	292
3.4 Registers.....	294
3.4.1 Time-Base Submodule Registers.....	294
3.4.2 Counter-Compare Submodule Registers.....	302
3.4.3 Action-Qualifier Submodule Registers.....	308



3.4.4 Dead-Band Submodule Registers.....	313
3.4.5 Trip-Zone Submodule Control and Status Registers.....	316
3.4.6 Event-Trigger Submodule Registers.....	325
3.4.7 PWM-Chopper Submodule Control Register.....	331
3.4.8 Digital Compare Submodule Registers.....	333
3.4.9 Proper Interrupt Initialization Procedure.....	341
<b>4 High-Resolution Pulse Width Modulator (HRPWM).....</b>	<b>343</b>
4.1 Introduction.....	344
4.2 Operational Description of HRPWM.....	345
4.2.1 Controlling the HRPWM Capabilities.....	346
4.2.2 Configuring the HRPWM.....	349
4.2.3 Principle of Operation.....	350
4.2.4 Scale Factor Optimizing Software (SFO).....	358
4.2.5 HRPWM Examples Using Optimized Assembly Code.....	358
4.3 SFO Library Software - SFO_TI_Build_V6.lib.....	363
4.3.1 Scale Factor Optimizer Function - int SFO().....	363
4.3.2 Software Usage.....	364
4.3.3 SFO Library Version Software Differences.....	366
4.4 HRPWM Registers.....	367
4.4.1 HRPWM Configuration (HRCNFG) Register.....	367
4.4.2 High Resolution Power (HRPWR) Register.....	369
4.4.3 High Resolution Micro Step (HRMSTEP) Register.....	369
4.4.4 High Resolution Period Control (HRPCTL) Register.....	370
<b>5 Enhanced Capture (eCAP).....</b>	<b>371</b>
5.1 Introduction.....	372
5.1.1 Features.....	372
5.1.2 ECAP Related Collateral.....	372
5.2 Description.....	372
5.3 Capture and APWM Operating Mode.....	373
5.4 Capture Mode Description.....	375
5.4.1 Event Prescaler.....	376
5.4.2 Edge Polarity Select and Qualifier.....	377
5.4.3 Continuous/One-Shot Control.....	377
5.4.4 32-Bit Counter and Phase Control.....	378
5.4.5 CAP1-CAP4 Registers.....	378
5.4.6 Interrupt Control.....	379
5.4.7 Shadow Load and Lockout Control.....	381
5.4.8 APWM Mode Operation.....	381
5.5 Application of the eCAP Module.....	383
5.5.1 Example 1 - Absolute Time-Stamp Operation Rising Edge Trigger.....	383
5.5.2 Example 2 - Absolute Time-Stamp Operation Rising and Falling Edge Trigger.....	384
5.5.3 Example 3 - Time Difference (Delta) Operation Rising Edge Trigger.....	385
5.5.4 Example 4 - Time Difference (Delta) Operation Rising and Falling Edge Trigger.....	386
5.6 Application of the APWM Mode.....	387
5.6.1 Example 1 - Simple PWM Generation (Independent Channel/s).....	387
5.7 eCAP Registers.....	387
5.7.1 eCAP Base Addresses.....	387
5.7.2 ECAP_REGS Registers.....	388
<b>6 Analog-to-Digital Converter (ADC).....</b>	<b>399</b>
6.1 Introduction.....	400
6.1.1 Features.....	400
6.1.2 ADC Related Collateral.....	400
6.1.3 Block Diagram.....	402
6.2 SOC Principle of Operation.....	403
6.2.1 ADC Acquisition (Sample and Hold) Window.....	404
6.2.2 Trigger Operation.....	409
6.2.3 Channel Selection.....	409
6.3 ONESHOT Single Conversion Support.....	409
6.4 ADC Conversion Priority.....	410
6.5 Sequential Sampling Mode.....	413
6.6 Simultaneous Sampling Mode.....	413

6.7 EOC and Interrupt Operation.....	414
6.8 Power-Up Sequence.....	415
6.9 ADC Calibration.....	415
6.9.1 Factory Settings and Calibration Function.....	415
6.9.2 ADC Zero Offset Calibration.....	415
6.9.3 ADC Full Scale Gain Calibration.....	416
6.9.4 ADC Bias Current Calibration.....	416
6.10 Internal/External Reference Voltage Selection.....	416
6.10.1 Internal Reference Voltage.....	416
6.10.2 External Reference Voltage.....	417
6.11 ADC Timings.....	417
6.12 Internal Temperature Sensor.....	422
6.12.1 Transfer Function.....	422
6.13 ADC Registers.....	424
6.13.1 ADC Control Register 1 (ADCCTL1).....	425
6.13.2 ADC Control Register 2 (ADCCTL2).....	427
6.13.3 ADC Interrupt Registers.....	428
6.13.4 ADC Start of Conversion Priority Control Register (SOCPRCTL).....	435
6.13.5 ADC SOC Registers.....	437
6.13.6 ADC Calibration Registers.....	446
6.13.7 Comparator Hysteresis Control Register (COMPHYSTCTL).....	447
6.13.8 ADC Revision Register (ADCREV).....	448
6.13.9 ADC RESULT0-RESULT15 Registers (ADCRESULTx).....	448
<b>7 Comparator (COMP)</b> .....	449
7.1 Introduction.....	450
7.1.1 Features.....	450
7.1.2 Block Diagram.....	450
7.2 Comparator Function.....	451
7.3 DAC Reference.....	451
7.4 Ramp Generator Input.....	452
7.5 Initialization.....	453
7.6 Digital Domain Manipulation.....	453
7.7 Comparator Registers.....	454
7.7.1 Comparator Control (COMPCTL) Register.....	455
7.7.2 Compare Output Status (COMPSTS) Register.....	456
7.7.3 DAC Control (DACCTL) Register.....	456
7.7.4 DAC Value (DACVAL) Register.....	457
7.7.5 Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register.....	457
7.7.6 Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register.....	457
7.7.7 Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register.....	458
7.7.8 Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register.....	458
7.7.9 Ramp Generator Status (RAMPSTS) Register.....	458
<b>8 Serial Peripheral Interface (SPI)</b> .....	459
8.1 Introduction.....	460
8.1.1 Features.....	460
8.1.2 Block Diagram.....	461
8.2 System-Level Integration.....	462
8.2.1 SPI Module Signals.....	462
8.2.2 Configuring Device Pins.....	462
8.2.3 SPI Interrupts.....	463
8.3 SPI Operation.....	465
8.3.1 Introduction to Operation.....	465
8.3.2 Master Mode.....	467
8.3.3 Slave Mode.....	467
8.3.4 Data Format.....	468
8.3.5 Baud Rate Selection.....	469
8.3.6 SPI Clocking Schemes.....	470
8.3.7 SPI FIFO Description.....	471
8.3.8 SPI 3-Wire Mode Description.....	472
8.4 Programming Procedure.....	474
8.4.1 Initialization Upon Reset.....	474

8.4.2 Configuring the SPI.....	474
8.4.3 Data Transfer Example.....	475
8.4.4 SPI 3-Wire Mode Code Examples.....	476
8.5 SPI Registers.....	477
8.5.1 SPI Base Addresses.....	477
8.5.2 SPI_REGS Registers.....	477
<b>9 Serial Communications Interface (SCI)</b> .....	<b>497</b>
9.1 Introduction.....	498
9.1.1 Features.....	498
9.1.2 SCI Related Collateral.....	499
9.1.3 Block Diagram.....	499
9.2 Architecture.....	499
9.3 SCI Module Signal Summary.....	499
9.4 Configuring Device Pins.....	500
9.5 Multiprocessor and Asynchronous Communication Modes.....	501
9.6 SCI Programmable Data Format.....	501
9.7 SCI Multiprocessor Communication.....	502
9.7.1 Recognizing the Address Byte.....	502
9.7.2 Controlling the SCI TX and RX Features.....	502
9.7.3 Receipt Sequence.....	502
9.8 Idle-Line Multiprocessor Mode.....	503
9.8.1 Idle-Line Mode Steps.....	503
9.8.2 Block Start Signal.....	503
9.8.3 Wake-UP Temporary (WUT) Flag.....	504
9.8.4 Receiver Operation.....	504
9.9 Address-Bit Multiprocessor Mode.....	505
9.9.1 Sending an Address.....	505
9.10 SCI Communication Format.....	506
9.10.1 Receiver Signals in Communication Modes.....	506
9.10.2 Transmitter Signals in Communication Modes.....	507
9.11 SCI Port Interrupts.....	508
9.12 SCI Baud Rate Calculations.....	508
9.13 SCI Enhanced Features.....	509
9.13.1 SCI FIFO Description.....	509
9.13.2 SCI Auto-Baud.....	511
9.13.3 Autobaud-Detect Sequence.....	511
9.14 SCI Registers.....	512
9.14.1 SCI Base Addresses.....	512
9.14.2 SCI_REGS Registers.....	512
<b>10 Inter-Integrated Circuit Module (I2C)</b> .....	<b>531</b>
10.1 Introduction.....	532
10.1.1 I2C Related Collateral.....	532
10.1.2 Features.....	533
10.1.3 Features Not Supported.....	533
10.1.4 Functional Overview.....	534
10.1.5 Clock Generation.....	535
10.1.6 I2C Clock Divider Registers (I2CCLKL and I2CCLKH).....	536
10.2 Configuring Device Pins.....	537
10.3 I2C Module Operational Details.....	537
10.3.1 Input and Output Voltage Levels.....	537
10.3.2 Data Validity.....	537
10.3.3 Operating Modes.....	537
10.3.4 I2C Module START and STOP Conditions.....	541
10.3.5 Non-repeat Mode versus Repeat Mode.....	542
10.3.6 Serial Data Formats.....	543
10.3.7 Clock Synchronization.....	545
10.3.8 Arbitration.....	546
10.3.9 Digital Loopback Mode.....	547
10.3.10 NACK Bit Generation.....	548
10.4 Interrupt Requests Generated by the I2C Module.....	549
10.4.1 Basic I2C Interrupt Requests.....	549

10.4.2 I2C FIFO Interrupts.....	551
10.5 Resetting or Disabling the I2C Module.....	551
10.6 I2C Registers.....	552
10.6.1 I2C Base Address Table (C28).....	552
10.6.2 I2C_REGS Registers.....	552
<b>11 Revision History.....</b>	<b>575</b>

## List of Figures

Figure 1-1. Flash Power Mode State Diagram.....	26
Figure 1-2. Flash Pipeline.....	28
Figure 1-3. Flash Configuration Access Flow Diagram.....	29
Figure 1-4. Flash Options Register (FOPT).....	31
Figure 1-5. Flash Power Register (FPWR).....	31
Figure 1-6. Flash Status Register (FSTATUS).....	32
Figure 1-7. Flash Standby Wait Register (FSTDBYWAIT).....	33
Figure 1-8. Flash Standby to Active Wait Counter Register (FACTIVEWAIT).....	33
Figure 1-9. Flash Wait-State Register (FBANKWAIT).....	34
Figure 1-10. OTP Wait-State Register (FOTPWAIT).....	35
Figure 1-11. Password Match Flow (PMF).....	41
Figure 1-12. CSM Status and Control Register (CSMSCR).....	45
Figure 1-13. Clock and Reset Domains.....	46
Figure 1-14. Peripheral Clock Control 0 Register (PCLKCR0).....	47
Figure 1-15. Peripheral Clock Control 1 Register (PCLKCR1).....	49
Figure 1-16. Peripheral Clock Control 3 Register (PCLKCR3).....	50
Figure 1-17. Low-Speed Peripheral Clock Prescaler Register (LOSPCP).....	51
Figure 1-18. Clocking Options.....	52
Figure 1-19. Internal Oscillator n Trim (INTOSCnTRIM) Register.....	53
Figure 1-20. Clocking (XCLK) Register.....	54
Figure 1-21. Clock Control (CLKCTL) Register.....	55
Figure 1-22. OSC and PLL Block.....	58
Figure 1-23. PLLCR Change Procedure Flow Chart.....	59
Figure 1-24. PLL Control Register (PLLCR).....	60
Figure 1-25. PLL Status (PLLSTS) Register.....	61
Figure 1-26. PLL Lock Period (PLLLOCKPRD) Register.....	63
Figure 1-27. Clocking and Reset Logic.....	64
Figure 1-28. Clock Fail Interrupt.....	67
Figure 1-29. NMI Configuration (NMICFG) Register Bit Definitions (EALLOW).....	68
Figure 1-30. NMI Flag (NMIFLG) Register Bit Definitions (EALLOW Protected).....	69
Figure 1-31. NMI Flag Clear (NMIFLGCLR) Register Bit Definitions (EALLOW Protected).....	69
Figure 1-32. NMI Flag Force (NMIFLGFRC) Register Bit Definitions (EALLOW Protected).....	70
Figure 1-33. NMI Watchdog Counter (NMIWDCNT) Register Bit Definitions.....	70
Figure 1-34. NMI Watchdog Period (NMIWDPRD) Register Bit Definitions (EALLOW Protected).....	71
Figure 1-35. XCLKOUT Generation.....	72
Figure 1-36. Low Power Mode Control 0 Register (LPMCR0).....	74
Figure 1-37. CPU Watchdog Module.....	75
Figure 1-38. System Control and Status Register (SCSR).....	78
Figure 1-39. Watchdog Counter Register (WDCNTR).....	79
Figure 1-40. Watchdog Reset Key Register (WDKEY).....	79
Figure 1-41. Watchdog Control Register (WDCR).....	80
Figure 1-42. CPU-Timers.....	81
Figure 1-43. CPU-Timer Interrupts Signals and Output Signal.....	81
Figure 1-44. CPU-Timer x Counter (TIMERxTIM) Register (x = 0, 1, 2).....	82
Figure 1-45. CPU-Timer x Counter (TIMERxTIMH) Register High (x = 0, 1, 2).....	83
Figure 1-46. CPU-Timer x Period (TIMERxPRD) Register (x = 0, 1, 2).....	83
Figure 1-47. CPU-Timer x Period (TIMERxPRDH) Register High (x = 0, 1, 2).....	83
Figure 1-48. CPU-Timer x Control (TIMERxTCR) Register (x = 0, 1, 2).....	84
Figure 1-49. CPU-Timer x Prescale (TIMERxTPR) Register (x = 0, 1, 2).....	85
Figure 1-50. CPU-Timer x Prescale (TIMERxTPRH) Register High (x = 0, 1, 2).....	86
Figure 1-51. GPIO0 to GPIO31 Multiplexing Diagram.....	87
Figure 1-52. GPIO32, GPIO33 Multiplexing Diagram.....	88
Figure 1-53. JTAG Port/GPIO Multiplexing.....	89

Figure 1-54. JTAGDEBUG Register (Address 0x702A, EALLOW protected).....	90
Figure 1-55. Analog/GPIO Multiplexing.....	91
Figure 1-56. Input Qualification Using a Sampling Window.....	96
Figure 1-57. Input Qualifier Clock Cycles.....	98
Figure 1-58. GPIO Port A Qualification Control (GPACTRL) Register.....	103
Figure 1-59. GPIO Port A Qualification Select 1 (GPAQSEL1) Register.....	104
Figure 1-60. GPIO Port A Qualification Select 2 (GPAQSEL2) Register.....	105
Figure 1-61. GPIO Port A MUX 1 (GPAMUX1) Register.....	105
Figure 1-62. GPIO Port A MUX 2 (GPAMUX2) Register.....	107
Figure 1-63. GPIO Port A Direction (GPADIR) Register.....	109
Figure 1-64. GPIO Port A Pullup Disable (GPAPUD) Register.....	110
Figure 1-65. GPIO Port B Qualification Control (GPBCTRL) Register.....	111
Figure 1-66. GPIO Port B Qualification Select 1 (GPBQSEL1) Register.....	112
Figure 1-67. GPIO Port B MUX 1 (GPBMUX1) Register.....	112
Figure 1-68. GPIO Port B Direction (GPBDIR) Register.....	114
Figure 1-69. GPIO Port B Pullup Disable (GPBPUD) Register.....	115
Figure 1-70. Analog I/O MUX (AIOMUX1) Register.....	116
Figure 1-71. Analog I/O Direction (AIODIR) Register.....	117
Figure 1-72. GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register.....	118
Figure 1-73. GPIO Low Power Mode Wakeup Select (GPIO_LPMSEL) Register.....	119
Figure 1-74. GPIO Port A Data (GPADAT) Register.....	120
Figure 1-75. GPIO Port A Set, Clear, and Toggle (GPASET, GPACLEAR, GPATOGGLE) Registers.....	121
Figure 1-76. GPIO Port B Data (GPBDAT) Register.....	122
Figure 1-77. GPIO Port B Set, Clear, and Toggle (GPBSET, GPBCLEAR, GPBTOGGLE) Registers.....	123
Figure 1-78. Analog I/O Data (AIODAT) Register.....	124
Figure 1-79. Analog I/O Set, Clear, and Toggle (AIOSET, AIOCLEAR, AIOTOGGLE) Registers.....	125
Figure 1-80. Device Configuration (DEVICECNF) Register.....	132
Figure 1-81. Part ID (PARTID) Register.....	133
Figure 1-82. Class ID (CLASSID) Register.....	134
Figure 1-83. Revision ID (REVID) Register.....	134
Figure 1-84. Overview: Multiplexing of Interrupts Using the PIE Block.....	136
Figure 1-85. Typical PIE/CPU Interrupt Response - INTx.y.....	138
Figure 1-86. Reset Flow Diagram.....	140
Figure 1-87. PIE Interrupt Sources and External Interrupts XINT1/XINT2/XINT3.....	141
Figure 1-88. Multiplexed Interrupt Request Flow Diagram.....	144
Figure 1-89. PIE Control (PIECTRL) Register.....	151
Figure 1-90. PIE Interrupt Acknowledge (PIEACK) Register.....	151
Figure 1-91. PIE Interrupt Enable (PIEIERn) Registers (n = 1 to 12).....	152
Figure 1-92. PIE Interrupt Flag (PIEIFRn) Registers (n = 1 to 12).....	153
Figure 1-93. Interrupt Flag Register (IFR) — CPU Register.....	154
Figure 1-94. Interrupt Enable Register (IER) — CPU Register.....	157
Figure 1-95. Debug Interrupt Enable Register (DBGIER) — CPU Register.....	159
Figure 1-96. External Interrupt Control Registers (XINTnCR) (n = 1 to 3).....	161
Figure 1-97. External Interrupt n Counter Registers (XINTnCTR).....	162
Figure 1-98. BOR Configuration (BORCFG) Register.....	164
Figure 2-1. Memory Map of On-Chip ROM.....	166
Figure 2-2. Vector Table Map.....	169
Figure 2-3. Bootloader Flow Diagram.....	171
Figure 2-4. Boot ROM Stack.....	173
Figure 2-5. Boot ROM Function Overview.....	175
Figure 2-6. Bootloader Basic Transfer Procedure.....	183
Figure 2-7. Overview of InitBoot Assembly Function.....	184
Figure 2-8. Overview of the SelectBootMode Function.....	185
Figure 2-9. Overview of Get_mode() Function.....	186
Figure 2-10. Overview of CopyData Function.....	187
Figure 2-11. Overview of SCI Bootloader Operation.....	187
Figure 2-12. Overview of SCI_Boot Function.....	188
Figure 2-13. Overview of SCI_GetWordData Function.....	189
Figure 2-14. Overview of Parallel GPIO Bootloader Operation.....	189
Figure 2-15. Parallel GPIO Boot Loader Handshake Protocol.....	190
Figure 2-16. Parallel GPIO Mode Overview.....	191



Figure 2-17. Parallel GPIO Mode - Host Transfer Flow.....	192
Figure 2-18. 8-Bit Parallel GetWord Function.....	193
Figure 2-19. SPI Loader.....	194
Figure 2-20. Data Transfer From EEPROM Flow.....	196
Figure 2-21. Overview of SPIA_GetWordData Function.....	196
Figure 2-22. EEPROM Device at Address 0x50.....	197
Figure 2-23. Overview of I2C_Boot Function.....	198
Figure 2-24. Random Read.....	199
Figure 2-25. Sequential Read.....	199
Figure 2-26. ExitBoot Procedure Flow.....	200
Figure 3-1. Multiple ePWM Modules.....	210
Figure 3-2. Submodules and Signal Connections for an ePWM Module.....	211
Figure 3-3. ePWM Submodules and Critical Internal Signal Interconnects.....	212
Figure 3-4. Time-Base Submodule Block Diagram.....	217
Figure 3-5. Time-Base Submodule Signals and Registers.....	218
Figure 3-6. Time-Base Frequency and Period.....	220
Figure 3-7. Time-Base Counter Synchronization Scheme 1.....	221
Figure 3-8. Time-Base Up-Count Mode Waveforms.....	223
Figure 3-9. Time-Base Down-Count Mode Waveforms.....	224
Figure 3-10. Time-Base Up-Down-Count Waveforms, TBCTL[PHSDIR = 0] Count Down On Synchronization Event.....	224
Figure 3-11. Time-Base Up-Down Count Waveforms, TBCTL[PHSDIR = 1] Count Up On Synchronization Event.....	225
Figure 3-12. Counter-Compare Submodule.....	226
Figure 3-13. Detailed View of the Counter-Compare Submodule.....	226
Figure 3-14. Counter-Compare Event Waveforms in Up-Count Mode.....	229
Figure 3-15. Counter-Compare Events in Down-Count Mode.....	229
Figure 3-16. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 0] Count Down On Synchronization Event.....	230
Figure 3-17. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 1] Count Up On Synchronization Event.....	230
Figure 3-18. Action-Qualifier Submodule.....	231
Figure 3-19. Action-Qualifier Submodule Inputs and Outputs.....	232
Figure 3-20. Possible Action-Qualifier Actions for EPWMxA and EPWMxB Outputs.....	233
Figure 3-21. Up-Down-Count Mode Symmetrical Waveform.....	236
Figure 3-22. Up, Single Edge Asymmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB—Active High.....	237
Figure 3-23. Up, Single Edge Asymmetric Waveform With Independent Modulation on EPWMxA and EPWMxB—Active Low.....	238
Figure 3-24. Up-Count, Pulse Placement Asymmetric Waveform With Independent Modulation on EPWMxA.....	240
Figure 3-25. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Active Low.....	241
Figure 3-26. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Complementary.....	242
Figure 3-27. Up-Down-Count, Dual Edge Asymmetric Waveform, With Independent Modulation on EPWMxA—Active Low.....	243
Figure 3-28. Dead-Band Submodule.....	244
Figure 3-29. Configuration Options for the Dead-Band Submodule.....	245
Figure 3-30. Dead-Band Waveforms for Typical Cases (0% < Duty < 100%).....	247
Figure 3-31. PWM-Chopper Submodule.....	248
Figure 3-32. PWM-Chopper Submodule Operational Details.....	249
Figure 3-33. Simple PWM-Chopper Submodule Waveforms Showing Chopping Action Only.....	249
Figure 3-34. PWM-Chopper Submodule Waveforms Showing the First Pulse and Subsequent Sustaining Pulses.....	250
Figure 3-35. PWM-Chopper Submodule Waveforms Showing the Pulse Width (Duty Cycle) Control of Sustaining Pulses.....	251
Figure 3-36. Trip-Zone Submodule.....	252
Figure 3-37. Trip-Zone Submodule Mode Control Logic.....	256
Figure 3-38. Trip-Zone Submodule Interrupt Logic.....	257
Figure 3-39. Event-Trigger Submodule.....	258
Figure 3-40. Event-Trigger Submodule Inter-Connectivity of ADC Start of Conversion.....	259
Figure 3-41. Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs.....	259
Figure 3-42. Event-Trigger Interrupt Generator.....	261
Figure 3-43. Event-Trigger SOCA Pulse Generator.....	262
Figure 3-44. Event-Trigger SOCB Pulse Generator.....	262
Figure 3-45. Digital-Compare Submodule High-Level Block Diagram.....	263

Figure 3-46. DCAEVT1 Event Triggering.....	265
Figure 3-47. DCAEVT2 Event Triggering.....	265
Figure 3-48. DCBEVT1 Event Triggering.....	266
Figure 3-49. DCBEVT2 Event Triggering.....	266
Figure 3-50. Event Filtering.....	267
Figure 3-51. Blanking Window Timing Diagram.....	268
Figure 3-52. Simplified ePWM Module.....	268
Figure 3-53. EPWM1 Configured as a Typical Master, EPWM2 Configured as a Slave.....	269
Figure 3-54. Control of Four Buck Stages. Here $F_{P_{PWM1}} \neq F_{P_{PWM2}} \neq F_{P_{PWM3}} \neq F_{P_{PWM4}}$ .....	270
Figure 3-55. Buck Waveforms for Figure 3-54 (Note: Only three bucks shown here).....	271
Figure 3-56. Control of Four Buck Stages. (Note: $F_{P_{PWM2}} = N \times F_{P_{PWM1}}$ ).....	273
Figure 3-57. Buck Waveforms for Figure 3-56 (Note: $F_{P_{PWM2}} = F_{P_{PWM1}}$ ).....	274
Figure 3-58. Control of Two Half-H Bridge Stages ( $F_{P_{PWM2}} = N \times F_{P_{PWM1}}$ ).....	276
Figure 3-59. Half-H Bridge Waveforms for Figure 3-58 (Note: Here $F_{P_{PWM2}} = F_{P_{PWM1}}$ ).....	277
Figure 3-60. Control of Dual 3-Phase Inverter Stages as Is Commonly Used in Motor Control.....	279
Figure 3-61. 3-Phase Inverter Waveforms for Figure 3-60 (Only One Inverter Shown).....	280
Figure 3-62. Configuring Two PWM Modules for Phase Control.....	282
Figure 3-63. Timing Waveforms Associated With Phase Control Between 2 Modules.....	283
Figure 3-64. Control of a 3-Phase Interleaved DC/DC Converter.....	284
Figure 3-65. 3-Phase Interleaved DC/DC Converter Waveforms for Figure 3-64.....	285
Figure 3-66. Controlling a Full-H Bridge Stage ( $F_{P_{PWM2}} = F_{P_{PWM1}}$ ).....	287
Figure 3-67. ZVS Full-H Bridge Waveforms.....	288
Figure 3-68. Peak Current Mode Control of a Buck Converter.....	290
Figure 3-69. Peak Current Mode Control Waveforms for Figure 3-68.....	290
Figure 3-70. Control of Two Resonant Converter Stages.....	292
Figure 3-71. H-Bridge LLC Resonant Converter PWM Waveforms.....	292
Figure 3-72. Time-Base Control Register (TBCTL).....	294
Figure 3-73. Time-Base Status Register (TBSTS).....	296
Figure 3-74. Time-Base Phase High Resolution Register (TBPHSHR).....	297
Figure 3-75. Time-Base Phase Register (TBPHS).....	297
Figure 3-76. Time-Base Counter Register (TBCTR).....	298
Figure 3-77. Time-Base Period Register (TBPRD).....	298
Figure 3-78. Time Base Period High Resolution Register (TBPRDHR).....	299
Figure 3-79. Time-Base Period High Resolution Mirror Register (TBPRDHMR).....	300
Figure 3-80. Time-Base Period Mirror Register (TBPRDM).....	301
Figure 3-81. Counter-Compare Control (CMPCTL) Register.....	302
Figure 3-82. Compare A High Resolution (CMPAHR) Register.....	303
Figure 3-83. Counter-Compare A (CMPA) Register.....	304
Figure 3-84. Counter-Compare B (CMPB) Register.....	305
Figure 3-85. Compare A High-Resolution Mirror (CMPAHRM) Register.....	306
Figure 3-86. Counter-Compare A Mirror (CMPAM) Register.....	307
Figure 3-87. Action-Qualifier Output A Control Register (AQCTLA).....	308
Figure 3-88. Action-Qualifier Output B Control Register (AQCTLB).....	309
Figure 3-89. Action-Qualifier Software Force Register (AQSFRC).....	311
Figure 3-90. Action-Qualifier Continuous Software Force Register (AQCSFRC).....	312
Figure 3-91. Dead-Band Generator Control (DBCTL) Register.....	313
Figure 3-92. Dead-Band Generator Rising Edge Delay (DBRED) Register.....	315
Figure 3-93. Dead-Band Generator Falling Edge Delay (DBFED) Register.....	315
Figure 3-94. Trip-Zone Select Register (TZSEL).....	316
Figure 3-95. Trip Zone Digital Compare Event Select Register (TZDCSEL).....	317
Figure 3-96. Trip-Zone Control Register (TZCTL).....	319
Figure 3-97. Trip-Zone Enable Interrupt Register (TZEINT).....	320
Figure 3-98. Trip-Zone Flag Register (TZFLG).....	321
Figure 3-99. Trip-Zone Clear Register (TZCLR).....	323
Figure 3-100. Trip-Zone Force Register (TZFRC).....	324
Figure 3-101. Event-Trigger Selection Register (ETSEL).....	325
Figure 3-102. Event-Trigger Prescale Register (ETPS).....	326
Figure 3-103. Event-Trigger Flag Register (ETFLG).....	328
Figure 3-104. Event-Trigger Clear Register (ETCLR).....	329
Figure 3-105. Event-Trigger Force Register (ETFRC).....	330
Figure 3-106. PWM-Chopper Control (PCCTL) Register.....	331

Figure 3-107. Digital Compare Trip Select (DCTRIPSEL) Register.....	333
Figure 3-108. Digital Compare A Control (DCACTL) Register.....	335
Figure 3-109. Digital Compare B Control (DCBCTL) Register.....	336
Figure 3-110. Digital Compare Filter Control (DCFCTL) Register.....	337
Figure 3-111. Digital Compare Capture Control (DCCAPCTL) Register.....	338
Figure 3-112. Digital Compare Filter Offset (DCFOFFSET) Register.....	339
Figure 3-113. Digital Compare Filter Offset Counter (DCFOFFSETCNT) Register.....	339
Figure 3-114. Digital Compare Filter Window (DCFWINDOW) Register.....	340
Figure 3-115. Digital Compare Filter Window Counter (DCFWINDOWCNT) Register.....	340
Figure 3-116. Digital Compare Counter Capture (DCCAP) Register.....	341
Figure 4-1. Resolution Calculations for Conventionally Generated PWM.....	344
Figure 4-2. Operating Logic Using MEP.....	345
Figure 4-3. HRPWM Extension Registers and Memory Configuration.....	346
Figure 4-4. HRPWM System Interface.....	347
Figure 4-5. HRPWM Block Diagram.....	348
Figure 4-6. Required PWM Waveform for a Requested Duty = 30.0%.....	350
Figure 4-7. Low % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0).....	353
Figure 4-8. High % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0).....	354
Figure 4-9. Up-Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1).....	354
Figure 4-10. Up-Down Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1).....	355
Figure 4-11. Simple Buck Controlled Converter Using a Single PWM.....	359
Figure 4-12. PWM Waveform Generated for Simple Buck Controlled Converter.....	359
Figure 4-13. Simple Reconstruction Filter for a PWM Based DAC.....	361
Figure 4-14. PWM Waveform Generated for the PWM DAC Function.....	361
Figure 4-15. HRPWM Configuration (HRCNFG) Register.....	367
Figure 4-16. High Resolution Power (HRPWR) Register.....	369
Figure 4-17. High Resolution Micro Step (HRMSTEP) Register.....	369
Figure 4-18. High Resolution Period Control (HRPCTL) Register.....	370
Figure 5-1. Capture and APWM Modes of Operation.....	373
Figure 5-2. Counter Compare and PRD Effects on the eCAP Output in APWM Mode.....	374
Figure 5-3. eCAP Block Diagram.....	375
Figure 5-4. Event Prescale Control.....	376
Figure 5-5. Prescale Function Waveforms.....	376
Figure 5-6. Details of the Continuous/One-shot Block.....	377
Figure 5-7. Details of the Counter and Synchronization Block.....	378
Figure 5-8. Interrupts in eCAP Module.....	380
Figure 5-9. PWM Waveform Details Of APWM Mode Operation.....	381
Figure 5-10. Time-Base Frequency and Period Calculation.....	382
Figure 5-11. Capture Sequence for Absolute Time-stamp and Rising Edge Detect.....	383
Figure 5-12. Capture Sequence for Absolute Time-stamp With Rising and Falling Edge Detect.....	384
Figure 5-13. Capture Sequence for Delta Mode Time-stamp and Rising Edge Detect.....	385
Figure 5-14. Capture Sequence for Delta Mode Time-stamp With Rising and Falling Edge Detect.....	386
Figure 5-15. PWM Waveform Details of APWM Mode Operation.....	387
Figure 5-16. TSCTR Register.....	389
Figure 5-17. CTRPHS Register.....	389
Figure 5-18. CAP1 Register.....	389
Figure 5-19. CAP2 Register.....	390
Figure 5-20. CAP3 Register.....	390
Figure 5-21. CAP4 Register.....	390
Figure 5-22. ECCTL1 Register.....	391
Figure 5-23. ECCTL2 Register.....	393
Figure 5-24. ECEINT Register.....	395
Figure 5-25. ECFLG Register.....	396
Figure 5-26. ECCLR Register.....	397
Figure 5-27. ECFRC Register.....	398
Figure 6-1. ADC Block Diagram.....	402
Figure 6-2. SOC Block Diagram.....	403
Figure 6-3. ADCINx Input Model.....	405
Figure 6-4. ONESHOT Single Conversion.....	409
Figure 6-5. Round Robin Priority Example.....	411
Figure 6-6. High Priority Example.....	412



Figure 6-7. Interrupt Structure.....	414
Figure 6-8. Timing Example For Sequential Mode / Late Interrupt Pulse.....	417
Figure 6-9. Timing Example For Sequential Mode / Early Interrupt Pulse.....	418
Figure 6-10. Timing Example For Simultaneous Mode / Late Interrupt Pulse.....	419
Figure 6-11. Timing Example For Simultaneous Mode / Early Interrupt Pulse.....	420
Figure 6-12. Timing Example for NONOVERLAP Mode.....	421
Figure 6-13. Temperature Sensor Transfer Function.....	422
Figure 6-14. ADC Control Register 1 (ADCCTL1).....	425
Figure 6-15. ADC Control Register 2 (ADCCTL2).....	427
Figure 6-16. ADC Interrupt Flag Register (ADCINTFLG).....	428
Figure 6-17. ADC Interrupt Flag Clear Register (ADCINTFLGCLR).....	429
Figure 6-18. ADC Interrupt Overflow Register (ADCINTOVF).....	430
Figure 6-19. ADC Interrupt Overflow Clear Register (ADCINTOVFCLR).....	431
Figure 6-20. Interrupt Select 1 and 2 Register (INTSEL1N2).....	432
Figure 6-21. Interrupt Select 3 and 4 Register (INTSEL3N4).....	432
Figure 6-22. Interrupt Select 5 and 6 Register (INTSEL5N6).....	432
Figure 6-23. Interrupt Select 7 and 8 Register (INTSEL7N8).....	433
Figure 6-24. Interrupt Select 9 and 10 Register (INTSEL9N10).....	433
Figure 6-25. ADC Start of Conversion Priority Control Register (SOCPRCTL).....	435
Figure 6-26. ADC Sample Mode Register (ADCSAMPLEMODE).....	437
Figure 6-27. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1).....	439
Figure 6-28. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2).....	439
Figure 6-29. ADC SOC Flag 1 Register (ADCSOCFLG1).....	440
Figure 6-30. ADC SOC Force 1 Register (ADCSOCFRC1).....	441
Figure 6-31. ADC SOC Overflow 1 Register (ADCSOCOVF1).....	442
Figure 6-32. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1).....	442
Figure 6-33. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL).....	443
Figure 6-34. ADC Reference/Gain Trim Register (ADCREFTRIM).....	446
Figure 6-35. ADC Offset Trim Register (ADCOFFTRIM).....	446
Figure 6-36. Comparator Hysteresis Control Register (COMPHYSTCTL).....	447
Figure 6-37. ADC Revision Register (ADCREV).....	448
Figure 6-38. ADC RESULT0-RESULT15 Registers (ADCRESULTx).....	448
Figure 7-1. Comparator Block Diagram.....	450
Figure 7-2. Comparator.....	451
Figure 7-3. Ramp Generator Block Diagram.....	452
Figure 7-4. Ramp Generator Behavior.....	453
Figure 7-5. Comparator Control (COMPCTL) Register.....	455
Figure 7-6. Compare Output Status (COMPSTS) Register.....	456
Figure 7-7. DAC Control (DACCTL) Register.....	456
Figure 7-8. DAC Value (DACVAL) Register.....	457
Figure 7-9. Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register.....	457
Figure 7-10. Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register.....	457
Figure 7-11. Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register.....	458
Figure 7-12. Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register.....	458
Figure 7-13. Ramp Generator Status (RAMPSTS) Register.....	458
Figure 8-1. SPI CPU Interface.....	461
Figure 8-2. SPI Interrupt Flags and Enable Logic Generation.....	464
Figure 8-3. SPI Master/Slave Connection.....	465
Figure 8-4. Serial Peripheral Interface Block Diagram.....	466
Figure 8-5. SPICLK Signal Options.....	470
Figure 8-6. SPI: SPICLK-LSPCLK Characteristic When (BRR + 1) is Odd, BRR > 3, and CLKPOLARITY = 1.....	471
Figure 8-7. SPI 3-wire Master Mode.....	472
Figure 8-8. SPI 3-wire Slave Mode.....	473
Figure 8-9. Five Bits per Character.....	475
Figure 8-10. SPI Configuration Control Register (SPICCR).....	479
Figure 8-11. SPI Operation Control (SPICTL) Register.....	482
Figure 8-12. SPI Status (SPISTS) Register.....	484
Figure 8-13. SPI Baud Rate Register (SPIBRR).....	486
Figure 8-14. SPI Emulation Buffer (SPIRXEMU) Register.....	487
Figure 8-15. SPI Serial Input Buffer (SPIRXBUF) Register.....	488
Figure 8-16. SPI Serial Output Buffer (SPITXBUF) Register.....	488

Figure 8-17. SPI Serial Data (SPIDAT) Register.....	489
Figure 8-18. SPI FIFO Transmit (SPIFFTX) Register.....	490
Figure 8-19. SPI FIFO Receive (SPIFFRX) Register.....	492
Figure 8-20. SPI FIFO Control (SPIFFCT) Register.....	494
Figure 8-21. SPI Priority Control (SPIPRI) Register.....	495
Figure 9-1. SCI CPU Interface.....	498
Figure 9-2. Serial Communications Interface (SCI) Module Block Diagram.....	500
Figure 9-3. Typical SCI Data Frame Formats.....	501
Figure 9-4. Idle-Line Multiprocessor Communication Format.....	503
Figure 9-5. Double-Buffered WUT and TXSHF.....	504
Figure 9-6. Address-Bit Multiprocessor Communication Format.....	505
Figure 9-7. SCI Asynchronous Communications Format.....	506
Figure 9-8. SCI RX Signals in Communication Modes.....	506
Figure 9-9. SCI TX Signals in Communications Mode.....	507
Figure 9-10. SCI FIFO Interrupt Flags and Enable Logic.....	510
Figure 9-11. SCI Communications Control Register (SCICCR).....	514
Figure 9-12. SCI Control Register 1 (SCICTL1).....	516
Figure 9-13. SCI Baud Rate (high) (SCIHBAUD) Register.....	518
Figure 9-14. SCI Baud Rate (low) (SCILBAUD) Register.....	519
Figure 9-15. SCI Control Register 2 (SCICTL2).....	519
Figure 9-16. SCI Receive Status (SCIRXST) Register.....	521
Figure 9-17. SCI Receive Emulation Buffer (SCIRXEMU) Register.....	523
Figure 9-18. SCI Receive Data Buffer (SCIRXBUF) Register.....	524
Figure 9-19. SCI Transmit Data Buffer (SCITXBUF) Register.....	525
Figure 9-20. SCI FIFO Transmit (SCIFFTX) Register.....	525
Figure 9-21. SCI FIFO Receive (SCIFFRX) Register.....	527
Figure 9-22. SCI FIFO Control (SCIFFCT) Register.....	529
Figure 9-23. SCI Priority Control (SCIPRI) Register.....	530
Figure 10-1. Multiple I2C Modules Connected.....	532
Figure 10-2. I2C Module Conceptual Block Diagram.....	535
Figure 10-3. Clocking Diagram for the I2C Module.....	535
Figure 10-4. Roles of the Clock Divide-Down Values (ICCL and ICCH).....	536
Figure 10-5. Bit Transfer on the I2C bus.....	537
Figure 10-6. I2C Slave TX / RX Flowchart.....	539
Figure 10-7. I2C Master TX / RX Flowchart.....	540
Figure 10-8. I2C Module START and STOP Conditions.....	541
Figure 10-9. I2C Module Data Transfer (7-Bit Addressing with 8-bit Data Configuration Shown).....	543
Figure 10-10. I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMADR).....	543
Figure 10-11. I2C Module 10-Bit Addressing Format (FDF = 0, XA = 1 in I2CMADR).....	544
Figure 10-12. I2C Module Free Data Format (FDF = 1 in I2CMADR).....	544
Figure 10-13. Repeated START Condition (in This Case, 7-Bit Addressing Format).....	545
Figure 10-14. Synchronization of Two I2C Clock Generators During Arbitration.....	545
Figure 10-15. Arbitration Procedure Between Two Master-Transmitters.....	546
Figure 10-16. Pin Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit.....	547
Figure 10-17. Enable Paths of the I2C Interrupt Requests.....	550
Figure 10-18. I2C FIFO Interrupt.....	551
Figure 10-19. I2C Own Address Register (I2COAR).....	554
Figure 10-20. I2C Interrupt Enable Register (I2CIER).....	555
Figure 10-21. I2C Status Register (I2CSTR).....	556
Figure 10-22. I2C Clock Low-time Divider (I2CCLKL) Register.....	560
Figure 10-23. I2C Clock High-time Divider (I2CCLKH) Register.....	560
Figure 10-24. I2C Data Count (I2CCNT) Register.....	561
Figure 10-25. I2C Data Receive Register (I2CDRR).....	562
Figure 10-26. I2C Slave Address Register (I2CSAR).....	563
Figure 10-27. I2C Data Transmit Register (I2CDXR).....	564
Figure 10-28. I2C Mode Register (I2CMADR).....	565
Figure 10-29. I2C Interrupt Source (I2CISRC) Register.....	569
Figure 10-30. I2C Extended Mode Register (I2CEMDR).....	570
Figure 10-31. I2C Prescaler (I2CPSR) Register.....	571
Figure 10-32. I2C FIFO Transmit (I2CFFTX) Register.....	572
Figure 10-33. I2C FIFO Receive (I2CFFRX) Register.....	574

## List of Tables

Table 1-1. Flash/OTP Configuration Registers.....	30
Table 1-2. Flash Options Register (FOPT) Field Descriptions.....	31
Table 1-3. Flash Power Register (FPWR) Field Descriptions.....	31
Table 1-4. Flash Status Register (FSTATUS) Field Descriptions.....	32
Table 1-5. Flash Standby Wait Register (FSTDBYWAIT) Field Descriptions.....	33
Table 1-6. Flash Standby to Active Wait Counter Register (FACTIVEWAIT) Field Descriptions.....	33
Table 1-7. Flash Wait-State Register (FBANKWAIT) Field Descriptions.....	34
Table 1-8. OTP Wait-State Register (FOTPWAIT) Field Descriptions.....	35
Table 1-9. Security Levels.....	36
Table 1-10. Resources Affected by the CSM.....	39
Table 1-11. Resources Not Affected by the CSM.....	39
Table 1-12. Code Security Module (CSM) Registers.....	45
Table 1-13. CSM Status and Control Register (CSMSCR) Field Descriptions.....	45
Table 1-14. PLL, Clocking, Watchdog, and Low-Power Mode Registers.....	46
Table 1-15. Peripheral Clock Control 0 Register (PCLKCR0) Field Descriptions.....	47
Table 1-16. Peripheral Clock Control 1 Register (PCLKCR1) Field Descriptions.....	49
Table 1-17. Peripheral Clock Control 3 Register (PCLKCR3) Field Descriptions.....	50
Table 1-18. Low-Speed Peripheral Clock Prescaler Register (LOSPCP) Field Descriptions.....	51
Table 1-19. Internal Oscillator n Trim (INTOSCnTRIM) Register Field Descriptions.....	53
Table 1-20. Clocking (XCLK) Field Descriptions.....	54
Table 1-21. Clock Control (CLKCTL) Register Field Descriptions.....	55
Table 1-22. Possible PLL Configuration Modes.....	58
Table 1-23. PLL Settings.....	60
Table 1-24. PLL Status (PLLSTS) Register Field Descriptions.....	61
Table 1-25. PLL Lock Period (PLLLOCKPRD) Register Field Descriptions.....	63
Table 1-26. NMI Interrupt Registers.....	68
Table 1-27. NMI Configuration (NMICFG) Register Bit Definitions (EALLOW).....	68
Table 1-28. NMI Flag (NMIFLG) Register Bit Definitions (EALLOW Protected).....	69
Table 1-29. NMI Flag Clear (NMIFLGCLR) Register Bit Definitions (EALLOW Protected).....	69
Table 1-30. NMI Flag Force (NMIFLGFRC) Register Bit Definitions (EALLOW Protected).....	70
Table 1-31. NMI Watchdog Counter (NMIWDCNT) Register Bit Definitions.....	70
Table 1-32. NMI Watchdog Period (NMIWDPRD) Register Bit Definitions (EALLOW Protected).....	71
Table 1-33. Low-Power Mode Summary.....	72
Table 1-34. Low Power Modes.....	73
Table 1-35. Low Power Mode Control 0 Register (LPMCR0) Field Descriptions.....	74
Table 1-36. Example Watchdog Key Sequences.....	76
Table 1-37. System Control and Status Register (SCSR) Field Descriptions.....	78
Table 1-38. Watchdog Counter Register (WDCNTR) Field Descriptions.....	79
Table 1-39. Watchdog Reset Key Register (WDKEY) Field Descriptions.....	79
Table 1-40. Watchdog Control Register (WDCR) Field Descriptions.....	80
Table 1-41. CPU-Timers 0, 1, 2 Configuration and Control Registers.....	82
Table 1-42. CPU-Timer x Counter (TIMERxTIM) Register Field Descriptions.....	82
Table 1-43. CPU-Timer x Counter (TIMERxTIMH) Register High Field Descriptions.....	83
Table 1-44. CPU-Timer x Period (TIMERxPRD) Register Field Descriptions.....	83
Table 1-45. CPU-Timer x Period (TIMERxPRDH) Register High Field Descriptions.....	83
Table 1-46. CPU-Timer x Control (TIMERxTCR) Register Field Descriptions.....	84
Table 1-47. CPU-Timer x Prescale (TIMERxTPR) Register Field Descriptions.....	85
Table 1-48. CPU-Timer x Prescale (TIMERxTPRH) Register High Field Descriptions.....	86
Table 1-49. JTAG Port Mode.....	90
Table 1-50. JTAGDEBUG Register Field Descriptions.....	90
Table 1-51. GPIO Control Registers.....	92
Table 1-52. GPIO Interrupt and Low Power Mode Select Registers.....	92
Table 1-53. GPIO Data Registers.....	94
Table 1-54. Sampling Period.....	96
Table 1-55. Sampling Frequency.....	96
Table 1-56. Case 1: Three-Sample Sampling Window Width.....	97
Table 1-57. Case 2: Six-Sample Sampling Window Width.....	97
Table 1-58. Default State of Peripheral Input.....	100
Table 1-59. 2802x GPIOA MUX.....	101

Table 1-60. 2802x GPIOB MUX.....	102
Table 1-61. Analog MUX.....	102
Table 1-62. GPIO Port A Qualification Control (GPACTRL) Register Field Descriptions.....	103
Table 1-63. GPIO Port A Qualification Select 1 (GPAQSEL1) Register Field Descriptions.....	104
Table 1-64. GPIO Port A Qualification Select 2 (GPAQSEL2) Register Field Descriptions.....	105
Table 1-65. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions.....	106
Table 1-66. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions.....	107
Table 1-67. GPIO Port A Direction (GPADIR) Register Field Descriptions.....	109
Table 1-68. GPIO Port A Internal Pullup Disable (GPAPUD) Register Field Descriptions.....	110
Table 1-69. GPIO Port B Qualification Control (GPBCTRL) Register Field Descriptions.....	111
Table 1-70. GPIO Port B Qualification Select 1 (GPBQSEL1) Register Field Descriptions.....	112
Table 1-71. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions.....	113
Table 1-72. GPIO Port B Direction (GPBDIR) Register Field Descriptions.....	114
Table 1-73. GPIO Port B Internal Pullup Disable (GPBPUD) Register Field Descriptions.....	115
Table 1-74. Analog I/O MUX (AIOMUX1) Register Field Descriptions.....	116
Table 1-75. Analog I/O Direction (AIODIR) Register Field Descriptions.....	117
Table 1-76. GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register Field Descriptions.....	118
Table 1-77. XINT1/XINT2/XINT3 Interrupt Select and Configuration Registers.....	118
Table 1-78. GPIO Low Power Mode Wakeup Select (GPIOLPMSEL) Register Field Descriptions.....	119
Table 1-79. GPIO Port A Data (GPADAT) Register Field Descriptions.....	120
Table 1-80. GPIO Port A Set (GPASET) Register Field Descriptions.....	121
Table 1-81. GPIO Port A Clear (GPACLEAR) Register Field Descriptions.....	121
Table 1-82. GPIO Port A Toggle (GPATOGGLE) Register Field Descriptions.....	122
Table 1-83. GPIO Port B Data (GPBDAT) Register Field Descriptions.....	122
Table 1-84. GPIO Port B Set (GPBSET) Register Field Descriptions.....	123
Table 1-85. GPIO Port B Clear (GPBCLEAR) Register Field Descriptions.....	123
Table 1-86. GPIO Port B Toggle (GPBTOGGLE) Register Field Descriptions.....	123
Table 1-87. Analog I/O Data (AIODAT) Register Field Descriptions.....	124
Table 1-88. Analog I/O Set (AIOSET) Register Field Descriptions.....	125
Table 1-89. Analog I/O Clear (AIOCLEAR) Register Field Descriptions.....	125
Table 1-90. Analog I/O Toggle (AIOTOGGLE) Register Field Descriptions.....	126
Table 1-91. Peripheral Frame 0 Registers.....	126
Table 1-92. Peripheral Frame 1 Registers.....	127
Table 1-93. Peripheral Frame 2 Registers.....	127
Table 1-94. Access to EALLOW-Protected Registers.....	128
Table 1-95. EALLOW-Protected Device Emulation Registers.....	128
Table 1-96. EALLOW-Protected Flash/OTP Configuration Registers.....	128
Table 1-97. EALLOW-Protected Code Security Module (CSM) Registers.....	129
Table 1-98. EALLOW-Protected PLL, Clocking, Watchdog, and Low-Power Mode Registers.....	129
Table 1-99. EALLOW-Protected GPIO Registers.....	130
Table 1-100. EALLOW-Protected PIE Vector Table.....	131
Table 1-101. EALLOW-Protected ePWM1-ePWM4 Registers.....	131
Table 1-102. Device Emulation Registers.....	132
Table 1-103. DEVICECNF Register Field Descriptions.....	132
Table 1-104. Part ID (PARTID) Register Field Descriptions.....	133
Table 1-105. Class ID (CLASSID) Register Field Descriptions.....	134
Table 1-106. Revision ID (REVID) Register Field Descriptions.....	134
Table 1-107. Enabling Interrupt.....	138
Table 1-108. Interrupt Vector Table Mapping.....	139
Table 1-109. Vector Table Mapping After Reset Operation.....	139
Table 1-110. PIE MUXed Peripheral Interrupt Vector Table.....	146
Table 1-111. PIE Vector Table.....	147
Table 1-112. PIE Configuration and Control Registers.....	150
Table 1-113. PIE Control (PIECTRL) Register Field Descriptions.....	151
Table 1-114. PIE Interrupt Acknowledge (PIEACK) Register Field Descriptions.....	151
Table 1-115. PIE Interrupt Enable (PIEIERn) Registers Field Descriptions.....	152
Table 1-116. PIE Interrupt Flag (PIEIFRn) Registers Field Descriptions.....	153
Table 1-117. Interrupt Flag Register (IFR) — CPU Register Field Descriptions.....	154
Table 1-118. Interrupt Enable Register (IER) Field Descriptions.....	157
Table 1-119. Debug Interrupt Enable Register (DBGIER) Field Descriptions.....	159
Table 1-120. Interrupt Control and Counter Registers (not EALLOW Protected).....	161



Table 1-121. External Interrupt Control Registers (XINTnCR) Field Descriptions.....	161
Table 1-122. External Interrupt n Counter Registers (XINTnCTR)Field Descriptions.....	162
Table 1-123. BOR Configuration (BORCFG) Field Descriptions.....	164
Table 2-1. Vector Locations.....	170
Table 2-2. Configuration for Device Modes.....	172
Table 2-3. PIE Vector SARAM Locations Used by the Boot ROM.....	174
Table 2-4. Boot Mode Selection.....	174
Table 2-5. Valid EMU_KEY and EMU_BMODE Values.....	176
Table 2-6. OTP Values for GetMode.....	178
Table 2-7. Emulation Boot Modes (TRST = 1).....	179
Table 2-8. Standalone Boot Modes (TRST = 0).....	179
Table 2-9. LSB/MSB Loading Sequence in 8-bit Data Stream.....	181
Table 2-10. Parallel GPIO Boot 8-Bit Data Stream.....	190
Table 2-11. SPI 8-Bit Data Stream.....	194
Table 2-12. I2C 8-Bit Data Stream.....	199
Table 2-13. CPU Register Restored Values.....	201
Table 2-14. Bootloader Options.....	202
Table 2-15. Bootloader Revision and Checksum Information.....	205
Table 2-16. Bootloader Revision Per Device.....	205
Table 3-1. ePWM Module Control and Status Register Set Grouped by Submodule.....	213
Table 3-2. Submodule Configuration Parameters.....	215
Table 3-3. Time-Base Submodule Registers.....	218
Table 3-4. Key Time-Base Signals.....	219
Table 3-5. Counter-Compare Submodule Registers.....	227
Table 3-6. Counter-Compare Submodule Key Signals.....	227
Table 3-7. Action-Qualifier Submodule Registers.....	232
Table 3-8. Action-Qualifier Submodule Possible Input Events.....	232
Table 3-9. Action-Qualifier Event Priority for Up-Down-Count Mode.....	234
Table 3-10. Action-Qualifier Event Priority for Up-Count Mode.....	234
Table 3-11. Action-Qualifier Event Priority for Down-Count Mode.....	234
Table 3-12. Behavior if CMPA/CMPB is Greater than the Period.....	234
Table 3-13. Dead-Band Generator Submodule Registers.....	244
Table 3-14. Classical Dead-Band Operating Modes.....	246
Table 3-15. PWM-Chopper Submodule Registers.....	248
Table 3-16. Possible Pulse Width Values for SYSCLKOUT = 60 MHz.....	250
Table 3-17. Trip-Zone Submodule Registers.....	253
Table 3-18. Possible Actions On a Trip Event.....	254
Table 3-19. Event-Trigger Submodule Registers.....	260
Table 3-20. Digital Compare Submodule Registers.....	264
Table 3-21. Time-Base Control Register (TBCTL) Field Descriptions.....	294
Table 3-22. Time-Base Status Register (TBSTS) Field Descriptions.....	296
Table 3-23. Time-Base Phase High Resolution Register (TBPHSHR) Field Descriptions.....	297
Table 3-24. Time-Base Phase Register (TBPHS) Field Descriptions.....	297
Table 3-25. Time-Base Counter Register (TBCTR) Field Descriptions.....	298
Table 3-26. Time-Base Period Register (TBPRD) Field Descriptions.....	298
Table 3-27. Time Base Period High Resolution Register (TBPRDHR) Field Descriptions.....	299
Table 3-28. Time-Base Period High Resolution Mirror Register (TBPRDHRM) Field Descriptions.....	300
Table 3-29. Time-Base Period Mirror Register (TBPRDM) Field Descriptions.....	301
Table 3-30. Counter-Compare Control (CMPCTL) Register Field Descriptions.....	302
Table 3-31. Compare A High Resolution (CMPAHR) Register Field Descriptions.....	303
Table 3-32. Counter-Compare A (CMPA) Register Field Descriptions.....	304
Table 3-33. Counter-Compare B (CMPB) Register Field Descriptions.....	305
Table 3-34. Compare A High-Resolution Mirror (CMPAHRM) Register Field Descriptions.....	306
Table 3-35. Counter-Compare A Mirror (CMPAM) Register Field Descriptions.....	307
Table 3-36. Action-Qualifier Output A Control Register (AQCTLA) Field Descriptions.....	308
Table 3-37. Action-Qualifier Output B Control Register (AQCTLB) Field Descriptions.....	309
Table 3-38. Action-Qualifier Software Force Register (AQSFR) Field Descriptions.....	311
Table 3-39. Action-Qualifier Continuous Software Force Register (AQCSFR) Field Descriptions.....	312
Table 3-40. Dead-Band Generator Control (DBCTL) Register Field Descriptions.....	313
Table 3-41. Dead-Band Generator Rising Edge Delay (DBRED) Register Field Descriptions.....	315
Table 3-42. Dead-Band Generator Falling Edge Delay (DBFED) Register Field Descriptions.....	315

Table 3-43. Trip-Zone Submodule Select Register (TZSEL) Field Descriptions.....	316
Table 3-44. Trip Zone Digital Compare Event Select Register (TZDCSEL) Field Descriptions.....	317
Table 3-45. Trip-Zone Control Register Field Descriptions.....	319
Table 3-46. Trip-Zone Enable Interrupt Register (TZEINT) Field Descriptions.....	320
Table 3-47. Trip-Zone Flag Register Field Descriptions.....	321
Table 3-48. Trip-Zone Clear Register (TZCLR) Field Descriptions.....	323
Table 3-49. Trip-Zone Force Register (TZFRC) Field Descriptions.....	324
Table 3-50. Event-Trigger Selection Register (ETSEL) Field Descriptions.....	325
Table 3-51. Event-Trigger Prescale Register (ETPS) Field Descriptions.....	326
Table 3-52. Event-Trigger Flag Register (ETFLG) Field Descriptions.....	328
Table 3-53. Event-Trigger Clear Register (ETCLR) Field Descriptions.....	329
Table 3-54. Event-Trigger Force Register (ETFRC) Field Descriptions.....	330
Table 3-55. PWM-Chopper Control (PCCTL) Register Field Descriptions.....	331
Table 3-56. Digital Compare Trip Select (DCTRIPSEL) Register Field Descriptions.....	333
Table 3-57. Digital Compare A Control (DCACTL) Register Field Descriptions.....	335
Table 3-58. Digital Compare B Control (DCBCTL) Register Field Descriptions.....	336
Table 3-59. Digital Compare Filter Control (DCFCTL) Register Field Descriptions.....	337
Table 3-60. Digital Compare Capture Control (DCCAPCTL) Register Field Descriptions.....	338
Table 3-61. Digital Compare Filter Offset (DCFOFFSET) Register Field Descriptions.....	339
Table 3-62. Digital Compare Filter Offset Counter (DCFOFFSETCNT) Register Field Descriptions.....	339
Table 3-63. Digital Compare Filter Window (DCFWINDOW) Register Field Descriptions.....	340
Table 3-64. Digital Compare Filter Window Counter (DCFWINDOWCNT) Register Field Descriptions.....	340
Table 3-65. Digital Compare Counter Capture (DCCAP) Register Field Descriptions.....	341
Table 4-1. Resolution for PWM and HRPWM.....	344
Table 4-2. HRPWM Registers.....	345
Table 4-3. Relationship Between MEP Steps, PWM Frequency, and Resolution.....	350
Table 4-4. CMPA versus Duty (left), and [CMPA:CMPAHR] versus Duty (right).....	351
Table 4-5. Duty Cycle Range Limitation for 3 SYSCLK/TBCLK Cycles.....	353
Table 4-6. SFO Library Features.....	363
Table 4-7. Factor Values.....	364
Table 4-8. HRPWM Registers.....	367
Table 4-9. HRPWM Configuration (HRCNFG) Register Field Descriptions.....	368
Table 4-10. High Resolution Power (HRPWR) Register Field Descriptions.....	369
Table 4-11. High Resolution Micro Step (HRMSTEP) Register Field Descriptions.....	369
Table 4-12. High Resolution Period Control (HRPCTL) Register Field Descriptions.....	370
Table 5-1. eCAP Base Address Table (C28).....	387
Table 5-2. ECAP_REGS Registers.....	388
Table 5-3. ECAP_REGS Access Type Codes.....	388
Table 5-4. TSCTR Register Field Descriptions.....	389
Table 5-5. CTRPHS Register Field Descriptions.....	389
Table 5-6. CAP1 Register Field Descriptions.....	389
Table 5-7. CAP2 Register Field Descriptions.....	390
Table 5-8. CAP3 Register Field Descriptions.....	390
Table 5-9. CAP4 Register Field Descriptions.....	390
Table 5-10. ECCTL1 Register Field Descriptions.....	391
Table 5-11. ECCTL2 Register Field Descriptions.....	393
Table 5-12. ECEINT Register Field Descriptions.....	395
Table 5-13. ECFLG Register Field Descriptions.....	396
Table 5-14. ECCLR Register Field Descriptions.....	397
Table 5-15. ECFRC Register Field Descriptions.....	398
Table 6-1. Sample Timings with Different Values of ACQPS.....	404
Table 6-2. Estimated Droop Error from $n_r$ Value.....	408
Table 6-3. ADC Configuration and Control Registers (AdcRegs and AdcResult).....	424
Table 6-4. ADC Control Register 1 (ADCCTL1) Field Descriptions.....	425
Table 6-5. ADC Control Register 2 (ADCCTL2) Field Descriptions.....	427
Table 6-6. ADC Interrupt Flag Register (ADCINTFLG) Field Descriptions.....	428
Table 6-7. ADC Interrupt Flag Clear Register (ADCINTFLGCLR) Field Descriptions.....	429
Table 6-8. ADC Interrupt Overflow Register (ADCINTOVF) Field Descriptions.....	430
Table 6-9. ADC Interrupt Overflow Clear Register (ADCINTOVFCLR) Field Descriptions.....	431
Table 6-10. Interrupt Select Register (INTSELxNy) Field Descriptions.....	433
Table 6-11. SOCPRICTL Register Field Descriptions.....	435

Table 6-12. ADC Sample Mode Register (ADCSAMPLEMODE) Field Descriptions.....	437
Table 6-13. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1) Register Field Descriptions.....	439
Table 6-14. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2) Field Descriptions.....	439
Table 6-15. ADC SOC Flag 1 Register (ADCSOCFLG1) Field Descriptions.....	440
Table 6-16. ADC SOC Force 1 Register (ADCSOCFRC1) Field Descriptions.....	441
Table 6-17. ADC SOC Overflow 1 Register (ADCSOCOVF1) Field Descriptions.....	442
Table 6-18. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1) Field Descriptions.....	442
Table 6-19. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions.....	443
Table 6-20. ADC Reference/Gain Trim Register (ADCREFTRIM) Field Descriptions.....	446
Table 6-21. ADC Offset Trim Register (ADCOFFTRIM) Field Descriptions.....	446
Table 6-22. Comparator Hysteresis Control Register (COMPHYSTCTL) Field Descriptions.....	447
Table 6-23. ADC Revision Register (ADCREV) Field Descriptions.....	448
Table 6-24. ADC RESULT0-ADCRESULT15 Registers (ADCRESULTx) Field Descriptions.....	448
Table 7-1. Comparator Truth Table.....	451
Table 7-2. Comparator Modules.....	454
Table 7-3. Comparator Module Registers.....	454
Table 7-4. Comparator Control (COMPCTL) Register Field Descriptions.....	455
Table 7-5. Compare Output Status (COMPSTS) Register Field Descriptions.....	456
Table 7-6. DAC Control (DACCTL) Register Field Descriptions.....	456
Table 7-7. DAC Value (DACVAL) Register Field Descriptions.....	457
Table 7-8. Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register Field Descriptions.....	457
Table 7-9. Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register Field Descriptions.....	457
Table 7-10. Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register Field Descriptions.....	458
Table 7-11. Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register Field Descriptions.....	458
Table 7-12. Ramp Generator Status (RAMPSTS) Register Field Descriptions.....	458
Table 8-1. SPI Module Signal Summary.....	462
Table 8-2. SPI Interrupt Flag Modes.....	464
Table 8-3. SPI Clocking Scheme Selection Guide.....	470
Table 8-4. 4-wire vs. 3-wire SPI Pin Functions.....	472
Table 8-5. 3-Wire SPI Pin Configuration.....	473
Table 8-6. SPI Base Address Table (C28).....	477
Table 8-7. SPI_REGS Registers.....	477
Table 8-8. SPI_REGS Access Type Codes.....	478
Table 8-9. SPI Configuration Control Register (SPICCR) Field Descriptions.....	479
Table 8-10. SPI Operation Control (SPICTL) Register Field Descriptions.....	482
Table 8-11. SPI Status (SPISTS) Register Field Descriptions.....	484
Table 8-12. SPI Baud Rate Register (SPIBRR) Field Descriptions.....	486
Table 8-13. SPI Emulation Buffer (SPIRXEMU) Register Field Descriptions.....	487
Table 8-14. SPI Serial Input Buffer (SPIRXBUF) Register Field Descriptions.....	488
Table 8-15. SPI Serial Output Buffer (SPITXBUF) Register Field Descriptions.....	488
Table 8-16. SPI Serial Data (SPIDAT) Register Field Descriptions.....	489
Table 8-17. SPI FIFO Transmit (SPIFFTX) Register Field Descriptions.....	490
Table 8-18. SPI FIFO Receive (SPIFFRX) Register Field Descriptions.....	492
Table 8-19. SPI FIFO Control (SPIFFCT) Register Field Descriptions.....	494
Table 8-20. SPI Priority Control (SPIPRI) Register Field Descriptions.....	495
Table 9-1. SCI Module Signal Summary.....	499
Table 9-2. Programming the Data Format Using SCICCR.....	501
Table 9-3. Asynchronous Baud Register Values for Common SCI Bit Rates.....	508
Table 9-4. SCI Interrupt Flags.....	510
Table 9-5. SCI Base Address Table (C28).....	512
Table 9-6. SCI_REGS Registers.....	512
Table 9-7. SCI_REGS Access Type Codes.....	513
Table 9-8. SCI Communications Control Register (SCICCR) Field Descriptions.....	514
Table 9-9. SCI Control Register 1 (SCICTL1) Field Descriptions.....	516
Table 9-10. SCI Baud Rate (high) (SCIHBAUD) Register Field Descriptions.....	518
Table 9-11. SCI Baud Rate (low) (SCILBAUD) Register Field Descriptions.....	519
Table 9-12. SCI Control Register 2 (SCICTL2) Field Descriptions.....	520
Table 9-13. SCI Receive Status (SCIRXST) Register Field Descriptions.....	521
Table 9-14. SCI Receive Emulation Buffer (SCIRXEMU) Register Field Descriptions.....	523
Table 9-15. SCI Receive Data Buffer (SCIRXBUF) Register Field Descriptions.....	524
Table 9-16. SCI Transmit Data Buffer (SCITXBUF) Register Field Descriptions.....	525

Table 9-17. SCI FIFO Transmit (SCIFFTX) Register Field Descriptions.....	526
Table 9-18. SCI FIFO Receive (SCIFFRX) Register Field Descriptions.....	527
Table 9-19. SCI FIFO Control (SCIFFCT) Register Field Descriptions.....	529
Table 9-20. SCI Priority Control (SCIPRI) Register Field Descriptions.....	530
Table 10-1. Dependency of Delay d on the Divide-Down Value IPSC.....	536
Table 10-2. Operating Modes of the I2C Module.....	538
Table 10-3. Master-Transmitter/Receiver Bus Activity Defined by the RM, STT, and STP Bits of I2CMR.....	538
Table 10-4. How the MST and FDF Bits of I2CMR Affect the Role of the TRX Bit of I2CMR.....	544
Table 10-5. Ways to Generate a NACK Bit.....	548
Table 10-6. Descriptions of the Basic I2C Interrupt Requests.....	549
Table 10-7. I2C Base Address Table (C28).....	552
Table 10-8. I2C_REGS Registers.....	552
Table 10-9. I2C_REGS Access Type Codes.....	553
Table 10-10. I2C Own Address Register (I2COAR) Field Descriptions.....	554
Table 10-11. I2C Interrupt Enable Register (I2CIER) Field Descriptions.....	555
Table 10-12. I2C Status Register (I2CSTR) Field Descriptions.....	556
Table 10-13. I2C Clock Low-time Divider (I2CCLKL) Register Field Descriptions.....	560
Table 10-14. I2C Clock High-time Divider (I2CCLKH) Register Field Descriptions.....	560
Table 10-15. I2C Data Count (I2CCNT) Register Field Descriptions.....	561
Table 10-16. I2C Data Receive Register (I2CDRR) Field Descriptions.....	562
Table 10-17. I2C Slave Address Register (I2CSAR) Field Descriptions.....	563
Table 10-18. I2C Data Transmit Register (I2CDXR) Field Descriptions.....	564
Table 10-19. I2C Mode Register (I2CMR) Field Descriptions.....	565
Table 10-20. I2C Interrupt Source (I2CISRC) Register Field Descriptions.....	569
Table 10-21. I2C Extended Mode Register (I2CEMR) Field Descriptions.....	570
Table 10-22. I2C Prescaler (I2CPSC) Register Field Descriptions.....	571
Table 10-23. I2C FIFO Transmit (I2CFFTX) Register Field Descriptions.....	572
Table 10-24. I2C FIFO Receive (I2CFFRX) Register Field Descriptions.....	574



## About This Manual

This Technical Reference Manual (TRM) details the integration, the environment, the functional description, and the programming models for each peripheral and subsystem in the device.

The TRM should not be considered a substitute for the data manual, rather a companion guide that should be used alongside the device-specific data manual to understand the details to program the device. The primary purpose of the TRM is to abstract the programming details of the device from the data manual. This allows the data manual to outline the high-level features of the device without unnecessary information about register descriptions or programming models.

## Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers can be shown with the suffix h or the prefix 0x. For example, the following number is 40 hexadecimal (decimal 64): 40h or 0x40.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties with default reset value below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure can have one of multiple meanings:
    - Not implemented on the device
    - Reserved for future device expansion
    - Reserved for TI testing
    - Reserved configurations of the device that are not supported
  - Writing nondefault values to the Reserved bits could cause unexpected behavior and should be avoided.

## Glossary

[TI Glossary](#) This glossary lists and explains terms, acronyms, and definitions.

## Related Documentation From Texas Instruments

For a complete listing of related documentation and development-support tools for these devices, visit the Texas Instruments website at <http://www.ti.com>.

Additionally, the [TMS320C28x DSP CPU and Instruction Set Reference Guide](#) and the [TMS320C28x Floating Point Unit and Instruction Set Reference Guide](#) must be used in conjunction with this TRM.

## Support Resources

[TI E2E™ support forums](#) are an engineer's go-to source for fast, verified answers and design help — straight from the experts. Search existing answers or ask your own question to get the quick design help you need.

Linked content is provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's [Terms of Use](#).

## Trademarks

TI E2E™ and Code Composer Studio™ are trademarks of Texas Instruments.  
All trademarks are the property of their respective owners.

This page intentionally left blank.

This chapter describes how various system controls and interrupts work and provides information on the:

- Flash and one-time programmable (OTP) memories
- Code security module (CSM), which is a security feature
- Clocking mechanisms including the oscillator, PLL, XCLKOUT, watchdog module, and the low-power modes. In addition, the 32-bit CPU timers are also described.
- GPIO multiplexing (MUX) registers used to select the operation of shared pins on the device
- Accessing the peripheral frames to write to and read from various peripheral registers on the device
- Interrupt sources both external and the peripheral interrupt expansion (PIE) block that multiplexes numerous interrupt sources into a smaller set of interrupt inputs

<b>1.1 Flash and OTP Memory</b> .....	<b>24</b>
<b>1.2 Code Security Module (CSM)</b> .....	<b>36</b>
<b>1.3 Clocking</b> .....	<b>46</b>
<b>1.4 General-Purpose Input/Output (GPIO)</b> .....	<b>86</b>
<b>1.5 Peripheral Frames</b> .....	<b>126</b>
<b>1.6 Peripheral Interrupt Expansion (PIE)</b> .....	<b>136</b>
<b>1.7 VREG/BOR/POR</b> .....	<b>163</b>

## 1.1 Flash and OTP Memory

This section describes the proper sequence to configure the wait states and operating mode of flash and one-time programmable (OTP) memories. It also includes information on flash and OTP power modes and how to improve flash performance by enabling the flash pipeline mode.

### 1.1.1 Flash Memory

The on-chip flash is uniformly mapped in both program and data memory space. This flash memory is always enabled and features:

- **Multiple sectors**

The minimum amount of flash memory that can be erased is a sector. Having multiple sectors provides the option of leaving some sectors programmed and only erasing specific sectors.

- **Code security**

The flash is protected by the Code Security Module (CSM). By programming a password into the flash, the user can prevent access to the flash by unauthorized persons. See [Section 1.2](#) for information in using the Code Security Module.

- **Low power modes**

To save power when the flash is not in use, two levels of low power modes are available. See [Section 1.1.3](#) for more information on the available flash power modes.

- **Configurable wait states**

Configurable wait states can be adjusted based on CPU frequency to give the best performance for a given execution speed.

- **Enhanced performance**

A flash pipeline mode is provided to improve performance of linear code execution.

### 1.1.2 OTP Memory

The 1K x 16 block of one-time programmable (OTP) memory is uniformly mapped in both program and data memory space. Thus, the OTP can be used to program data or code. This block, unlike flash, can be programmed only one time and cannot be erased.

### 1.1.3 Flash and OTP Power Modes

The following operating states apply to the flash and OTP memory:

- **Reset or Sleep State**

This is the state after a device reset. In this state, the bank and pump are in a sleep state (lowest power). When the flash is in the sleep state, a CPU data read or opcode fetch to the flash or OTP memory map area will automatically initiate a change in power modes to the standby state and then to the active state. During this transition time to the active state, the CPU will automatically be stalled. Once the transition to the active state is completed, the CPU access will complete as normal.

- **Standby State**

In this state, the bank and pump are in standby power mode state. This state uses more power than the sleep state, but takes a shorter time to transition to the active or read state. When the flash is in the standby state, a CPU data read or opcode fetch to the flash or OTP memory map area will automatically initiate a change in power modes to the active state. During this transition time to the active state, the CPU will automatically be stalled. Once the flash/OTP has reached the active state, the CPU access will complete as normal.

- **Active or Read State**

In this state, the bank and pump are in active power mode state (highest power). The CPU read or fetch access wait states to the flash/OTP memory map area is controlled by the FBANKWAIT and FOTPWAIT registers. A prefetch mechanism called flash pipeline can also be enabled to improve fetch performance for linear code execution.

---

#### Note

During the boot process, the Boot ROM performs a dummy read of the Code Security Module (CSM) password locations located in the flash. This read is performed to unlock a new or erased device that has no password stored in it so that flash programming or loading of code into CSM protected SARAM can be performed. On devices with a password stored, this read has no effect and the CSM remains locked (see [Section 1.2](#) for information on the CSM). One effect of this read is that the flash will transition from the sleep (reset) state to the active state.

---

The flash/OTP bank and pump are always in the same power mode. See [Figure 1-1](#) for a graphic depiction of the available power states. You can change the current flash/OTP memory power state as follows:

- **To move to a lower power state**

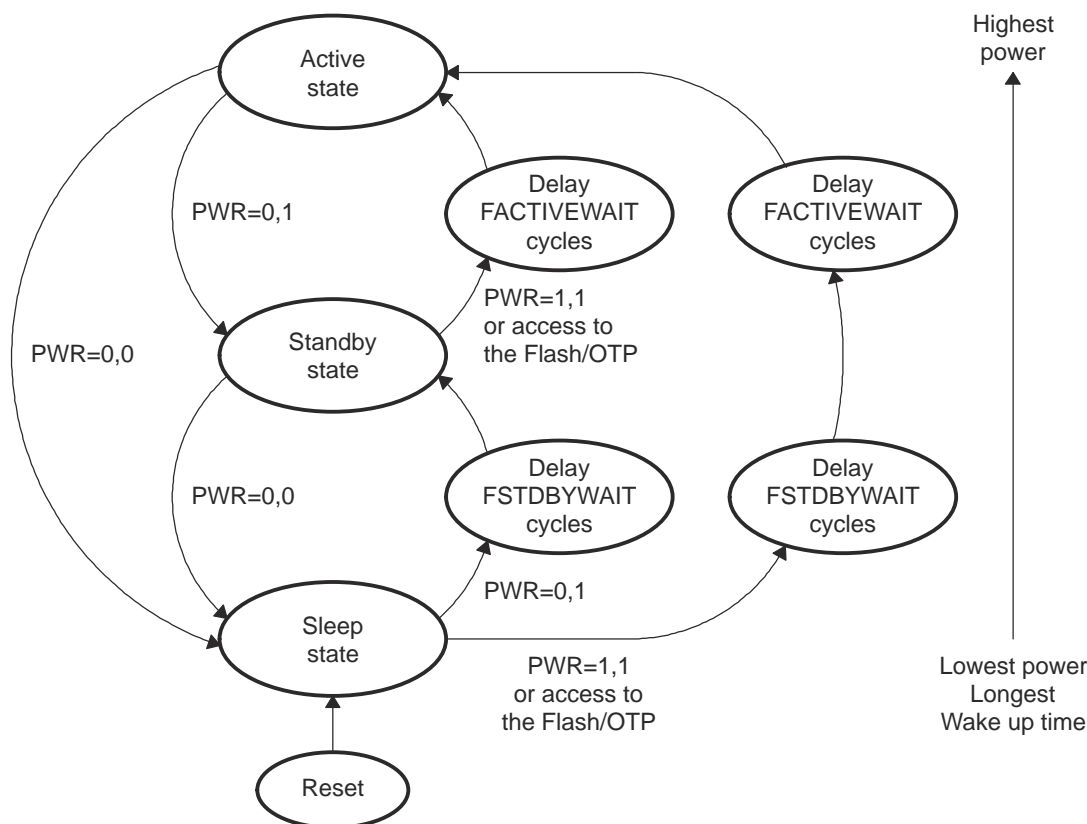
Change the PWR mode bits from a higher power mode to a lower power mode. This change instantaneously moves the flash/OTP bank to the lower power state. This register should be accessed only by code running outside the flash/OTP memory.

- **To move to a higher power state**

To move from a lower power state to a higher power state, there are two options.

1. Change the FPWR register from a lower state to a higher state. This access brings the flash/OTP memory to the higher state.
2. Access the flash or OTP memory by a read access or program opcode fetch access. This access automatically brings the flash/OTP memory to the active state.

There is a delay when moving from a lower power state to a higher one. See [Figure 1-1](#). This delay is required to allow the flash to stabilize at the higher power mode. If any access to the flash/OTP memory occurs during this delay the CPU automatically stalls until the delay is complete.



**Figure 1-1. Flash Power Mode State Diagram**

The duration of the delay is determined by the FSTDBYWAIT and FACTIVEWAIT registers. Moving from the sleep state to a standby state is delayed by a count determined by the FSTDBYWAIT register. Moving from the standby state to the active state is delayed by a count determined by the FACTIVEWAIT register. Moving from the sleep mode (lowest power) to the active mode (highest power) is delayed by FSTDBYWAIT + FACTIVEWAIT. These registers should be left in their default state.

### 1.1.3.1 Flash and OTP Performance

CPU read or data fetch operations to the flash/OTP can take one of the following forms:

- 32-bit instruction fetch
- 16-bit or 32-bit data space read
- 16-bit program space read

Once flash is in the active power state, then a read or fetch access to the bank memory map area can be classified as a flash access or an OTP access.

The main flash array is organized into rows and columns. The rows contain 2048 bits of information. Accesses to flash and OTP are one of three types:

#### 1. Flash Memory Random Access

The first access to a 2048 bit row is considered a random access.

#### 2. Flash Memory Paged Access

While the first access to a row is considered a random access, subsequent accesses within the same row are termed paged accesses.

The number of wait states for both a random and a paged access can be configured by programming the FBANKWAIT register. The number of wait states used by a random access is controlled by the RANDWAIT bits and the number of wait states used by a paged access is controlled by the PAGEWAIT bits. The

FBANKWAIT register defaults to a worst-case wait state count and, thus, needs to be initialized for the appropriate number of wait states to improve performance based on the CPU clock rate and the access time of the flash. The flash supports 0-wait accesses when the PAGEWAIT bits are set to zero. This assumes that the CPU speed is low enough to accommodate the access time. To determine the random and paged access time requirements, refer to the Data Manual for your particular device.

### 3. OTP Access

Read or fetch accesses to the OTP are controlled by the OTPWAIT bits in the FOTPWAIT register. Accesses to the OTP take longer than the flash and there is no paged mode. To determine OTP access time requirements, see the data manual for your particular device.

Some other points to keep in mind when working with flash:

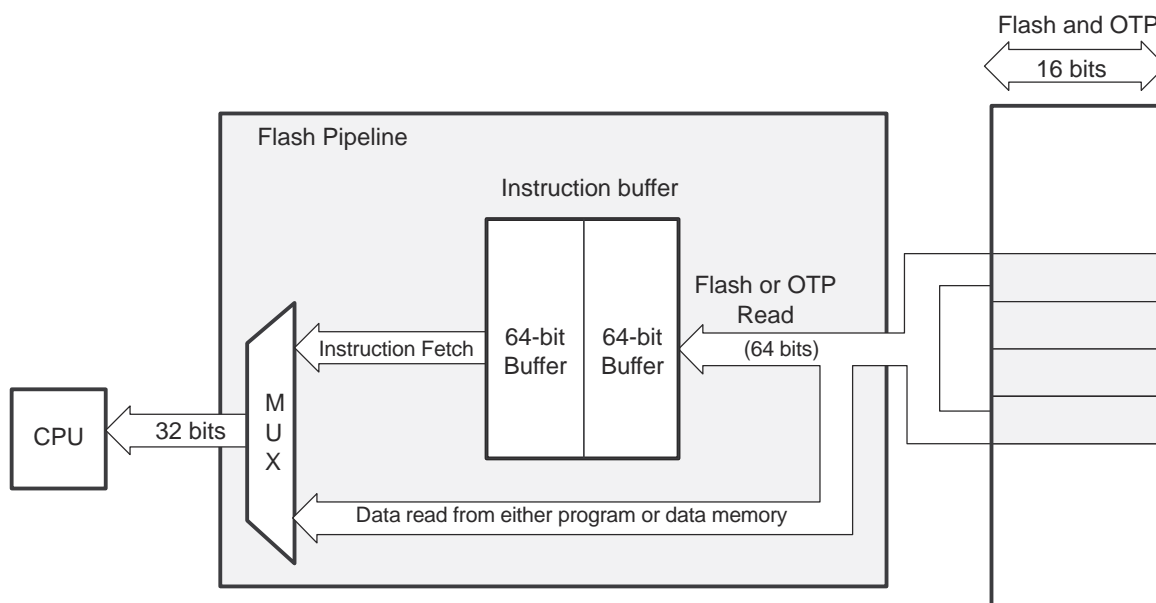
- CPU writes to the flash or OTP memory map area are ignored. They complete in a single cycle.
- When the Code Security Module (CSM) is secured, reads to the flash/OTP memory map area from outside the secure zone take the same number of cycles as a normal access. However, the read operation returns a zero.
- Reads of the CSM password locations are hardwired for 16 wait-states. The PAGEWAIT and RANDOMWAIT bits have no effect on these locations. See [Section 1.2](#) for more information on the CSM.

#### 1.1.3.2 Flash Pipeline Mode

Flash memory is typically used to store application code. During code execution, instructions are fetched from sequential memory addresses, except when a discontinuity occurs. Usually the portion of the code that resides in sequential addresses makes up the majority of the application code and is referred to as linear code. To improve the performance of linear code execution, a flash pipeline mode has been implemented. The flash pipeline feature is disabled by default. Setting the ENPIPE bit in the FOPT register enables this mode. The flash pipeline mode is independent of the CPU pipeline.

An instruction fetch from the flash or OTP reads out 64 bits per access. The starting address of the access from flash is automatically aligned to a 64-bit boundary such that the instruction location is within the 64 bits to be fetched. With flash pipeline mode enabled (see [Figure 1-2](#)), the 64 bits read from the instruction fetch are stored in a 64-bit wide by 2-level deep instruction prefetch buffer. The contents of this prefetch buffer are then sent to the CPU for processing as required.

Up to two 32-bit instructions or up to four 16-bit instructions can reside within a single 64-bit access. The majority of C28x instructions are 16 bits, so for every 64-bit instruction fetch from the flash bank it is likely that there are up to four instructions in the prefetch buffer ready to process through the CPU. During the time it takes to process these instructions, the flash pipeline automatically initiates another access to the flash bank to prefetch the next 64 bits. In this manner, the flash pipeline mode works in the background to keep the instruction prefetch buffers as full as possible. Using this technique, the overall efficiency of sequential code execution from flash or OTP is improved significantly.



**Figure 1-2. Flash Pipeline**

The flash pipeline prefetch is aborted only on a PC discontinuity caused by executing an instruction such as a branch, BANTZ, call, or loop. When this occurs, the prefetch is aborted and the contents of the prefetch buffer are flushed. There are two possible scenarios when this occurs:

1. If the destination address is within the flash or OTP, the prefetch aborts and then resumes at the destination address.
2. If the destination address is outside of the flash and OTP, the prefetch is aborted and begins again only when a branch is made back into the flash or OTP. The flash pipeline prefetch mechanism only applies to instruction fetches from program space. Data reads from data memory and from program memory do not utilize the prefetch buffer capability and thus bypass the prefetch buffer. For example, instructions such as MAC, DMAC, and PREAD read a data value from program memory. When this read happens, the prefetch buffer is bypassed but the buffer is not flushed. If an instruction prefetch is already in progress when a data read operation is initiated, then the data read will be stalled until the prefetch completes.

### 1.1.3.3 Reserved Locations Within Flash and OTP

When allocating code and data to flash and OTP memory, keep the following in mind:

1. Address locations 0x3F 7FF6 and 0x3F 7FF7 are reserved for an entry into flash branch instruction. When the boot to flash boot option is used, the boot ROM will jump to address 0x3F 7FF6. If you program a branch instruction here that will then redirect code execution to the entry point of the application.
2. For code security operation, all addresses between 0x3F 7F80 and 0x3F 7FF5 cannot be used for program code or data, but must be programmed to 0x0000 when the Code Security Password is programmed. If the code security feature is not used, addresses 0x3F7F80 to 0x3F7FEF may be used for code or data. Addresses 0x3F7FF0 to 0x3F7FF5 are reserved for data and should not contain program code. See [Section 1.2](#) for information in using the Code Security Module.



### 1.1.3.4 Procedure to Change the Flash Configuration Registers

During flash configuration, no accesses to the flash or OTP can be in progress. This includes instructions still in the CPU pipeline, data reads, and instruction prefetch operations. To be sure that no access takes place during the configuration change, you should follow the procedure shown in [Figure 1-3](#) for any code that modifies the FOPT, FPWR, FBANKWAIT, or FOTPWAIT registers.

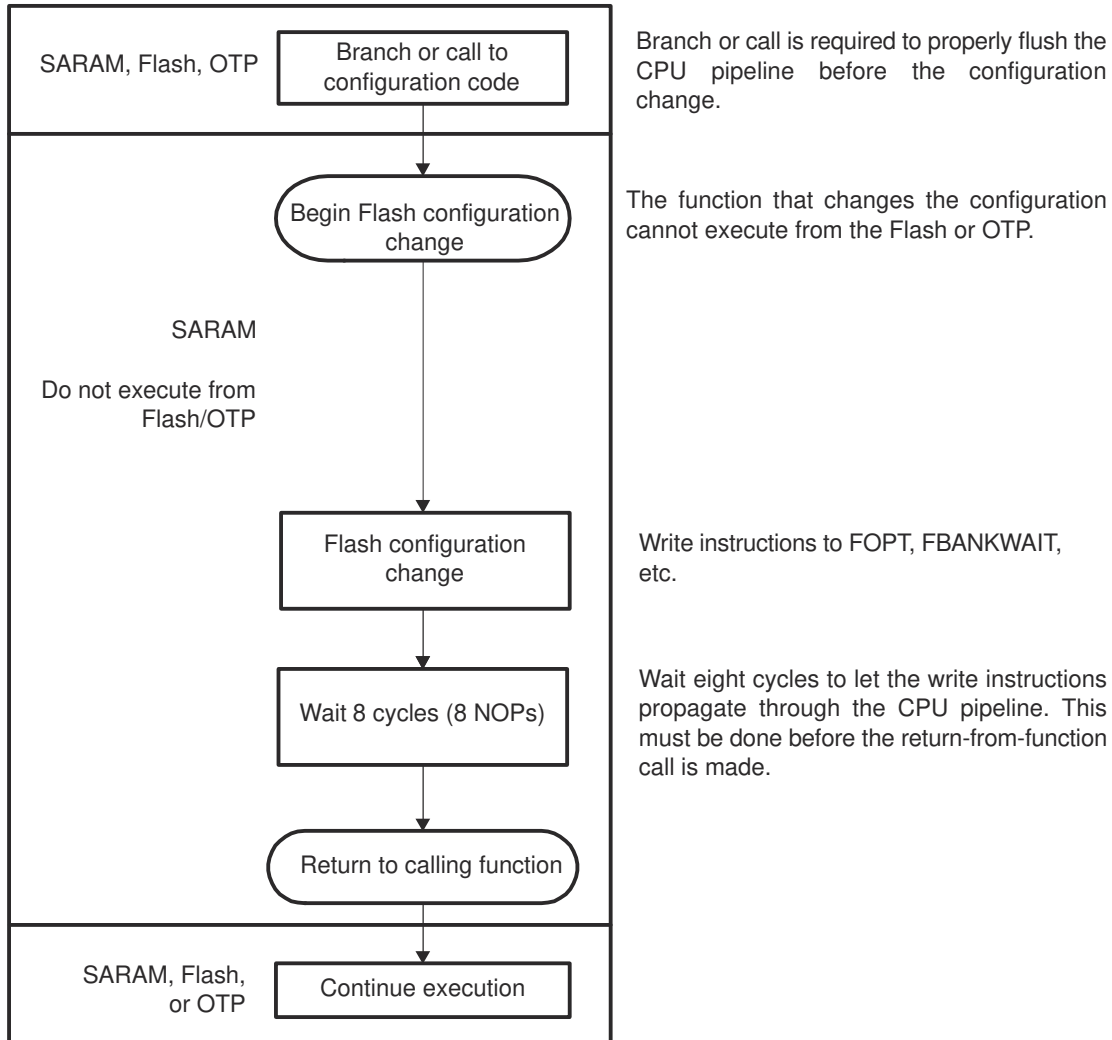


Figure 1-3. Flash Configuration Access Flow Diagram

### 1.1.4 Flash and OTP Registers

The flash and OTP memory can be configured by the registers shown in [Table 1-1](#). The configuration registers are all EALLOW protected.

**Table 1-1. Flash/OTP Configuration Registers**

Name <sup>(1) (2)</sup>	Address	Size (x16)	Description	Bit Description
FOPT	0x0A80	1	Flash Option Register	<a href="#">Section 1.1.4.1</a>
Reserved	0x0A81	1	Reserved	-
FPWR	0x0A82	1	Flash Power Modes Register	<a href="#">Section 1.1.4.2</a>
FSTATUS	0x0A83	1	Status Register	<a href="#">Section 1.1.4.3</a>
FSTDBYWAIT <sup>(3)</sup>	0x0A84	1	Flash Sleep To Standby Wait Register	<a href="#">Section 1.1.4.4</a>
FACTIVEWAIT <sup>(3)</sup>	0x0A85	1	Flash Standby To Active Wait Register	<a href="#">Section 1.1.4.5</a>
FBANKWAIT	0x0A86	1	Flash Read Access Wait State Register	<a href="#">Section 1.1.4.6</a>
FOTPWAIT	0x0A87	1	OTP Read Access Wait State Register	<a href="#">Section 1.1.4.7</a>

(1) These registers are EALLOW protected. See [Section 1.5.2](#) for information.

(2) These registers are protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.

(3) These registers should be left in their default state.

#### Note

The flash configuration registers should not be written to by code that is running from OTP or flash memory or while an access to flash or OTP may be in progress. All register accesses to the flash registers should be made from code executing outside of flash/OTP memory and an access should not be attempted until all activity on the flash/OTP has completed. No hardware is included to protect against this.

To summarize, you can read the flash registers from code executing in flash/OTP; however, do not write to the registers.

CPU write access to the flash configuration registers can be enabled only by executing the EALLOW instruction. Write access is disabled when the EDIS instruction is executed. This protects the registers from spurious accesses. Read access is always available. The registers can be accessed through the JTAG port without the need to execute EALLOW. See [Section 1.5.2](#) for information on EALLOW protection. These registers support both 16-bit and 32-bit accesses.

### 1.1.4.1 Flash Options Register (FOPT)

**Figure 1-4. Flash Options Register (FOPT)**

15		1	0
Reserved		ENPIPE	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-2. Flash Options Register (FOPT) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup> <sup>(2)</sup> <sup>(3)</sup>
15-1	Reserved		Any writes to these bit(s) must always have a value of 0.
0	ENPIPE		Enable Flash Pipeline Mode Bit. Flash pipeline mode is active when this bit is set. The pipeline mode improves performance of instruction fetches by prefetching instructions. See <a href="#">Section 1.1.3.2</a> for more information.  When pipeline mode is enabled, the flash wait states (paged and random) must be greater than zero.  On flash devices, ENPIPE affects fetches from flash and OTP.
		0	Flash Pipeline mode is not active. (default)
		1	Flash Pipeline mode is active.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) This register is protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.

(3) When writing to this register, follow the procedure described in [Section 1.1.3.4](#).

### 1.1.4.2 Flash Power Register (FPWR)

**Figure 1-5. Flash Power Register (FPWR)**

15		2	1	0
Reserved		PWR		
R-0		R/W-0		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-3. Flash Power Register (FPWR) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup> <sup>(2)</sup>
15-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1-0	PWR		Flash Power Mode Bits. Writing to these bits changes the current power mode of the flash bank and pump. See <a href="#">Section 1.1.3</a> for more information on changing the flash bank power mode.
		00	Pump and bank sleep (lowest power)
		01	Pump and bank standby
		10	Reserved (no effect)
		11	Pump and bank active (highest power)

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) This register is protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.

### 1.1.4.3 Flash Status Register (FSTATUS)

**Figure 1-6. Flash Status Register (FSTATUS)**

15	Reserved	9	8
			3VSTAT
	R-0		R/W1C-0
7	4	3	2
			1
			0
	Reserved	ACTIVEWAITS	STDBYWAITS
			PWRS
	R-0	R-0	R-0
		R-0	R-0

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -n = value after reset

**Table 1-4. Flash Status Register (FSTATUS) Field Descriptions**

Bit	Field	Value	Description <sup>(1) (2)</sup>
15-9	Reserved		Any writes to these bit(s) must always have a value of 0.
8	3VSTAT	0 1	Flash Voltage ( $V_{DD3VFL}$ ) Status Latch Bit. When set, this bit indicates that the 3VSTAT signal from the pump module went to a high level. This signal indicates that the flash 3.3-V supply went out of the allowable range. Writes of 0 are ignored. When this bit reads 1, it indicates that the flash 3.3-V supply went out of the allowable range. Clear this bit by writing a 1.
7-4	Reserved		Any writes to these bit(s) must always have a value of 0.
3	ACTIVEWAITS	0 1	Bank and Pump Standby To Active Wait Counter Status Bit. This bit indicates whether the respective wait counter is timing out an access. The counter is not counting. The counter is counting.
2	STDBYWAITS	0 1	Bank and Pump Sleep To Standby Wait Counter Status Bit. This bit indicates whether the respective wait counter is timing out an access. The counter is not counting. The counter is counting.
1-0	PWRS	00 01 10 11	Power Modes Status Bits. These bits indicate which power mode the flash/OTP is currently in. The PWRS bits are set to the new power mode only after the appropriate timing delays have expired. Pump and bank in sleep mode (lowest power) Pump and bank in standby mode Reserved Pump and bank active and in read mode (highest power)

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) This register is protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.

#### 1.1.4.4 Flash Standby Wait Register (FSTDBYWAIT)

**Figure 1-7. Flash Standby Wait Register (FSTDBYWAIT)**

15	9	8	0
Reserved		STDBYWAIT	
R-0		R/W-0x1FF	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-5. Flash Standby Wait Register (FSTDBYWAIT) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup> <sup>(2)</sup>
15-9	Reserved		Any writes to these bit(s) must always have a value of 0.
8-0	STDBYWAIT	0x1FF	<b>This register should be left in its default state.</b> Bank and Pump Sleep To Standby Wait Count: 511 SYSCLKOUT cycles (default)

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) This register is protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.

#### 1.1.4.5 Flash Standby to Active Wait Counter Register (FACTIVEWAIT)

**Figure 1-8. Flash Standby to Active Wait Counter Register (FACTIVEWAIT)**

15	9	8	0
Reserved		ACTIVEWAIT	
R-0		R/W-0x1FF	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-6. Flash Standby to Active Wait Counter Register (FACTIVEWAIT) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup> <sup>(2)</sup>
15-9	Reserved		Any writes to these bit(s) must always have a value of 0.
8-0	ACTIVEWAIT	0x1FF	<b>This register should be left in its default state.</b> Bank and Pump Standby To Active Wait Count: 511 SYSCLKOUT cycles (default)

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) This register is protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.

### 1.1.4.6 Flash Wait-State Register (FBANKWAIT)

**Figure 1-9. Flash Wait-State Register (FBANKWAIT)**

15	12	11	8	7	4	3	0
Reserved	PAGEWAIT	Reserved	RANDWAIT				
R-0	R/W-0xF	R-0	R/W-0xF				

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

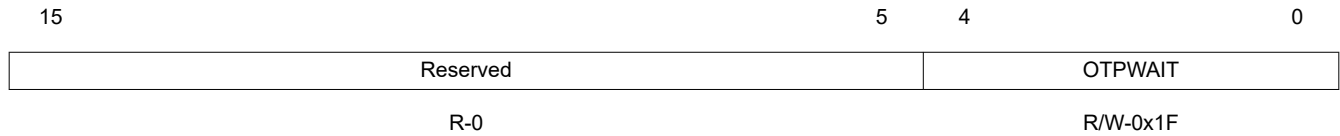
**Table 1-7. Flash Wait-State Register (FBANKWAIT) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup> <sup>(2)</sup> <sup>(3)</sup>
15-12	Reserved		Any writes to these bit(s) must always have a value of 0.
11-8	PAGEWAIT	0000 0001 0010 0011 ... 1111	Flash Paged Read Wait States. These register bits specify the number of wait states for a paged read operation in CPU clock cycles (0..15 SYSCLKOUT cycles) to the flash bank. See <a href="#">Section 1.1.3.1</a> for more information.  See the device-specific data manual for the minimum time required for a PAGED flash access.  You must set RANDWAIT to a value greater than or equal to the PAGEWAIT setting. No hardware is provided to detect a PAGEWAIT value that is greater than RANDWAIT.  Zero wait-state per paged flash access or one SYSCLKOUT cycle per access One wait state per paged flash access or a total of two SYSCLKOUT cycles per access Two wait states per paged flash access or a total of three SYSCLKOUT cycles per access Three wait states per paged flash access or a total of four SYSCLKOUT cycles per access ... 15 wait states per paged flash access or a total of 16 SYSCLKOUT cycles per access. (default)
7-4	Reserved		Any writes to these bit(s) must always have a value of 0.
3-0	RANDWAIT	0000 0001 0010 0011 ... 1111	Flash Random Read Wait States. These register bits specify the number of wait states for a random read operation in CPU clock cycles (1..15 SYSCLKOUT cycles) to the flash bank. See <a href="#">Section 1.1.3.1</a> for more information.  See the device-specific data manual for the minimum time required for a RANDOM flash access.  RANDWAIT must be set greater than 0. That is, at least 1 random wait state must be used. In addition, you must set RANDWAIT to a value greater than or equal to the PAGEWAIT setting. The device will not detect and correct a PAGEWAIT value that is greater than RANDWAIT.  Illegal value. RANDWAIT must be set greater than 0. One wait state per random flash access or a total of two SYSCLKOUT cycles per access. Two wait states per random flash access or a total of three SYSCLKOUT cycles per access. Three wait states per random flash access or a total of four SYSCLKOUT cycles per access. ... 15 wait states per random flash access or a total of 16 SYSCLKOUT cycles per access. (default)

- (1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.  
 (2) This register is protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.  
 (3) When writing to this register, follow the procedure described in [Section 1.1.3.4](#).

### 1.1.4.7 OTP Wait-State Register (FOTPWAIT)

**Figure 1-10. OTP Wait-State Register (FOTPWAIT)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-8. OTP Wait-State Register (FOTPWAIT) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup> <sup>(2)</sup> <sup>(3)</sup>
15-5	Reserved		Any writes to these bit(s) must always have a value of 0.
4-0	OTPWAIT		<p>OTP Read Wait States. These register bits specify the number of wait states for a read operation in CPU clock cycles (1..31 SYSCLKOUT cycles) to the OTP. See CPU Read Or Fetch Access From flash/OTP section for details. There is no PAGE mode in the OTP.</p> <p>OTPWAIT must be set greater than 0. That is, a minimum of 1 wait state must be used. See the device-specific data manual for the minimum time required for an OTP access.</p> <p>00000 Illegal value. OTPWAIT must be set to 1 or greater.</p> <p>00001 One wait state will be used each OTP access for a total of two SYSCLKOUT cycles per access.</p> <p>00010 Two wait states will be used for each OTP access for a total of three SYSCLKOUT cycles per access.</p> <p>00011 Three wait states will be used for each OTP access for a total of four SYSCLKOUT cycles per access.</p> <p>... ..</p> <p>11111 31 wait states will be used for an OTP access for a total of 32 SYSCLKOUT cycles per access.</p>

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) This register is protected by the Code Security Module (CSM). See [Section 1.2](#) for more information.

(3) When writing to this register, follow the procedure described in [Section 1.1.3.4](#).

## 1.2 Code Security Module (CSM)

The code security module (CSM) is a security feature incorporated in 28x devices. It prevents access or visibility to on-chip memory to unauthorized persons. In other words, it prevents duplication as well as reverse engineering of proprietary code.

The word secure means access to on-chip memory is protected. The word unsecure means access to on-chip secure memory is not protected — that is, the contents of the memory could be read by any means (for example, through a debugging tool such as the Code Composer Studio™ IDE).

### 1.2.1 Functional Description

The security module restricts the CPU access to certain on-chip memory without interrupting or stalling CPU execution. When a read occurs to a protected memory location, the read returns a zero value and CPU execution continues with the next instruction. This, in effect, blocks read and write access to various memories through the JTAG port or external peripherals. Security is defined with respect to the access of on-chip memory and prevents unauthorized copying of proprietary code or data.

The device is secure when CPU access to the on-chip secure memory locations is restricted. When secure, two levels of protection are possible, depending on where the program counter is currently pointing. If code is currently running from inside secure memory, only an access through JTAG is blocked (that is, through the JTAG debug probe). This allows secure code to access secure data. Conversely, if code is running from nonsecure memory, all accesses to secure memories are blocked. User code can dynamically jump in and out of secure memory, thereby allowing secure function calls from nonsecure memory. Similarly, interrupt service routines can be placed in secure memory, even if the main program loop is run from nonsecure memory.

Security is protected by a password of 128 bits of data (eight 16-bit words) that is used to secure or unsecure the device. This password is stored at the end of flash in 8 words referred to as the password locations.

The device is unsecured by executing the password match flow (PMF), described in [Section 1.2.3.2](#). The levels of security are shown in [Table 1-9](#).

**Table 1-9. Security Levels**

PMF Executed With Correct Password?	Operating Mode	Program Fetch Location	Security Description
No	Secure	Outside secure memory	Only instruction fetches by the CPU are allowed to secure memory. In other words, code can still be executed, but not read
No	Secure	Inside secure memory	CPU has full access. JTAG port cannot read the secured memory contents.
Yes	Not Secure	Anywhere	Full access for CPU and JTAG port to secure memory

The password is stored in code security password locations (PWL) in flash memory (0x3F 7FF8 - 0x3F 7FFF). These locations store the password predetermined by the system designer.

If the password locations have all 128 bits as ones, the device is labeled unsecure. Since new flash devices have erased flash (all ones), only a read of the password locations is required to bring the device into unsecure mode. If the password locations have all 128 bits as zeros, the device is secure, regardless of the contents of the KEY registers. Do not use all zeros as a password or reset the device during an erase of the flash. Resetting the device during an erase routine can result in either an all zero or unknown password. If a device is reset when the password locations are all zeros, the device cannot be unlocked by the password match flow described in [Section 1.2.3.2](#). Using a password of all zeros will seriously limit your ability to debug secure code or reprogram the flash.



---

**Note**

If a device is reset while the password locations are all zero or an unknown value, the device will be permanently locked unless a method to run the flash erase routine from secure SARAM is embedded into the flash or OTP. You must take care when implementing this procedure to avoid introducing a security hole.

---

User accessible registers (eight 16-bit words) that are used to unsecure the device are referred to as key registers. These registers are mapped in the memory space at addresses 0x00 0AE0 - 0x00 0AE7 and are EALLOW protected.

In addition to the CSM, the emulation code security logic (ECSL) has been implemented to prevent unauthorized users from stepping through secure code. Any code or data access to flash, user OTP, L0 memory while the JTAG debug probe is connected will trip the ECSL and break the emulation connection. To allow emulation of secure code, while maintaining the CSM protection against secure memory reads, you must write the correct value into the lower 64 bits of the KEY register, which matches the value stored in the lower 64 bits of the password locations within the flash. Note that dummy reads of all 128 bits of the password in the flash must still be performed. If the lower 64 bits of the password locations are all ones (unprogrammed), then the KEY value does not need to match.

When initially debugging a device with the password locations in flash programmed (that is, secured), the JTAG debug probe takes some time to take control of the CPU. During this time, the CPU will start running and may execute an instruction that performs an access to a protected ECSL area. If this happens, the ECSL will trip and cause the JTAG debug probe connection to be cut. Two solutions to this problem exist:

1. The first is to use the Wait-In-Reset emulation mode, which will hold the device in reset until the JTAG debug probe takes control. The JTAG debug probe must support this mode for this option.
2. The second option is to use the "Branch to check boot mode" boot option. This will sit in a loop and continuously poll the boot mode select pins. You can select this boot mode and then exit this mode once the JTAG debug probe is connected by re-mapping the PC to another address or by changing the boot mode selection pin to the desired boot mode.

---

**Note****Reserved Flash Locations When Using Code Security**

For code security operation, **all addresses between 0x3F 7F80 and 0x3F 7FF5 cannot be used as program code or data, but must be programmed to 0x0000** when the Code Security Password is programmed. If security is not a concern, then these addresses may be used for code or data. The 128-bit password (at 0x3F 7FF8 - 0x3F 7FFF) must not be programmed to zeros. Doing so would permanently lock the device.

Addresses 0x3F 7FF0 through 0x3F 7FF5 are reserved for data variables and should not contain program code.

---

**Note****Code Security Module Disclaimer**

The Code Security Module ( CSM ) included on this device was designed to password protect the data stored in the associated memory and is warranted by Texas Instruments (TI), in accordance with its standard terms and conditions, to conform to TI's published specifications for the warranty period applicable for this device.

TI DOES NOT, HOWEVER, WARRANT OR REPRESENT THAT THE CSM CANNOT BE COMPROMISED OR BREACHED OR THAT THE DATA STORED IN THE ASSOCIATED MEMORY CANNOT BE ACCESSED THROUGH OTHER MEANS. MOREOVER, EXCEPT AS SET FORTH ABOVE, TI MAKES NO WARRANTIES OR REPRESENTATIONS CONCERNING THE CSM OR OPERATION OF THIS DEVICE, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL TI BE LIABLE FOR ANY CONSEQUENTIAL, SPECIAL, INDIRECT, INCIDENTAL, OR PUNITIVE DAMAGES, HOWEVER CAUSED, ARISING IN ANY WAY OUT OF YOUR USE OF THE CSM OR THIS DEVICE, WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO LOSS OF DATA, LOSS OF GOODWILL, LOSS OF USE OR INTERRUPTION OF BUSINESS OR OTHER ECONOMIC LOSS.

---

## 1.2.2 CSM Impact on Other On-Chip Resources

The CSM affects access to the on-chip resources listed in [Table 1-10](#).

**Table 1-10. Resources Affected by the CSM**

Address	Block
0x00 0A80 - 0x00 0A87	Flash Configuration Registers
0x00 8000 - 0x00 83FF or 0x00 8000 - 0x00 8BFF or 0x00 8000 - 0x00 8FFF	L0 SARAM (1K X 16) L0 SARAM (3K X 16) L0 SARAM (4K X 16)
0x3F 4000 - 0x3F 7FFF or 0x3F 0000 - 0x3F 7FFF	Flash (16K X 16) Flash (32K X 16)
0x3D 7800 - 0x3D 7BFF	User One-Time Programmable (OTP) (1K X 16)
0x3D 7C00 - 0x3D 7FFF	TI One-Time Programmable (OTP) <sup>(1)</sup> (1K X 16)
0x3F 8000 - 0x3F 83FF or 0x3F 8000 - 0x3F 8BFF or 0x3F 8000 - 0x3F 8FFF	L0 SARAM (1K X 16) L0 SARAM (3K X 16) L0 SARAM (4K X 16)

(1) Not affected by ECSL

The Code Security Module has no impact whatsoever on the following on-chip resources:

- Single-access RAM (SARAM) blocks not designated as secure - These memory blocks can be freely accessed and code run from them, whether the device is in secure or unsecure mode.
- Boot ROM contents - Visibility to the boot ROM contents is not impacted by the CSM.
- On-chip peripheral registers - The peripheral registers can be initialized by code running from on-chip or off-chip memory, whether the device is in secure or unsecure mode.
- PIE Vector Table - Vector tables can be read and written regardless of whether the device is in secure or unsecure mode. [Table 1-10](#) and [Table 1-11](#) show which on-chip resources are affected (or are not affected) by the CSM.

**Table 1-11. Resources Not Affected by the CSM**

Address	Block
0x00 0000 - 0x00 03FF	M0 SARAM (1K x 16)
0x00 0400 - 0x00 07FF	M1 SARAM (1K x16)
0x00 0800 - 0x00 0CFF	Peripheral Frame 0 (2K x 16)
0x00 0D00 - 0x00 0FFF	PIE Vector RAM (256 x 16)
0x00 6000 - 0x00 6FFF	Peripheral Frame 1 (4K x 16)
0x00 7000 - 0x00 7FFF	Peripheral Frame 2 (4K x 16)
0x3F E000 0x3F FFFF	Boot ROM (4K x 16)

To summarize, it is possible to load code onto the unprotected on-chip program SARAM using the JTAG connector without any impact from the Code Security Module. The code can be debugged and the peripheral registers initialized, independent of whether the device is in secure or unsecure mode.

### 1.2.3 Incorporating Code Security in User Applications

Code security is typically not employed in the development phase of a project; however, security may be desired once the application code is finalized. Before such a code is programmed in the flash memory, a password should be chosen to secure the device. Once a password is in place, the device is secured (that is, programming a password at the appropriate locations and either performing a device reset or setting the FORCESEC bit (CSMSCR.15) is the action that secures the device). From that time on, access to debug the contents of secure memory by any means (by way of JTAG, code running off external/on-chip memory, and so on) requires the supply of a valid password. A password is not needed to run the code out of secure memory (such as in end-application usage); however, access to secure memory contents for debug purpose requires a password.

If the code security feature is used, any one of the following directives must be used when a function residing in secure memory calls another function which belongs to unsecure memory:

- Use unsecure memory as stack
- Switch stack to unsecure memory before calling the function
- Unlock security before calling the function

Note that the above directives apply for any address-based-parameters passed on to the called function, basically making sure that the called function can read/write to these address-based parameters.

#### 1.2.3.1 Environments That Require Security Unlocking

Following are the typical situations under which unsecuring can be required:

- Code development using debuggers (such as Code Composer Studio ).  
This is the most common environment during the design phase of a product.
- Flash programming using TI's flash utilities such as Code Composer Studio™ Flash Programmer plug-in or Uniflash.

Flash programming is common during code development and testing. Once the user supplies the necessary password, the flash utilities disable the security logic before attempting to program the flash. The flash utilities can disable the code security logic in new devices without any authorization, since new devices come with an erased flash. However, reprogramming devices (that already contain a custom password) require the password to be supplied to the flash utilities in order to unlock the device to enable programming. In custom programming solutions that use the flash API supplied by TI unlocking the CSM can be avoided by executing the flash programming algorithms from secure memory.

- Custom environment defined by the application

In addition to the above, access to secure memory contents can be required in situations such as:

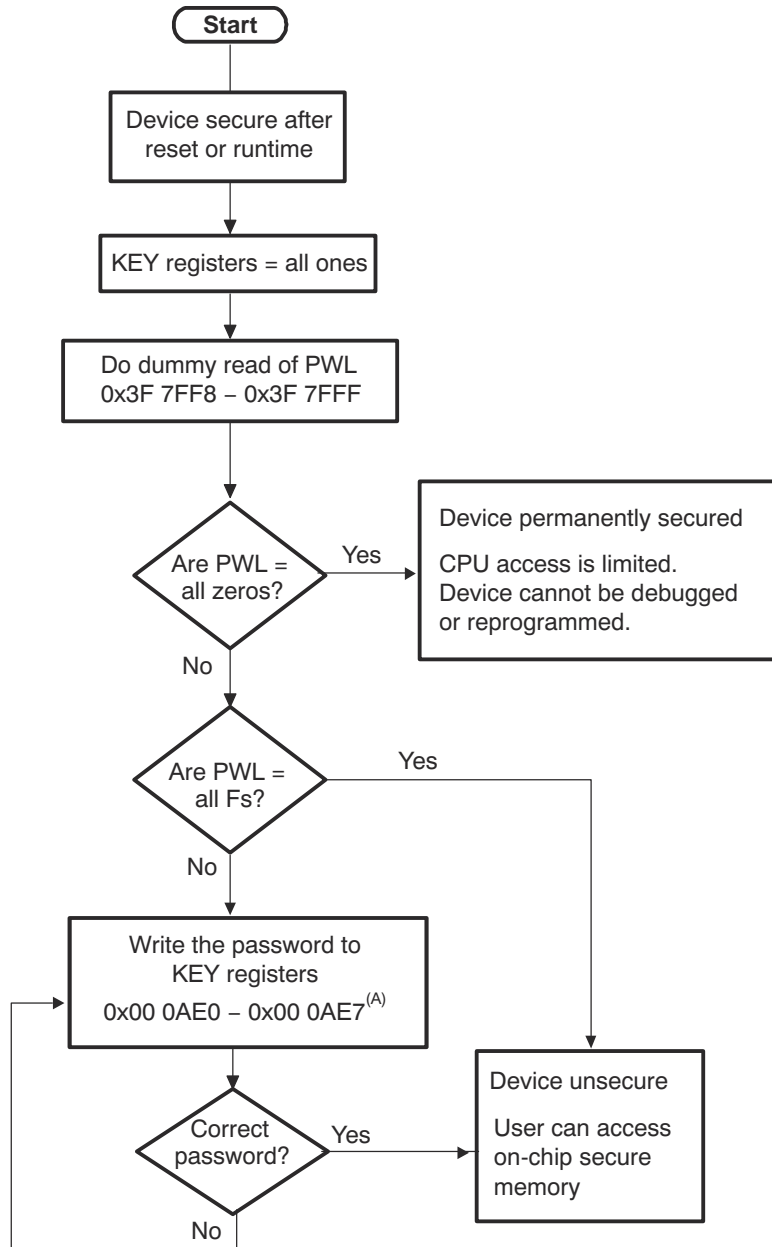
- Using the on-chip bootloader to load code or data into secure SARAM or to erase/program the flash.
- Executing code from on-chip unsecure memory and requiring access to secure memory for lookup table. This is not a suggested operating condition as supplying the password from external code could compromise code security.

The unsecuring sequence is identical in all the above situations. This sequence is referred to as the Password Match Flow (PMF) for simplicity. [Figure 1-11](#) explains the sequence of operation that is required every time the user attempts to unsecure a device. A code example is listed for clarity.

### 1.2.3.2 Password Match Flow

Password match flow (PMF) is essentially a sequence of eight dummy reads from password locations (PWL) followed by eight writes to KEY registers.

Figure 1-11 shows how the PMF helps to initialize the security logic registers and disable security logic.



A. The KEY registers are EALLOW protected.

Figure 1-11. Password Match Flow (PMF)

---

**Note**

**NOTE:** Any read of the CSM password would yield 0x0000 until the device is unlocked. These reads are labeled "dummy read" or a "fake read." The application reads the password locations, but will always get 0's no matter what the actual value is. What is important is the actual value of the password. If the actual value is all 0xFFFF, then doing this "dummy read" will unlock the device. If the actual value is all 0x0000, then no matter what the application code does, you will never be able to unlock the device. If the actual value is something other than all 0xFFFF or 0x0000, then when the dummy read is performed, the actual value must match the password the user provided.

---

### 1.2.3.3 Unsecuring Considerations for Devices With and Without Code Security

Case 1 and Case 2 provide unsecuring considerations for devices with and without code security.

#### Case 1: Device With Code Security

A device with code security should have a predetermined password stored in the password locations (0x3F 7FF8 - 0x3F 7FFF in memory). In addition, locations 0x3F 7F80 - 0x3F 7FF5 should be programmed with all 0x0000 and not used for program and/or data storage. The following are steps to unsecure this device:

1. Perform a dummy read of the password locations.
2. Write the password into the KEY registers (locations 0x00 0AE0 - 0x00 0AE7 in memory).
3. If the password is correct, the device becomes unsecure; otherwise, it stays secure.

#### Case 2: Device Without Code Security

A device without code security should have 0x FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF (128 bits of all ones) stored in the password locations. The following are steps to use this device:

1. At reset, the CSM will lock memory regions protected by the CSM.
2. Perform a dummy read of the password locations.
3. Since the password is all ones, this alone will unlock all memory regions. Secure memory is fully accessible immediately after this operation is completed.

---

**Note**

Even if a device is not protected with a password (all password locations all ones), the CSM will lock at reset. Thus, a dummy read operation must still be performed on these devices prior to reading, writing, or programming secure memory if the code performing the access is executing from outside of the CSM protected memory region. The Boot ROM code does this dummy read for convenience.

---

### 1.2.3.3.1 C Code Example to Unsecure

```
volatile int *CSM = (volatile int *)0x000AE0; //CSM register file
volatile int *PWL = (volatile int *)0x003F7FF8; //Password location
volatile int tmp;
int I;
    // Read the 128-bits of the password locations (PWL)
    // in flash at address 0x3F 7FF8 - 0x3F 7FFF
    // If the device is secure, then the values read will
    // not actually be loaded into the temp variable, so
    // this is called a dummy read.
for (I=0; i<8; I++) tmp = *PWL++;
    // If the password locations (PWL) are all = ones (0xFFFF),
    // then the device will now be unsecure. If the password
    // is not all ones (0xFFFF), then the code below is required
    // to unsecure the CSM.
    // Write the 128-bit password to the KEY registers
    // If this password matches that stored in the
    // PWL then the CSM will become unsecure. If it does not
    // match, then the device will remain secure.
    // An example password of:
    // 0x11112222333344445555666677778888 is used.
asm(" EALLOW"); // Key registers are EALLOW protected
*CSM++ = 0x1111; // Register KEY0 at 0xAE0
*CSM++ = 0x2222; // Register KEY1 at 0xAE1
*CSM++ = 0x3333; // Register KEY2 at 0xAE2
*CSM++ = 0x4444; // Register KEY3 at 0xAE3
*CSM++ = 0x5555; // Register KEY4 at 0xAE4
*CSM++ = 0x6666; // Register KEY5 at 0xAE5
*CSM++ = 0x7777; // Register KEY6 at 0xAE6
*CSM++ = 0x8888; // Register KEY7 at 0xAE7
asm(" EDIS");
```

### 1.2.3.3.2 C Code Example to Resecure

```
volatile int *CSMSCR = 0x00AEF; //CSMSCR register
                                //Set FORCESEC bit
asm(" EALLOW"); //CSMSCR register is EALLOW protected.
*CSMSCR = 0x8000;
asm("EDIS");
```



## 1.2.4 Do's and Don'ts to Protect Security Logic

### 1.2.4.1 Do's

- To keep the debug and code development phase simple, use the device in the unsecure mode; that is, use all 128 bits as 1s in the password locations (or use a password that is easy to remember). Use a password after the development phase when the code is frozen.
- Recheck the password stored in the password locations before programming the COFF file using flash utilities.
- The flow of code execution can freely toggle back and forth between secure memory and unsecure memory without compromising security. To access data variables located in secure memory when the device is secured, code execution must currently be running from secure memory.
- Program locations 0x3F 7F80 - 0x3F 7FF5 with 0x0000 when using the CSM.

### 1.2.4.2 Don'ts

- If code security is desired, do not embed the password in your application anywhere other than in the password locations or security can be compromised.
- Do not use 128 bits of all zeros as the password. This automatically secures the device, regardless of the contents of the KEY register. The device is not debuggable nor reprogrammable.
- Do not pull a reset during an erase operation on the flash array. This can leave either zeros or an unknown value in the password locations. If the password locations are all zero during a reset, the device will always be secure, regardless of the contents of the KEY register.
- Do not use locations 0x3F 7F80 - 0x3F 7FF5 to store program and/or data. These locations should be programmed to 0x0000 when using the CSM.

## 1.2.5 CSM Features - Summary

- The flash is secured after a reset until the password match flow described in [Section 1.2.3.2](#) is executed.
- The standard way of running code out of the flash is to program the flash with the code and power up the device. Since instruction fetches are always allowed from secure memory, regardless of the state of the CSM, the code functions correctly even without executing the password match flow.
- Secure memory cannot be modified by code executing from unsecure memory while the device is secured.
- Secure memory cannot be read from any code running from unsecure memory while the device is secured.
- Secure memory cannot be read or written to by the debugger (Code Composer Studio) at any time that the device is secured.
- Complete access to secure memory from both the CPU code and the debugger is granted while the device is unsecured.

## 1.2.6 CSM Status and Control Registers

**Table 1-12. Code Security Module (CSM) Registers**

Memory Address	Register Name	Reset Values	Register Description
<b>KEY Registers</b>			
0x00 - 0AE0	KEY0 <sup>(1)</sup>	0xFFFF	Low word of the 128-bit KEY register
0x00 - 0AE1	KEY1 <sup>(1)</sup>	0xFFFF	Second word of the 128-bit KEY register
0x00 - 0AE2	KEY2 <sup>(1)</sup>	0xFFFF	Third word of the 128-bit KEY register
0x00 - 0AE3	KEY3 <sup>(1)</sup>	0xFFFF	Fourth word of the 128-bit key
0x00 - 0AE4	KEY4 <sup>(1)</sup>	0xFFFF	Fifth word of the 128-bit key
0x00 - 0AE5	KEY5 <sup>(1)</sup>	0xFFFF	Sixth word of the 128-bit key
0x00 - 0AE6	KEY6 <sup>(1)</sup>	0xFFFF	Seventh word of the 128-bit key
0x00 - 0AE7	KEY7 <sup>(1)</sup>	0xFFFF	High word of the 128-bit KEY register
0x00 - 0AEF	CSMSCR <sup>(1)</sup>	0x002F	CSM status and control register
<b>Password Locations (PWL) in Flash Memory - Reserved for the CSM password only</b>			
0x3F - 7FF8	PWL0	User defined	Low word of the 128-bit password
0x3F - 7FF9	PWL1	User defined	Second word of the 128-bit password
0x3F - 7FFA	PWL2	User defined	Third word of the 128-bit password
0x3F - 7FFB	PWL3	User defined	Fourth word of the 128-bit password
0x3F - 7FFC	PWL4	User defined	Fifth word of the 128-bit password
0x3F - 7FFD	PWL5	User defined	Sixth word of the 128-bit password
0x3F - 7FFE	PWL6	User defined	Seventh word of the 128-bit password
0x3F - 7FFF	PWL7	User defined	High word of the 128-bit password

(1) These registers are EALLOW protected. Refer to [Section 1.5.2](#) for more information.

### 1.2.6.1 CSM Status and Control Register (CSMSCR)

**Figure 1-12. CSM Status and Control Register (CSMSCR)**

15	14	1	0
FORCESEC	Reserved	SECURE	
W-0	R-0x002E	R-1	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-13. CSM Status and Control Register (CSMSCR) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
15	FORCESEC	0	Writing a 1 clears the KEY registers and secures the device. A read always returns a zero.
		1	Clears the KEY registers and secures the device. The password match flow described in <a href="#">Section 1.2.3.2</a> must be followed to unsecure the device again.
14-1	Reserved		Reserved
0	SECURE	0	Read-only bit that reflects the security state of the device. Device is unsecure (CSM unlocked).
		1	Device is secure (CSM locked).

(1) This register is EALLOW protected. Refer to [Section 1.5.2](#) for more information.



**Table 1-14. PLL, Clocking, Watchdog, and Low-Power Mode Registers (continued)**

Name	Address	Size (x16)	Description	Bit Description
PLLCR	0x0000-7021	1	PLL Control Register	<a href="#">Section 1.3.2.4.1.1</a>
SCSR	0x0000-7022	1	System Control and Status Register	<a href="#">Section 1.3.4.5.1</a>
WDCNTR	0x0000-7023	1	Watchdog Counter Register	<a href="#">Section 1.3.4.5.2</a>
WDKEY	0x0000-7025	1	Watchdog Reset Key Register	<a href="#">Section 1.3.4.5.3</a>
WDCR	0x0000-7029	1	Watchdog Control Register	<a href="#">Section 1.3.4.5.4</a>
BORCFG	0x000985	1	BOR Configuration Register	<a href="#">Section 1.7.2</a>

### 1.3.1.1 Enabling/Disabling Clocks to the Peripheral Modules (PCLKCR0/1/3)

The (PCLKCR0/1/3) registers enable and disable clocks to the various peripheral modules. There is a 2-SYCLKOUT cycle delay from when a write to the (PCLKCR0/1/3) registers occurs to when the action is valid. This delay must be taken into account before attempting to access the peripheral configuration registers. Due to the peripheral-GPIO multiplexing at the pin level, all peripherals cannot be used at the same time. While it is possible to turn on the clocks to all the peripherals at the same time, such a configuration may not be useful. If this is done, the current drawn will be more than required. To avoid this, only enable the clocks required by the application.

**Figure 1-14. Peripheral Clock Control 0 Register (PCLKCR0)**

15	11	10	9	8		
Reserved			SCIA ENCLK	Reserved	SPIA ENCLK	
R-0			R/W-0	R-0	R/W-0	
7	5	4	3	2	1	0
Reserved		I2CA ENCLK	ADC ENCLK	TBCLK SYNC	Reserved	HRPWM ENCLK
R-0		R/W-0	R/W-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-15. Peripheral Clock Control 0 Register (PCLKCR0) Field Descriptions**

Bit	Field	Value	Description
15-11	Reserved		Any writes to these bit(s) must always have a value of 0.
10	SCIAENCLK	0 1	SCI-A clock enable The SCI-A module is not clocked. (default) <sup>(1)</sup> The SCI-A module is clocked by the low-speed clock (LSPCLK).
9	Reserved		Any writes to these bit(s) must always have a value of 0.
8	SPIAENCLK	0 1	SPI-A clock enable The SPI-A module is not clocked. (default) <sup>(1)</sup> The SPI-A module is clocked by the low-speed clock (LSPCLK).
7-5	Reserved		Any writes to these bit(s) must always have a value of 0.

**Table 1-15. Peripheral Clock Control 0 Register (PCLKCR0) Field Descriptions (continued)**

Bit	Field	Value	Description
4	I2CAENCLK		I <sup>2</sup> C clock enable
		0	The I <sup>2</sup> C module is not clocked. (default) <sup>(1)</sup>
3	ADCENCLK		ADC clock enable
		0	The ADC is not clocked. (default) <sup>(1)</sup>
2	TBCLKSYNC		ePWM Module Time Base Clock (TBCLK) Sync: Allows the user to globally synchronize all enabled ePWM modules to the time base clock (TBCLK):
		0	The TBCLK (Time Base Clock) within each enabled ePWM module is stopped. (default). If, however, the ePWM clock enable bit is set in the PCLKCR1 register, then the ePWM module will still be clocked by SYSCLKOUT even if TBCLKSYNC is 0.
1	Reserved		Any writes to these bit(s) must always have a value of 0.
		1	All enabled ePWM module clocks are started with the first rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescaler bits in the TBCTL register of each ePWM module must be set identically. The proper procedure for enabling ePWM clocks is as follows: <ul style="list-style-type: none"> <li>• Enable ePWM module clocks in the PCLKCR1 register.</li> <li>• Set TBCLKSYNC to 0.</li> <li>• Configure prescaler values and ePWM modes.</li> <li>• Set TBCLKSYNC to 1.</li> </ul>
0	HRPWMENCLK		HRPWM clock enable
		0	HRPWM is not enabled.
0	HRPWMENCLK	1	HRPWM is enabled.

(1) If a peripheral block is not used, the clock to that peripheral can be turned off to minimize power consumption.

**Figure 1-15. Peripheral Clock Control 1 Register (PCLKCR1)**

15					9	8
Reserved					ECAP1 ENCLK	
R-0					R/W-0	
7	4	3	2	1	0	
Reserved		EPWM4 ENCLK	EPWM3 ENCLK	EPWM2 ENCLK	EPWM1 ENCLK	
R-0		R/W-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-16. Peripheral Clock Control 1 Register (PCLKCR1) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
15-9	Reserved		Any writes to these bit(s) must always have a value of 0.
8	ECAP1ENCLK	0 1	eCAP1 clock enable The eCAP1 module is not clocked. (default) <sup>(2)</sup> The eCAP1 module is clocked by the system clock (SYSCLKOUT).
7-4	Reserved		Any writes to these bit(s) must always have a value of 0.
3	EPWM4ENCLK	0 1	ePWM4 clock enable. <sup>(3)</sup> The ePWM4 module is not clocked. (default) <sup>(2)</sup> The ePWM4 module is clocked by the system clock (SYSCLKOUT).
2	EPWM3ENCLK	0 1	ePWM3 clock enable. <sup>(3)</sup> The ePWM3 module is not clocked. (default) <sup>(2)</sup> The ePWM3 module is clocked by the system clock (SYSCLKOUT).
1	EPWM2ENCLK	0 1	ePWM2 clock enable. <sup>(3)</sup> The ePWM2 module is not clocked. (default) <sup>(2)</sup> The ePWM2 module is clocked by the system clock (SYSCLKOUT).
0	EPWM1ENCLK	0 1	ePWM1 clock enable. <sup>(3)</sup> The ePWM1 module is not clocked. (default) <sup>(2)</sup> The ePWM1 module is clocked by the system clock (SYSCLKOUT).

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) If a peripheral block is not used, the clock to that peripheral can be turned off to minimize power consumption.

(3) To start the ePWM Time-base clock (TBCLK) within the ePWM modules, the TBCLKSYNC bit in PCLKCR0 must also be set.

**Figure 1-16. Peripheral Clock Control 3 Register (PCLKCR3)**

15	14	13	12	11	10	9	8	
Reserved		GPIOIN ENCLK	Reserved		CPUTIMER2 ENCLK	CPUTIMER1 ENCLK	CPUTIMER0 ENCLK	
R-0		R/W-1	R-0		R/W-1	R/W-1	R/W-1	
7						2	1	0
Reserved						COMP2 ENCLK	COMP1 ENCLK	
R-0						R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-17. Peripheral Clock Control 3 Register (PCLKCR3) Field Descriptions**

Bit	Field	Value	Description
15-14	Reserved		Any writes to these bit(s) must always have a value of 0.
13	GPIOINENCLK	0 1	GPIO Input Clock Enable GPIO Module is not clocked. GPIO Module is clocked.
12-11	Reserved		Any writes to these bit(s) must always have a value of 0.
10	CPUTIMER2ENCLK	0 1	CPU Timer 2 Clock Enable CPU Timer 2 is not clocked. CPU Timer 2 is clocked.
9	CPUTIMER1ENCLK	0 1	CPU Timer 1 Clock Enable CPU Timer 1 is not clocked. CPU Timer 1 is clocked.
8	CPUTIMER0ENCLK	0 1	CPU Timer 0 Clock Enable CPU Timer 0 is not clocked. CPU Timer 0 is clocked.
7-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1	COMP2ENCLK	0 1	Comparator2 clock enable Comparator2 is not clocked Comparator2 is clocked
0	COMP1ENCLK	0 1	Comparator1 clock enable Comparator1 is not clocked Comparator1 is clocked



### 1.3.1.2 Configuring the Low-Speed Peripheral Clock Prescaler (LOSPCP)

The low-speed peripheral clock prescale (LOSPCP) registers are used to configure the low-speed peripheral clocks. See [Figure 1-17](#) for the LOSPCP layout.

**Figure 1-17. Low-Speed Peripheral Clock Prescaler Register (LOSPCP)**

15	3	2	0
Reserved		LSPCLK	
R-0		R/W-010	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-18. Low-Speed Peripheral Clock Prescaler Register (LOSPCP) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
15-3	Reserved		Any writes to these bit(s) must always have a value of 0.
2-0	LSPCLK		These bits configure the low-speed peripheral clock (LSPCLK) rate relative to SYSCLKOUT: If LOSPCP <sup>(2)</sup> ≠ 0, then LSPCLK = SYSCLKOUT/(LOSPCP X 2) If LOSPCP = 0, then LSPCLK = SYSCLKOUT
		000	Low speed clock = SYSCLKOUT/1
		001	Low speed clock = SYSCLKOUT/2
		010	Low speed clock = SYSCLKOUT/4 (reset default)
		011	Low speed clock = SYSCLKOUT/6
		100	Low speed clock = SYSCLKOUT/8
		101	Low speed clock = SYSCLKOUT/10
		110	Low speed clock = SYSCLKOUT/12
		111	Low speed clock = SYSCLKOUT/14

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) LOSPCP in this equation denotes the value of bits 2:0 in the LOSPCP register.

## 1.3.2 OSC and PLL Block

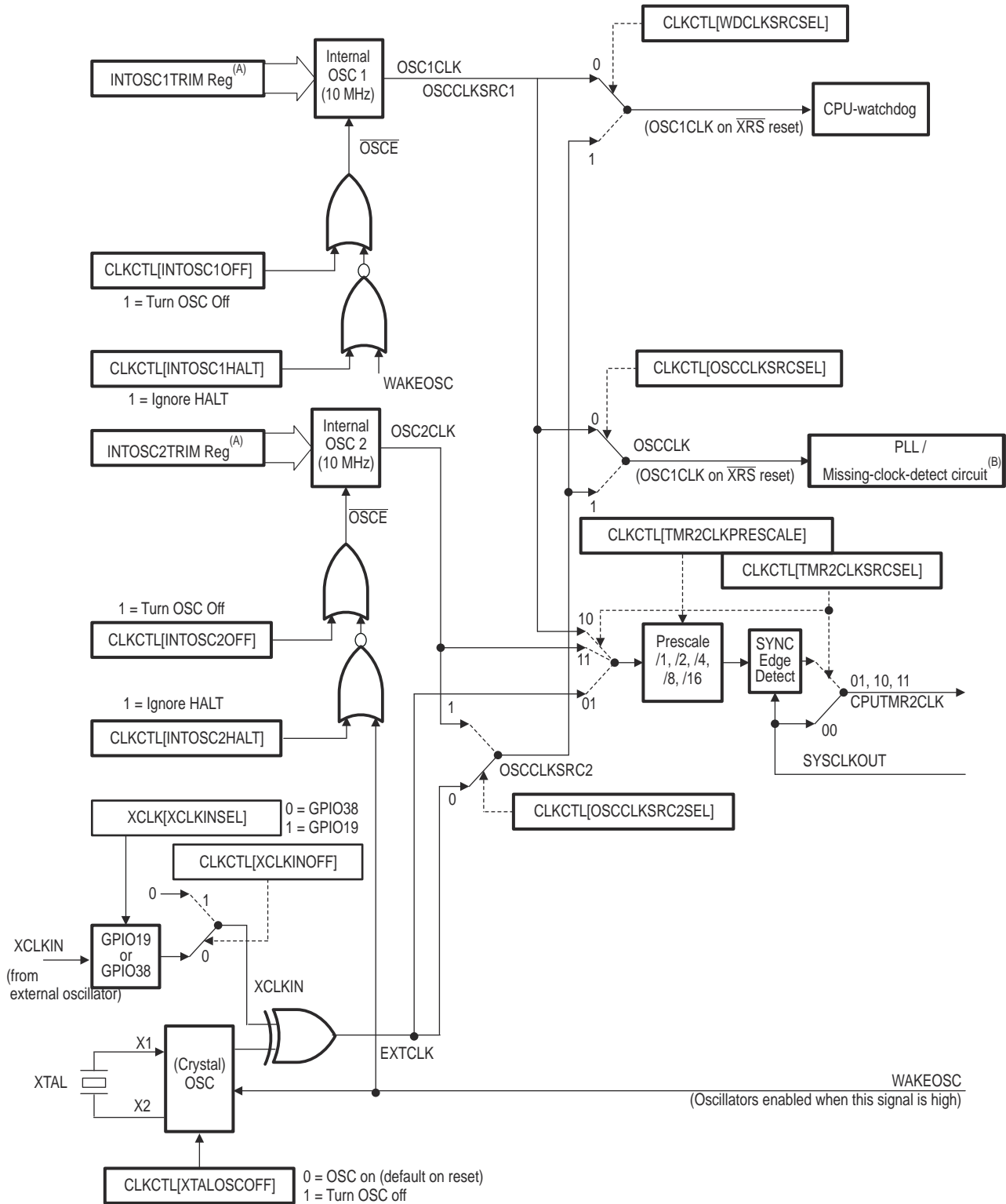
The on-chip oscillator and phase-locked loop (PLL) block provide the clocking signals for the device, as well as control for low-power mode (LPM) entry or exit.

### 1.3.2.1 Input Clock Options

The device has two internal oscillators (INTOSC1 and INTOSC2) that need no external components. It also has an on-chip, PLL-based clock module. [Figure 1-18](#) shows the different options that are available to clock the device. Following are the input clock options available:

- **INTOSC1 (Internal zero-pin Oscillator 1):** This is the on-chip internal oscillator 1. It can provide the clock for the Watchdog block, CPU-core and CPU-Timer 2. This is the default clock source upon reset.
- **INTOSC2 (Internal zero-pin Oscillator 2):** This is the on-chip internal oscillator 2. It can provide the clock for the Watchdog block, CPU-core and CPU-Timer 2. Both INTOSC1 and INTOSC2 can be independently chosen for the Watchdog block, CPU-core, and CPU-Timer 2. If using INTOSC2 as a clock source, refer to the *Advisory Oscillator: CPU clock switching to INTOSC2 may result in missing clock condition after reset* in the device errata.
- **XTAL OSC (Crystal or Resonator):** The on-chip crystal oscillator enables the use of an external quartz crystal or ceramic resonator. The crystal or resonator is connected to the X1/X2 pins.
- **XCLKIN (External clock source):** If the on-chip crystal oscillator is not used, this mode allows it to be bypassed. The device clock is generated from an external clock source input on the XCLKIN pin. Note that the XCLKIN is multiplexed with GPIO19 or GPIO38 pin. The XCLKIN input can be selected as GPIO19 or GPIO38 via the XCLKINSEL bit in XCLK register. The CLKCTL[XCLKINOFF] bit disables this clock input

(forced low). If the clock source is not used or the respective pins are used as GPIOs, the user should disable it at boot time.



A. Register loaded from TI OTP-based calibration function.

Figure 1-18. Clocking Options

### 1.3.2.1.1 Trimming INTOSCn

The nominal frequency of both INTOSC1 and INTOSC2 is 10 MHz. Two 16-bit registers are provided for trimming each oscillator at manufacturing time (called coarse trim) and also to provide you with a way to trim the oscillator using software (called fine trim). The bit layout for both registers is the same so only one is shown with "n" in place of the numbers 1 or 2.

**Figure 1-19. Internal Oscillator n Trim (INTOSCnTRIM) Register**

15	14	9	8	7	0
Reserved	FINETRIM	Reserved	COARSETRIM		
R-0	R/W-0	R-0	R/W-0		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-19. Internal Oscillator n Trim (INTOSCnTRIM) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1) (2)</sup>
15	Reserved		Any writes to these bit(s) must always have a value of 0.
14-9	FINETRIM		6-bit Fine Trim Value: Signed magnitude value (-31 to +31)
8	Reserved		Any writes to these bits must always have a value of 0.
7-0	COARSETRIM		8-bit Coarse Trim Value: Signed magnitude value (-127 to +127)

(1) These registers are EALLOW protected.

(2) The internal oscillators are software trimmed with parameters stored in OTP. During boot time, the boot-ROM copies this value to the above registers.

### 1.3.2.1.2 Device\_Cal

The Device\_cal() routine is programmed into TI reserved memory by the factory. The boot ROM automatically calls the Device\_cal() routine to calibrate the internal oscillators and ADC with device specific calibration data. During normal operation, this process occurs automatically and no action is required by the user.

If the boot ROM is bypassed by Code Composer Studio during the development process, then the calibration must be initialized by application. For working examples, see the system initialization in C2000Ware.

#### Note

Failure to initialize these registers will cause the oscillators and ADC to function out of specification. The following three steps describe how to call the Device\_cal routine from an application.

Step 1: Create a pointer to the Device\_cal function as shown in [Example 1-1](#). This #define is included in the Header Files and Peripheral Examples.

Step 2: Call the function pointed to by Device\_cal() as shown in [Example 1-1](#). The ADC clocks must be enabled before making this call.

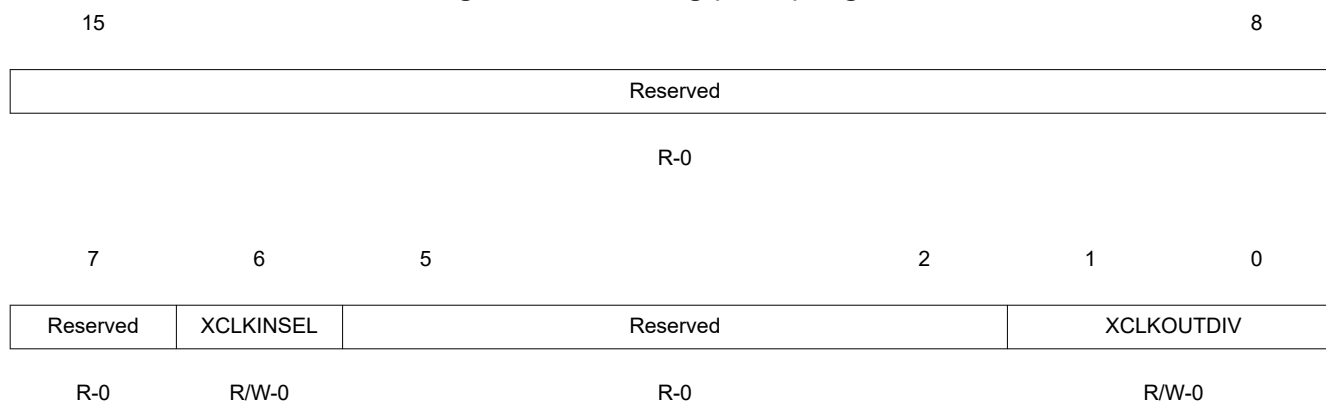
**Example 1-1. Calling the Device\_cal() function**

```

//Device_cal is a pointer to a function
//that begins at the address shown
# define Device_cal (void(*) (void))0x3D7C80
... ..
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
    (*Device_cal)();
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0;
    EDIS;
...
    
```

**1.3.2.2 Configuring Input Clock Source and XCLKOUT Options (XCLK)**

The XCLK register is used to choose the GPIO pin for XCLKIN input and to configure the XCLKOUT pin frequency.

**Figure 1-20. Clocking (XCLK) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-20. Clocking (XCLK) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-7	Reserved		Any writes to these bit(s) must always have a value of 0.
6	XCLKINSEL	0 1	XCLKIN Source Select Bit: This bit selects the source 0 GPIO38 is XCLKIN input source (this is also the JTAG port TCK source) 1 GPIO19 is XCLKIN input source
5-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1-0	XCLKOUTDIV <sup>(2)</sup>	00 01 10 11	XCLKOUT Divide Ratio: These two bits select the XCLKOUT frequency ratio relative to SYSCLKOUT. The ratios are: 00 XCLKOUT = SYSCLKOUT/4 01 XCLKOUT = SYSCLKOUT/2 10 XCLKOUT = SYSCLKOUT 11 XCLKOUT = Off

(1) The XCLKINSEL bit in the XCLK register is reset by  $\overline{XRS}$  input signal.

(2) Refer to the device data sheet for the maximum permissible XCLKOUT frequency.

### 1.3.2.3 Configuring Device Clock Domains (CLKCTL)

The CLKCTL register is used to choose between the available clock sources and also configure device behavior during clock failure.

**Figure 1-21. Clock Control (CLKCTL) Register**

15	14	13	12	11	10	9	8
NMIRESETSEL	XTALOSCOFF	XCLKINOFF	WDHALTI	INTOSC2HALTI	INTOSC2OFF	INTOSC1HALTI	INTOSC1OFF
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	5	4	3	2	1	0	
TMR2CLKPRESCALE		TMR2CLKSRCSEL		WDCLKSRCSEL	OSCCLKSRC2SEL	OSCCLKSRCSEL	
R/W-0		R/W-0		R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-21. Clock Control (CLKCTL) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15	NMIRESETSEL	0 1	NMI Reset Select Bit: This bit selects between generating the $\overline{\text{MCLKRS}}$ signal directly when a missing clock condition is detected or the $\overline{\text{NMIRS}}$ reset is used: 0 $\overline{\text{MCLKRS}}$ is driven without any delay (default on reset) 1 NMI Watchdog Reset ( $\overline{\text{NMIRS}}$ ) initiates $\overline{\text{MCLKRS}}$ <b>Note:</b> The $\overline{\text{CLOCKFAIL}}$ signal is generated regardless of this mode selection.
14	XTALOSCOFF	0 1	Crystal Oscillator Off Bit: This bit could be used to turn off the crystal oscillator if it is not used. 0 Crystal oscillator on (default on reset) 1 Crystal oscillator off
13	XCLKINOFF	0 1	XCLKIN Off Bit: This bit turns external XCLKIN oscillator input off: 0 XCLKIN oscillator input on (default on reset) 1 XCLKIN oscillator input off <b>Note:</b> You need to select XCLKIN GPIO pin source via the XCLKINSEL bit in the XCLK register. See the XCLK register description for more details. XTALOSCOFF must be set to 1 if XCLKIN is used.
12	WDHALTI	0 1	Watchdog HALT Mode Ignore Bit: This bit selects if the watchdog is automatically turned off by the HALT mode or not turned off. This feature can be used to allow the selected watchdog clock source to continue clocking the watchdog when HALT mode is active. This would enable the watchdog to periodically wake up the device. 0 Watchdog Automatically Turned Off By HALT (default on reset) 1 Watchdog continues to function in HALT mode
11	INTOSC2HALTI	0 1	Internal Oscillator 2 HALT Mode Ignore Bit: This bit selects if the internal oscillator 2 is automatically turned off by the HALT mode or not. This feature can be used to allow the internal oscillator to continue clocking when HALT mode is active. This would enable a quicker wake-up from HALT. 0 Internal Oscillator 2 Automatically Turned Off By HALT (default on reset) 1 Internal Oscillator 2 continues to function in HALT mode. This feature can be used to allow the internal oscillator to continue clocking when HALT mode is active. This would enable a quicker wake-up from HALT.

**Table 1-21. Clock Control (CLKCTL) Register Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
10	INTOSC2OFF	0 1	Internal Oscillator 2 Off Bit: This bit turns oscillator 2 off: 0 Internal Oscillator 2 On (default on reset) 1 Internal Oscillator 2 Off. This bit could be used by the user to turn off the internal oscillator 2 if it is not used. This selection is not affected by the missing clock detect circuit.
9	INTOSC1HALTI	0 1	Internal Oscillator 1 HALT Mode Ignore Bit: This bit selects if the internal oscillator 1 is automatically turned off by the HALT mode or not: 0 Internal Oscillator 1 Automatically Turned Off By HALT (default on reset) 1 Internal Oscillator 1 continues to function in HALT mode. This feature can be used to allow the internal oscillator to continue clocking when HALT mode is active. This would enable a quicker wake-up from HALT.
8	INTOSC1OFF	0 1	Internal Oscillator 1 Off Bit: This bit turns oscillator 1 off: 0 Internal Oscillator 1 On (default on reset) 1 Internal Oscillator 1 Off. This bit could be used by the user to turn off the internal oscillator 1 if it is not used. This selection is not affected by the missing clock detect circuit.
7-5	TMR2CLKPRESCALE	000 001 010 011 100 101 110 111	CPU Timer 2 Clock Pre-Scale Value: These bits select the pre-scale value for the selected clock source for CPU Timer 2. This selection is not affected by the missing clock detect circuit. /1 (default on reset) /2 /4 /8 /16 Reserved Reserved Reserved
4-3	TMR2CLKSRCSEL	00 01 10 11	CPU Timer 2 Clock Source Select Bit: This bit selects the source for CPU Timer 2: 00 SYSCLOCKOUT Selected (default on reset, pre-scaler is bypassed) 01 External Oscillator Selected (at XOR output) 10 Internal Oscillator 1 Selected 11 Internal Oscillator 2 Selected. This selection is not affected by the missing clock detect circuit.
2	WDCLKSRCSEL	0 1	Watchdog Clock Source Select Bit: This bit selects the source for the watchdog clock. On $\overline{XRS}$ low and after $\overline{XRS}$ goes high, internal oscillator 1 is selected by default. User would need to select external oscillator or Internal Oscillator 2 during their initialization process. If missing clock detect circuit detects a missing clock, then this bit is forced to 0 and internal oscillator 1 is selected. The user changing this bit does not affect the PLLCR value. 0 Internal Oscillator 1 Selected (default on reset) 1 External Oscillator or Internal Oscillator 2 Selected
1	OSCCLKSRC2SEL	0 1	Oscillator 2 Clock Source Select Bit: This bit selects between internal oscillator 2 or external oscillator. This selection is not affected by the missing clock detect circuit. 0 External Oscillator Selected (default on reset) 1 Internal Oscillator 2 Selected

**Table 1-21. Clock Control (CLKCTL) Register Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
0	OSCCLKSRCSEL		Oscillator Clock Source Select Bit. This bit selects the source for OSCCLK. On $\overline{XRS}$ low and after $\overline{XRS}$ goes high, internal oscillator 1 is selected by default. User would need to select external oscillator or Internal Oscillator 2 during their initialization process. Whenever the user changes the clock source using these bits, the PLLCR register will be automatically forced to zero. This prevents potential PLL overshoot. The user will then have to write to the PLLCR register to configure the appropriate PLL multiplier value. The user can also configure the PLL lock period using the PLLLOCKPRD register to reduce the lock time if necessary. If missing clock detect circuit detects a missing clock, then this bit is automatically forced to 0 and internal oscillator 1 is selected. The PLLCR register will also be automatically forced to zero to prevent any potential overshoot.
		0	Internal Oscillator 1 Selected (default on reset)
		1	External Oscillator or Internal Oscillator 2 Selected. Note: If users wish to use Oscillator 2 or External Oscillator to clock the CPU, they should configure the OSCCLKSRC2SEL bit first, and then write to the OSCCLKSRCSEL bit next.

(1) The internal oscillators are software trimmed with parameters stored in OTP. During boot time, the boot-ROM copies this value to the above registers.

### 1.3.2.3.1 Switching the Input Clock Source

The following procedure may be used to switch clock sources:

1. Use CPU Timer 2 to detect if clock sources are functional.
2. If any of the clock sources is not functional, turn off the respective clock source (using the respective CLKCTL bit).
3. Switch over to a new clock source.
4. If clock source switching occurred while in Limp Mode, then write a 1 to MCLKCLR to exit Limp Mode.

If OSCCLKSRC2 (an external Crystal [XTAL] or oscillator [XCLKIN input] or Internal Oscillator 2 [INTOSC2]) is selected as the clock source and a missing clock is detected, the missing clock detect circuit will automatically switch to Internal Oscillator 1 (OSCCLKSRC1) and generate a CLOCKFAIL signal. In addition, the PLLCR register is forced to zero (PLL is bypassed) to prevent any potential overshoot. The user can then write to the PLLCR register to re-lock the PLL. Under this situation, the missing clock detect circuit will be automatically re-enabled (PLLSTS[MCLKSTS] bit will be automatically cleared). If Internal Oscillator 1 (OSCCLKSRC1) should also fail, then under this situation, the missing clock detect circuit will remain in limp mode. The user will have to re-enable the logic via the PLLSTS[MCLKCLR] bit.

### 1.3.2.3.2 Switching to INTOSC2 in the Absence of External Clocks

For the device to work properly upon a switch from INTOSC1 to INTOSC2 in the absence of any external clock, the application code needs to write a 1 to the CLKCTL.XTALOSCOFF and CLKCTL.XCLKINOFF bits first. This is to indicate to the clock switching circuitry that external clocks are not present. Only after this should the OSCCLKSRCSEL and OSCCLKSRC2SEL bits be written to. Note that this sequence should be separated into two writes as follows:

First write → CLKCTL.XTALOSCOFF=1 and CLKCTL.XCLKINOFF=1

Second write → CLKCTL.OSCCLKSRCSEL=1 and CLKCTL.OSCCLKSRC2SEL=1

The second write should not alter the values of XTALOSCOFF and XCLKINOFF bits. If *C2000Ware*, supplied by Texas Instruments is used, clock switching can be achieved with the following code snip:

```
SysCtrlRegs.CLKCTL.all = 0x6000; // Set XTALOSCOFF=1 & XCLKINOFF=1
SysCtrlRegs.CLKCTL.all = 0x6003; // Set OSCCLKSRCSEL=1 & OSCCLKSRC2SEL=1
```

The system initialization file DSP2802x\_SysCtrl.c, provided as part of *C2000Ware* also contain functions to switch to different clock sources. If an attempt is made to switch from INTOSC1 to INTOSC2 without the write to the XTALOSCOFF and XCLKINOFF bits, a missing clock will be detected due to the absence of external clock source (even after the proper source selection). The PLLCR will be zeroed out and the device will automatically clear the MCLKSTS bit and switch back INTOSC1.



### 1.3.2.4 PLL-based Clock Module

Figure 1-22 shows the OSC and PLL block diagram.

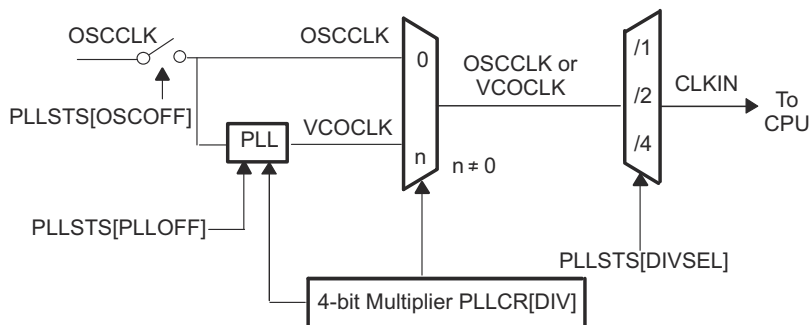


Figure 1-22. OSC and PLL Block

For devices that have X1 and X2 pins and when using XCLKIN as the external clock source, you must connect X1 low and leave X2 disconnected.

Table 1-22. Possible PLL Configuration Modes

PLL Mode	Remarks	PLLSTS[DIVSEL] <sup>(1)</sup>	CLKIN and SYSCLKOUT <sup>(2)</sup>
PLL Off	Invoked by the user setting the PLLOFF bit in the PLLSTS register. The PLL block is disabled in this mode. The CPU clock (CLKIN) can then be derived directly from any one of the following sources: INTOSC1, INTOSC2, XCLKIN pin or X1/X2 pins. This can be useful to reduce system noise and for low power operation. The PLLCR register must first be set to 0x0000 (PLL Bypass) before entering this mode. The CPU clock (CLKIN) is derived directly from the input clock on either X1/X2, X1 or XCLKIN.	0, 1 2 3	OSCCLK/4 OSCCLK/2 OSCCLK/1
PLL Bypass	PLL Bypass is the default PLL configuration upon power-up or after an external reset (XRS). This mode is selected when the PLLCR register is set to 0x0000 or while the PLL locks to a new frequency after the PLLCR register has been modified. In this mode, the PLL itself is bypassed but the PLL is not turned off.	0, 1 2 3	OSCCLK/4 OSCCLK/2 OSCCLK/1
PLL Enabled	Achieved by writing a non-zero value n into the PLLCR register. Upon writing to the PLLCR, the device will switch to PLL Bypass mode until the PLL locks.	0, 1 2 3	OSCCLK*n/4 OSCCLK*n/2 OSCCLK*n/1

(1) PLLSTS[DIVSEL] must be 0 before writing to the PLLCR and should be changed only after PLLSTS[PLLLOCKS] = 1. See Figure 1-23.  
 (2) The input clock and PLLCR[DIV] bits should be chosen in such a way that the output frequency of the PLL (VCOCLK) is a minimum of 50 MHz.

#### 1.3.2.4.1 PLL Control Registers

The PLLCR register is used to change the PLL multiplier of the device. Before writing to the PLLCR register, the following requirements must be met:

- The PLLSTS[DIVSEL] bit must be 0 (CLKIN divide by 4 enabled). Change PLLSTS[DIVSEL] only after the PLL has completed locking, that is, after PLLSTS[PLLLOCKS] = 1.

Once the PLL is stable and has locked at the new specified frequency, the PLL switches CLKIN to the new value as shown in Table 1-23. When this happens, the PLLLOCKS bit in the PLLSTS register is set, indicating that the PLL has finished locking and the device is now running at the new frequency. User software can monitor the PLLLOCKS bit to determine when the PLL has completed locking. Once PLLSTS[PLLLOCKS] = 1, DIVSEL can be changed.

Follow the procedure in Figure 1-23 any time you are writing to the PLLCR register.

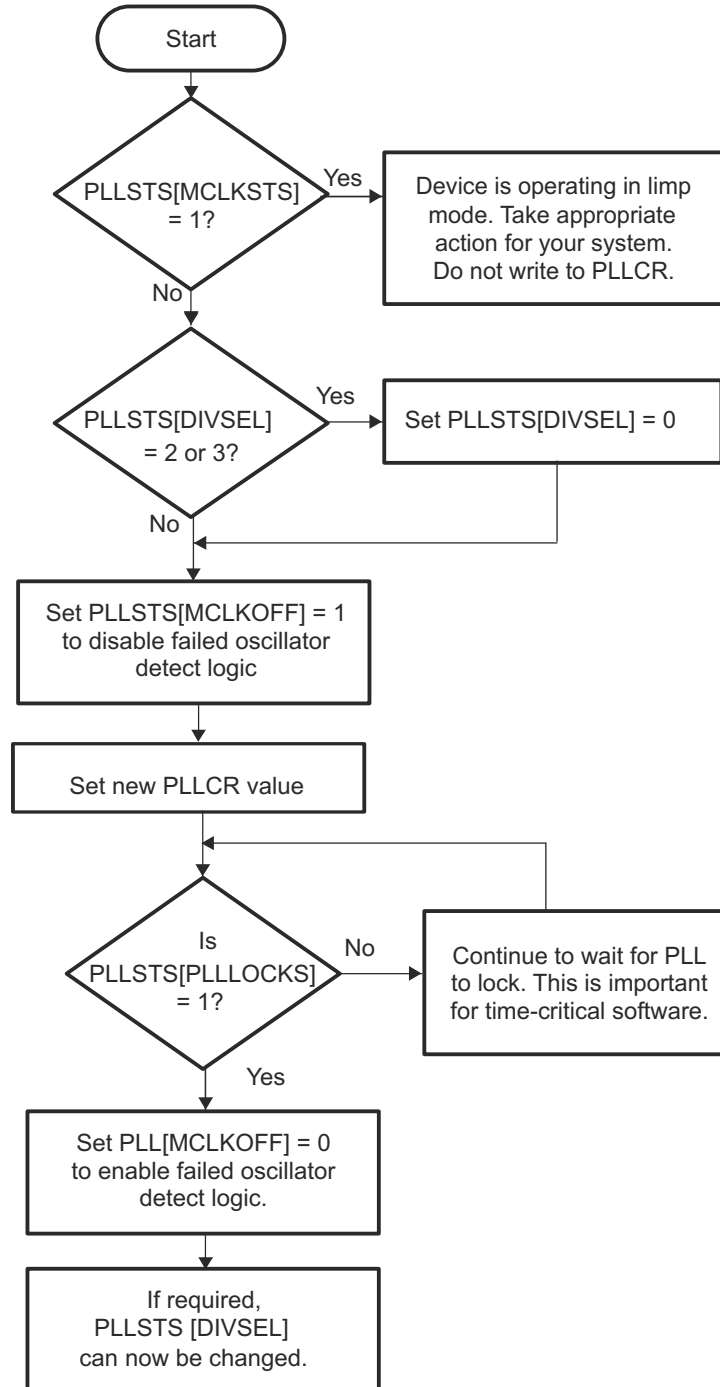


Figure 1-23. PLLCR Change Procedure Flow Chart

### 1.3.2.4.1.1 PLL Control Register (PLLCR)

The DIV field in the PLLCR register controls whether the PLL is bypassed or not and sets the PLL clocking ratio when it is not bypassed. PLL bypass is the default mode after reset. Do not write to the DIV field if the PLLSTS[DIVSEL] bit is 10 or 11, or if the PLL is operating in limp mode as indicated by the PLLSTS[MCLKSTS] bit being set. See the procedure for changing the PLLCR described in [Figure 1-23](#).

**Figure 1-24. PLL Control Register (PLLCR)**

15	4	3	0
Reserved		DIV	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-23. PLL Settings**

PLLCR[DIV] Value <sup>(2)</sup>	SYSCLOCKOUT (CLKIN) <sup>(1)</sup>		
	PLLSTS[DIVSEL] = 0 or 1	PLLSTS[DIVSEL] = 2	PLLSTS[DIVSEL] = 3
0000 (PLL bypass)	OSCCLK/4 (Default)	OSCCLK/2	OSCCLK/1
0001	(OSCCLK * 1)/4	(OSCCLK * 1)/2	(OSCCLK * 1)/1
0010	(OSCCLK * 2)/4	(OSCCLK * 2)/2	(OSCCLK * 2)/1
0011	(OSCCLK * 3)/4	(OSCCLK * 3)/2	(OSCCLK * 3)/1
0100	(OSCCLK * 4)/4	(OSCCLK * 4)/2	(OSCCLK * 4)/1
0101	(OSCCLK * 5)/4	(OSCCLK * 5)/2	(OSCCLK * 5)/1
0110	(OSCCLK * 6)/4	(OSCCLK * 6)/2	(OSCCLK * 6)/1
0111	(OSCCLK * 7)/4	(OSCCLK * 7)/2	(OSCCLK * 7)/1
1000	(OSCCLK * 8)/4	(OSCCLK * 8)/2	(OSCCLK * 8)/1
1001	(OSCCLK * 9)/4	(OSCCLK * 9)/2	(OSCCLK * 9)/1
1010	(OSCCLK * 10)/4	(OSCCLK * 10)/2	(OSCCLK * 10)/1
1011	(OSCCLK * 11)/4	(OSCCLK * 11)/2	(OSCCLK * 11)/1
1100	(OSCCLK * 12)/4	(OSCCLK * 12)/2	(OSCCLK * 12)/1
1101-1111	Reserved	Reserved	Reserved

- (1) PLLSTS[DIVSEL] must be 0 or 1 before writing to the PLLCR and should be changed only after PLLSTS[PLLLOCKS] = 1. See [Figure 1-23](#).
- (2) The PLL control register (PLLCR) and PLL Status Register (PLLSTS) are reset to their default state by the  $\overline{XRS}$  signal or a watchdog reset only. A reset issued by the debugger or the missing clock detect logic have no effect.

### 1.3.2.4.1.2 PLL Status (PLLSTS) Register

**Figure 1-25. PLL Status (PLLSTS) Register**

15	14						9	8
NORMRDYE	Reserved						DIVSEL	
R/W-0	R-0						R/W-0	
7	6	5	4	3	2	1	0	
DIVSEL	MCLKOFF	OSCOFF	MCLKCLR	MCLKSTS	PLLOFF	Reserved	PLLLOCKS	
R/W-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	R-0	R-1	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-24. PLL Status (PLLSTS) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1) (2)</sup>
15	NORMRDYE	0 1	<p>NORMRDY Enable Bit: This bit selects if NORMRDY signal from VREG gates the PLL from turning on when the VREG is out of regulation. It may be required to keep the PLL off while coming in and out of HALT mode and this signal can be used for that purpose:</p> <p>NORMRDY signal from VREG does not gate PLL (PLL ignores NORMRDY)</p> <p>NORMRDY signal from VREG will gate PLL (PLL off when NORMRDY low)</p> <p>The NORMRDY signal from the VREG is low when the VREG is out of regulation and this signal will go high if the VREG is within regulation.</p>
14-9	Reserved		Any writes to these bit(s) must always have a value of 0.
8:7	DIVSEL	00, 01 10 11	<p>Divide Select: This bit selects between /4, /2, and /1 for CLKIN to the CPU. The configuration of the DIVSEL bit is as follows:</p> <p>Select Divide By 4 for CLKIN</p> <p>Select Divide By 2 for CLKIN</p> <p>Select Divide By 1 for CLKIN</p>
6	MCLKOFF	0 1	<p>Missing clock-detect off bit</p> <p>0 Main oscillator fail-detect logic is enabled. (default)</p> <p>1 Main oscillator fail-detect logic is disabled and the PLL will not issue a limp-mode clock. Use this mode when code must not be affected by the detection circuit. For example, if external clocks are turned off.</p>
5	OSCOFF	0 1	<p>Oscillator Clock Off Bit</p> <p>0 The OSCCLK signal from X1/X2 or XCLKIN is fed to the PLL block. (default)</p> <p>1 The OSCCLK signal from X1/X2 or XCLKIN is not fed to the PLL block. This does not shut down the internal oscillator. The OSCOFF bit is used for testing the missing clock detection logic.</p> <p>When the OSCOFF bit is set, do not enter HALT or STANDBY modes or write to PLLCR as these operations can result in unpredictable behavior.</p> <p>When the OSCOFF bit is set, the behavior of the watchdog is different depending on which input clock source (X1, X1/X2 or XCLKIN) is being used:</p> <ul style="list-style-type: none"> <li>X1/X2: The watchdog is not functional.</li> <li>XCLKIN: The watchdog is functional and should be disabled before setting OSCOFF.</li> </ul>

**Table 1-24. PLL Status (PLLSTS) Register Field Descriptions (continued)**

Bits	Field	Value	Description <sup>(1) (2)</sup>
4	MCLKCLR	0 1	Missing Clock Clear Bit. Writing a 0 has no effect. This bit always reads 0. Forces the missing clock detection circuits to be cleared and reset. If OSCCLK is still missing, the detection circuit will again generate a reset to the system, set the missing clock status bit (MCLKSTS), and the CPU will be clocked by the PLL operating at a limp mode frequency.
3	MCLKSTS	0 1	Missing Clock Status Bit. Check the status of this bit after a reset to determine whether a missing oscillator condition was detected. Under normal conditions, this bit should be 0. Writes to this bit are ignored. This bit will be cleared by writing to the MCLKCLR bit or by forcing an external reset. Indicates normal operation. A missing clock condition has not been detected. Indicates that OSCCLK was detected as missing. The main oscillator fail detect logic has reset the device and the CPU is now clocked by the PLL operating at the limp mode frequency. When the missing clock detection circuit automatically switches between OSCCLKSRC2 to OSCCLKSRC1 (upon detecting OSCCLKSRC2 failure), this bit will be automatically cleared and the missing clock detection circuit will be re-enabled. For all other cases, the user needs to re-enable this mode by writing a 1 to the MCLKCLR bit.
2	PLLOFF	0 1	PLL Off Bit. This bit turns off the PLL. This is useful for system noise testing. This mode must only be used when the PLLCR register is set to 0x0000. PLL On (default) PLL Off. While the PLLOFF bit is set the PLL module will be kept powered down. The device must be in PLL bypass mode (PLLCR = 0x0000) before writing a 1 to PLLOFF. While the PLL is turned off (PLLOFF = 1), do not write a non-zero value to the PLLCR. The STANDBY and HALT low power modes will work as expected when PLLOFF = 1. After waking up from HALT or STANDBY the PLL module will remain powered down.
1	Reserved		Any writes to these bit(s) must always have a value of 0.
0	PLLLOCKS	0 1	PLL Lock Status Bit. Indicates that the PLLCR register has been written to and the PLL is currently locking. The CPU is clocked by OSCCLK/2 until the PLL locks. Indicates that the PLL has finished locking and is now stable.

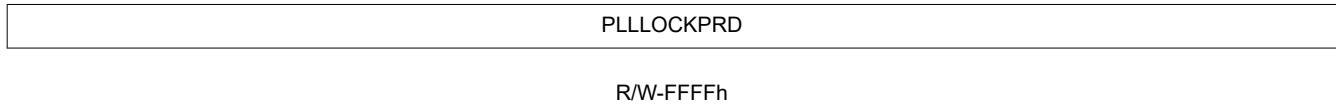
- (1) This register is reset to its default state only by the  $\overline{XRS}$  signal or a watchdog reset. It is not reset by a missing clock or debugger reset.  
(2) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

1.3.2.4.1.3 PLL Lock Period (PLLLOCKPRD) Register

Figure 1-26. PLL Lock Period (PLLLOCKPRD) Register

15

0



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 1-25. PLL Lock Period (PLLLOCKPRD) Register Field Descriptions

Bit	Field	Value	Description <sup>(1) (2)</sup>
15-0	PLLLOCKPRD		PLL Lock Counter Period Value
		0	These 16-bits select the PLL lock counter period. This value is programmable, so shorter PLL lock-time can be programmed by user. The user needs to compute the number of OSCCLK cycles (based on the OSCCLK value used in the design) and update this register.
		1	PLL Lock Period
		FFFFh	65535 OSCCLK Cycles (default on reset)
		FFFEh	65534 OSCCLK Cycles
		...	...
		0001h	1 OSCCLK Cycles
		0000h	0 OSCCLK Cycles (no PLL lock period)

- (1) PLLLOCKPRD is affected by XRSn signal only.
- (2) This register is EALLOW protected. See Section 1.5.2 for more information.

1.3.2.5 Input Clock Fail Detection

It is possible for the clock source of the device to fail. When the PLL is not disabled, the main oscillator fail logic allows the device to detect this condition and handle it as described in this section.

Two counters are used to monitor the presence of the OSCCLK signal as shown in Figure 1-27. The first counter is incremented by the OSCCLK signal itself. When the PLL is not turned off, the second counter is incremented by the VCOCLK coming out of the PLL block. These counters are configured such that when the 7-bit OSCCLK counter overflows, it clears the 13-bit VCOCLK counter. In normal operating mode, as long as OSCCLK is present, the VCOCLK counter will never overflow.

If the OSCCLK input signal is missing, then the PLL will output a default limp mode frequency and the VCOCLK counter will continue to increment. Since the OSCCLK signal is missing, the OSCCLK counter will not increment, and therefore, the VCOCLK counter is not periodically cleared. Eventually, the VCOCLK counter overflows. This signals a missing clock condition to the missing-clock-detection logic. What happens next is based on which clock source has been chosen for the PLL and the value of NMIRESETSEL.

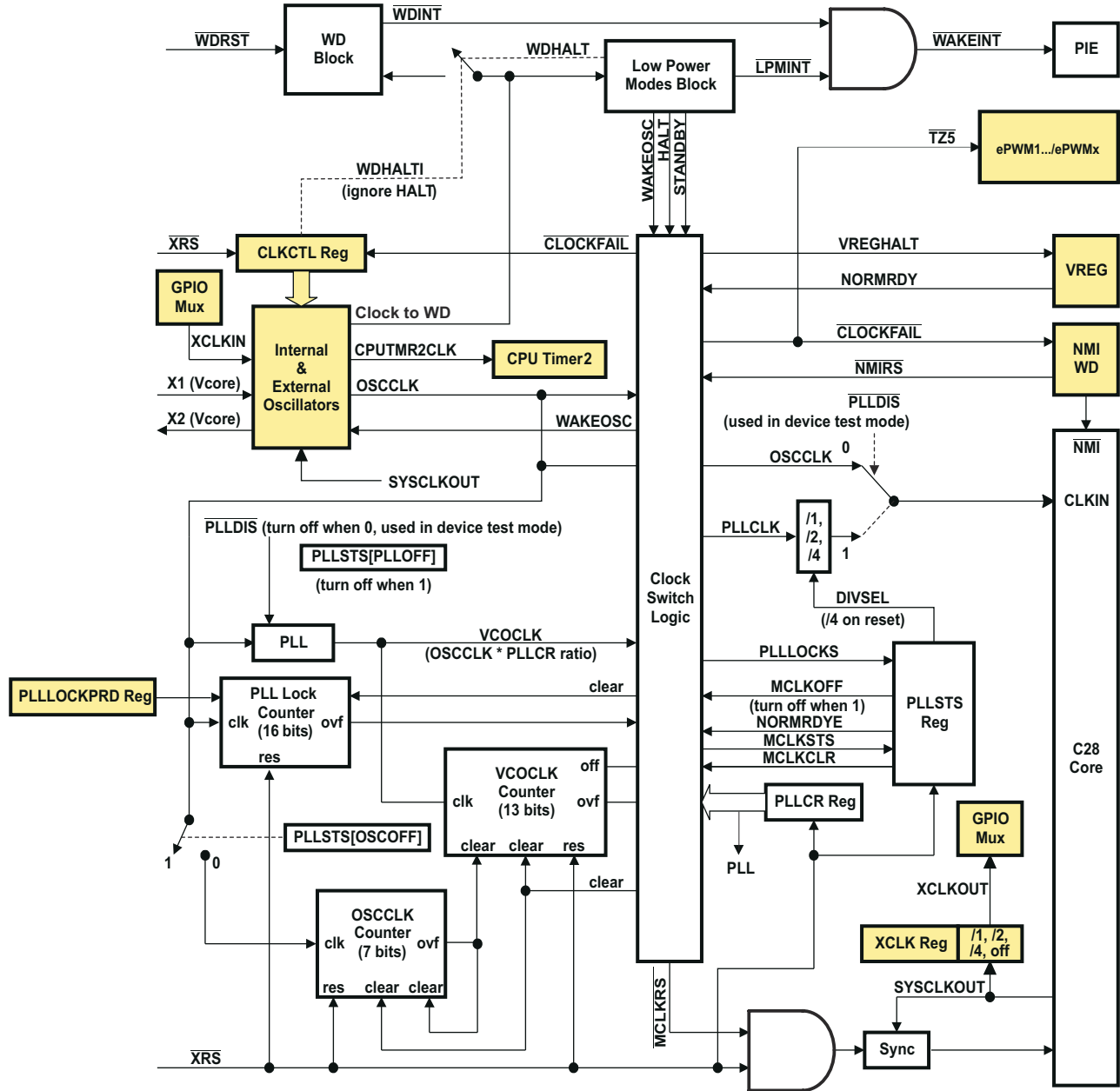


Figure 1-27. Clocking and Reset Logic



**Case A:****INTOSC1 is used as the clock source. NMIWD is disabled (NMIRESETSEL = 0)**

Failure of INTOSC1 causes PLL to issue a limp mode clock. The system continues to function with the limp clock and so does the VCOCLK counter. Eventually, VCOCLK counter overflows and issues a  $\overline{\text{CLOCKFAIL}}$  signal (MCLKSTS bit is set) and the missing clock detection logic resets the CPU, peripherals, and other device logic by way of  $\overline{\text{MCLKRS}}$ . The exact delay (from the time the clock was stopped to the time a reset is asserted) depends on the VCOCLK counter value when the INTOSC1 clock vanished. The MCLKSTS bit is only affected by  $\overline{\text{XRS}}$ , not by a missing clock reset. So, after a reset, code can examine this bit to determine if the reset was due to a missing clock and take appropriate action. Note that even though the  $\overline{\text{CLOCKFAIL}}$  signal is generated, the NMIWDCNTR will not count.

**Case B:****INTOSC1 is used as the clock source. NMIWD is enabled (NMIRESETSEL = 1)**

Failure of INTOSC1 causes PLL to issue a limp mode clock. The system continues to function with the limp clock and so does the VCOCLK counter. Eventually, the VCOCLK counter overflows and issues  $\overline{\text{CLOCKFAIL}}$  (MCLKSTS bit is set), which asserts the NMI and starts the NMIWDCNTR. If NMIWDCNTR is allowed to reach the NMIWDPRD value, a reset ( $\overline{\text{MCLKRS}}$ ) is asserted. In the interim period, the application could choose to gracefully shut down the system before a reset is generated. Inside the NMI\_ISR, the flags in NMIFLG register may be cleared, which prevents a reset.

In case A, reset is inevitable and cannot be delayed. In case B, the software can

- Choose to clear the flags to prevent a reset.
- Perform a graceful shutdown of the system.
- Switch to OSCCLKSRC2, if need be.

**Case C:****OSCCLKSRC2 (INTOSC2 or X1/X2 or XCLKIN) is used as the clock source. NMIWD is disabled (NMIRESETSEL = 0)**

When the VCOCLK counter overflows (due to loss of OSCCLKSRC2), the Missing-Clock-Detect circuit recognizes the missing clock condition.  $\overline{\text{CLOCKFAIL}}$  will be generated (but it is of no consequence). Since NMIRESETSEL=0, the device will be reset. No switching of clock source happens, since the device is reset. This is similar to Case A.

**Case D:****OSCCLKSRC2 (INTOSC2 or X1/X2 or XCLKIN) is used as the clock source. NMIWD is enabled (NMIRESETSEL = 1)**

When the VCOCLK counter overflows (due to loss of OSCCLKSRC2), the Missing-Clock-Detect circuit recognizes the missing clock condition.  $\overline{\text{CLOCKFAIL}}$  is generated and OSCCLK is switched to INTOSC1. For this reason, INTOSC1 should not be disabled in user code. The MCLKSTS bit is set, but cleared automatically after the clock switch. PLLCR is zeroed. The user must reconfigure PLLCR. Since NMIRESETSEL=1, NMI interrupt will be triggered and PLL could be reconfigured there. Inside the NMI\_ISR, the flags in the NMIFLG register may be cleared, which prevents a reset. If INTOSC1 also fails, this becomes similar to Case B. The advantage of using OSCCLKSRC2 as the source for the PLL is that the clock source is automatically switched to INTOSC1 upon loss of OSCCLKSRC2.

### 1.3.2.6 Missing Clock Reset and Missing Clock Status

The  $\overline{\text{MCLKRS}}$  is an internal reset only. The external  $\overline{\text{XRS}}$  pin of the device is not pulled low by  $\overline{\text{MCLKRS}}$ , and the PLLCR and PLLSTS registers are not reset. In addition to resetting the device, the missing clock detect logic sets the PLLSTS[MCLKSTS] register bit. When the MCLKSTS bit is 1, this indicates that the missing oscillator detect logic has reset the part and that the CPU is now running at the limp mode frequency.

Software should check the PLLSTS[MCLKSTS] bit after a reset to determine if the device was reset by  $\overline{\text{MCLKRS}}$  due to a missing clock condition. If MCLKSTS is set, then the firmware should take the action appropriate for the system such as a system shutdown. The missing clock status can be cleared by writing a 1 to the PLLSTS[MCLKCLR] bit. This will reset the missing clock detection circuits and counters. If OSCCLK is still missing after writing to the MCLKCLR bit, then the VCOCLK counter again overflows and the process will repeat.

#### Note

Applications in which the correct CPU operating frequency is absolutely critical should implement a mechanism by which the DSP will be held in reset should the input clocks ever fail. For example, an R-C circuit may be used to trigger the  $\overline{\text{XRS}}$  pin of the DSP should the capacitor ever get fully charged. An I/O pin may be used to discharge the capacitor on a periodic basis to prevent it from getting fully charged. Such a circuit would also help in detecting failure of the flash memory.

The following precautions and limitations should be kept in mind:

- **Use the proper procedure when changing the PLL Control Register.** Always follow the procedure outlined in [Figure 1-23](#) when modifying the PLLCR register.
- **Do not write to the PLLCR register when the device is operating in limp mode.** When writing to the PLLCR register, the device switches to the CPU's CLKIN input to OSCCLK/2. When operating after limp mode has been detected, OSCCLK may not be present and the clocks to the system will stop. Always check that the PLLSTS[MCLKSTS] bit = 0 before writing to the PLLCR register as described in [Figure 1-23](#).
- **Do not enter HALT low power mode when the device is operating in limp mode.** If you try to enter HALT mode when the device is already operating in limp mode then the device may not properly enter HALT. The device may instead enter STANDBY mode or may hang and you may not be able to exit HALT mode. For this reason, always check that the PLLSTS[MCLKSTS] bit = 0 before entering HALT mode.

The following list describes the behavior of the missing clock detect logic in various operating modes:

- **PLL by-pass mode**

When the PLL control register is set to 0x0000, the PLL is bypassed. Depending on the state of the PLLSTS[DIVSEL] bit, OSCCLK, OSCCLK/2, or OSCCLK/4 is connected directly to the CPU's input clock, CLKIN. If the OSCCLK is detected as missing, the device will automatically switch to the PLL's limp mode clock. Further behavior is determined by the clock source used for OSCCLK and the value of NMIRESETSEL bit as explained before.

- **STANDBY low power mode**

In this mode, the CLKIN to the CPU is stopped. If a missing input clock is detected, the missing clock status bit will be set and the device will generate a missing clock reset. If the PLL is in by-pass mode when this occurs, then one-half of the PLL limp frequency will automatically be routed to the CPU. The device will now run at the PLL limp mode frequency or at one-half or one-fourth of the PLL limp mode frequency, depending on the state of the PLLSTS[DIVSEL] bit.

- **HALT low power mode**

In HALT low power mode, all of the clocks to the device are turned off. When the device comes out of HALT mode, the oscillator and PLL will power up. The counters that are used to detect a missing input clock (VCOCLK and OSCCLK) will be enabled only after this power-up has completed. If VCOCLK counter overflows, the missing clock detect status bit will be set and the device will generate a missing clock reset. If the PLL is in by-pass mode when the overflow occurs, then one-half of the PLL limp frequency will automatically be routed to the CPU. The device will now run at the PLL limp mode frequency or at one-half or one-fourth of the PLL limp mode frequency depending on the state of the PLLSTS[DIVSEL] bit.



### 1.3.2.7.1 NMI Interrupt Registers

The NMI Interrupt support registers are listed in [Table 1-26](#).

**Table 1-26. NMI Interrupt Registers**

Name	Address Range	Size (x16)	EALLOW	Description	Bit Description
NMICFG	0x7060	1	Yes	NMI Configuration Register	<a href="#">Section 1.3.2.7.1.1</a>
NMIFLG	0x7061	1	Yes	NMI Flag Register	<a href="#">Section 1.3.2.7.1.2</a>
NMIFLGCLR	0x7062	1	Yes	NMI Flag Clear Register	<a href="#">Section 1.3.2.7.1.3</a>
NMIFLGFRC	0x7063	1	Yes	NMI Flag Force Register	<a href="#">Section 1.3.2.7.1.4</a>
NMIWDCNT	0x7064	1	-	NMI Watchdog Counter Register	<a href="#">Section 1.3.2.7.1.5</a>
NMIWDPRD	0x7065	1	Yes	NMI Watchdog Period Register	<a href="#">Section 1.3.2.7.1.6</a>

#### 1.3.2.7.1.1 NMI Configuration (NMICFG) Register

**Figure 1-29. NMI Configuration (NMICFG) Register Bit Definitions (EALLOW)**

15	2	1	0
Reserved	CLOCKFAIL	Reserved	
R-0	R/W-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-27. NMI Configuration (NMICFG) Register Bit Definitions (EALLOW)**

Bits	Name	Value	Description
15-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1	CLOCKFAIL	0 1	CLOCKFAIL-interrupt Enable Bit: This bit, when set to 1 enables the CLOCKFAIL condition to generate an NMI interrupt. Once enabled, the flag cannot be cleared by the user. Only a device reset clears the flag. Writes of 0 are ignored. Reading the bit will indicate if the flag is enabled or disabled: 0 CLOCKFAIL Interrupt Disabled 1 CLOCKFAIL Interrupt Enabled
0	Reserved		Any writes to these bit(s) must always have a value of 0.

### 1.3.2.7.1.2 NMI Flag (NMIFLG) Register

**Figure 1-30. NMI Flag (NMIFLG) Register Bit Definitions (EALLOW Protected)**

15		2	1	0
	Reserved		CLOCKFAIL	NMIINT
	R-0		R-0	R-0

LEGEND: R = Read only; -n = value after reset

**Table 1-28. NMI Flag (NMIFLG) Register Bit Definitions (EALLOW Protected)**

Bits	Name	Value	Description <sup>(1)</sup>
15-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1	CLOCKFAIL	0 1	CLOCKFAIL Interrupt Flag: This bit indicates if the CLOCKFAIL condition is latched. This bit can be cleared only by writing to the respective bit in the NMIFLGCLR register or by a device reset ( $\overline{XRS}$ ): 0 No CLOCKFAIL condition pending 1 CLOCKFAIL condition detected. This bit will be set in the event of any clock failure.
0	NMIINT	0 1	NMI Interrupt Flag: This bit indicates if an NMI interrupt was generated. This bit can only be cleared by writing to the respective bit in the NMIFLGCLR register or by an $\overline{XRS}$ reset: 0 No NMI interrupt generated 1 NMI interrupt generated No further NMI interrupts are generated until you clear this flag.

(1) The NMIFLG register is only reset by the  $\overline{XRS}$  signal, not  $\overline{SYSRS}$

### 1.3.2.7.1.3 NMI Flag Clear (NMIFLGCLR) Register

**Figure 1-31. NMI Flag Clear (NMIFLGCLR) Register Bit Definitions (EALLOW Protected)**

15		2	1	0
	Reserved		CLOCKFAIL	NMIINT
	R-0		W-0	W-0

LEGEND: R = Read only; W = Write only; -n = value after reset

**Table 1-29. NMI Flag Clear (NMIFLGCLR) Register Bit Definitions (EALLOW Protected)**

Bits	Name	Value	Description
15-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1	CLOCKFAIL <sup>(1)</sup>	0 1	CLOCKFAIL Flag Clear 0 Writes of 0 are ignored. Always reads back 0. 1 Writing a 1 to the respective bit clears the corresponding flag bit in the NMIFLG register.
0	NMIINT <sup>(1)</sup>	0 1	NMI Flag Clear 0 Writes of 0 are ignored. Always reads back 0. 1 Writing a 1 to the respective bit clears the corresponding flag bit in the NMIFLG register.

(1) If hardware is trying to set a bit to 1 while software is trying to clear a bit to 0 on the same cycle, hardware has priority. You should clear the pending CLOCKFAIL flag first and then clear the NMIINT flag.

#### 1.3.2.7.1.4 NMI Flag Force (NMIFLGFR) Register

**Figure 1-32. NMI Flag Force (NMIFLGFR) Register Bit Definitions (EALLOW Protected)**

15	2	1	0
Reserved	CLOCKFAIL	Reserved	
R-0	W-0	R-0	

LEGEND: R = Read only; W = Write only; -n = value after reset

**Table 1-30. NMI Flag Force (NMIFLGFR) Register Bit Definitions (EALLOW Protected)**

Bits	Name	Value	Description
15-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1	CLOCKFAIL	0	CLOCKFAIL flag force Writes of 0 are ignored. Always reads back 0. This can be used as a means to test the NMI mechanisms.
		1	Writing a 1 sets the CLOCKFAIL flag.
0	Reserved		Any writes to these bit(s) must always have a value of zero.

#### 1.3.2.7.1.5 NMI Watchdog Counter (NMIWDCNT) Register

**Figure 1-33. NMI Watchdog Counter (NMIWDCNT) Register Bit Definitions**

15	NMIWDCNT	0
NMIWDCNT		
R-0		

LEGEND: R = Read only; -n = value after reset

**Table 1-31. NMI Watchdog Counter (NMIWDCNT) Register Bit Definitions**

Bits	Name	Value	Description
15-0	NMIWDCNT		NMI Watchdog Counter: This 16-bit incremental counter will start incrementing whenever any one of the enabled FAIL flags are set. If the counter reaches the period value, an NMIRS signal is fired, which then resets the system. The counter resets to zero when it reaches the period value and then restarts counting if any of the enabled FAIL flags are set.
		0	If no enabled FAIL flag is set, then the counter resets to zero and remains at zero until an enabled FAIL flag is set.
		1	Normally, the software would respond to the NMI interrupt generated and clear the offending FLAGS before the NMI watchdog triggers a reset. In some situations, the software may decide to allow the watchdog to reset the device anyway.
			The counter is clocked at the SYSCLKOUT rate. Reset value of this counter is zero.

### 1.3.2.7.1.6 NMI Watchdog Period (NMIWDPRD) Register

**Figure 1-34. NMI Watchdog Period (NMIWDPRD) Register Bit Definitions (EALLOW Protected)**

15

0

NMIWDPRD
R/W-0xFFFF

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-32. NMI Watchdog Period (NMIWDPRD) Register Bit Definitions (EALLOW Protected)**

Bits	Name	Type	Description
15-0	NMIWDPRD	R/W	<p>NMI Watchdog Period: This 16-bit value contains the period value at which a reset is generated when the watchdog counter matches. At reset this value is set at the maximum. The software can decrease the period value at initialization time.</p> <p>Writing a PERIOD value that is equal to the current counter value automatically forces an <math>\overline{\text{NMIRS}}</math> and resets the watchdog counter. If a PERIOD value is written that is smaller than the current counter value, the counter will continue counting until it overflows and starts counting up again from 0. After the overflow, once the COUNTER value equals the new PERIOD value, an <math>\overline{\text{NMIRS}}</math> is forced which resets the watchdog counter.</p>

### 1.3.2.7.2 NMI Watchdog Emulation Considerations

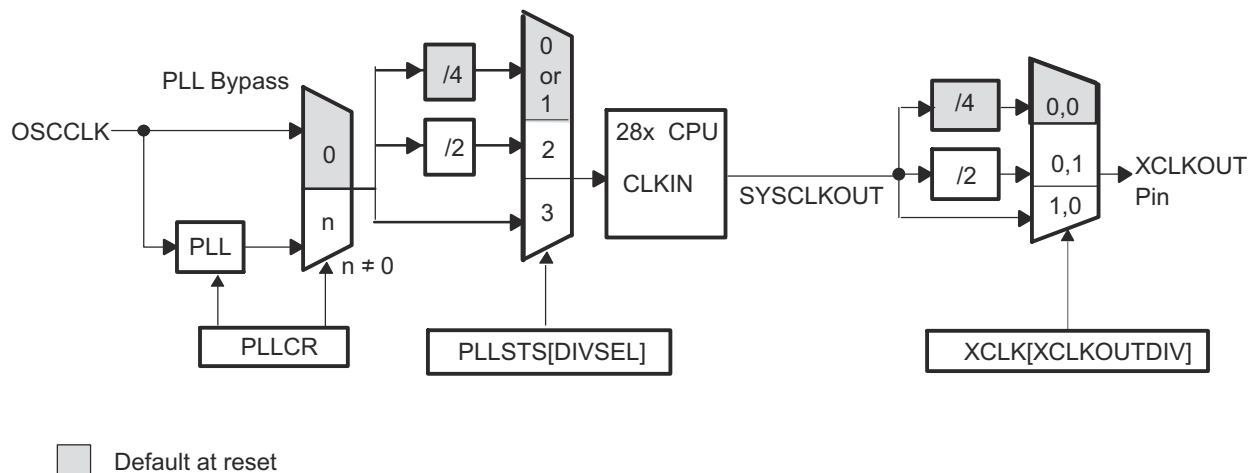
The NMI watchdog module does not operate when trying to debug the target device (emulation suspend such as breakpoint). The NMI watchdog module behaves as follows under various debug conditions:

<i>CPU Suspended:</i>	When the CPU is suspended, the NMI watchdog counter is suspended.
<i>Run-Free Mode:</i>	When the CPU is placed in run-free mode, the NMI watchdog counter resumes operation as normal.
<i>Real-Time Single-Step Mode:</i>	When the CPU is in real-time single-step mode, the NMI watchdog counter is suspended. The counter remains suspended even within real-time interrupts.
<i>Real-Time Run-Free Mode:</i>	When the CPU is in real-time run-free mode, the NMI watchdog counter operates as normal.

### 1.3.2.8 XCLKOUT Generation

The XCLKOUT signal is directly derived from the system clock SYSCLKOUT as shown in [Figure 1-35](#). XCLKOUT can be either equal to, one-half, or one-fourth of SYSCLKOUT. By default, at power-up,  $XCLKOUT = SYSCLKOUT/4$  or  $XCLKOUT = OSCCLK/16$ .

If XCLKOUT is not being used, it can be turned off by setting the XCLKOUTDIV bit to 3 in the XCLK register.



**Figure 1-35. XCLKOUT Generation**

### 1.3.2.9 External Reference Oscillator Clock Option

TI recommends that customers have the resonator/crystal vendor characterize the operation of their device with the device chip. The resonator/crystal vendor has the equipment and expertise to tune the tank circuit. The vendor can also advise the customer regarding the proper tank component values to provide proper start-up and stability over the entire operating range.

### 1.3.3 Low-Power Modes Block

[Table 1-33](#) summarizes the various modes. The various low-power modes operate as shown in [Table 1-34](#).

See the [TMS320F2802x Microcontrollers Data Manual](#) for exact timing for entering and exiting the low power modes.

The low-power modes are controlled by the LPMCR0 register described in [Section 1.3.3.1](#).

**Table 1-33. Low-Power Mode Summary**

Mode	LPMCR0[1:0]	OSCCLK	CLKIN	SYSCLKOUT	Exit <sup>(1)</sup>
IDLE	00	On	On	On	$\overline{XRS}$ , Watchdog interrupt, Any enabled interrupt
STANDBY	01	On (watchdog still running)	Off	Off	$\overline{XRS}$ , Watchdog interrupt, GPIO Port A signal, Debugger <sup>(2)</sup>
HALT	1X	Off (oscillator and PLL turned off, watchdog not functional)	Off	Off	$\overline{XRS}$ , GPIO Port A Signal, Debugger <sup>(2)</sup>

(1) The Exit column lists which signals or under what conditions the low power mode is exited. This signal must be kept low long enough for an interrupt to be recognized by the device. Otherwise the IDLE mode is not exited and the device goes back into the indicated low power mode.

(2) On the 28x, the JTAG port can still function even if the clock to the CPU (CLKIN) is turned off.



**Table 1-34. Low Power Modes**

Mode	Description
IDLE Mode:	This mode is exited by any enabled interrupt. The LPM block itself performs no tasks during this mode.
STANDBY Mode:	<p>If the LPM bits in the LPMCR0 register are set to 01, the device enters STANDBY mode when the IDLE instruction is executed. In STANDBY mode the clock input to the CPU (CLKIN) is disabled, which disables all clocks derived from SYSCLKOUT. The oscillator and PLL and watchdog will still function. Before entering the STANDBY mode, you should perform the following tasks:</p> <ul style="list-style-type: none"> <li>• Enable the WAKEINT interrupt in the PIE module. This interrupt is connected to both the watchdog and the low power mode module interrupt.</li> <li>• If desired, specify one of the GPIO port A signals to wake the device in the GPIOLPMSEL register. The GPIOLPMSEL register is part of the GPIO module. In addition to the selected GPIO signal, the <math>\overline{XRS}</math> input and the watchdog interrupt, if enabled in the LPMCR0 register, can wake the device from the STANDBY mode.</li> <li>• Select the input qualification in the LPMCR0 register for the signal that will wake the device.</li> </ul> <p>When the selected external signal goes low, it must remain low a number of OSCCLK cycles as specified by the qualification period in the LPMCR0 register. If the signal should be sampled high during this time, the qualification will restart. At the end of the qualification period, the PLL enables the CLKIN to the CPU and the WAKEINT interrupt is latched in the PIE block. The CPU then responds to the WAKEINT interrupt if it is enabled.</p>
HALT Mode:	<p>If the LPM bits in the LPMCR0 register are set to 1x, the device enters the HALT mode when the IDLE instruction is executed. In HALT mode all of the device clocks, including the PLL and oscillator, are shut down. Before entering the HALT mode, you should perform the following tasks:</p> <ul style="list-style-type: none"> <li>• Enable the WAKEINT interrupt in the PIE module (PIEIER1.8 = 1). This interrupt is connected to both the watchdog and the Low-Power-Mode module interrupt.</li> <li>• Specify one of the GPIO port A signals to wake the device in the GPIOLPMSEL register. The GPIOLPMSEL register is part of the GPIO module. In addition to the selected GPIO signal, the XRS input can also wake the device from the HALT mode.</li> <li>• Disable all interrupts with the possible exception of the HALT mode wakeup interrupt. The interrupts can be re-enabled after the device is brought out of HALT mode.</li> </ul> <ol style="list-style-type: none"> <li>1. For device to exit HALT mode properly, the following conditions must be met: <ul style="list-style-type: none"> <li>Bit 7 (INT1.8) of PIEIER1 register should be 1.</li> <li>Bit 0 (INT1) of IER register must be 1.</li> </ul> </li> <li>2. If the above conditions are met, <ol style="list-style-type: none"> <li>a. WAKE_INT ISR will be executed first, followed by the instructions after IDLE, if INTM = 0.</li> <li>b. WAKE_INT ISR will not be executed and instructions after IDLE will be executed, if INTM = 1.</li> </ol> </li> </ol> <p><b>Do not enter HALT low power mode when the device is operating in limp mode (PLLSTS[MCLKSTS] = 1).</b> If you try to enter HALT mode when the device is already operating in limp mode then the device may not properly enter HALT. The device may instead enter STANDBY mode or may hang and you may not be able to exit HALT mode. For this reason, always check that the PLLSTS[MCLKSTS] bit = 0 before entering HALT mode.</p> <p>When the selected external signal goes low, it is fed asynchronously to the LPM block. The oscillator is turned on and begins to power up. You must hold the signal low long enough for the oscillator to complete power up. When the signal is held low for enough time and driven high, this will asynchronously release the PLL and it will begin to lock. Once the PLL has locked, it feeds the CLKIN to the CPU at which time the CPU responds to the WAKEINT interrupt if enabled.</p>

### 1.3.3.1 Low Power Mode Control 0 Register (LPMCR0)

**Figure 1-36. Low Power Mode Control 0 Register (LPMCR0)**

15	14	8	7	2	1	0
WDINTE	Reserved		QUALSTDBY		LPM	
R/W-0	R-0		R/W-0x3F		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-35. Low Power Mode Control 0 Register (LPMCR0) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
15	WDINTE	0 1	Watchdog interrupt enable The watchdog interrupt is not allowed to wake the device from STANDBY. (default) The watchdog is allowed to wake the device from STANDBY. The watchdog interrupt must also be enabled in the SCSR Register.
14-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7-2	QUALSTDBY	000000 000001 ... 111111	Select number of OSCCLK clock cycles to qualify the selected GPIO inputs that wake the device from STANDBY mode. This qualification is only used when in STANDBY mode. The GPIO signals that can wake the device from STANDBY are specified in the GPIO_LPMSEL register. 2 OSCCLKs 3 OSCCLKs ... 65 OSCCLKs (default)
1-0	LPM <sup>(2)</sup>	00 01 10 11	These bits set the low-power mode for the device. Set the low-power mode to IDLE (default) Set the low-power mode to STANDBY Set the low-power mode to HALT <sup>(3)</sup> Set the low-power mode to HALT <sup>(3)</sup>

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) The low-power mode bits (LPM) only take effect when the IDLE instruction is executed. Therefore, you must set the LPM bits to the appropriate mode before executing the IDLE instruction.

(3) If you try to enter HALT mode when the device is already operating in limp mode, then the device may not properly enter HALT. The device may instead enter STANDBY mode or may hang and you may not be able to exit HALT mode. For this reason, always check that the PLLSTS[MCLKSTS] bit = 0 before entering HALT mode.

### 1.3.3.2 Options for Automatic Wakeup in Low-power Modes

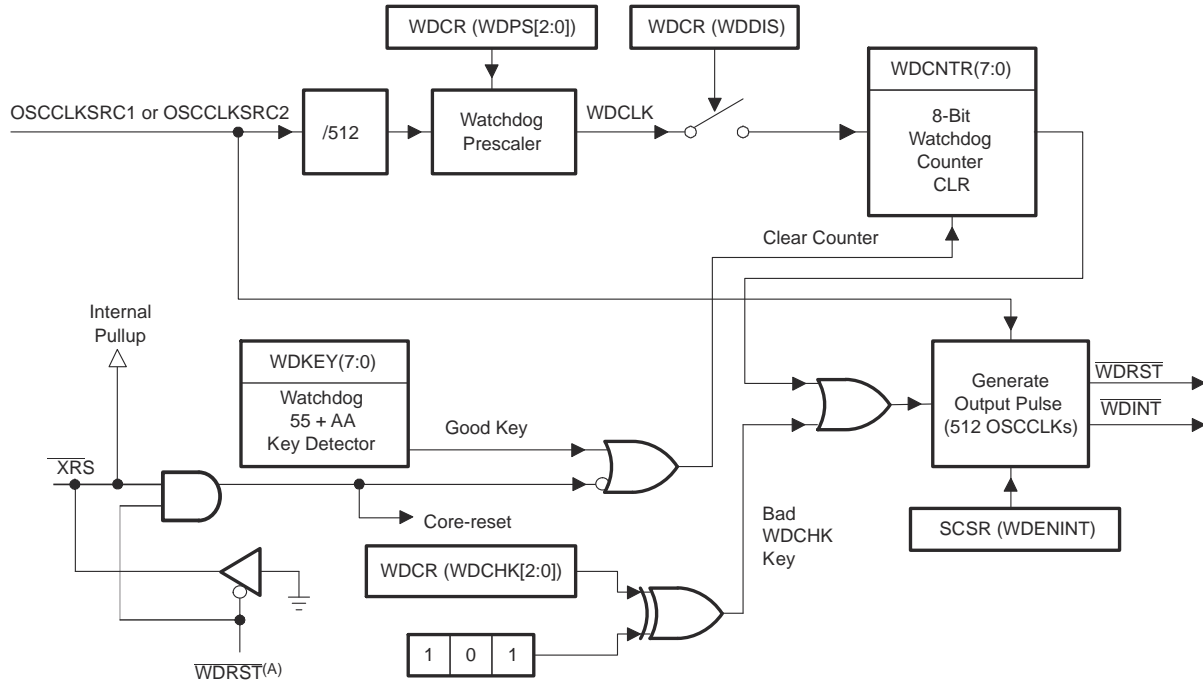
The device provides two options to automatically wake up from HALT and STANDBY modes, without the need for an external stimulus:

**Wakeup from HALT:** Set WDHalti bit in CLKCTL register to 1. When the device wakes up from HALT, it will be through a CPU-watchdog reset. The WDFLAG bit in the WDCR register can be used to differentiate between a CPU-watchdog-reset and a device reset.

**Wakeup from STANDBY:** Set WDINTE bit in LPMCR0 register to 1. When the device wakes up from STANDBY, it will be through the WAKEINT interrupt (Interrupt 1.8 in the PIE).

### 1.3.4 CPU Watchdog Block

The watchdog module generates an output pulse, 512 oscillator-clocks (OSCCLK) wide whenever the 8-bit watchdog up counter has reached its maximum value. To prevent this, either disable the counter or the software must periodically write a 0x55 + 0xAA sequence into the watchdog key register, which resets the watchdog counter. Figure 1-37 shows the various functional blocks within the watchdog module.



- A. The  $\overline{WDRST}$  and  $\overline{XRS}$  signals are driven low for 512 OSCCLK cycles when a watchdog reset occurs. Likewise, if the watchdog interrupt is enabled, the  $\overline{WDINT}$  signal will be driven low for 512 OSCCLK cycles when an interrupt occurs.

Figure 1-37. CPU Watchdog Module

### 1.3.4.1 Servicing The Watchdog Timer

The WDCNTR is reset when the proper sequence is written to the WDKEY register before the 8-bit watchdog counter (WDCNTR) overflows. The WDCNTR is reset-enabled when a value of 0x55 is written to the WDKEY. When the next value written to the WDKEY register is 0xAA then the WDCNTR is reset. Any value written to the WDKEY other than 0x55 or 0xAA causes no action. Any sequence of 0x55 and 0xAA values can be written to the WDKEY without causing a system reset; only a write of 0x55 followed by a write of 0xAA to the WDKEY resets the WDCNTR.

**Table 1-36. Example Watchdog Key Sequences**

Step	Value Written to WDKEY	Result
1	0xAA	No action
2	0xAA	No action
3	0x55	WDCNTR is enabled to be reset if next value is 0xAA.
4	0x55	WDCNTR is enabled to be reset if next value is 0xAA.
5	0x55	WDCNTR is enabled to be reset if next value is 0xAA.
6	0xAA	WDCNTR is reset.
7	0xAA	No action
8	0x55	WDCNTR is enabled to be reset if next value is 0xAA.
9	0xAA	WDCNTR is reset.
10	0x55	WDCNTR is enabled to be reset if next value is 0xAA.
11	0x32	Improper value written to WDKEY. No action, WDCNTR no longer enabled to be reset by next 0xAA.
12	0xAA	No action due to previous invalid value.
13	0x55	WDCNTR is enabled to be reset if next value is 0xAA.
14	0xAA	WDCNTR is reset.

Step 3 in [Table 1-36](#) is the first action that enables the WDCNTR to be reset. The WDCNTR is not actually reset until step 6. Step 8 again re-enables the WDCNTR to be reset and step 9 resets the WDCNTR. Step 10 again re-enables the WDCNTR to be reset. Writing the wrong key value to the WDKEY in step 11 causes no action, however the WDCNTR is no longer enabled to be reset and the 0xAA in step 12 now has no effect.

If the watchdog is configured to reset the device, then a WDCR overflow or writing the incorrect value to the WDCR[WDCHK] bits will reset the device and set the watchdog flag (WDFLAG) in the WDCR register. After a reset, the program can read the state of this flag to determine the source of the reset. After reset, the WDFLAG should be cleared by software to allow the source of subsequent resets to be determined. Watchdog resets are not prevented when the flag is set.

### 1.3.4.2 Watchdog Reset or Watchdog Interrupt Mode

The watchdog can be configured in the SCSR register to either reset the device ( $\overline{\text{WDRST}}$ ) or assert an interrupt ( $\overline{\text{WDINT}}$ ) if the watchdog counter reaches its maximum value. The behavior of each condition is described below:

- **Reset mode:**

If the watchdog is configured to reset the device, then the  $\overline{\text{WDRST}}$  signal will pull the device reset ( $\overline{\text{XRS}}$ ) pin low for 512 OSCCLK cycles when the watchdog counter reaches its maximum value.

- **Interrupt mode:**

If the watchdog is configured to assert an interrupt, then the  $\overline{\text{WDINT}}$  signal will be driven low for 512 OSCCLK cycles, causing the WAKEINT interrupt in the PIE to be taken if it is enabled in the PIE module. The watchdog interrupt is edge triggered on the falling edge of  $\overline{\text{WDINT}}$ . Thus, if the WAKEINT interrupt is re-enabled before  $\overline{\text{WDINT}}$  goes inactive, you will not immediately get another interrupt. The next WAKEINT interrupt will occur at the next watchdog timeout. If the watchdog is disabled before  $\overline{\text{WDINT}}$  goes inactive, the 512-cycle count will halt and  $\overline{\text{WDINT}}$  will remain active. The count will resume when the watchdog is enabled again.

If the watchdog is reconfigured from interrupt mode to reset mode while  $\overline{\text{WDINT}}$  is still active low, then the device will reset immediately. The WDINTS bit in the SCSR register can be read to determine the current state of the  $\overline{\text{WDINT}}$  signal before reconfiguring the watchdog to reset mode.

### 1.3.4.3 Watchdog Operation in Low Power Modes

In STANDBY mode, all of the clocks to the peripherals are turned off on the device. The only peripheral that remains functional is the watchdog since the watchdog module runs off the oscillator clock (OSCCLK). The  $\overline{\text{WDINT}}$  signal is fed to the Low Power Modes (LPM) block so that it can be used to wake the device from STANDBY low power mode (if enabled). See the Low Power Modes Block section of the device data manual for details.

In IDLE mode, the watchdog interrupt ( $\overline{\text{WDINT}}$ ) signal can generate an interrupt to the CPU to take the CPU out of IDLE mode. The watchdog is connected to the WAKEINT interrupt in the PIE.

---

#### Note

If the watchdog interrupt is used to wake-up from an IDLE or STANDBY low power mode condition, then make sure that the  $\overline{\text{WDINT}}$  signal goes back high again before attempting to go back into the IDLE or STANDBY mode. The  $\overline{\text{WDINT}}$  signal will be held low for 512 OSCCLK cycles when the watchdog interrupt is generated. You can determine the current state of  $\overline{\text{WDINT}}$  by reading the watchdog interrupt status bit (WDINTS) bit in the SCSR register. WDINTS follows the state of  $\overline{\text{WDINT}}$  by two SYSCLKOUT cycles.

---

By using WDHalti and INTOSC1HALTI bits, INTOSC1 and the watchdog module could be kept alive in HALT mode. The device can then wake-up from HALT mode through the watchdog, but it is through the watchdog reset, not the interrupt. When this happens, the RAM contents are not disturbed, but the peripherals will have to be re-initialized.

### 1.3.4.4 Emulation Considerations

The watchdog module behaves as follows under various debug conditions:

CPU Suspended:	When the CPU is suspended, the watchdog clock (WDCLK) is suspended
Run-Free Mode:	When the CPU is placed in run-free mode, then the watchdog module resumes operation as normal.
Real-Time Single-Step Mode:	When the CPU is in real-time single-step mode, the watchdog clock (WDCLK) is suspended. The watchdog remains suspended even within real-time interrupts.
Real-Time Run-Free Mode:	When the CPU is in real-time run-free mode, the watchdog operates as normal.

### 1.3.4.5 Watchdog Registers

#### 1.3.4.5.1 System Control and Status Register (SCSR)

The system control and status register (SCSR) contains the watchdog override bit and the watchdog interrupt enable/disable bit.

**Figure 1-38. System Control and Status Register (SCSR)**

15	Reserved				8
R-0					
7	3	2	1	0	
Reserved			WDINTS	WDENINT	WDOVERRIDE
R-0			R-1	R/W-0	R/W1C-1

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

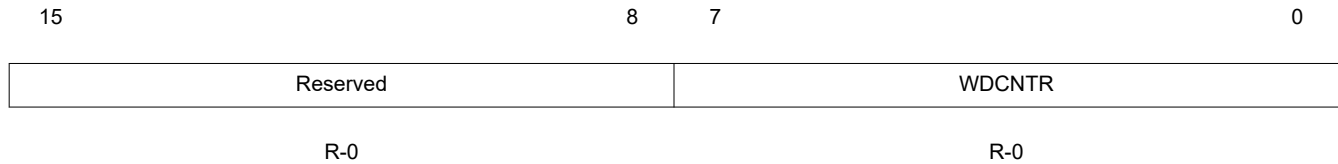
**Table 1-37. System Control and Status Register (SCSR) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-3	Reserved		Any writes to these bit(s) must always have a value of 0.
2	WDINTS	0 1	<p>Watchdog interrupt status bit. WDINTS reflects the current state of the <math>\overline{\text{WDINT}}</math> signal from the watchdog block. WDINTS follows the state of <math>\overline{\text{WDINT}}</math> by two SYSCLKOUT cycles.</p> <p>If the watchdog interrupt is used to wake the device from IDLE or STANDBY low power mode, use this bit to make sure <math>\overline{\text{WDINT}}</math> is not active before attempting to go back into IDLE or STANDBY mode.</p> <p>0 Watchdog interrupt signal (<math>\overline{\text{WDINT}}</math>) is active.</p> <p>1 Watchdog interrupt signal (<math>\overline{\text{WDINT}}</math>) is not active.</p>
1	WDENINT	0 1	<p>Watchdog interrupt enable.</p> <p>0 The watchdog reset (<math>\overline{\text{WDRST}}</math>) output signal is enabled and the watchdog interrupt (<math>\overline{\text{WDINT}}</math>) output signal is disabled. This is the default state on reset (<math>\overline{\text{XRS}}</math>). When the watchdog interrupt occurs the <math>\overline{\text{WDRST}}</math> signal will stay low for 512 OSCCLK cycles.</p> <p>If the WDENINT bit is cleared while <math>\overline{\text{WDINT}}</math> is low, a reset will immediately occur. The WDINTS bit can be read to determine the state of the <math>\overline{\text{WDINT}}</math> signal.</p> <p>1 The <math>\overline{\text{WDRST}}</math> output signal is disabled and the <math>\overline{\text{WDINT}}</math> output signal is enabled. When the watchdog interrupt occurs, the <math>\overline{\text{WDINT}}</math> signal will stay low for 512 OSCCLK cycles.</p> <p>If the watchdog interrupt is used to wake the device from IDLE or STANDBY low power mode, use the WDINTS bit to make sure <math>\overline{\text{WDINT}}</math> is not active before attempting to go back into IDLE or STANDBY mode.</p>
0	WDOVERRIDE	0 1	<p>Watchdog override</p> <p>0 Writing a 0 has no effect. If this bit is cleared, it remains in this state until a reset occurs. The current state of this bit is readable by the user.</p> <p>1 You can change the state of the watchdog disable (WDDIS) bit in the watchdog control (WDCR) register. If the WDOVERRIDE bit is cleared by writing a 1, you cannot modify the WDDIS bit.</p>

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.3.4.5.2 Watchdog Counter Register (WDCNTR)

**Figure 1-39. Watchdog Counter Register (WDCNTR)**



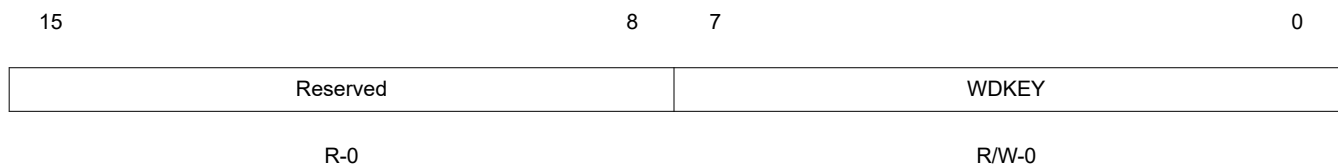
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-38. Watchdog Counter Register (WDCNTR) Field Descriptions**

Bits	Field	Description
15-8	Reserved	Any writes to these bit(s) must always have a value of 0.
7-0	WDCNTR	These bits contain the current value of the WD counter. The 8-bit counter continually increments at the watchdog clock (WDCLK), rate. If the counter overflows, then the watchdog initiates a reset. If the WDKEY register is written with a valid combination, then the counter is reset to zero. The watchdog clock rate is configured in the WDCR register.

### 1.3.4.5.3 Watchdog Reset Key Register (WDKEY)

**Figure 1-40. Watchdog Reset Key Register (WDKEY)**

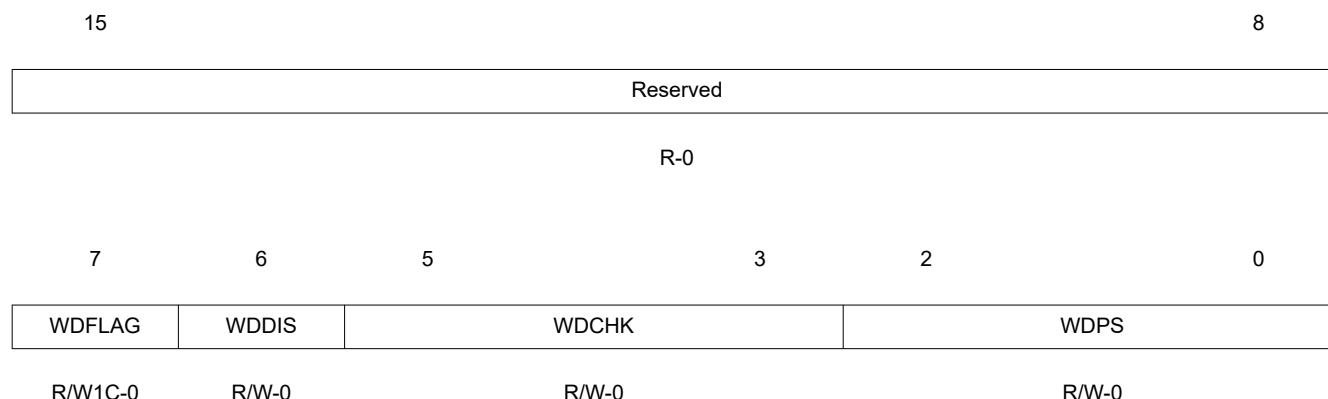


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-39. Watchdog Reset Key Register (WDKEY) Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7-0	WDKEY	0x55 + 0xAA Other value	Refer to <a href="#">Table 1-36</a> for examples of different WDKEY write sequences. Writing 0x55 followed by 0xAA to WDKEY causes the WDCNTR bits to be cleared. Writing any value other than 0x55 or 0xAA causes no action to be generated. If any value other than 0xAA is written after 0x55, then the sequence must restart with 0x55. Reads from WDKEY return the value of the WDCR register.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

**1.3.4.5.4 Watchdog Control Register (WDCR)**
**Figure 1-41. Watchdog Control Register (WDCR)**




signal connects to the  $\overline{XRS}$  input. Hence to distinguish between a watchdog reset and an external device reset, an external reset must be longer in duration than the watchdog pulse.

### 1.3.5 32-Bit CPU Timers 0/1/2

This section describes the three 32-bit CPU-timers (TIMER0/1/2) shown in Figure 1-42.

The CPU Timer-0 and CPU-Timer 1 can be used in user applications. Timer 2 is reserved for device/BIOS. If the application is not using device/BIOS, then Timer 2 can be used in the application. The CPU-timer interrupt signals (TINT0, TINT1, TINT2) are connected as shown in Figure 1-43.

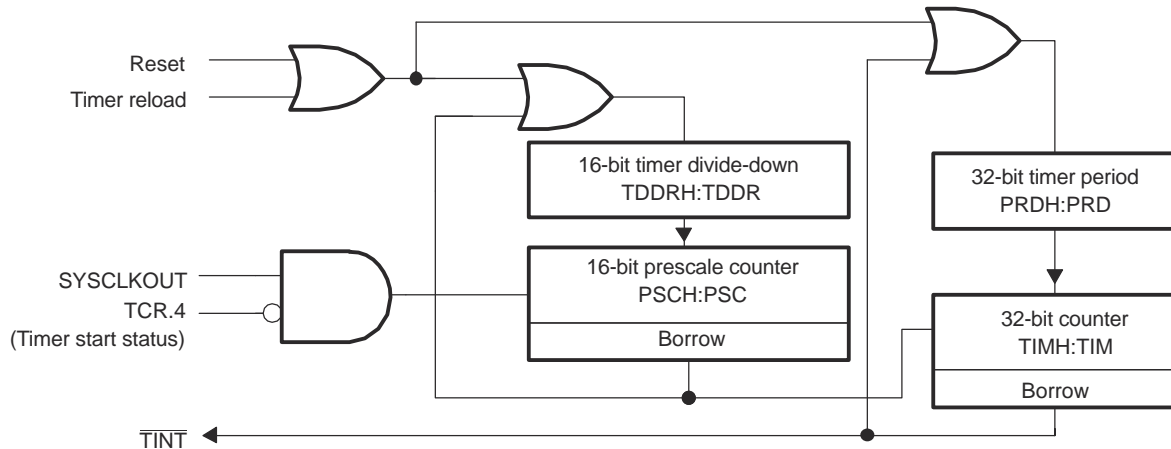
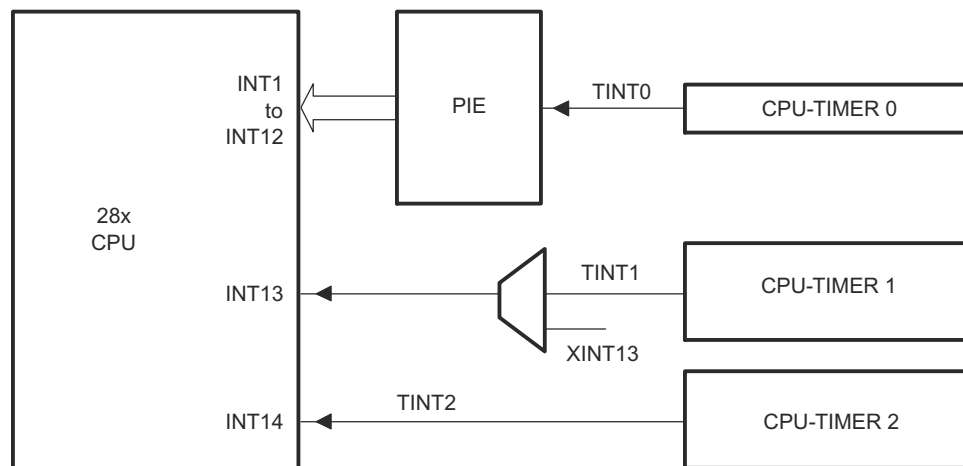


Figure 1-42. CPU-Timers



- A. The timer registers are connected to the Memory Bus of the 28x processor.
- B. The timing of the timers is synchronized to SYSCLKOUT of the processor clock.

Figure 1-43. CPU-Timer Interrupts Signals and Output Signal

The general operation of the CPU-timer is as follows: The 32-bit counter register TIMH:TIM is loaded with the value in the period register PRDH:PRD. The counter decrements once every  $(TPR[TDDRH:TDDR]+1)$  SYCLKOUT cycles, where TDDRH:TDDR is the timer divider. When the counter reaches 0, a timer interrupt output signal generates an interrupt pulse. The registers listed in [Table 1-41](#) are used to configure the timers.

**Table 1-41. CPU-Timers 0, 1, 2 Configuration and Control Registers**

Name	Address	Size (x16)	Description	Bit Description
TIMER0TIM	0x0C00	1	CPU-Timer 0, Counter Register	<a href="#">Section 1.3.5.1</a>
TIMER0TIMH	0x0C01	1	CPU-Timer 0, Counter Register High	<a href="#">Section 1.3.5.2</a>
TIMER0PRD	0x0C02	1	CPU-Timer 0, Period Register	<a href="#">Section 1.3.5.3</a>
TIMER0PRDH	0x0C03	1	CPU-Timer 0, Period Register High	<a href="#">Section 1.3.5.4</a>
TIMER0TCR	0x0C04	1	CPU-Timer 0, Control Register	<a href="#">Section 1.3.5.5</a>
TIMER0TPR	0x0C06	1	CPU-Timer 0, Prescale Register	<a href="#">Section 1.3.5.6</a>
TIMER0TPRH	0x0C07	1	CPU-Timer 0, Prescale Register High	<a href="#">Section 1.3.5.7</a>
TIMER1TIM	0x0C08	1	CPU-Timer 1, Counter Register	<a href="#">Section 1.3.5.1</a>
TIMER1TIMH	0x0C09	1	CPU-Timer 1, Counter Register High	<a href="#">Section 1.3.5.2</a>
TIMER1PRD	0x0C0A	1	CPU-Timer 1, Period Register	<a href="#">Section 1.3.5.3</a>
TIMER1PRDH	0x0C0B	1	CPU-Timer 1, Period Register High	<a href="#">Section 1.3.5.4</a>
TIMER1TCR	0x0C0C	1	CPU-Timer 1, Control Register	<a href="#">Section 1.3.5.5</a>
TIMER1TPR	0x0C0E	1	CPU-Timer 1, Prescale Register	<a href="#">Section 1.3.5.6</a>
TIMER1TPRH	0x0C0F	1	CPU-Timer 1, Prescale Register High	<a href="#">Section 1.3.5.7</a>
TIMER2TIM	0x0C10	1	CPU-Timer 2, Counter Register	<a href="#">Section 1.3.5.1</a>
TIMER2TIMH	0x0C11	1	CPU-Timer 2, Counter Register High	<a href="#">Section 1.3.5.2</a>
TIMER2PRD	0x0C12	1	CPU-Timer 2, Period Register	<a href="#">Section 1.3.5.3</a>
TIMER2PRDH	0x0C13	1	CPU-Timer 2, Period Register High	<a href="#">Section 1.3.5.4</a>
TIMER2TCR	0x0C14	1	CPU-Timer 2, Control Register	<a href="#">Section 1.3.5.5</a>
TIMER2TPR	0x0C16	1	CPU-Timer 2, Prescale Register	<a href="#">Section 1.3.5.6</a>
TIMER2TPRH	0x0C17	1	CPU-Timer 2, Prescale Register High	<a href="#">Section 1.3.5.7</a>

### 1.3.5.1 CPU-Timer x Counter (TIMERxTIM) Register

**Figure 1-44. CPU-Timer x Counter (TIMERxTIM) Register (x = 0, 1, 2)**

15

0

TIM
-----

R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-42. CPU-Timer x Counter (TIMERxTIM) Register Field Descriptions**

Bits	Field	Description
15-0	TIM	CPU-Timer Counter Registers (TIMH:TIM): The TIM register holds the low 16 bits of the current 32-bit count of the timer. The TIMH register holds the high 16 bits of the current 32-bit count of the timer. The TIMH:TIM decrements by one every $(TDDRH:TDDR+1)$ clock cycles, where TDDRH:TDDR is the timer prescale divide-down value. When the TIMH:TIM decrements to zero, the TIMH:TIM register is reloaded with the period value contained in the PRDH:PRD registers. The timer interrupt (TINT) signal is generated.

### 1.3.5.2 CPU-Timer x Counter (TIMERxTIMH) Register High

**Figure 1-45. CPU-Timer x Counter (TIMERxTIMH) Register High (x = 0, 1, 2)**

15 0

TIMH
------

R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-43. CPU-Timer x Counter (TIMERxTIMH) Register High Field Descriptions**

Bits	Field	Description
15-0	TIMH	See description for TIMERxTIM (see <a href="#">Section 1.3.5.1</a> ).

### 1.3.5.3 CPU-Timer x Period (TIMERxPRD) Register

**Figure 1-46. CPU-Timer x Period (TIMERxPRD) Register (x = 0, 1, 2)**

15 0

PRD
-----

R/W-1

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-44. CPU-Timer x Period (TIMERxPRD) Register Field Descriptions**

Bits	Field	Description
15-0	PRD	CPU-Timer Period Registers (PRDH:PRD): The PRD register holds the low 16 bits of the 32-bit period. The PRDH register holds the high 16 bits of the 32-bit period. When the TIMH:TIM decrements to zero, the TIMH:TIM register is reloaded with the period value contained in the PRDH:PRD registers, at the start of the next timer input clock cycle (the output of the prescaler). The PRDH:PRD contents are also loaded into the TIMH:TIM when you set the timer reload bit (TRB) in the Timer Control Register (TCR).

### 1.3.5.4 CPU-Timer x Period (TIMERxPRDH) Register High

**Figure 1-47. CPU-Timer x Period (TIMERxPRDH) Register High (x = 0, 1, 2)**

15 0

PRDH
------

R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-45. CPU-Timer x Period (TIMERxPRDH) Register High Field Descriptions**

Bits	Field	Description
15-0	PRDH	See description for TIMERxPRD (see <a href="#">Section 1.3.5.3</a> ).

### 1.3.5.5 CPU-Timer x Control (TIMERxTCR) Register

**Figure 1-48. CPU-Timer x Control (TIMERxTCR) Register (x = 0, 1, 2)**

15	14	13	12	11	10	9	8
TIF	TIE	Reserved		FREE	SOFT	Reserved	
R/W-0	R/W-0	R-0		R/W-0	R/W-0	R-0	
7	6	5	4	3	0		
Reserved		TRB	TSS	Reserved			
R-0		R/W-0	R/W-0	R-0			

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-46. CPU-Timer x Control (TIMERxTCR) Register Field Descriptions**

Bits	Field	Value	Description														
15	TIF	0 1	CPU-Timer Overflow Flag TIF indicates whether a timer overflow has happened since TIF was last cleared. TIF is not cleared automatically and does not need to be cleared to enable the next timer interrupt. The CPU-Timer has not decremented to zero. Writes of 0 are ignored. This flag gets set when the CPU-timer decrements to zero. Writing a 1 to this bit clears the flag.														
14	TIE	0 1	CPU-Timer Interrupt Enable. The CPU-Timer interrupt is disabled. The CPU-Timer interrupt is enabled. If the timer decrements to zero, and TIE is set, the timer asserts its interrupt request.														
13-12	Reserved		Any writes to these bit(s) must always have a value of 0.														
11-10	FREE SOFT	<table border="1"> <thead> <tr> <th>FREE</th> <th>SOFT</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Stop after the next decrement of the TIMH:TIM (hard stop)</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stop after the TIMH:TIM decrements to 0 (soft stop)</td> </tr> <tr> <td>1</td> <td>0</td> <td>Free run</td> </tr> <tr> <td>1</td> <td>1</td> <td>Free run</td> </tr> </tbody> </table> CPU-Timer Emulation Modes: These bits are special emulation bits that determine the state of the timer when a breakpoint is encountered in the high-level language debugger. If the FREE bit is set to 1, then, upon a software breakpoint, the timer continues to run (that is, free runs). In this case, SOFT is a <i>don't care</i> . But if FREE is 0, then SOFT takes effect. In this case, if SOFT = 0, the timer halts the next time the TIMH:TIM decrements. If the SOFT bit is 1, then the timer halts when the TIMH:TIM has decremented to zero. CPU-Timer Emulation Mode In the SOFT STOP mode, the timer generates an interrupt before shutting down (since reaching 0 is the interrupt causing condition).	FREE	SOFT	Description	0	0	Stop after the next decrement of the TIMH:TIM (hard stop)	0	1	Stop after the TIMH:TIM decrements to 0 (soft stop)	1	0	Free run	1	1	Free run
FREE	SOFT	Description															
0	0	Stop after the next decrement of the TIMH:TIM (hard stop)															
0	1	Stop after the TIMH:TIM decrements to 0 (soft stop)															
1	0	Free run															
1	1	Free run															
9-6	Reserved		Any writes to these bit(s) must always have a value of 0.														
5	TRB	0 1	CPU-Timer Reload bit. The TRB bit is always read as zero. Writes of 0 are ignored. When you write a 1 to TRB, the TIMH:TIM is loaded with the value in the PRDH:PRD, and the prescaler counter (PSCH:PSC) is loaded with the value in the timer divide-down register (TDDR:TDDR).														

**Table 1-46. CPU-Timer x Control (TIMERxTCR) Register Field Descriptions (continued)**

Bits	Field	Value	Description
4	TSS	0	CPU-Timer stop status bit. TSS is a 1-bit flag that stops or starts the CPU-timer. Reads of 0 indicate the CPU-timer is running. To start or restart the CPU-timer, set TSS to 0. At reset, TSS is cleared to 0 and the CPU-timer immediately starts.
		1	Reads of 1 indicate that the CPU-timer is stopped. To stop the CPU-timer, set TSS to 1.
3-0	Reserved		Any writes to these bit(s) must always have a value of 0.

### 1.3.5.6 CPU-Timer x Prescale (TIMERxTPR) Register

**Figure 1-49. CPU-Timer x Prescale (TIMERxTPR) Register (x = 0, 1, 2)**

15	8	7	0
PSC		TDDR	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-47. CPU-Timer x Prescale (TIMERxTPR) Register Field Descriptions**

Bits	Field	Description
15-8	PSC	CPU-Timer Prescale Counter. These bits hold the current prescale count for the timer. For every timer clock source cycle that the PSCH:PSC value is greater than 0, the PSCH:PSC decrements by one. One timer clock (output of the timer prescaler) cycle after the PSCH:PSC reaches 0, the PSCH:PSC is loaded with the contents of the TDDRH:TDDR, and the timer counter register (TIMH:TIM) decrements by one. The PSCH:PSC is also reloaded whenever the timer reload bit (TRB) is set by software. The PSCH:PSC can be checked by reading the register, but it cannot be set directly. It must get its value from the timer divide-down register (TDDRH:TDDR). At reset, the PSCH:PSC is set to 0.
7-0	TDDR	CPU-Timer Divide-Down. Every (TDDRH:TDDR + 1) timer clock source cycles, the timer counter register (TIMH:TIM) decrements by one. At reset, the TDDRH:TDDR bits are cleared to 0. To increase the overall timer count by an integer factor, write this factor minus one to the TDDRH:TDDR bits. When the prescaler counter (PSCH:PSC) value is 0, one timer clock source cycle later, the contents of the TDDRH:TDDR reload the PSCH:PSC, and the TIMH:TIM decrements by one. TDDRH:TDDR also reloads the PSCH:PSC whenever the timer reload bit (TRB) is set by software.

### 1.3.5.7 CPU-Timer x Prescale (TIMERxTPRH) Register High

**Figure 1-50. CPU-Timer x Prescale (TIMERxTPRH) Register High (x = 0, 1, 2)**

15	8      7	0
PSCH	TDDRH	
R-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-48. CPU-Timer x Prescale (TIMERxTPRH) Register High Field Descriptions**

Bits	Field	Description
15-8	PSCH	See description of TIMERxTPR (see <a href="#">Section 1.3.5.6</a> ).
7-0	TDDRH	See description of TIMERxTPR (see <a href="#">Section 1.3.5.6</a> ).

## 1.4 General-Purpose Input/Output (GPIO)

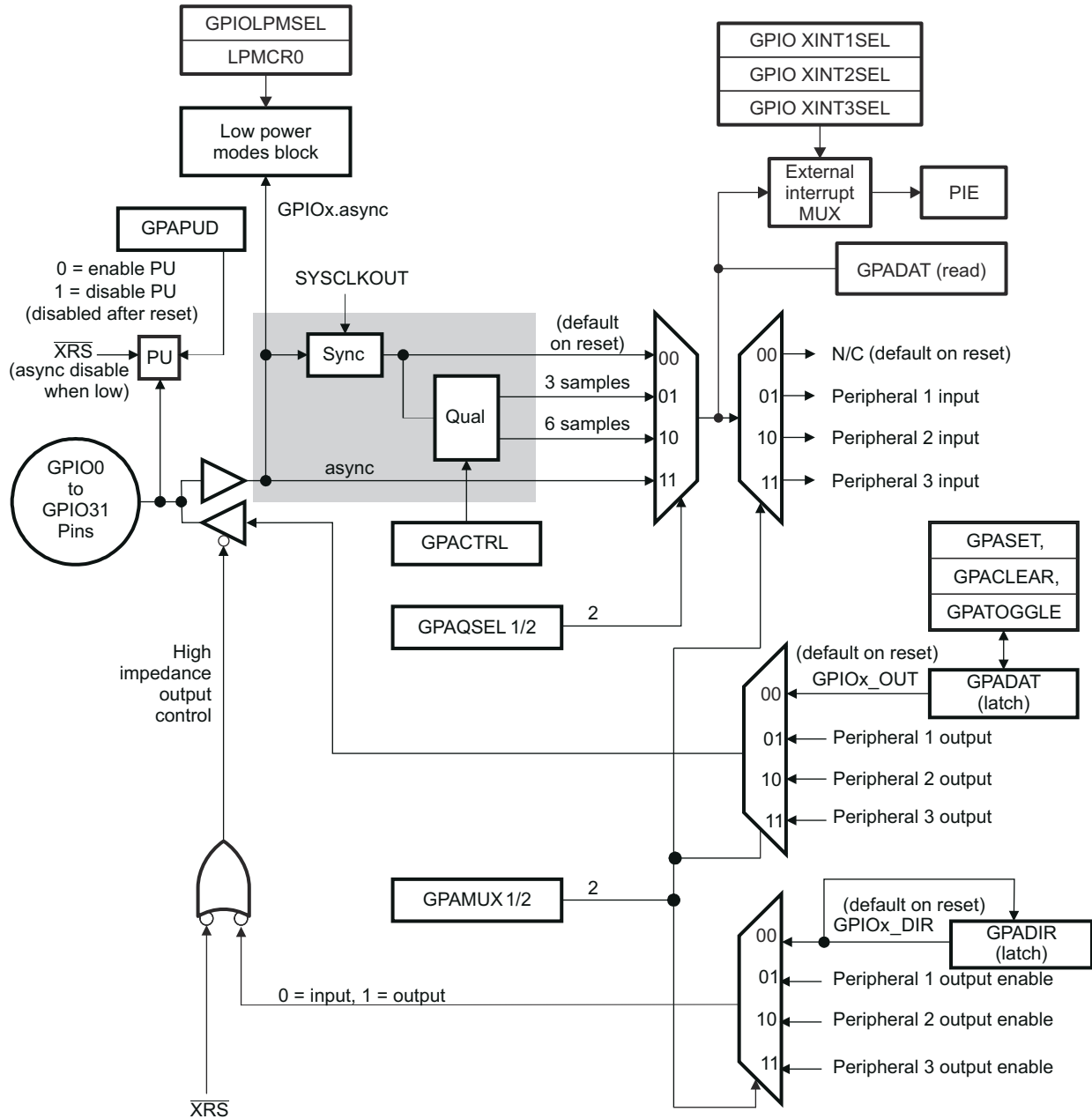
The GPIO multiplexing (MUX) registers are used to select the operation of shared pins. The pins are named by their general-purpose I/O name (that is, GPIO0 - GPIO38). These pins can be individually selected to operate as digital I/O, referred to as GPIO, or connected to one of up to three peripheral I/O signals (by the GPxMUXn registers). If selected for digital I/O mode, registers are provided to configure the pin direction (by the GPxDIR registers). You can also qualify the input signals to remove unwanted noise (by the GPxQSELn, GPaCTRL, and GPBCTRL registers).

### 1.4.1 GPIO Module Overview

Up to three independent peripheral signals are multiplexed on a single GPIO-enabled pin in addition to individual pin bit-I/O capability. There are three I/O ports:

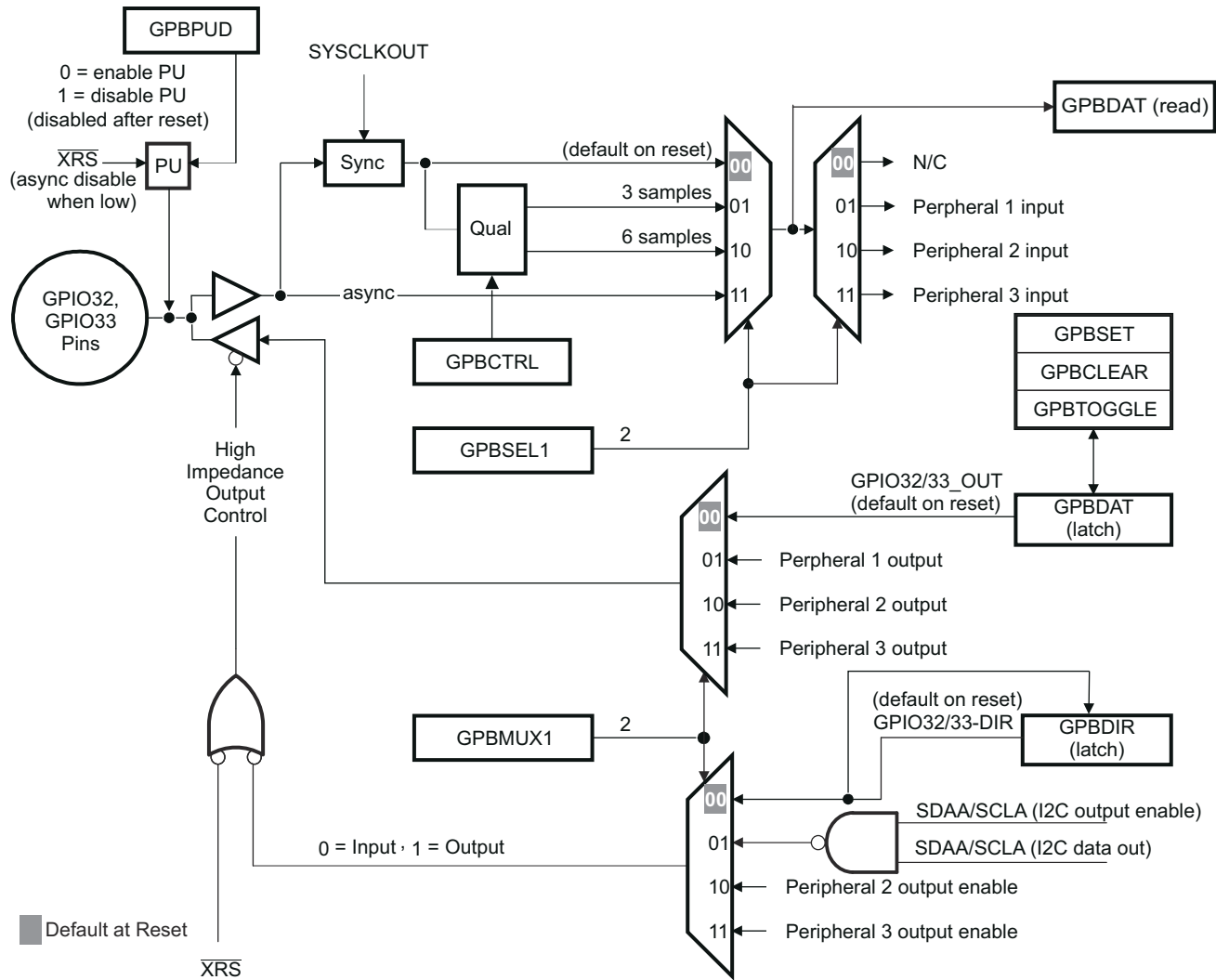
- Port A consists of GPIO0-GPIO31
- Port B consists of GPIO32-GPIO38
- Analog port consists of AIO0-AIO15

[Figure 1-51](#) shows the basic modes of operation for the GPIO module. Note that GPIO functionality is provided on JTAG pins as well.



A. GPxDAT latch/read are accessed at the same memory location.

**Figure 1-51. GPIO0 to GPIO31 Multiplexing Diagram**



- A. The GPIOINENCLK bit in the PCLKCR3 register does not affect the above GPIOs (I<sup>2</sup>C pins) since the pins are bi-directional.
- B. The input qualification circuit is not reset when modes are changed (such as changing from output to input mode). Any state will get flushed by the circuit eventually.

**Figure 1-52. GPIO32, GPIO33 Multiplexing Diagram**



### 1.4.1.1 JTAG Port

On this device, the JTAG port is reduced to five pins ( $\overline{\text{TRST}}$ , TCK, TDI, TMS, and TDO). The TCK, TDI, TMS, and TDO pins are also GPIO pins. The  $\overline{\text{TRST}}$  signal selects either the JTAG or GPIO operating mode for the pins in Figure 1-53.

#### Note

The JTAG pins may also be used as GPIO pins. Care should be taken in the board design to ensure that the circuitry connected to these pins do not affect the emulation capabilities of the JTAG pin function. Any circuitry connected to these pins should not prevent the JTAG debug probe from driving (or being driven by) the JTAG pins for successful debug.

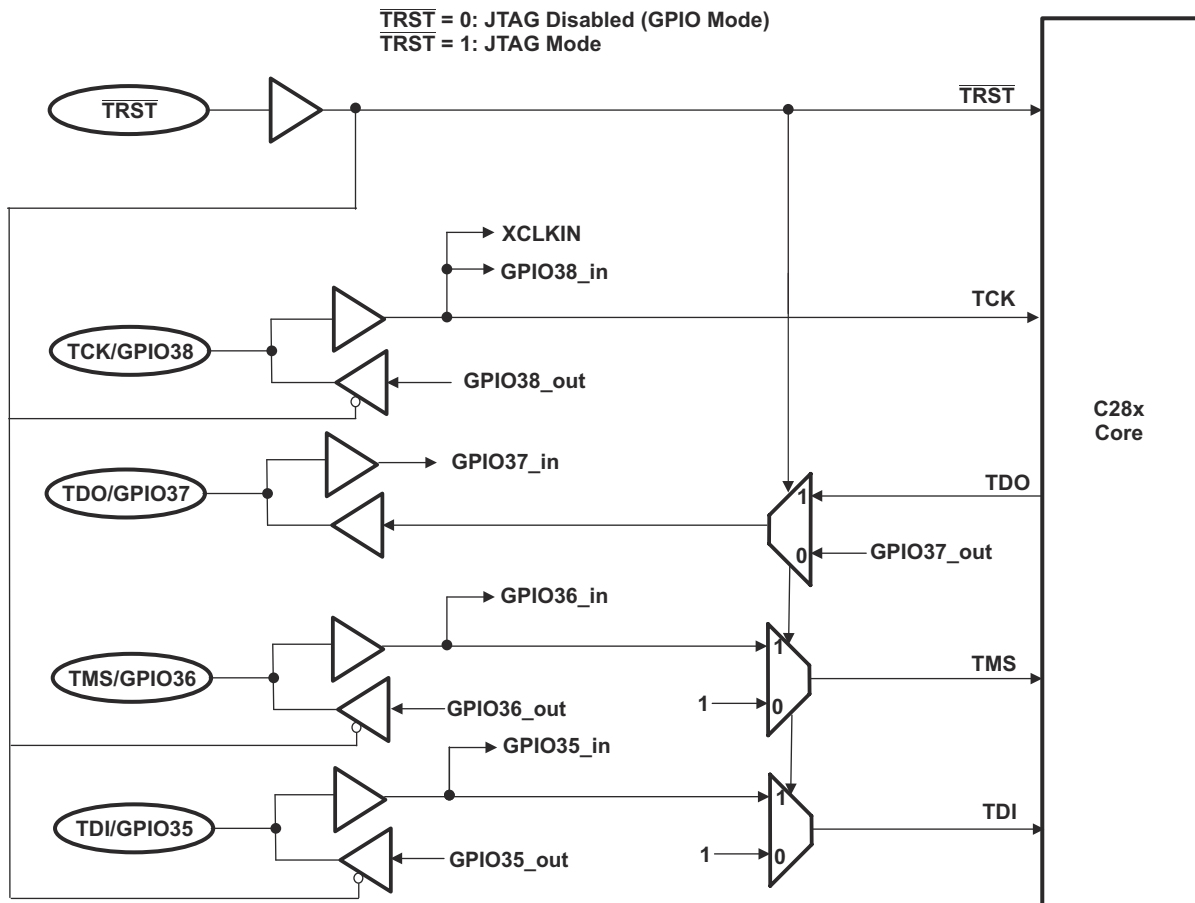


Figure 1-53. JTAG Port/GPIO Multiplexing

### 1.4.1.2 Choosing JTAG or GPIO Functionality

The  $\overline{\text{TRST}}$  signal selects the functionality of the JTAG signals, in combination with the JTAGDIS bit in the JTAGDEBUG register as shown in [Table 1-49](#).

**Table 1-49. JTAG Port Mode**

TRST	JTAGDIS bit	JTAG Port Mode
0	X	GPIO mode enabled, JTAG port disabled
1	0	JTAG port enabled (GPIOs should be configured as inputs)
1	1	GPIO mode enabled, JTAG port disabled

The JTAGDEBUG register is shown in [Figure 1-54](#) and described in [Table 1-50](#).

**Figure 1-54. JTAGDEBUG Register (Address 0x702A, EALLOW protected)**

15	1	0
Reserved		JTAGDIS
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-50. JTAGDEBUG Register Field Descriptions**

Bits	Field	Value	Description
15-1	Reserved		Any writes to these bits must always have a value of 0.
0	JTAGDIS	0 JTAG Port Enabled 1 JTAG Port Disabled (GPIO Mode)	JTAG Port Disable Bit: This bit enables/disables the JTAG port. When disabled, the JTAG pins can be used as GPIOs.  This bit is reset by $\overline{\text{TRST}}$ . The bit is forced to "0" when $\overline{\text{TRST}}$ is "0". When $\overline{\text{TRST}}$ is "1", then JTAGDIS bit can be modified by CPU.  <b>Note:</b> Ensure no contention with the debug probe signals when JTAGDIS=1.



## 1.4.2 Configuration Overview

The pin function assignments, input qualification, and the external interrupt sources are all controlled by the GPIO configuration control registers. In addition, you can assign pins to wake the device from the HALT and STANDBY low power modes and enable/disable internal pullup resistors. [Table 1-51](#) and [Table 1-52](#) list the registers that are used to configure the GPIO pins to match the system requirements.

**Table 1-51. GPIO Control Registers**

Name <sup>(1)</sup>	Address	Size (x16)	Register Description	Bit Description
GPACTRL	0x6F80	2	GPIO A Control Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.1</a>
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register (GPIO0-GPIO15)	<a href="#">Section 1.4.6.2</a>
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register (GPIO16-GPIO31)	<a href="#">Section 1.4.6.3</a>
GPAMUX1	0x6F86	2	GPIO A MUX 1 Register (GPIO0-GPIO15)	<a href="#">Section 1.4.6.4</a>
GPAMUX2	0x6F88	2	GPIO A MUX 2 Register (GPIO16-GPIO31)	<a href="#">Section 1.4.6.5</a>
GPADIR	0x6F8A	2	GPIO A Direction Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.6</a>
GPAPUD	0x6F8C	2	GPIO A Pullup Disable Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.7</a>
GPBCTRL	0x6F90	2	GPIO B Control Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.8</a>
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.9</a>
GPBMUX1	0x6F96	2	GPIO B MUX 1 Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.10</a>
GPBDIR	0x6F9A	2	GPIO B Direction Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.11</a>
GPBPUD	0x6F9C	2	GPIO B Pullup Disable Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.12</a>
AIOMUX1	0x6FB6	2	Analog, I/O MUX 1 register (AIO0-AIO15)	<a href="#">Section 1.4.6.13</a>
AIODIR	0x6FBA	2	Analog, I/O Direction Register (AIO0-AIO15)	<a href="#">Section 1.4.6.14</a>

(1) The registers in this table are EALLOW protected. See [Section 1.5.2](#) for more information.

**Table 1-52. GPIO Interrupt and Low Power Mode Select Registers**

Name <sup>(1)</sup>	Address	Size (x16)	Register Description	Bit Description
GPIOXINT1SEL	0x6FE0	1	XINT1 Source Select Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.15</a>
GPIOXINT2SEL	0x6FE1	1	XINT2 Source Select Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.15</a>
GPIOXINT3SEL	0x6FE2	1	XINT3 Source Select Register (GPIO0 - GPIO31)	<a href="#">Section 1.4.6.15</a>
GPIOLPMSEL	0x6FE8	1	Low Power Mode Wakeup Source Select Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.16</a>

(1) The registers in this table are EALLOW protected. See [Section 1.5.2](#) for more information.

To plan configuration of the GPIO module, consider the following steps:

### 1. Plan the device pin-out:

Through a pin multiplexing scheme, a lot of flexibility is provided for assigning functionality to the GPIO-capable pins. Before getting started, look at the peripheral options available for each pin, and plan pin-out for your specific system. Will the pin be used as a general purpose input or output (GPIO) or as one of up to three available peripheral functions? Knowing this information will help determine how to further configure the pin.

### 2. Enable or disable internal pull-up resistors:

To enable or disable the internal pullup resistors, write to the respective bits in the GPIO pullup disable (GPAPUD and GPBPUD) registers. For pins that can function as ePWM output pins, the internal pullup resistors are disabled by default. All other GPIO-capable pins have the pullup enabled by default. The AIOx pins do not have internal pull-up resistors.

**3. Select input qualification:**

If the pin will be used as an input, specify the required input qualification, if any. The input qualification is specified in the GPACTRL, GPBCTRL, GPAQSEL1, GPAQSEL2, GPBQSEL1, and GPBQSEL2 registers. By default, all of the input signals are synchronized to SYSCLKOUT only.

**4. Select the pin function:**

Configure the GPxMUXn or AIOMUXn registers such that the pin is a GPIO or one of three available peripheral functions. By default, all GPIO-capable pins are configured at reset as general purpose input pins.

**5. For digital general purpose I/O, select the direction of the pin:**

If the pin is configured as an GPIO, specify the direction of the pin as either input or output in the GPADIR, GPBDIR, or AIODIR registers. By default, all GPIO pins are inputs. To change the pin from input to output, first load the output latch with the value to be driven by writing the appropriate value to the GPxCLEAR, GPxSET, or GPxTOGGLE (or AIOCLEAR, AIOSET, or AIOTOGGLE) registers. Once the output latch is loaded, change the pin direction from input to output via the GPxDIR registers. The output latch for all pins is cleared at reset.

**6. Select low power mode wake-up sources:**

Specify which pins, if any, will be able to wake the device from HALT and STANDBY low power modes. The pins are specified in the GPIOLPMSEL register.

**7. Select external interrupt sources:**

Specify the source for the XINT1 - XINT3 interrupts. For each interrupt you can specify one of the port A signals as the source. This is done by specifying the source in the GPIOXINTnSEL register. The polarity of the interrupts can be configured in the XINTnCR register as described in [Section 1.6.5](#).

---

**Note**

There is a 2-SYSCLKOUT cycle delay from when a write to configuration registers such as GPxMUXn and GPxQSELn occurs to when the action is valid.

---

### 1.4.3 Digital General-Purpose I/O Control

For pins that are configured as GPIO, the values on the pins are changed by using the registers in [Table 1-53](#).

**Table 1-53. GPIO Data Registers**

Name <sup>(1)</sup>	Address	Size (x16)	Register Description	Bit Description
GPADAT	0x6FC0	2	GPIO A Data Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.17</a>
GPASET	0x6FC2	2	GPIO A Set Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.18</a>
GPACLEAR	0x6FC4	2	GPIO A Clear Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.18</a>
GPATOGGLE	0x6FC6	2	GPIO A Toggle Register (GPIO0-GPIO31)	<a href="#">Section 1.4.6.18</a>
GPBDAT	0x6FC8	2	GPIO B Data Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.19</a>
GPBSET	0x6FCA	2	GPIO B Set Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.20</a>
GPBCLEAR	0x6FCC	2	GPIO B Clear Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.20</a>
GPBTOGGLE	0x6FCE	2	GPIO B Toggle Register (GPIO32-GPIO38)	<a href="#">Section 1.4.6.20</a>
AIODAT	0x6FD8	2	Analog I/O Data Register (AIO0-AIO15)	<a href="#">Section 1.4.6.21</a>
AIOSET	0x6FDA	2	Analog I/O Data Set Register (AIO0-AIO15)	<a href="#">Section 1.4.6.22</a>
AIOCLEAR	0x6FDC	2	Analog I/O Clear Register (AIO0-AIO15)	<a href="#">Section 1.4.6.22</a>
AIOGGLE	0x6FDE	2	Analog I/O Toggle Register (AIO0-AIO15)	<a href="#">Section 1.4.6.22</a>

(1) The registers in this table are not EALLOW protected.

#### • GPxDAT/AIODAT Registers

Each I/O port has one data register. Each bit in the data register corresponds to one GPIO pin. No matter how the pin is configured (GPIO or peripheral function), the corresponding bit in the data register reflects the current state of the pin after qualification (This does not apply to AIOx pins). Writing to the GPxDAT/AIODAT register clears or sets the corresponding output latch and if the pin is enabled as a general purpose output (GPIO output) the pin will also be driven either low or high. If the pin is not configured as a GPIO output then the value will be latched, but the pin will not be driven. Only if the pin is later configured as a GPIO output, will the latched value be driven onto the pin.

When using the GPxDAT register to change the level of an output pin, you should be cautious not to accidentally change the level of another pin. For example, if you mean to change the output latch level of GPIOA1 by writing to the GPADAT register bit 0 using a read-modify-write instruction, a problem can occur if another I/O port A signal changes level between the read and the write stage of the instruction. Following is an analysis of why this happens:

The GPxDAT registers reflect the state of the pin, not the latch. This means the register reflects the actual pin value. However, there is a lag between when the register is written to when the new pin value is reflected back in the register. This may pose a problem when this register is used in subsequent program statements to alter the state of GPIO pins. An example is shown below where two program statements attempt to drive two different GPIO pins that are currently low to a high state.

If Read-Modify-Write operations are used on the GPxDAT registers, because of the delay between the output and the input of the first instruction (I1), the second instruction (I2) will read the old value and write it back.

```
GpioDataRegs.GPADAT.bit.GPIO1 = 1 ; I1 performs read-modify-write of GPADAT
GpioDataRegs.GPADAT.bit.GPIO2 = 1 ; I2 also a read-modify-write of GPADAT.
                                ; It gets the old value of GPIO1 due to the delay
```

The second instruction will wait for the first to finish its write due to the write-followed-by-read protection on this peripheral frame. There will be some lag, however, between the write of (I1) and the GPxDAT bit reflecting the new value (1) on the pin. During this lag, the second instruction will read the old value of GPIO1 (0) and write it back along with the new value of GPIO2 (1). Therefore, GPIO1 pin stays low.

One solution is to put some NOP's between instructions. A better solution is to use the GPxSET/GPxCLEAR/GPxTOGGLE registers instead of the GPxDAT registers. These registers always read back a 0 and writes of

0 have no effect. Only bits that need to be changed can be specified without disturbing any other bits that are currently in the process of changing.

- **GPxSET/AIOSET Registers**

The set registers are used to drive specified GPIO pins high without disturbing other pins. Each I/O port has one set register and each bit corresponds to one GPIO pin. The set registers always read back 0. If the corresponding pin is configured as an output, then writing a 1 to that bit in the set register will set the output latch high and the corresponding pin will be driven high. If the pin is not configured as a GPIO output, then the value will be latched but the pin will not be driven. Only if the pin is later configured as a GPIO output will the latched value will be driven onto the pin. Writing a 0 to any bit in the set registers has no effect.

- **GPxCLEAR/AIOCLEAR Registers**

The clear registers are used to drive specified GPIO pins low without disturbing other pins. Each I/O port has one clear register. The clear registers always read back 0. If the corresponding pin is configured as a general purpose output, then writing a 1 to the corresponding bit in the clear register will clear the output latch and the pin will be driven low. If the pin is not configured as a GPIO output, then the value will be latched but the pin will not be driven. Only if the pin is later configured as a GPIO output will the latched value will be driven onto the pin. Writing a 0 to any bit in the clear registers has no effect.

- **GPxTOGGLE/AIOTOGGLE Registers**

The toggle registers are used to drive specified GPIO pins to the opposite level without disturbing other pins. Each I/O port has one toggle register. The toggle registers always read back 0. If the corresponding pin is configured as an output, then writing a 1 to that bit in the toggle register flips the output latch and pulls the corresponding pin in the opposite direction. That is, if the output pin is driven low, then writing a 1 to the corresponding bit in the toggle register will pull the pin high. Likewise, if the output pin is high, then writing a 1 to the corresponding bit in the toggle register will pull the pin low. If the pin is not configured as a GPIO output, then the value will be latched but the pin will not be driven. Only if the pin is later configured as a GPIO output will the latched value will be driven onto the pin. Writing a 0 to any bit in the toggle registers has no effect.

#### 1.4.4 Input Qualification

The input qualification scheme has been designed to be very flexible. You can select the type of input qualification for each GPIO pin by configuring the GPAQSEL1, GPAQSEL2, GPBQSEL1 and GPBQSEL2 registers. In the case of a GPIO input pin, the qualification can be specified as only synchronize to SYSCLKOUT or qualification by a sampling window. For pins that are configured as peripheral inputs, the input can also be asynchronous in addition to synchronized to SYSCLKOUT or qualified by a sampling window. The remainder of this section describes the options available.

##### 1.4.4.1 No Synchronization (asynchronous input)

This mode is used for peripherals where input synchronization is not required or the peripheral itself performs the synchronization. Examples include communication ports SCI, SPI, and I<sup>2</sup>C. In addition, it may be desirable to have the ePWM trip zone ( $\overline{TZn}$ ) signals function independent of the presence of SYSCLKOUT.

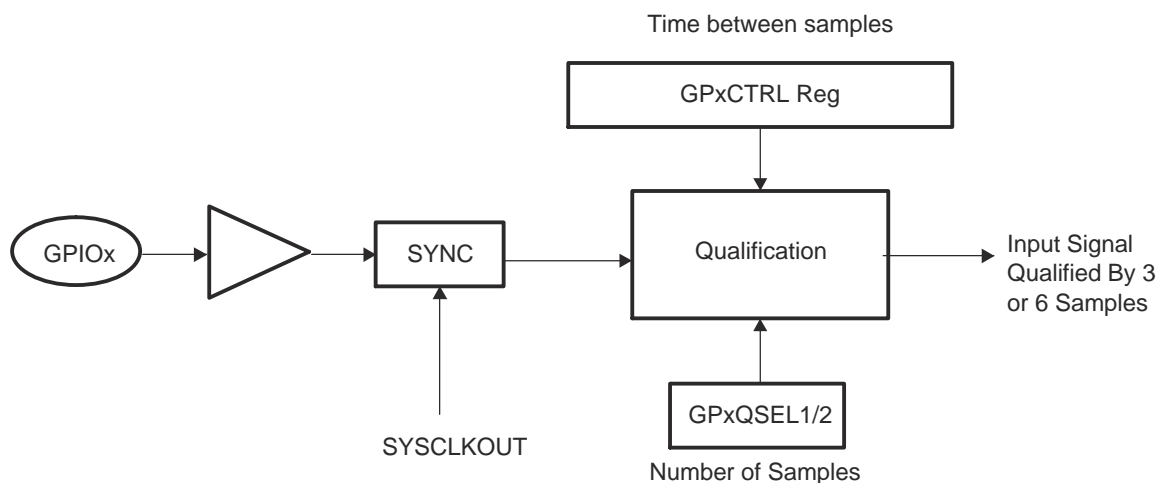
The asynchronous option is not valid if the pin is used as a general purpose digital input pin (GPIO). If the pin is configured as a GPIO input and the asynchronous option is selected then the qualification defaults to synchronization to SYSCLKOUT as described in [Section 1.4.4.2](#).

##### 1.4.4.2 Synchronization to SYSCLKOUT Only

This is the default qualification mode of all the pins at reset. In this mode, the input signal is only synchronized to the system clock (SYSCLKOUT). Because the incoming signal is asynchronous, it can take up to a SYSCLKOUT period of delay in order for the input to the device to be changed. No further qualification is performed on the signal.

### 1.4.4.3 Qualification Using a Sampling Window

In this mode, the signal is first synchronized to the system clock (SYSCLKOUT) and then qualified by a specified number of cycles before the input is allowed to change. Figure 1-56 and Figure 1-57 show how the input qualification is performed to eliminate unwanted noise. Two parameters are specified by the user for this type of qualification: 1) the sampling period, or how often the signal is sampled, and 2) the number of samples to be taken.



**Figure 1-56. Input Qualification Using a Sampling Window**

#### Time between samples (sampling period):

To qualify the signal, the input signal is sampled at a regular period. The sampling period is specified by the user and determines the time duration between samples, or how often the signal will be sampled, relative to the CPU clock (SYSCLKOUT).

The sampling period is specified by the qualification period (QUALPRDn) bits in the GPxCTRL register. The sampling period is configurable in groups of 8 input signals. For example, GPIO0 to GPIO7 use GPCCTRL[QUALPRD0] setting and GPIO8 to GPIO15 use GPCCTRL[QUALPRD1]. Table 1-54 and Table 1-55 show the relationship between the sampling period or sampling frequency and the GPxCTRL[QUALPRDn] setting.

**Table 1-54. Sampling Period**

Sampling Period	
If GPxCTRL[QUALPRDn] = 0	$1 \times T_{\text{SYSCLKOUT}}$
If GPxCTRL[QUALPRDn] ≠ 0	$2 \times \text{GPxCTRL[QUALPRDn]} \times T_{\text{SYSCLKOUT}}$
Where $T_{\text{SYSCLKOUT}}$ is the period in time of SYSCLKOUT	

**Table 1-55. Sampling Frequency**

Sampling Period	
If GPxCTRL[QUALPRDn] = 0	$f_{\text{SYSCLKOUT}}$
If GPxCTRL[QUALPRDn] ≠ 0	$f_{\text{SYSCLKOUT}} \times 1 \div (2 \times \text{GPxCTRL[QUALPRDn]})$
Where $f_{\text{SYSCLKOUT}}$ is the frequency of SYSCLKOUT	



From these equations, the minimum and maximum time between samples can be calculated for a given SYSCLKOUT frequency:

**Example: Maximum Sampling Frequency**

If GPxCTRL[QUALPRDn] = 0 then the sampling frequency is  $f_{\text{SYSCLKOUT}}$   
 If, for example,  $f_{\text{SYSCLKOUT}} = 60 \text{ MHz}$  then the signal will be sampled at 60 MHz or one sample every 16.67 ns.

**Example: Minimum Sampling Frequency**

If GPxCTRL[QUALPRDn] = 0xFF (that is, 255) then the sampling frequency is  $f_{\text{SYSCLKOUT}} \times 1 \div (2 \times \text{GPxCTRL[QUALPRDn]})$   
 If, for example,  $f_{\text{SYSCLKOUT}} = 60 \text{ MHz}$  then the signal will be sampled at  $60 \text{ MHz} \times 1 \div (2 \times 255)$  or one sample every 8.5  $\mu\text{s}$ .

**Number of samples:**

The number of times the signal is sampled is either 3 samples or 6 samples as specified in the qualification selection (GPAQSEL1, GPAQSEL2, GPBQSEL1, and GPBQSEL2) registers. When 3 or 6 consecutive cycles are the same, then the input change will be passed through to the device.

**Total Sampling Window Width:**

The sampling window is the time during which the input signal will be sampled as shown in [Figure 1-57](#). By using the equation for the sampling period along with the number of samples to be taken, the total width of the window can be determined.

For the input qualifier to detect a change in the input, the level of the signal must be stable for the duration of the sampling window width or longer.

The number of sampling periods within the window is always one less than the number of samples taken. For a three-sample window, the sampling window width is 2 sampling periods wide where the sampling period is defined in [Table 1-54](#). Likewise, for a six-sample window, the sampling window width is 5 sampling periods wide. [Table 1-56](#) and [Table 1-57](#) show the calculations that can be used to determine the total sampling window width based on GPxCTRL[QUALPRDn] and the number of samples taken.

**Table 1-56. Case 1: Three-Sample Sampling Window Width**

Total Sampling Window Width	
If GPxCTRL[QUALPRDn] = 0	$2 \times T_{\text{SYSCLKOUT}}$
If GPxCTRL[QUALPRDn] $\neq$ 0	$2 \times 2 \times \text{GPxCTRL[QUALPRDn]} \times T_{\text{SYSCLKOUT}}$
Where $T_{\text{SYSCLKOUT}}$ is the period in time of SYSCLKOUT	

**Table 1-57. Case 2: Six-Sample Sampling Window Width**

Total Sampling Window Width	
If GPxCTRL[QUALPRDn] = 0	$5 \times T_{\text{SYSCLKOUT}}$
If GPxCTRL[QUALPRDn] $\neq$ 0	$5 \times 2 \times \text{GPxCTRL[QUALPRDn]} \times T_{\text{SYSCLKOUT}}$
Where $T_{\text{SYSCLKOUT}}$ is the period in time of SYSCLKOUT	

### Note

The external signal change is asynchronous with respect to both the sampling period and SYSCLKOUT. Due to the asynchronous nature of the external signal, the input should be held stable for a time greater than the sampling window width to make sure the logic detects a change in the signal. The extra time required can be up to an additional sampling period +  $T_{\text{SYSCLKOUT}}$ .

The required duration for an input signal to be stable for the qualification logic to detect a change is described in the device specific data manual.

### Example Qualification Window:

For the example shown in Figure 1-57, the input qualification has been configured as follows:

- $\text{GPxQSEL1/2} = 1,0$ . This indicates a six-sample qualification.
- $\text{GPxCTRL[QUALPRDn]} = 1$ . The sampling period is  $t_w(\text{SP}) = 2 \times \text{GPxCTRL[QUALPRDn]} \times T_{\text{SYSCLKOUT}}$ .

This configuration results in the following:

- The width of the sampling window is: .

$$t_w(\text{IQSW}) = 5 \times t_w(\text{SP}) = 5 \times 2 \times \text{GPxCTRL[QUALPRDn]} \times T_{\text{SYSCLKOUT}} \text{ or } 5 \times 2 \times T_{\text{SYSCLKOUT}}$$

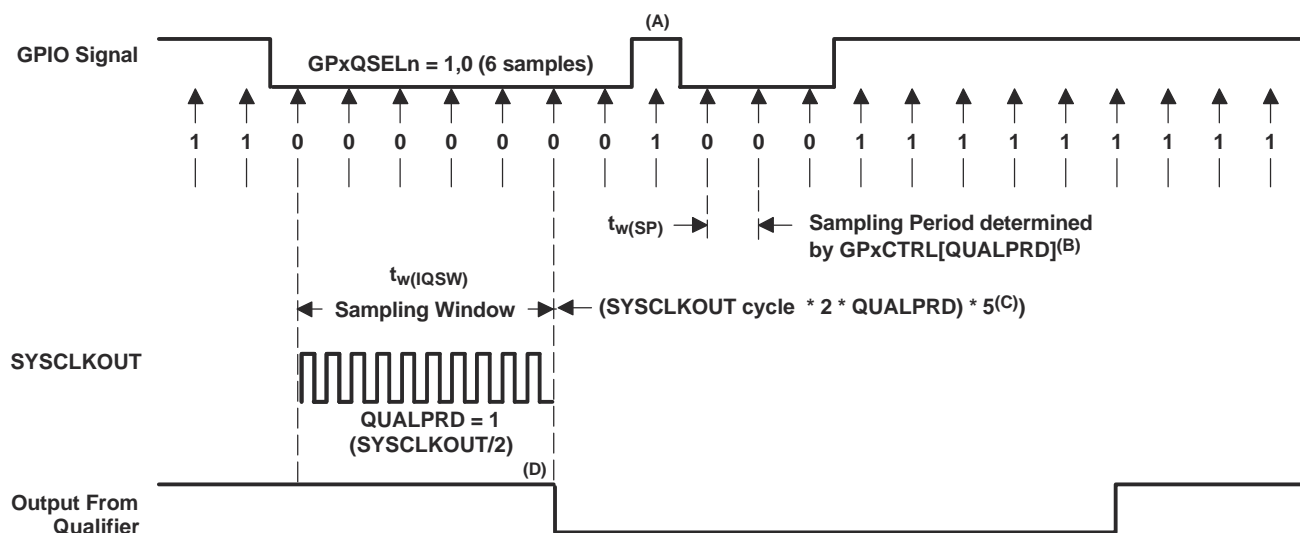
- If, for example,  $T_{\text{SYSCLKOUT}} = 16.67 \text{ ns}$ , then the duration of the sampling window is:

$$t_w(\text{IQSW}) = 5 \times 2 \times 16.67 \text{ ns} = 166.7 \text{ ns}.$$

- To account for the asynchronous nature of the input relative to the sampling period and SYSCLKOUT, up to an additional sampling period,  $t_w(\text{SP})$ , +  $T_{\text{SYSCLKOUT}}$  may be required to detect a change in the input signal. For this example:

$$t_w(\text{SP}) + T_{\text{SYSCLKOUT}} = 333.4 \text{ ns} + 166.67 \text{ ns} = 500.1 \text{ ns}$$

- In Figure 1-57, the glitch (A) is shorter than the qualification window and will be ignored by the input qualifier.



- This glitch will be ignored by the input qualifier. The QUALPRD bit field specifies the qualification sampling period. It can vary from 00 to 0xFF. If QUALPRD = 00, then the sampling period is 1 SYSCLKOUT cycle. For any other value "n", the qualification sampling period is 2n SYSCLKOUT cycles (i.e., at every 2n SYSCLKOUT cycles, the GPIO pin will be sampled).
- The qualification period selected via the GPxCTRL register applies to groups of 8 GPIO pins.
- The qualification block can take either three or six samples. The GPxQSELn Register selects which sample mode is used.
- In the example shown, for the qualifier to detect the change, the input should be stable for 10 SYSCLKOUT cycles or greater. In other words, the inputs should be stable for  $(5 \times \text{QUALPRD} \times 2)$  SYSCLKOUT cycles. That would ensure 5 sampling periods for detection to occur. Since external signals are driven asynchronously, a 13-SYSCLKOUT-wide pulse ensures reliable recognition.

**Figure 1-57. Input Qualifier Clock Cycles**

### 1.4.5 GPIO and Peripheral Multiplexing (MUX)

Up to three different peripheral functions are multiplexed along with a general input/output (GPIO) function per pin. This allows you to pick and choose a peripheral mix that will work best for the particular application.

[Table 1-59](#) and [Table 1-60](#) show an overview of the possible multiplexing combinations sorted by GPIO pin. The second column indicates the I/O name of the pin on the device. Since the I/O name is unique, it is the best way to identify a particular pin. Therefore, the register descriptions in this section refer only to the GPIO name of a particular pin. The MUX register and particular bits that control the selection for each pin are indicated in the first column.

For example, the multiplexing for the GPIO6 pin is controlled by writing to GPAMUX[13:12]. By writing to these bits, the pin is configured as either GPIO6, or one of up to three peripheral functions. The GPIO6 pin can be configured as:

GPAMUX1[13:12] Bit Setting	Pin Functionality Selected
If GPAMUX1[13:12] = 0,0	Pin configured as GPIO6
If GPAMUX1[13:12] = 0,1	Pin configured as EPWM4A (O)
If GPAMUX1[13:12] = 1,0	Pin configured as EPWMSYNCl (I)
If GPAMUX1[13:12] = 1,1	Pin configured as EPWMSYNCO (O)

If a peripheral is not available on a particular device, that MUX selection is reserved on that device and should not be used.

#### Note

If you should select a reserved GPIO MUX configuration that is not mapped to a peripheral, the state of the pin will be undefined and the pin may be driven. Reserved configurations are for future expansion and should not be selected. In the device MUX tables ([Table 1-59](#) and [Table 1-60](#)), these options are indicated as Reserved.

Some peripherals can be assigned to more than one pin with the MUX registers. For example, in the 2802x device, the SCIRXDA can be assigned to either the GPIO19 or GPIO28 pin, depending on individual system requirements as:

Pin Assigned to SCIRXDA	MUX Configuration
Choice 1 - GPIO19	GPAMUX2[7:6] = 1,0
or Choice 2 - GPIO28	GPAMUX2[25:24] = 0,1

If no pin is configured as an input to a peripheral or if more than one pin is configured as an input for the same peripheral, then the input to the peripheral will either default to a 0 or a 1 as shown in [Table 1-58](#). For example, if SCIRXDA were assigned to both GPIO19 and GPIO28, the input to the SPI peripheral would default to a high state as shown in [Table 1-58](#) and the input would not be connected to GPIO19 or GPIO28.

**Table 1-58. Default State of Peripheral Input**

Peripheral Input	Description	Default Input <sup>(1)</sup>
TZ1- TZ3	Trip zone 1-3	1
EPWMSYNCI	ePWM Synch Input	0
ECAP1	eCAP1 input	1
SPICLKA	SPI-A clock	1
SPISTEA	SPI-A transmit enable	0
SPISIMOA	SPI-A Slave-in, master-out	1
SPISOMIA	SPI-A Slave-out, master-in	1
SCIRXDA - SCIRXDB	SCI-A - SCI-B receive	1
SDAA	I <sup>2</sup> C data	1
SCLA1	I <sup>2</sup> C clock	1

- (1) This value will be assigned to the peripheral input, if more then one pin has been assigned to the peripheral function in the GPxMUX1/2 registers or if no pin has been assigned.

**Table 1-59. 2802x GPIOA MUX**

	Default at Reset Primary I/O Function	Peripheral Selection 1	Peripheral Selection 2	Peripheral Selection 3
GPAMUX1 Register Bits	(GPAMUX1 bits = 00)	(GPAMUX1 bits = 01)	(GPAMUX1 bits = 10)	(GPAMUX1 bits = 11)
1-0	GPIO0	EPWM1A (O)	Reserved <sup>(1)</sup>	Reserved <sup>(1)</sup>
3-2	GPIO1	EPWM1B (O)	Reserved	COMP1OUT (O)
5-4	GPIO2	EPWM2A (O)	Reserved	Reserved <sup>(1)</sup>
7-6	GPIO3	EPWM2B (O)	Reserved	COMP2OUT (O)
9-8	GPIO4	EPWM3A (O)	Reserved	Reserved <sup>(1)</sup>
11-10	GPIO5	EPWM3B (O)	Reserved	ECAP1 (I/O)
13-12	GPIO6	EPWM4A (O)	EPWMSYNCl (I)	EPWMSYNCO (O)
15-14	GPIO7	EPWM4B (O)	SCIRXDA (I)	Reserved
17-16	Reserved	Reserved	Reserved	Reserved
19-18	Reserved	Reserved	Reserved	Reserved
21-20	Reserved	Reserved	Reserved	Reserved
23-22	Reserved	Reserved	Reserved	Reserved
25-24	GPIO12	TZ1 (I)	SCITXDA (O)	Reserved
27-26	Reserved	Reserved	Reserved	Reserved
29-28	Reserved	Reserved	Reserved	Reserved
31-30	Reserved	Reserved	Reserved	Reserved
GPAMUX2 Register Bits	(GPAMUX2 bits = 00)	(GPAMUX2 bits = 01)	(GPAMUX2 bits = 10)	(GPAMUX2 bits = 11)
1-0	GPIO16	SPISIMOA (I/O)	Reserved	TZ2 (I)
3-2	GPIO17	SPISOMIA (I/O)	Reserved	TZ3 (I)
5-4	GPIO18	SPICLKA (I/O)	SCITXDA (O)	XCLKOUT (O)
7-6	GPIO19/XCLKIN	SPISTEA (I/O)	SCIRXDA (I)	ECAP1 (I/O)
9-8	Reserved	Reserved	Reserved	Reserved
11-10	Reserved	Reserved	Reserved	Reserved
13-12	Reserved	Reserved	Reserved	Reserved
15-14	Reserved	Reserved	Reserved	Reserved
17-16	Reserved	Reserved	Reserved	Reserved
19-18	Reserved	Reserved	Reserved	Reserved
21-20	Reserved	Reserved	Reserved	Reserved
23-22	Reserved	Reserved	Reserved	Reserved
25-24	GPIO28	SCIRXDA (I)	SDAA (I/OC)	TZ2 (O)
27-26	GPIO29	SCITXDA (O)	SCLA (I/OC)	TZ3 (O)
29-28	Reserved	Reserved	Reserved	Reserved
31-30	Reserved	Reserved	Reserved	Reserved

- (1) The word Reserved means that there is no peripheral assigned to this GPxMUX1/2 register setting. Should it be selected, the state of the pin will be undefined and the pin may be driven. This selection is a reserved configuration for future expansion.

**Table 1-60. 2802x GPIOB MUX**

GPBMUX1 Register Bits	Default at Reset			
	Primary I/O Function (GPBMUX1 bits = 00)	Peripheral Selection 1 (GPBMUX1 bits = 01)	Peripheral Selection 2 (GPBMUX1 bits = 10)	Peripheral Selection 3 (GPBMUX1 bits = 11)
1,0	GPIO32	SDAA (I/OC)	EPWMSYNCI (I)	ADCSOCAO (O)
3,2	GPIO33	SCLA (I/OC)	EPWMSYNCO (O)	ADCSOCBO (O)
5,4	GPIO34	COMP2OUT (O)	Reserved	Reserved
7,6	GPIO35 (TDI)	Reserved	Reserved	Reserved
9,8	GPIO36 (TMS)	Reserved	Reserved	Reserved
11,10	GPIO37 (TDO)	Reserved	Reserved	Reserved
13,12	GPIO38/XCLKIN (TCK)	Reserved	Reserved	Reserved
15,14	Reserved	Reserved	Reserved	Reserved
17,16	Reserved	Reserved	Reserved	Reserved
19,18	Reserved	Reserved	Reserved	Reserved
21,20	Reserved	Reserved	Reserved	Reserved
23,22	Reserved	Reserved	Reserved	Reserved
25,24	Reserved	Reserved	Reserved	Reserved
27,26	Reserved	Reserved	Reserved	Reserved
29,28	Reserved	Reserved	Reserved	Reserved
31,30	Reserved	Reserved	Reserved	Reserved

**Table 1-61. Analog MUX**

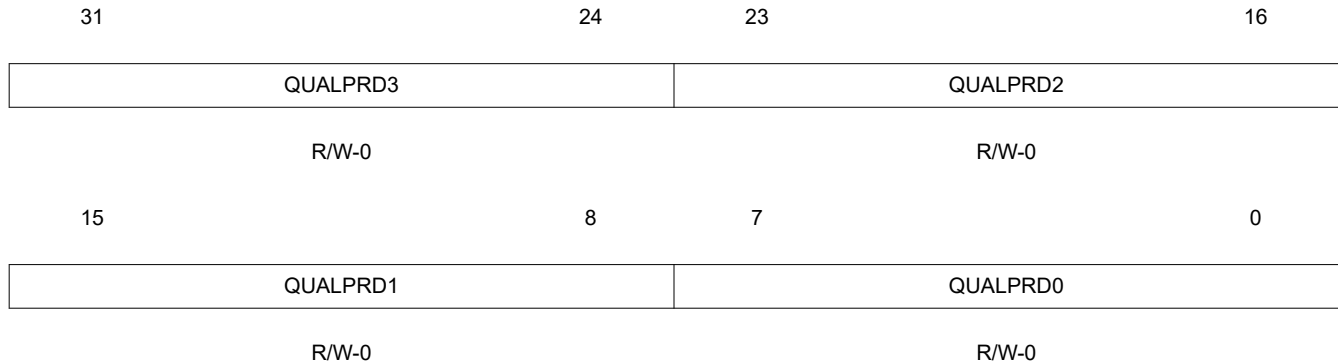
AIOMUX1 Register Bits	Default at Reset	
	AIOx and Peripheral Selection 1	Peripheral Selection 2 and Peripheral Selection 3
	AIOMUX1 Bits = 0,x	AIOMUX1 Bits = 1,x
1-0	ADCINA0 (I)	ADCINA0 (I)
3-2	ADCINA1 (I)	ADCINA1 (I)
5-4	AIO2 (I/O)	ADCINA2 (I), COMP1A (I)
7-6	ADCINA3 (I)	ADCINA3 (I)
9-8	AIO4 (I/O)	ADCINA4 (I), COMP2A (I)
11-10	ADCINA5 (I)	ADCINA5 (I)
13-12	AIO6 (I/O)	ADCINA6 (I), COMP3A (1)
15-14	ADCINA7 (I)	ADCINA7 (I)
17-16	ADCINB0 (I)	ADCINB0 (I)
19-18	ADCINB1 (I)	ADCINB1 (I)
21-20	AIO10 (I/O)	ADCINB2 (I), COMP1B (I)
23-22	ADCINB3 (I)	ADCINB3 (I)
25-24	AIO12 (I/O)	ADCINB4 (I), COMP2B (I)
27-26	ADCINB5 (I)	ADCINB5 (I)
29-28	AIO14 (I/O)	ADCINB6 (I), COMP3B (1)
31-30	ADCINB7 (I)	ADCINB7 (I)

## 1.4.6 GPIO Registers

### 1.4.6.1 GPIO Port A Qualification Control (GPACTRL) Register

The GPACTRL registers specify the sampling period for input pins when configured for input qualification using a window of 3 or 6 samples. The sampling period is the amount of time between qualification samples relative to the period of SYSCLKOUT. The number of samples is specified in the GPAQSELn registers.

**Figure 1-58. GPIO Port A Qualification Control (GPACTRL) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-62. GPIO Port A Qualification Control (GPACTRL) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-24	QUALPRD3	0x00 0x01 0x02 ... 0xFF	Specifies the sampling period for pins GPIO24 to GPIO31. Sampling Period = $T_{SYSCLKOUT}$ <sup>(2)</sup> Sampling Period = $2 \times T_{SYSCLKOUT}$ Sampling Period = $4 \times T_{SYSCLKOUT}$ ... Sampling Period = $510 \times T_{SYSCLKOUT}$
23-16	QUALPRD2	0x00 0x01 0x02 ... 0xFF	Specifies the sampling period for pins GPIO16 to GPIO23. Sampling Period = $T_{SYSCLKOUT}$ <sup>(2)</sup> Sampling Period = $2 \times T_{SYSCLKOUT}$ Sampling Period = $4 \times T_{SYSCLKOUT}$ ... Sampling Period = $510 \times T_{SYSCLKOUT}$
15-8	QUALPRD1	0x00 0x01 0x02 ... 0xFF	Specifies the sampling period for pins GPIO8 to GPIO15. Sampling Period = $T_{SYSCLKOUT}$ <sup>(2)</sup> Sampling Period = $2 \times T_{SYSCLKOUT}$ Sampling Period = $4 \times T_{SYSCLKOUT}$ ... Sampling Period = $510 \times T_{SYSCLKOUT}$
7-0	QUALPRD0	0x00 0x01 0x02 ... 0xFF	Specifies the sampling period for pins GPIO0 to GPIO7. Sampling Period = $T_{SYSCLKOUT}$ <sup>(2)</sup> Sampling Period = $2 \times T_{SYSCLKOUT}$ Sampling Period = $4 \times T_{SYSCLKOUT}$ ... Sampling Period = $510 \times T_{SYSCLKOUT}$

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2)  $T_{SYSCLKOUT}$  indicates the period of SYSCLKOUT.

### 1.4.6.2 GPIO Port A Qualification Select 1 (GPAQSEL1) Register

**Figure 1-59. GPIO Port A Qualification Select 1 (GPAQSEL1) Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GPIO15		GPIO14		GPIO13		GPIO12		GPIO11		GPIO10		GPIO9		GPIO8	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO7		GPIO6		GPIO5		GPIO4		GPIO3		GPIO2		GPIO1		GPIO0	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-63. GPIO Port A Qualification Select 1 (GPAQSEL1) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-0	GPIO15-GPIO0		Select input qualification type for GPIO0 to GPIO15. The input qualification of each GPIO input is controlled by two bits as shown in <a href="#">Figure 1-59</a> .
		00	Synchronize to SYSCLKOUT only. Valid for both peripheral and GPIO pins.
		01	Qualification using 3 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		10	Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		11	Asynchronous. (no synchronization or qualification). This option applies to pins configured as peripherals only. If the pin is configured as a GPIO input, then this option is the same as 0,0 or synchronize to SYSCLKOUT.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.



### 1.4.6.3 GPIO Port A Qualification Select 2 (GPAQSEL2) Register

**Figure 1-60. GPIO Port A Qualification Select 2 (GPAQSEL2) Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GPIO31		GPIO30		GPIO29		GPIO28		GPIO27		GPIO26		GPIO25		GPIO24	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO23		GPIO22		GPIO21		GPIO20		GPIO19		GPIO18		GPIO17		GPIO16	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-64. GPIO Port A Qualification Select 2 (GPAQSEL2) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-0	GPIO31-GPIO16	00	Select input qualification type for GPIO16 to GPIO31. The input qualification of each GPIO input is controlled by two bits as shown in <a href="#">Figure 1-60</a> .
		01	Synchronize to SYSCLKOUT only. Valid for both peripheral and GPIO pins.
		10	Qualification using 3 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		11	Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		11	Asynchronous. (no synchronization or qualification). This option applies to pins configured as peripherals only. If the pin is configured as a GPIO input, then this option is the same as 0,0 or synchronize to SYSCLKOUT.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.4 GPIO Port A MUX 1 (GPAMUX1) Register

**Figure 1-61. GPIO Port A MUX 1 (GPAMUX1) Register**

31					26	25	24	23								16
Reserved						GPIO12		Reserved								
R-0						R/W-0		R-0								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
GPIO7		GPIO6		GPIO5		GPIO4		GPIO3		GPIO2		GPIO1		GPIO0		
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-65. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-26	Reserved		Reserved
25-24	GPIO12	00 01 10 11	Configure the GPIO12 pin as: GPIO12 - General purpose I/O 12 (default) (I/O) TZ1 - Trip zone 1 (I) SCITXDA - SCI-A Transmit (O) Reserved
23-16	Reserved		Reserved
15-14	GPIO7	00 01 10 11	Configure the GPIO7 pin as: GPIO7 - General purpose I/O 7 (default) (I/O) EPWM4B - ePWM4 output B (O) SCIRXDA (I) - SCI-A Receive (I) Reserved
13-12	GPIO6	00 01 10 11	Configure the GPIO6 pin as: GPIO6 - General purpose I/O 6 (default) EPWM4A - ePWM4 output A (O) EPWMSYNCI - ePWM Synch-in (I) EPWMSYNCO - ePWM Synch-out (O)
11-10	GPIO5	00 01 10 11	Configure the GPIO5 pin as: GPIO5 - General purpose I/O 5 (default) (I/O) EPWM3B - ePWM3 output B Reserved ECAP1 - eCAP1 (I/O)
9-8	GPIO4	00 01 10 11	Configure the GPIO4 pin as: GPIO4 - General purpose I/O 4 (default) (I/O) EPWM3A - ePWM3 output A (O) Reserved. <sup>(2)</sup> Reserved. <sup>(2)</sup>
7-6	GPIO3	00 01 10 11	Configure the GPIO3 pin as: GPIO3 - General purpose I/O 3 (default) (I/O) EPWM2B - ePWM2 output B (O) Reserved COMP2OUT (O)
5-4	GPIO2	00 01 10 11	Configure the GPIO2 pin as: GPIO2 (I/O) General purpose I/O 2 (default) (I/O) EPWM2A - ePWM2 output A (O) Reserved. <sup>(2)</sup> Reserved. <sup>(2)</sup>
3-2	GPIO1	00 01 10 11	Configure the GPIO1 pin as: GPIO1 - General purpose I/O 1 (default) (I/O) EPWM1B - ePWM1 output B (O) Reserved COMP1OUT (O) - Comparator 1 output

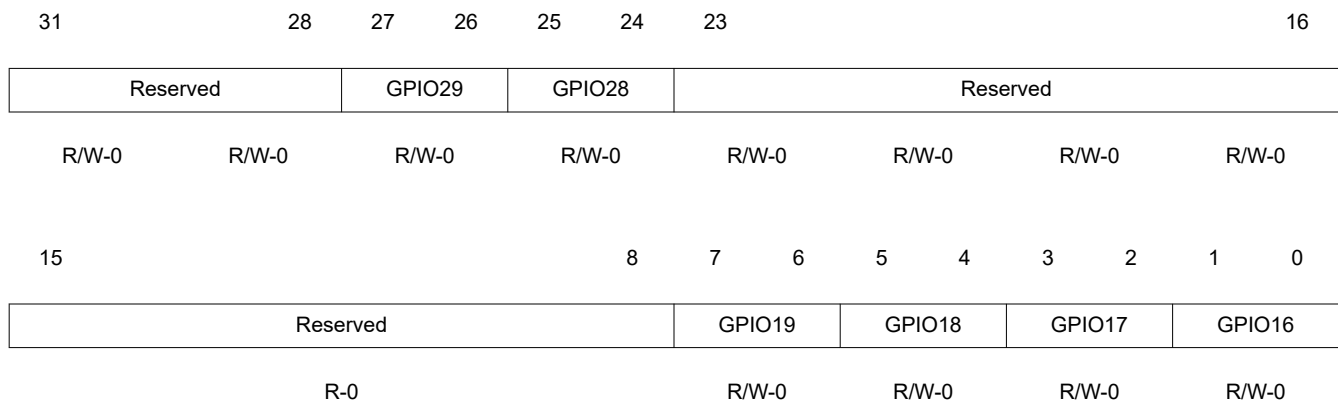
**Table 1-65. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions (continued)**

Bits	Field	Value	Description <sup>(1)</sup>
1-0	GPIO0		Configure the GPIO0 pin as:
		00	GPIO0 - General purpose I/O 0 (default) (I/O)
		01	EPWM1A - ePWM1 output A (O)
		10	Reserved. <sup>(2)</sup>
		11	Reserved. <sup>(2)</sup>

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2) If reserved configurations are selected, then the state of the pin will be undefined and the pin may be driven. These selections are reserved for future expansion and should not be used.

#### 1.4.6.5 GPIO Port A MUX 2 (GPAMUX2) Register

**Figure 1-62. GPIO Port A MUX 2 (GPAMUX2) Register**

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-66. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-28	Reserved		Reserved
27-26	GPIO29		Configure the GPIO29 pin as:
		00	GPIO29 (I/O) General purpose I/O 29 (default) (I/O)
		01	SCITXDA - SCI-A transmit. (O)
		10	SCLA (I/OC)
		11	$\overline{TZ3}$ - Trip zone 3(I)
25-24	GPIO28		Configure the GPIO28 pin as:
		00	GPIO28 (I/O) General purpose I/O 28 (default) (I/O)
		01	SCIRXDA - SCI-A receive (I)
		10	SDAA (I/OC)
		11	$\overline{TZ2}$ - Trip zone 2(I)
23-8	Reserved	11	Reserved
7-6	GPIO19/XCLKIN		Configure the GPIO19 pin as:
		00	GPIO19 - General purpose I/O 19 (default) (I/O)
		01	$\overline{SPISTEA}$ - SPI-A slave transmit enable (I/O)
		10	SCIRXDA (I)
		11	ECAP1 (I/O)

**Table 1-66. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions (continued)**

Bits	Field	Value	Description <sup>(1)</sup>
5-4	GPIO18		Configure the GPIO18 pin as: 00 GPIO18 - General purpose I/O 18 (default) (I/O) 01 SPICLKA - SPI-A clock (I/O) 10 SCITXDA (O) 11 XCLKOUT (O) - External clock output
3-2	GPIO17		Configure the GPIO17 pin as: 00 GPIO17 - General purpose I/O 17 (default) (I/O) 01 SPISOMIA - SPI-A slave-out, master-in (I/O) 10 Reserved 11 $\overline{TZ3}$ - Trip zone 3 (I)
1-0	GPIO16		Configure the GPIO16 pin as: 00 GPIO16 - General purpose I/O 16 (default) (I/O) 01 SPISIMOA - SPI-A slave-in, master-out (I/O), 10 Reserved 11 $\overline{TZ2}$ - Trip zone 2 (I)

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.6 GPIO Port A Direction (GPADIR) Register

The GPADIR register controls the direction of the pins when they are configured as a GPIO in the appropriate MUX register. The direction register has no effect on pins configured as peripheral functions.

**Figure 1-63. GPIO Port A Direction (GPADIR) Register**

31	30	29	28	27	26	25	24
GPIO31	GPIO30	GPIO29	GPIO28	GPIO27	GPIO26	GPIO25	GPIO24
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
23	22	21	20	19	18	17	16
GPIO23	GPIO22	GPIO21	GPIO20	GPIO19	GPIO18	GPIO17	GPIO16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8
GPIO15	GPIO14	GPIO13	GPIO12	GPIO11	GPIO10	GPIO9	GPIO8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-67. GPIO Port A Direction (GPADIR) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-0	GPIO31-GPIO0	0 1	Controls direction of GPIO Port A pins when the specified pin is configured as a GPIO in the appropriate GPAMUX1 or GPAMUX2 register. Configures the GPIO pin as an input. (default) Configures the GPIO pin as an output. The value currently in the GPADAT output latch is driven on the pin. To initialize the GPADAT latch prior to changing the pin from an input to an output, use the GPASET, GPACLEAR, and GPATOGGLE registers.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.7 GPIO Port A Pullup Disable (GPAPUD) Register

The pullup disable (GPAPUD) register allows you to specify which pins should have an internal pullup resistor enabled. The internal pullups on the pins that can be configured as ePWM outputs (GPIO0-GPIO11) are all disabled asynchronously when the external reset signal ( $\overline{XRS}$ ) is low. The internal pullups on all other pins are enabled on reset. When coming out of reset, the pullups remain in their default state until you enable or disable them selectively in software by writing to this register. The pullup configuration applies both to pins configured as I/O and those configured as peripheral functions.

**Figure 1-64. GPIO Port A Pullup Disable (GPAPUD) Register**

31	30	29	28	27	26	25	24
GPIO31	GPIO30	GPIO29	GPIO28	GPIO27	GPIO26	GPIO25	GPIO24
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
23	22	21	20	19	18	17	16
GPIO23	GPIO22	GPIO21	GPIO20	GPIO19	GPIO18	GPIO17	GPIO16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8
GPIO15	GPIO14	GPIO13	GPIO12	GPIO11	GPIO10	GPIO9	GPIO8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1
7	6	5	4	3	2	1	0
GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

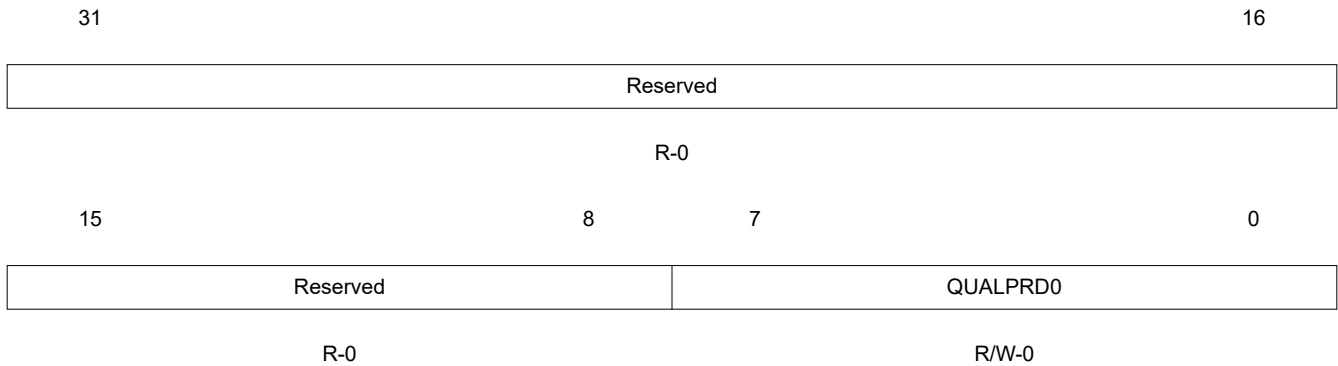
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-68. GPIO Port A Internal Pullup Disable (GPAPUD) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-0	GPIO31-GPIO0		Configure the internal pullup resistor on the selected GPIO Port A pin. Each GPIO pin corresponds to one bit in this register.
		0	Enable the internal pullup on the specified pin. (default for GPIO12-GPIO31)
		1	Disable the internal pullup on the specified pin. (default for GPIO0-GPIO11)

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.8 GPIO Port B Qualification Control (GPBCTRL) Register

**Figure 1-65. GPIO Port B Qualification Control (GPBCTRL) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-69. GPIO Port B Qualification Control (GPBCTRL) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31- 8	Reserved		Reserved
7-0	QUALPRD0		Specifies the sampling period for pins GPIO32 to GPIO38
		0xFF	Sampling Period = $510 \times T_{\text{SYSCLKOUT}}$
		0x00	Sampling Period = $T_{\text{SYSCLKOUT}}$ <sup>(2)</sup>
		0x01	Sampling Period = $2 \times T_{\text{SYSCLKOUT}}$
		0x02	Sampling Period = $4 \times T_{\text{SYSCLKOUT}}$
		...	...
		0xFF	Sampling Period = $510 \times T_{\text{SYSCLKOUT}}$

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

(2)  $T_{\text{SYSCLKOUT}}$  indicates the period of SYSCLKOUT.

### 1.4.6.9 GPIO Port B Qualification Select 1 (GPBQSEL1) Register

**Figure 1-66. GPIO Port B Qualification Select 1 (GPBQSEL1) Register**

31

16

Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		GPIO38		GPIO37		GPIO36		GPIO35		GPIO34		GPIO33		GPIO32	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-70. GPIO Port B Qualification Select 1 (GPBQSEL1) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-14	Reserved		
13-0	GPIO38 -GPIO32	00 01 10 11	Select input qualification type for GPIO32 to GPIO38. The input qualification of each GPIO input is controlled by two bits as shown in <a href="#">Figure 1-66</a> .  Synchronize to SYSCLKOUT only. Valid for both peripheral and GPIO pins.  Qualification using 3 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.  Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.  Asynchronous. (no synchronization or qualification). This option applies to pins configured as peripherals only. If the pin is configured as a GPIO input, then this option is the same as 0,0 or synchronize to SYSCLKOUT.

 (1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.10 GPIO Port B MUX 1 (GPBMUX1) Register

**Figure 1-67. GPIO Port B MUX 1 (GPBMUX1) Register**

31

16

Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		GPIO38		GPIO37		GPIO36		GPIO35		GPIO34		GPIO33		GPIO32	
R-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset



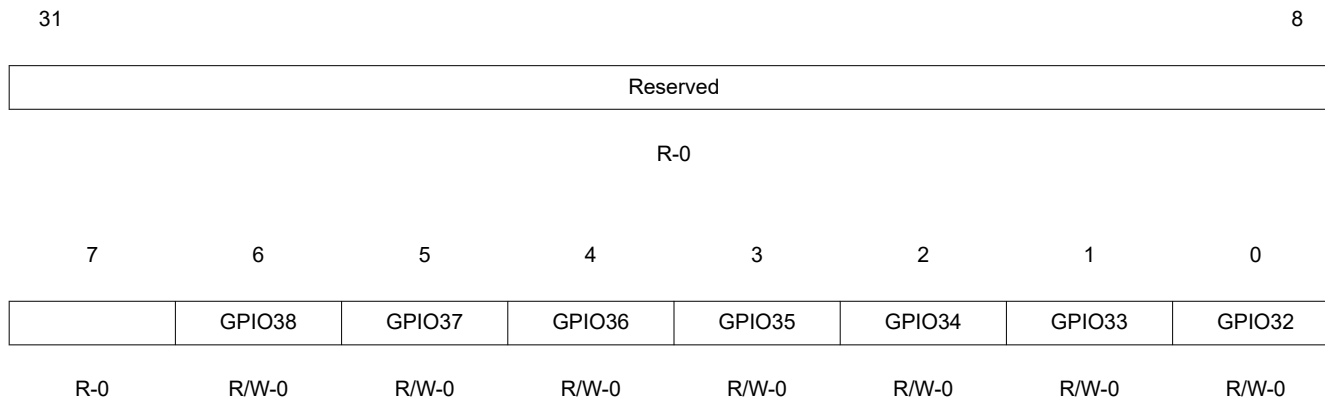
**Table 1-71. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions**

Bit	Field	Value	Description
31-14	Reserved		Reserved
13-12	GPIO38/ XCLKIN/TCK	00 01 10 or 11	Configure this pin as: GPIO 38 - general purpose I/O 38 (default) (I/O). If $\overline{\text{TRST}} = 1$ , JTAG TCK function is chosen for this pin. This pin can also be used to provide a clock from an external oscillator to the core. Reserved Reserved
11-10	GPIO37/TDO	00 01 10 or 11	Configure this pin as: GPIO 37 - general purpose I/O 37 (default). If $\overline{\text{TRST}} = 1$ , JTAG TDO function is chosen for this pin. Reserved Reserved
9-8	GPIO36/TMS	00 01 10 or 11	Configure this pin as: GPIO 36 - general purpose I/O 36 (default). If $\overline{\text{TRST}} = 1$ , JTAG TMS function is chosen for this pin. Reserved Reserved
7-6	GPIO35/TDI	00 01 10 or 11	Configure this pin as: GPIO 35 - general purpose I/O 35 (default). If $\overline{\text{TRST}} = 1$ , JTAG TDI function is chosen for this pin. Reserved Reserved
5-4	GPIO34	00 01 10 11	Configure this pin as: GPIO 34 - general purpose I/O 34 (default) COMP2OUT (O) Reserved Reserved
3-2	GPIO33	00 01 10 11	Configure this pin as: GPIO 33 - general purpose I/O 33 (default) SCLA - I <sup>2</sup> C clock open drain bidirectional port (I/O) EPWMSYNCO - External ePWM sync pulse output (O) $\overline{\text{ADCSOCBO}}$ - ADC start-of-conversion B (O)
1-0	GPIO32	00 01 10 11	Configure this pin as: GPIO 32 - general purpose I/O 32 (default) SDAA - I <sup>2</sup> C data open drain bidirectional port (I/O) EPWMSYNCI - External ePWM sync pulse input (I) $\overline{\text{ADCSOCAO}}$ - ADC start-of-conversion A (O)

### 1.4.6.11 GPIO Port B Direction (GPBDIR) Register

The GPBDIR register controls the direction of the pins when they are configured as a GPIO in the appropriate MUX register. The direction register has no effect on pins configured as peripheral functions.

**Figure 1-68. GPIO Port B Direction (GPBDIR) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-72. GPIO Port B Direction (GPBDIR) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-7	Reserved		Reserved
6-0	GPIO38-GPIO32	0	Controls direction of GPIO pin when GPIO mode is selected. Reading the register returns the current value of the register setting Configures the GPIO pin as an input. (default)
		1	Configures the GPIO pin as an output.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.12 GPIO Port B Pullup Disable (GPBPUD) Register

The pullup disable (GPBPUD) register allows you to specify which pins should have an internal pullup resistor enabled. The internal pullups on all pins are enabled on reset. When coming out of reset, the pullups remain in their default state until you enable or disable them selectively in software by writing to this register. The pullup configuration applies both to pins configured as I/O and those configured as peripheral functions.

**Figure 1-69. GPIO Port B Pullup Disable (GPBPUD) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-73. GPIO Port B Internal Pullup Disable (GPBPUD) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-7	Reserved		Reserved
6-0	GPIO38 - GPIO32	0	Configure the internal pullup resistor on the selected GPIO Port B pin. Each GPIO pin corresponds to one bit in this register. Enable the internal pullup on the specified pin. (default)
		1	Disable the internal pullup on the specified pin.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.13 Analog I/O MUX (AIOMUX1) Register

**Figure 1-70. Analog I/O MUX (AIOMUX1) Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	16
Reserved		AIO14		Reserved		AIO12		Reserved		AIO10		Reserved	
R-0		R/W-1,x		R-0		R/W-1,x		R-0		R/W-1,x		R-0	
15	14	13	12	11	10	9	8	7	6	5	4	3	0
Reserved		AIO6		Reserved		AIO4		Reserved		AIO2		Reserved	
R-0		R/W-1,x		R-0		R/W-1,x		R-0		R/W-1,x		R-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-74. Analog I/O MUX (AIOMUX1) Register Field Descriptions**

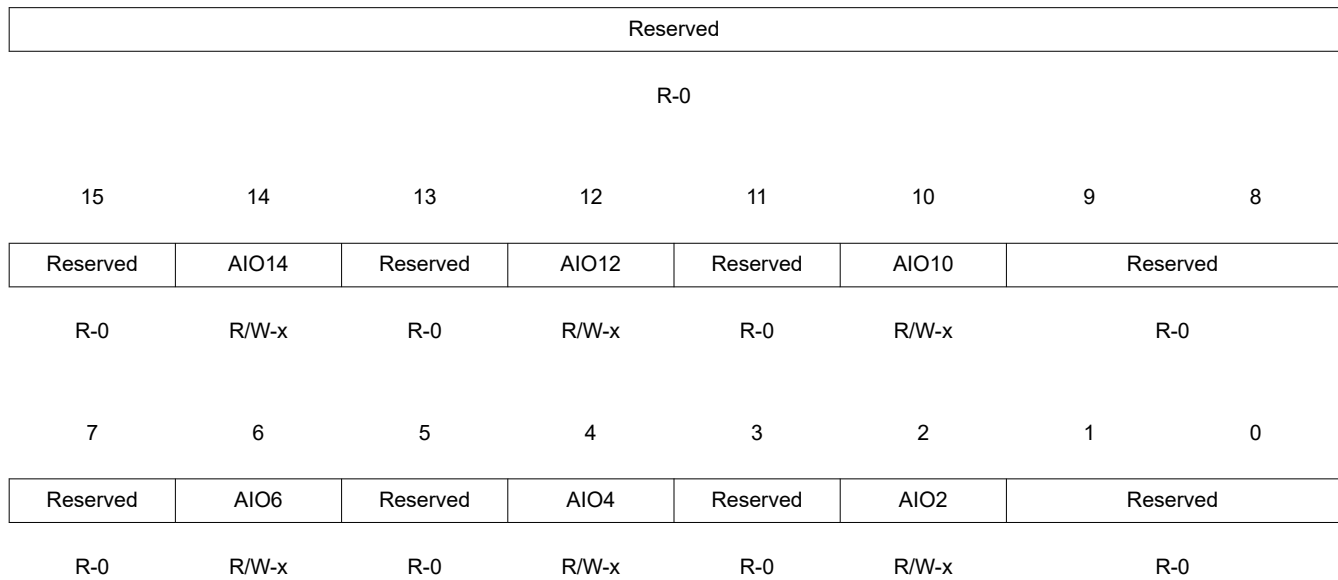
Bit	Field	Value	Description
31-30	Reserved		Any writes to these bits must always have a value of 0.
29-28	AIO14	00 or 01 10 or 11	AIO14 enabled AIO14 disabled (default)
27-26	Reserved		Any writes to these bits must always have a value of 0.
25-24	AIO12	00 or 01 10 or 11	AIO12 enabled AIO12 disabled (default)
23-22	Reserved		Any writes to these bits must always have a value of 0.
21-20	AIO10	00 or 01 10 or 11	AIO10 enabled AIO10 disabled (default)
19-14	Reserved		Any writes to these bits must always have a value of 0.
13-12	AIO6	00 or 01 10 or 11	AIO6 enabled AIO6 disabled (default)
11-10	Reserved		Any writes to these bits must always have a value of 0.
9-8	AIO4	00 or 01 10 or 11	AIO4 enabled AIO4 disabled (default)
7-6	Reserved		Any writes to these bits must always have a value of 0.
5-4	AIO2	00 or 01 10 or 11	AIO2 enabled AIO2 disabled (default)
3-0	Reserved		Any writes to these bits must always have a value of 0.

### 1.4.6.14 Analog I/O Direction (AIODIR) Register

**Figure 1-71. Analog I/O Direction (AIODIR) Register**

31

16



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-75. Analog I/O Direction (AIODIR) Register Field Descriptions**

Bit	Field	Value	Description
31-15	Reserved		
14-0	AIO <sub>n</sub>	0	Controls direction of the available AIO pin when AIO mode is selected. Reading the register returns the current value of the register setting
		1	Configures the AIO pin as an input. (default)
			Configures the AIO pin as an output.

### 1.4.6.15 GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register

**Figure 1-72. GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-76. GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
15-5	Reserved		Reserved
4-0	GPIOXINTnSEL		Select the port A GPIO signal (GPIO0-GPIO31) that is used as the XINT1, XINT2, or XINT3 interrupt source. In addition, you can configure the interrupt in the XINT1CR, XINT2CR, or XINT3CR registers described in <a href="#">Section 1.6.5</a> . To use XINT2 as ADC start of conversion, enable it in the desired ADCSOCxCTL register. The ADCSOC signal is always rising edge sensitive.
		00000	Select the GPIO0 pin as the XINTn interrupt source. (default)
		00001	Select the GPIO1 pin as the XINTn interrupt source.
		...	...
		11110	Select the GPIO30 pin as the XINTn interrupt source.
		11111	Select the GPIO31 pin as the XINTn interrupt source.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

**Table 1-77. XINT1/XINT2/XINT3 Interrupt Select and Configuration Registers**

n	Interrupt	Interrupt Select Register	Configuration Register
1	XINT1	GPIOXINT1SEL	XINT1CR
2	XINT2	GPIOXINT2SEL	XINT2CR
3	XINT3	GPIOXINT3SEL	XINT3CR

### 1.4.6.16 GPIO Low Power Mode Wakeup Select (GPIOLPMSEL) Register

**Figure 1-73. GPIO Low Power Mode Wakeup Select (GPIOLPMSEL) Register**

31	30	29	28	27	26	25	24
GPIO31	GPIO30	GPIO29	GPIO28	GPIO27	GPIO26	GPIO25	GPIO24
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
23	22	21	20	19	18	17	16
GPIO23	GPIO22	GPIO21	GPIO20	GPIO19	GPIO18	GPIO17	GPIO16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8
GPIO15	GPIO14	GPIO13	GPIO12	GPIO11	GPIO10	GPIO9	GPIO8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-78. GPIO Low Power Mode Wakeup Select (GPIOLPMSEL) Register Field Descriptions**

Bits	Field	Value	Description <sup>(1)</sup>
31-0	GPIO31-GPIO0	0	Low Power Mode Wakeup Selection. Each bit in this register corresponds to one GPIO port A pin (GPIO0 - GPIO31) as shown in <a href="#">Figure 1-73</a> . If the bit is cleared, the signal on the corresponding pin has no effect on the HALT and STANDBY low power modes.
		1	If the respective bit is set to 1, the signal on the corresponding pin is able to wake the device from both HALT and STANDBY low power modes.

(1) This register is EALLOW protected. See [Section 1.5.2](#) for more information.

### 1.4.6.17 GPIO Port A Data (GPADAT) Register

The GPIO data registers indicate the current status of the GPIO pin, irrespective of which mode the pin is in. Writing to this register will set the respective GPIO pin high or low if the pin is enabled as a GPIO output, otherwise the value written is latched but ignored. The state of the output register latch will remain in its current state until the next write operation. A reset will clear all bits and latched values to zero. The value read from the GPxDAT registers reflect the state of the pin (after qualification), not the state of the output latch of the GPxDAT register.

Typically the DAT registers are used for reading the current state of the pins. To easily modify the output level of the pin, refer to the SET, CLEAR, and TOGGLE registers.

**Figure 1-74. GPIO Port A Data (GPADAT) Register**

31	30	29	28	27	26	25	24
GPIO31	GPIO30	GPIO29	GPIO28	GPIO27	GPIO26	GPIO25	GPIO24
R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
23	22	21	20	19	18	17	16
GPIO23	GPIO22	GPIO21	GPIO20	GPIO19	GPIO18	GPIO17	GPIO16
R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
15	14	13	12	11	10	9	8
GPIO15	GPIO14	GPIO13	GPIO12	GPIO11	GPIO10	GPIO9	GPIO8
R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
7	6	5	4	3	2	1	0
GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset; x = state of the GPADAT register is unknown after reset (depends on the level of the pin after reset).

**Table 1-79. GPIO Port A Data (GPADAT) Register Field Descriptions**

Bits	Field	Value	Description
31-0	GPIO31-GPIO0	0	Each bit corresponds to one GPIO port A pin (GPIO0-GPIO31) as shown in <a href="#">Figure 1-74</a> . Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured. Writing a 0 forces an output of 0, if the pin is configured as a GPIO output in the appropriate GPAMUX1/2 and GPADIR registers; otherwise, the value is latched but not used to drive the pin.
		1	Reading a 1 indicates that the state of the pin is currently high irrespective of the mode the pin is configured. Writing a 1 forces an output of 1, if the pin is configured as a GPIO output in the appropriate GPAMUX1/2 and GPADIR registers; otherwise, the value is latched but not used to drive the pin.



### 1.4.6.18 GPIO Port A Set, Clear, and Toggle (GPASET, GPACLEAR, GPATOGGLE) Registers

**Figure 1-75. GPIO Port A Set, Clear, and Toggle (GPASET, GPACLEAR, GPATOGGLE) Registers**

31	30	29	28	27	26	25	24
GPIO31	GPIO30	GPIO29	GPIO28	GPIO27	GPIO26	GPIO25	GPIO24
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
23	22	21	20	19	18	17	16
GPIO23	GPIO22	GPIO21	GPIO20	GPIO19	GPIO18	GPIO17	GPIO16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8
GPIO15	GPIO14	GPIO13	GPIO12	GPIO11	GPIO10	GPIO9	GPIO8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-80. GPIO Port A Set (GPASET) Register Field Descriptions**

Bits	Field	Value	Description
31-0	GPIO31-GPIO0	0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to high. If the pin is configured as a GPIO output, then it will be driven high. If the pin is not configured as a GPIO output, then the latch is set high but the pin is not driven.

**Table 1-81. GPIO Port A Clear (GPACLEAR) Register Field Descriptions**

Bits	Field	Value	Description
31-0	GPIO31 - GPIO0	0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to low. If the pin is configured as a GPIO output, then it will be driven low. If the pin is not configured as a GPIO output, then the latch is cleared but the pin is not driven.

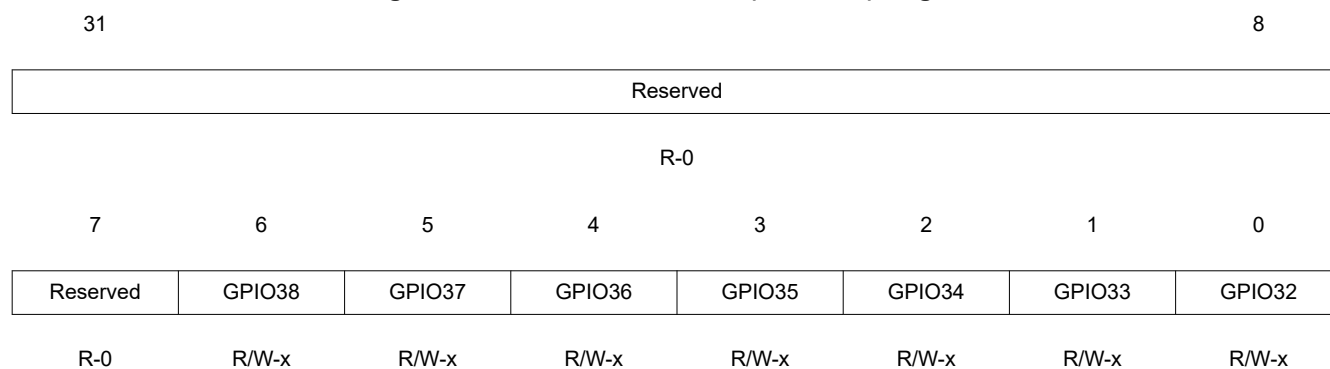
**Table 1-82. GPIO Port A Toggle (GPATOGGLE) Register Field Descriptions**

Bits	Field	Value	Description
31-0	GPIO31-GPIO0	0	Each GPIO port A pin (GPIO0-GPIO31) corresponds to one bit in this register as shown in <a href="#">Figure 1-75</a> . Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to toggle from its current state. If the pin is configured as a GPIO output, then it will be driven in the opposite direction of its current state. If the pin is not configured as a GPIO output, then the latch is toggled but the pin is not driven.

#### 1.4.6.19 GPIO Port B Data (GPBDAT) Register

The GPIO data registers indicate the current status of the GPIO pin, irrespective of which mode the pin is in. Writing to this register will set the respective GPIO pin high or low if the pin is enabled as a GPIO output, otherwise the value written is latched but ignored. The state of the output register latch will remain in its current state until the next write operation. A reset will clear all bits and latched values to zero. The value read from the GPxDAT registers reflect the state of the pin (after qualification), not the state of the output latch of the GPxDAT register.

Typically the DAT registers are used for reading the current state of the pins. To easily modify the output level of the pin, refer to the SET, CLEAR, and TOGGLE registers.

**Figure 1-76. GPIO Port B Data (GPBDAT) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset; x = state of the GPADAT register is unknown after reset (depends on the level of the pin after reset).

**Table 1-83. GPIO Port B Data (GPBDAT) Register Field Descriptions**

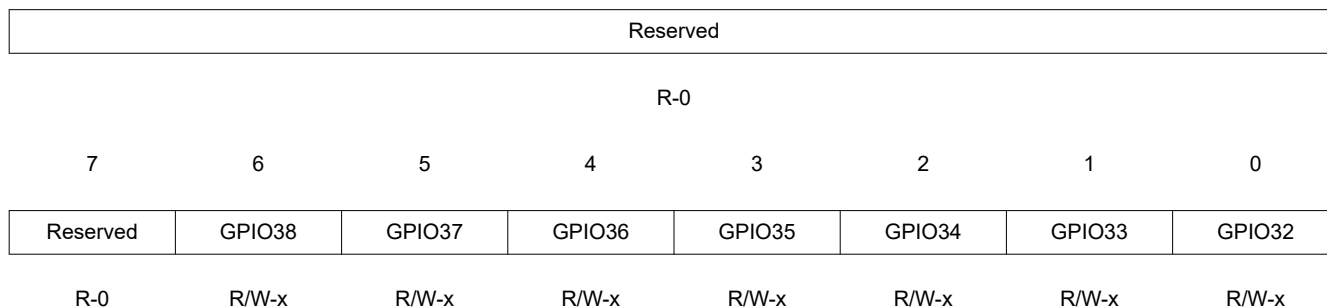
Bit	Field	Value	Description
31-7	Reserved		Reserved
6-0	GPIO38 -GPIO32	0	Each bit corresponds to one GPIO port B pin (GPIO32-GPIO38 ) as shown in <a href="#">Figure 1-76</a> . Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured. Writing a 0 forces an output of 0, if the pin is configured as a GPIO output in the appropriate GPBMUX1 and GPBDIR registers; otherwise, the value is latched but not used to drive the pin.
		1	Reading a 1 indicates that the state of the pin is currently high irrespective of the mode the pin is configured. Writing a 1 forces an output of 1, if the pin is configured as a GPIO output in the GPBMUX1 and GPBDIR registers; otherwise, the value is latched but not used to drive the pin.

### 1.4.6.20 GPIO Port B Set, Clear, and Toggle (GPBSET, GPBCLEAR, GPBTOGGLE) Registers

**Figure 1-77. GPIO Port B Set, Clear, and Toggle (GPBSET, GPBCLEAR, GPBTOGGLE) Registers**

31

8



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-84. GPIO Port B Set (GPBSET) Register Field Descriptions**

Bits	Field	Value	Description
31-7	Reserved		Reserved
6-0	GPIO38 -GPIO32	0 1	Each GPIO port B pin (GPIO32-GPIO38 ) corresponds to one bit in this register as shown in <a href="#">Figure 1-77</a> . Writes of 0 are ignored. This register always reads back a 0. Writing a 1 forces the respective output data latch to high. If the pin is configured as a GPIO output, then it will be driven high. If the pin is not configured as a GPIO output, then the latch is set but the pin is not driven.

**Table 1-85. GPIO Port B Clear (GPBCLEAR) Register Field Descriptions**

Bits	Field	Value	Description
31-7	Reserved		Reserved
6-0	GPIO38 -GPIO32	0 1	Each GPIO port B pin (GPIO32-GPIO38 ) corresponds to one bit in this register as shown in <a href="#">Figure 1-77</a> . Writes of 0 are ignored. This register always reads back a 0. Writing a 1 forces the respective output data latch to low. If the pin is configured as a GPIO output, then it will be driven low. If the pin is not configured as a GPIO output, then the latch is cleared but the pin is not driven.

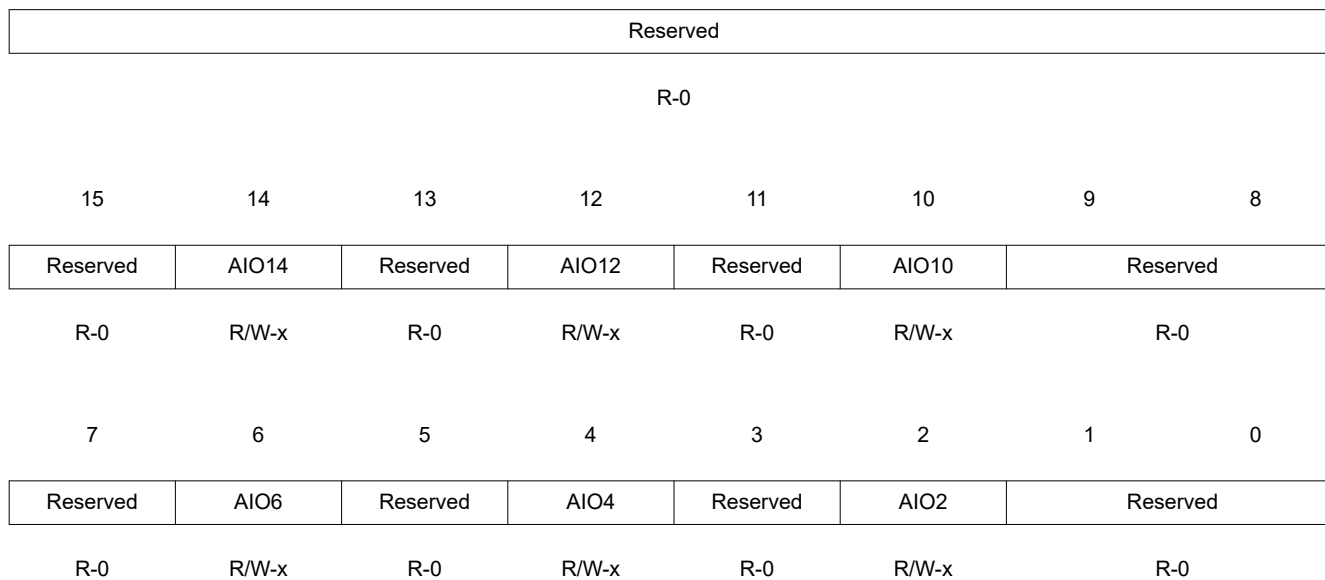
**Table 1-86. GPIO Port B Toggle (GPBTOGGLE) Register Field Descriptions**

Bits	Field	Value	Description
31-7	Reserved		Reserved
6-0	GPIO38 -GPIO32	0 1	Each GPIO port B pin (GPIO32-GPIO38 ) corresponds to one bit in this register as shown in <a href="#">Figure 1-77</a> . Writes of 0 are ignored. This register always reads back a 0. Writing a 1 forces the respective output data latch to toggle from its current state. If the pin is configured as a GPIO output, then it will be driven in the opposite direction of its current state. If the pin is not configured as a GPIO output, then the latch is cleared but the pin is not driven.

**1.4.6.21 Analog I/O Data (AIODAT) Register**
**Figure 1-78. Analog I/O Data (AIODAT) Register**

31

16



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-87. Analog I/O Data (AIODAT) Register Field Descriptions**

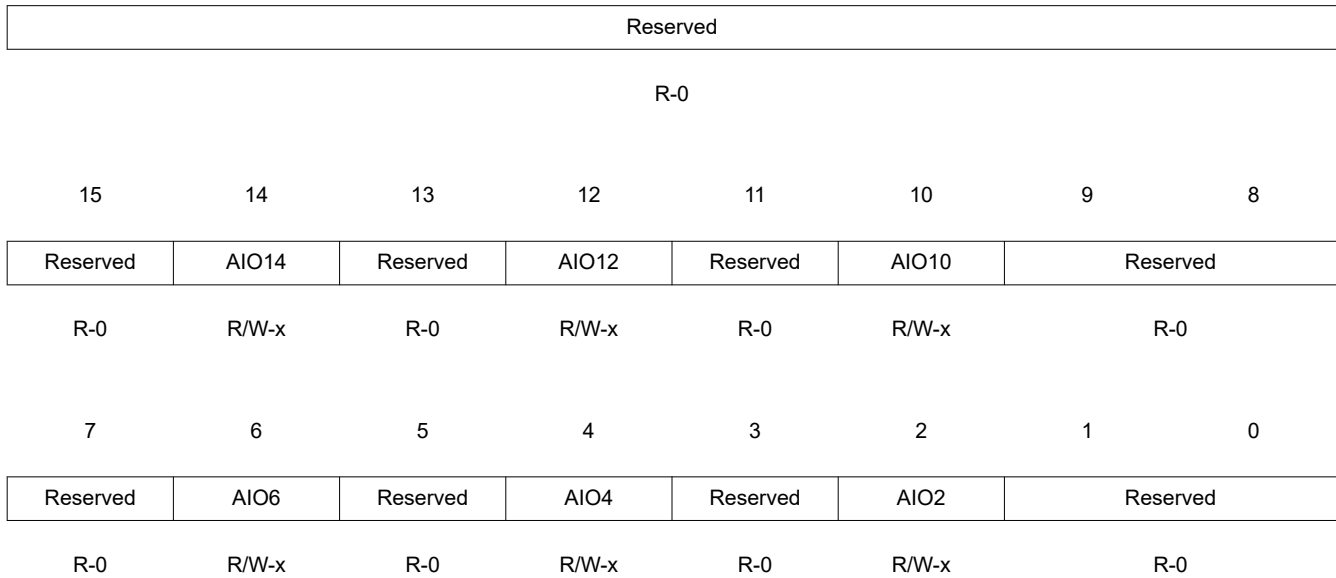
Bit	Field	Value	Description
31-15	Reserved		Reserved
14-0	AIO <sub>n</sub>	0	Each bit corresponds to one AIO port pin. Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured. Writing a 0 forces an output of 0, if the pin is configured as a AIO output in the appropriate registers; otherwise, the value is latched but not used to drive the pin.
		1	Reading a 1 indicates that the state of the pin is currently high irrespective of the mode the pin is configured. Writing a 1 forces an output of 1, if the pin is configured as a AIO output in the appropriate registers; otherwise, the value is latched but not used to drive the pin.

### 1.4.6.22 Analog I/O Set, Clear, and Toggle (AIOSET, AIOCLEAR, AIOTOGGLE) Registers

Figure 1-79. Analog I/O Set, Clear, and Toggle (AIOSET, AIOCLEAR, AIOTOGGLE) Registers

31

16



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-88. Analog I/O Set (AIOSET) Register Field Descriptions**

Bits	Field	Value	Description
31-15	Reserved		Reserved
14-0	AIO <sub>n</sub>	0	Each AIO pin corresponds to one bit in this register. Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to high. If the pin is configured as a AIO output, then it will be driven high. If the pin is not configured as a AIO output, then the latch is set but the pin is not driven.

**Table 1-89. Analog I/O Clear (AIOCLEAR) Register Field Descriptions**

Bits	Field	Value	Description
31-15	Reserved		Reserved
14-0	AIO <sub>n</sub>	0	Each AIO pin corresponds to one bit in this register. Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to low. If the pin is configured as a AIO output, then it will be driven low. If the pin is not configured as a AIO output, then the latch is cleared but the pin is not driven.

**Table 1-90. Analog I/O Toggle (AIOTOGGLE) Register Field Descriptions**

Bits	Field	Value	Description
31-15	Reserved		
14-0	AION	0 1	Each AIO pin corresponds to one bit in this register. Writes of 0 are ignored. This register always reads back a 0. Writing a 1 forces the respective output data latch to toggle from its current state. If the pin is configured as a AIO output, then it will be driven in the opposite direction of its current state. If the pin is not configured as a AIO output, then the latch is cleared but the pin is not driven.

## 1.5 Peripheral Frames

This section describes the peripheral frames. It also describes the device emulation registers.

### 1.5.1 Peripheral Frame Registers

The device contains three peripheral register spaces. The spaces are categorized as:

- Peripheral Frame 0: These are peripherals that are mapped directly to the CPU memory bus. See [Table 1-91](#).
- Peripheral Frame 1: These are peripherals that are mapped to the 32-bit peripheral bus. See [Table 1-92](#).
- Peripheral Frame 2: These are peripherals that are mapped to the 16-bit peripheral bus. See [Table 1-93](#).

**Table 1-91. Peripheral Frame 0 Registers**

Name <sup>(1)</sup>	Address Range	Size (x16)	Access Type <sup>(2)</sup>
Device Emulation Registers	0x00 0880 - 0x00 0984	261	EALLOW protected
System Power Control Registers	0x00 0985 - 0x00 0987	3	EALLOW protected
FLASH Registers <sup>(3)</sup>	0x00 0A80 - 0x00 0ADF	96	EALLOW protected
Code Security Module Registers	0x00 0AE0 - 0x00 0AEF	16	EALLOW protected
ADC registers (dual-mapped) (0 wait, read only, CPU )	0x00 0B00 - 0x00 0B1F	32	Not EALLOW protected
CPU-TIMER0/1/2 Registers	0x00 0C00 - 0x00 0C3F	64	Not EALLOW protected
PIE Registers	0x00 0CE0 - 0x00 0CFF	32	Not EALLOW protected
PIE Vector Table	0x00 0D00 - 0x00 0DFF	256	EALLOW protected

(1) Registers in Frame 0 support 16-bit and 32-bit accesses.

(2) If registers are EALLOW protected, then writes cannot be performed until the EALLOW instruction is executed. The EDIS instruction disables writes to prevent stray code or pointers from corrupting register contents.

(3) The Flash Registers are also protected by the Code Security Module (CSM).

**Table 1-92. Peripheral Frame 1 Registers**

Name <sup>(1)</sup>	Address Range	Size (x16)	Access Type <sup>(2)</sup>
Comparator1 Registers	0x6400 - 0x641F	32	<sup>(3)</sup>
Comparator2 Registers	0x6420 - 0x643F	32	<sup>(3)</sup>
ePWM1 + HRPWM1 Registers	0x6800 - 0x683F	64	<sup>(3)</sup>
ePWM2 + HRPWM2 Registers	0x6840 - 0x687F	64	<sup>(3)</sup>
ePWM3 + HRPWM3 Registers	0x6880 - 0x68BF	64	<sup>(3)</sup>
ePWM4 + HRPWM4 Registers	0x68C0 - 0x68FF	64	<sup>(3)</sup>
eCAP1 Registers	0x6A00 - 0x6A1F	32	Not EALLOW-protected
GPIO Control Registers	0x6F80 - 0x6FBF	128	EALLOW-protected
GPIO Data Registers	0x6FC0 - 0x6FDF	32	Not EALLOW-protected
GPIO Interrupt and LPM Select Registers	0x6FE0 - 0x6FFF	32	EALLOW-protected

(1) Back-to-back write operations to Peripheral Frame 1 registers will incur a 1-cycle stall (1 cycle delay).

(2) Peripheral Frame 1 allows 16-bit and 32-bit accesses. All 32-bit accesses are aligned to even address boundaries.

(3) Some Registers/Bits are EALLOW protected. See the module reference guide for more information.

**Table 1-93. Peripheral Frame 2 Registers**

Name	Address Range	Size (x16)	Access Type <sup>(1)</sup>
System Control Registers	0x7010 - 0x702F	32	EALLOW-protected
SPI-A Registers	0x7040 - 0x704F	16	Not EALLOW protected
SCI-A Registers	0x7050 - 0x705F	16	Not EALLOW protected
NMI Watchdog Interrupt Registers	0x7060 - 0x706F	16	
External Interrupt Registers	0x7070 - 0x707F	16	Not EALLOW protected
ADC Registers	0x7100 - 0x711F	32	Not EALLOW protected
I <sup>2</sup> C Registers	0x7900 - 0x793F	64	Not EALLOW protected

(1) Peripheral Frame 2 only allows 16-bit accesses. All 32-bit accesses are ignored (invalid data can be returned or written).

## 1.5.2 EALLOW-Protected Registers

Several control registers are protected from spurious CPU writes by the EALLOW protection mechanism. The EALLOW bit in status register 1 (ST1) indicates if the state of protection as shown in [Table 1-94](#).

**Table 1-94. Access to EALLOW-Protected Registers**

EALLOW Bit	CPU Writes	CPU Reads	JTAG Writes	JTAG Reads
0	Ignored	Allowed	Allowed <sup>(1)</sup>	Allowed
1	Allowed	Allowed	Allowed	Allowed

(1) The EALLOW bit is overridden via the JTAG port, allowing full access of protected registers during debug from the Code Composer Studio™ IDE.

At reset the EALLOW bit is cleared enabling EALLOW protection. While protected, all writes to protected registers by the CPU are ignored and only CPU reads, JTAG reads, and JTAG writes are allowed. If this bit is set, by executing the EALLOW instruction, then the CPU is allowed to write freely to protected registers. After modifying registers, they can once again be protected by executing the EDI instruction to clear the EALLOW bit.

The following registers are EALLOW-protected:

- Device Emulation Registers ([Table 1-95](#))
- Flash Registers ([Table 1-96](#))
- CSM Registers ([Table 1-97](#))
- System Control Registers ([Table 1-98](#))
- GPIO MUX Registers ([Table 1-99](#))
- PIE Vector Table ([Table 1-100](#))

**Table 1-95. EALLOW-Protected Device Emulation Registers**

Name	Address	Size (x16)	Description
DEVICECNF	0x0880 0x0881	2	Device Configuration Register

**Table 1-96. EALLOW-Protected Flash/OTP Configuration Registers**

Name	Address	Size (x16)	Description
FOPT	0x0A80	1	Flash Option Register
FPWR	0x0A82	1	Flash Power Modes Register
FSTATUS	0x0A83	1	Status Register
FSTDBYWAIT	0x0A84	1	Flash Sleep To Standby Wait State Register
FACTIVEWAIT	0x0A85	1	Flash Standby To Active Wait State Register
FBANKWAIT	0x0A86	1	Flash Read Access Wait State Register
FOTPWAIT	0x0A87	1	OTP Read Access Wait State Register



**Table 1-97. EALLOW-Protected Code Security Module (CSM) Registers**

Name	Address	Size (x16)	Description
KEY0	0x0AE0	1	Low word of the 128-bit KEY register
KEY1	0x0AE1	1	Second word of the 128-bit KEY register
KEY2	0x0AE2	1	Third word of the 128-bit KEY register
KEY3	0x0AE3	1	Fourth word of the 128-bit KEY register
KEY4	0x0AE4	1	Fifth word of the 128-bit KEY register
KEY5	0x0AE5	1	Sixth word of the 128-bit KEY register
KEY6	0x0AE6	1	Seventh word of the 128-bit KEY register
KEY7	0x0AE7	1	High word of the 128-bit KEY register
CSMSCR	0x0AEF	1	CSM status and control register

**Table 1-98. EALLOW-Protected PLL, Clocking, Watchdog, and Low-Power Mode Registers**

Name	Address	Size (x16)	Description
BORCFG	0x0000-0985	1	BOR Configuration Register
XCLK	0x0000-7010	1	XCLKOUT/XCLKIN Control
PLLSTS	0x0000-7011	1	PLL Status Register
CLKCTL	0x0000-7012	1	Clock Control Register
PLLLOCKPRD	0x0000-7013	1	PLL Lock Period Register
INTOSC1TRIM	0x0000-7014	1	Internal Oscillator 1 Trim Register
INTOSC2TRIM	0x0000-7016	1	Internal Oscillator 2 Trim Register
LOSPCP	0x0000-701B	1	Low-Speed Peripheral Clock Pre-Scaler Register
PCLKCR0	0x0000-701C	1	Peripheral Clock Control Register 0
PCLKCR1	0x0000-701D	1	Peripheral Clock Control Register 1
LPMCR0	0x0000-701E	1	Low Power Mode Control Register 0
PCLKCR3	0x0000-7020	1	Peripheral Clock Control Register 3
PLLCR	0x0000-7021	1	PLL Control Register
SCSR	0x0000-7022	1	System Control and Status Register
WDCNTR	0x0000-7023	1	Watchdog Counter Register
WDKEY	0x0000-7025	1	Watchdog Reset Key Register
WDCR	0x0000-7029	1	Watchdog Control Register

**Table 1-99. EALLOW-Protected GPIO Registers**

<b>Name <sup>(1)</sup></b>	<b>Address</b>	<b>Size (x16)</b>	<b>Description</b>
GPACTRL	0x6F80	2	GPIO A Control Register
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register
GPAMUX1	0x6F86	2	GPIO A MUX 1 Register
GPAMUX2	0x6F88	2	GPIO A MUX 2 Register
GPADIR	0x6F8A	2	GPIO A Direction Register
GPAPUD	0x6F8C	2	GPIO A Pull Up Disable Register
GPBCTRL	0x6F90	2	GPIO B Control Register
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register
GPBMUX1	0x6F96	2	GPIO B MUX 1 Register
GPBMUX2	0x6F98	2	GPIO B MUX 2 Register
GPBDIR	0x6F9A	2	GPIO B Direction Register
GPBPUD	0x6F9C	2	GPIO B Pull Up Disable Register
AIOMUX1	0x6FB6	2	Analog, I/O MUX 1 register
AIODIR	0x6FBA	2	Analog, IO Direction Register
GPIOXINT1SEL	0x6FE0	1	XINT1 Source Select Register (GPIO0-GPIO31)
GPIOXINT2SEL	0x6FE1	1	XINT2 Source Select Register (GPIO0-GPIO31)
GPIOXINT3SEL	0x6FE2	1	XINT3 Source Select Register (GPIO0 - GPIO31)
GPIOLPMSEL	0x6FE8	1	LPM wakeup Source Select Register (GPIO0-GPIO31)

(1) The registers in this table are EALLOW protected. See [Section 1.5.2](#) for more information.

**Table 1-100. EALLOW-Protected PIE Vector Table**

Name	Address	Size (x16)	Description
Not used	0x0D00	2	Reserved
	0x0D02		
	0x0D04		
	0x0D06		
	0x0D08		
	0x0D0A		
	0x0D0C		
	0x0D0E		
	0x0D10		
	0x0D12		
	0x0D14		
	0x0D16		
	0x0D18		
INT13	0x0D1A	2	CPU-Timer 1
INT14	0x0D1C	2	CPU-Timer 2
DATALOG	0x0D1E	2	CPU Data Logging Interrupt
RTOSINT	0x0D20	2	CPU Real-Time OS Interrupt
EMUINT	0x0D22	2	CPU Emulation Interrupt
NMI	0x0D24	2	External Non-Maskable Interrupt
ILLEGAL	0x0D26	2	Illegal Operation
USER1	0x0D28	2	User-Defined Trap
.	.	.	.
USER12	0x0D3E	2	User-Defined Trap
INT1.1	0x0D40	2	Group 1 Interrupt Vectors
.	.	.	.
INT1.8	0x0D4E	2	
.	.	.	Group 2 Interrupt Vectors
.	.	.	to Group 11 Interrupt Vectors
.	.	.	.
INT12.1	0x0DF0	2	Group 12 Interrupt Vectors
.	.	.	.
INT12.8	0x0DFE	2	

Table 1-101 shows addresses for the following ePWM EALLOW-protected registers:

- Trip Zone Select Register (TZSEL)
- Trip Zone Control Register (TZCTL)
- Trip Zone Enable Interrupt Register (TZEINT)
- Trip Zone Clear Register (TZCLR)
- Trip Zone Force Register (TZFRC)
- HRPWM Configuration Register (HRCNFG)

**Table 1-101. EALLOW-Protected ePWM1-ePWM4 Registers**

ePWMn	TZSEL	TZCTL	TZEINT	TZCLR	TZFRC	HRCNFG	Size x16
ePWM1	0x6812	0x6814	0x6815	0x6817	0x6818	0x6820	1
ePWM2	0x6852	0x6854	0x6855	0x6857	0x6858	0x6860	1
ePWM3	0x6892	0x6894	0x6895	0x6897	0x6898	0x68A0	1
ePWM4	0x68D2	0x68D4	0x68D5	0x68D7	0x68D8	0x68E0	1

### 1.5.3 Device Emulation Registers

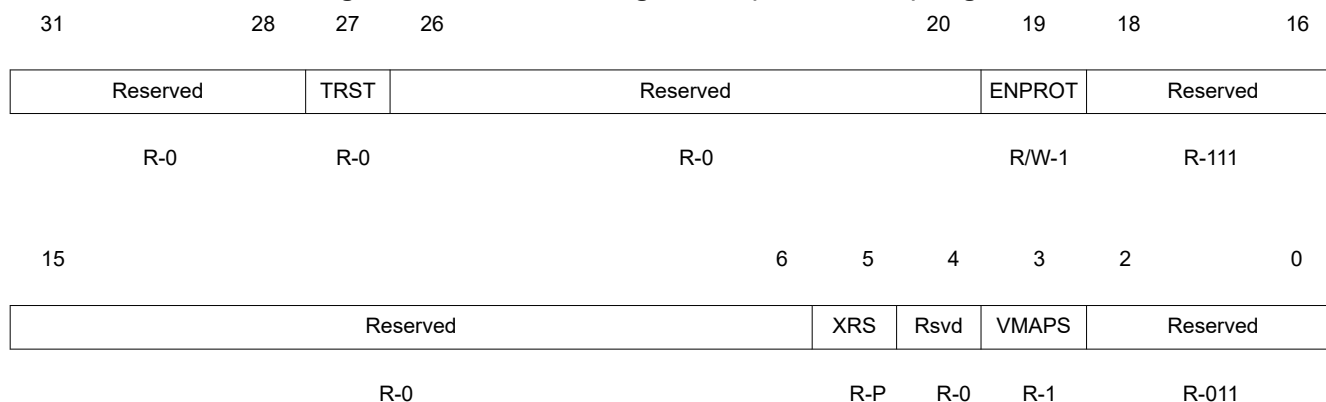
The registers are listed in [Table 1-102](#) are used to control the protection mode of the C28x CPU and to monitor some critical device signals.

**Table 1-102. Device Emulation Registers**

Name	Address	Size (x16)	Description	Bit Description
DEVICECNF	0x0880 0x0881	2	Device Configuration Register	<a href="#">Section 1.5.3.1</a>
PARTID	0x3D7FFF	1	Part ID Register	<a href="#">Section 1.5.3.2</a>
CLASSID	0x0882	1	Class ID Register	<a href="#">Section 1.5.3.3</a>
REVID	0x0883	1	Revision ID Register	<a href="#">Section 1.5.3.4</a>

#### 1.5.3.1 Device Configuration (DEVICECNF) Register

**Figure 1-80. Device Configuration (DEVICECNF) Register**



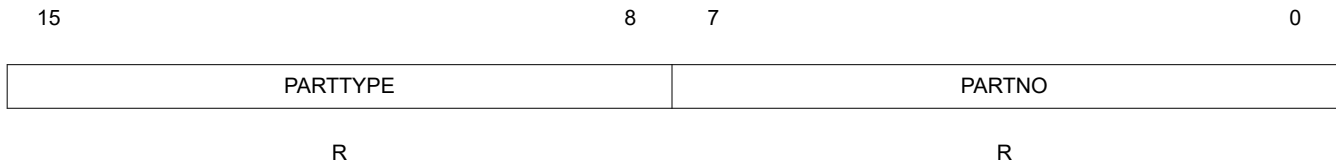
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-103. DEVICECNF Register Field Descriptions**

Bits	Field	Value	Description
31-28	Reserved		Reserved
27	TRST	0 1	Read status of $\overline{\text{TRST}}$ signal. Reading this bit gives the current status of the $\overline{\text{TRST}}$ signal. No JTAG debug probe is connected. JTAG debug probe is connected.
26:20	Reserved		Reserved
19	ENPROT	0 1	Enable Write-Read Protection Mode Bit. Disables write-read protection mode Enables write-read protection for the address range 0x4000-0x7FFF
18-6	Reserved		Reserved
5	XRS		Reset Input Signal Status. This is connected directly to the $\overline{\text{XRS}}$ input pin.
4	Reserved		Reserved
3	VMAPS		VMAP Configure Status. This indicates the status of VMAP.
2-0	Reserved		Reserved

### 1.5.3.2 Part ID (PARTID) Register

**Figure 1-81. Part ID (PARTID) Register**



LEGEND: R = Read only; -n = value after reset

**Table 1-104. Part ID (PARTID) Register Field Descriptions**

Bit	Field	Value <sup>(1)</sup>	Description
15-8	PARTTYPE	0x00	These 8 bits specify the type of device such as Flash-based. Flash-based device All other values are reserved.
7-0	PARTNO	0x04	These 8 bits specify the feature set of the device as follows: All other values are reserved. TMS320F280220DA
		0x05	TMS320F280220PT
		0x06	TMS320F280260DA
		0x07	TMS320F280260PT
		0x0C	TMS320F280230DA
		0x0D	TMS320F280230PT
		0x0E	TMS320F280270DA
		0x0F	TMS320F280270PT
		0xC0	TMS320F280200DA
		0xC1	TMS320F280200PT
		0xC2	TMS320F28020DA
		0xC3	TMS320F28020PT
		0xC4	TMS320F28022DA
		0xC5	TMS320F28022PT
		0xC6	TMS320F28026DA
		0xC7	TMS320F28026PT
		0xCA	TMS320F28021DA
		0xCB	TMS320F28021PT
		0xCC	TMS320F28023DA
		0xCD	TMS320F28023PT
		0xCE	TMS320F28027DA
		00CF	TMS320F28027PT

(1) The reset value depends on the device as indicated in the register description.

### 1.5.3.3 Class ID (CLASSID) Register

**Figure 1-82. Class ID (CLASSID) Register**

15

0



R

LEGEND: R = Read only; -n = value after reset

**Table 1-105. Class ID (CLASSID) Register Field Descriptions**

Bit	Field	Value <sup>(1)</sup>	Description
15-0	CLASSID	0xC7 0xCF	These 16 bits specify the feature set of the device as follows: All other values are reserved. TMS320F28020/022/026, TMS320F280200/220/230/260/270 TMS320F28021/023/027

(1) The reset value depends on the device as indicated in the register description.

### 1.5.3.4 Revision ID (REVID) Register

**Figure 1-83. Revision ID (REVID) Register**

15

0



R

LEGEND: R = Read only; -n = value after reset

**Table 1-106. Revision ID (REVID) Register Field Descriptions**

Bits	Field	Value <sup>(1)</sup>	Description
15-0	REVID	0x0000 0x0001 0x0002	These 16 bits specify the silicon revision number for the particular part. This number always starts with 0x0000 on the first revision of the silicon and is incremented on any subsequent revisions. Silicon Revision 0 - This silicon revision is available as TMX and TMS. Silicon Revision A - TMS Silicon Revision B - TMS

(1) The reset value depends on the silicon revision as described in the register field description.

### 1.5.4 Write-Followed-by-Read Protection

The memory address range for which CPU write followed by read operations are protected is 0x4000 - 0x7FFF (operations occur in sequence rather than in their natural pipeline order). This is necessary protection for certain peripheral operations.

**Example:** The following lines of code perform a write to register 1 (REG1) location and the next instruction performs a read from Register 2 (REG2) location. On the processor memory bus, with block protection disabled, the read operation is issued before the write as shown.

```

MOV   @REG1,AL      -----+
TBIT  @REG2,#BIT_X  -----|-----> Read
                                     +-----> Write
If block protection is enabled, then the read is stalled until the write occurs as shown:
MOV   @REG1,AL      -----+
TBIT  @REG2,#BIT_X  -----|-----+
                                     +-----|----> Write
                                     +----> Read
  
```

## 1.6 Peripheral Interrupt Expansion (PIE)

The peripheral interrupt expansion (PIE) block multiplexes numerous interrupt sources into a smaller set of interrupt inputs. The PIE block can support 96 individual interrupts that are grouped into blocks of eight. Each group is fed into one of 12 core interrupt lines (INT1 to INT12). Each of the 96 interrupts is supported by its own vector stored in a dedicated RAM block that you can modify. The CPU, upon servicing the interrupt, automatically fetches the appropriate interrupt vector. It takes nine CPU clock cycles to fetch the vector and save critical CPU registers. Therefore, the CPU can respond quickly to interrupt events. Prioritization of interrupts is controlled in hardware and software. Each individual interrupt can be enabled/disabled within the PIE block.

### 1.6.1 Overview of the PIE Controller

The 28x CPU supports one nonmaskable interrupt (NMI) and 16 maskable prioritized interrupt requests (INT1-INT14, RTOSINT, and DLOGINT) at the CPU level. The 28x devices have many peripherals and each peripheral is capable of generating one or more interrupts in response to many events at the peripheral level. Because the CPU does not have sufficient capacity to handle all peripheral interrupt requests at the CPU level, a centralized peripheral interrupt expansion (PIE) controller is required to arbitrate the interrupt requests from various sources such as peripherals and other external pins.

The PIE vector table is used to store the address (vector) of each interrupt service routine (ISR) within the system. There is one vector per interrupt source including all MUXed and nonMUXed interrupts. You populate the vector table during device initialization and you can update it during operation.

#### 1.6.1.1 Interrupt Operation Sequence

Figure 1-84 shows an overview of the interrupt operation sequence for all multiplexed PIE interrupts. Interrupt sources that are not multiplexed are fed directly to the CPU.

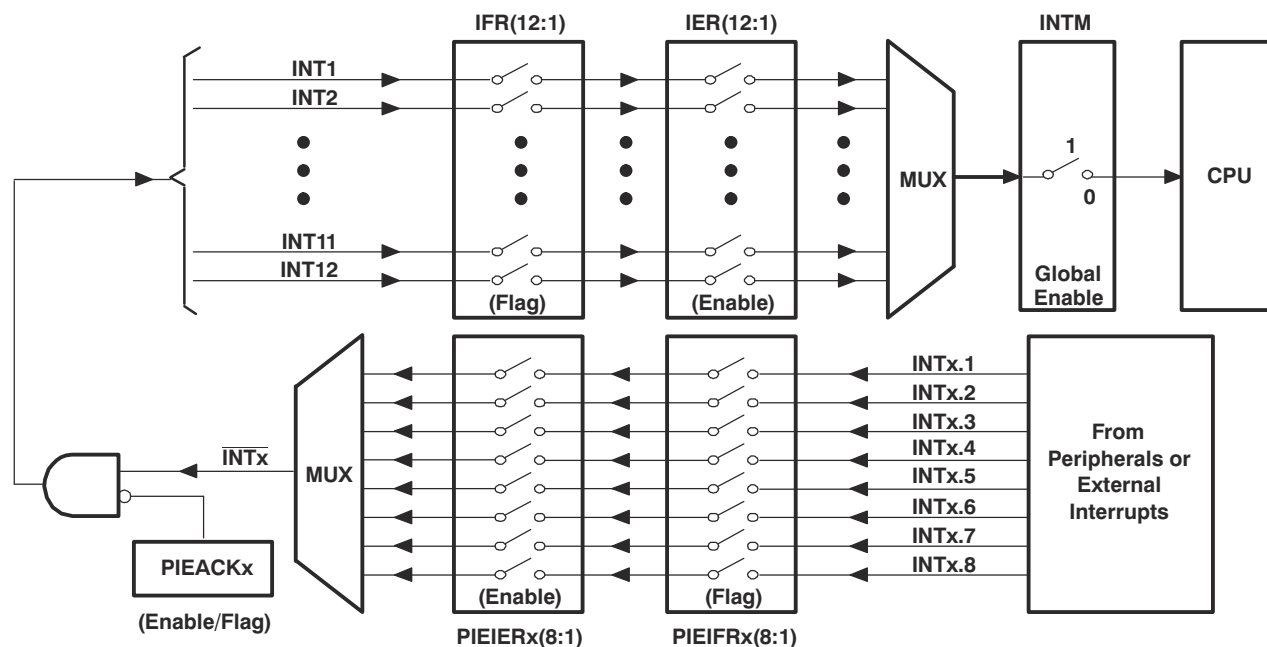


Figure 1-84. Overview: Multiplexing of Interrupts Using the PIE Block



- **Peripheral Level**

An interrupt-generating event occurs in a peripheral. The interrupt flag (IF) bit corresponding to that event is set in a register for that particular peripheral.

If the corresponding interrupt enable (IE) bit is set, the peripheral generates an interrupt request to the PIE controller. If the interrupt is not enabled at the peripheral level, the IF remains set until cleared by software. If the interrupt is enabled at a later time, and the interrupt flag is still set, the interrupt request is asserted to the PIE.

Interrupt flags within the peripheral registers must be manually cleared. See the peripheral reference guide for a specific peripheral for more information.

- **PIE Level**

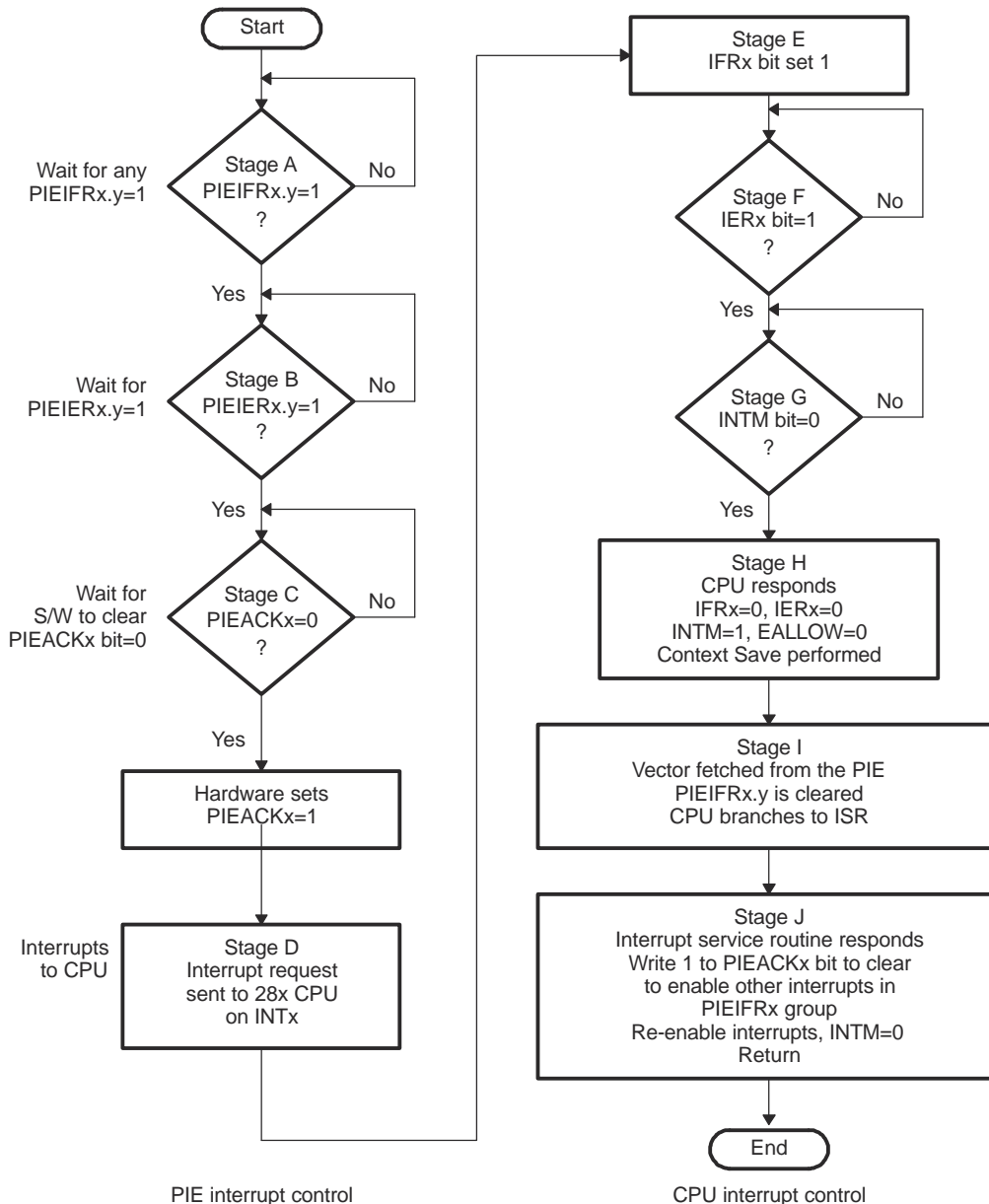
The PIE block multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 - PIE group 12. The interrupts within a group are multiplexed into one CPU interrupt. For example, PIE group 1 is multiplexed into CPU interrupt 1 (INT1) while PIE group 12 is multiplexed into CPU interrupt 12 (INT12). Interrupt sources connected to the remaining CPU interrupts are not multiplexed. For the nonmultiplexed interrupts, the PIE passes the request directly to the CPU.

For multiplexed interrupt sources, each interrupt group in the PIE block has an associated flag register (PIEIFRx) and enable (PIEIERx) register (x = PIE group 1 - PIE group 12). Each bit, referred to as y, corresponds to one of the 8 MUXed interrupts within the group. Thus PIEIFRx.y and PIEIERx.y correspond to interrupt y (y = 1-8) in PIE group x (x = 1-12). In addition, there is one acknowledge bit (PIEACK) for every PIE interrupt group referred to as PIEACKx (x = 1-12). [Figure 1-85](#) illustrates the behavior of the PIE hardware under various PIEIFR and PIEIER register conditions.

Once the request is made to the PIE controller, the corresponding PIE interrupt flag (PIEIFRx.y) bit is set. If the PIE interrupt enable (PIEIERx.y) bit is also set for the given interrupt, the PIE checks the corresponding PIEACKx bit to determine if the CPU is ready for an interrupt from that group. If the PIEACKx bit is clear for that group, the PIE sends the interrupt request to the CPU. If PIEACKx is set, the PIE waits until it is cleared to send the request for INTx. See [Figure 1-88](#) for details.

- **CPU Level**

Once the request is sent to the CPU, the CPU level interrupt flag (IFR) bit corresponding to INTx is set. After a flag has been latched in the IFR, the corresponding interrupt is not serviced until it is appropriately enabled in the CPU interrupt enable (IER) register or the debug interrupt enable register (DBGIER) and the global interrupt mask (INTM) bit.



- A. For multiplexed interrupts, the PIE responds with the highest priority interrupt that is both flagged and enabled. If there is no interrupt both flagged and enabled, then the highest priority interrupt within the group (INTx.1 where x is the PIE group) is used. See [Section 1.6.3.3](#) for details.

**Figure 1-85. Typical PIE/CPU Interrupt Response - INTx.y**

As shown in [Table 1-107](#), the requirements for enabling the maskable interrupt at the CPU level depends on the interrupt handling process being used. In the standard process, which happens most of the time, the DBGIER register is not used. When the 28x is in real-time emulation mode and the CPU is halted, a different process is used. In this special case, the DBGIER is used and the INTM bit is ignored. If the device is in real-time mode and the CPU is running, the standard interrupt-handling process applies.

**Table 1-107. Enabling Interrupt**

Interrupt Handling Process	Interrupt Enabled If...
Standard	INTM = 0 and bit in IER is 1
Device in real-time mode and halted	Bit in IER is 1 and DBGIER is 1

The CPU then prepares to service the interrupt. This preparation process is described in the [TMS320C28x DSP CPU and Instruction Set Reference Guide](#). In preparation, the corresponding CPU IFR and IER bits are cleared, EALLOW and LOOP are cleared, INTM and DBGM are set, the pipeline is flushed and the return address is stored, and the automatic context save is performed. The vector of the ISR is then fetched from the PIE module. If the interrupt request comes from a multiplexed interrupt, the PIE module uses the group PIEIERx and PIEIFRx registers to decode which interrupt needs to be serviced. This decode process is described in [Section 1.6.3.3](#).

The address for the interrupt service routine that is executed is fetched directly from the PIE interrupt vector table. There is one 32-bit vector for each of the possible 96 interrupts within the PIE. Interrupt flags within the PIE module (PIEIFRx.y) are automatically cleared when the interrupt vector is fetched. The PIE acknowledge bit for a given interrupt group, however, must be cleared manually when ready to receive more interrupts from the PIE group.

## 1.6.2 Vector Table Mapping

On 28xx devices, the interrupt vector table can be mapped to four distinct locations in memory. In practice only the PIE vector table mapping is used.

This vector mapping is controlled by the following mode bits/signals:

VMAP:	VMAP is found in Status Register 1 ST1 (bit 3). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC VMAP instructions. For normal operation leave this bit set.
M0M1MAP:	M0M1MAP is found in Status Register 1 ST1 (bit 11). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC M0M1MAP instructions. For normal 28xx device operation, this bit should remain set. M0M1MAP = 0 is reserved for TI testing only.
ENPIE:	ENPIE is found in PIECTRL Register (bit 0). The default value of this bit, on reset, is set to 0 (PIE disabled). The state of this bit can be modified after reset by writing to the PIECTRL register (address 0x0000 0CE0).

Using these bits and signals the possible vector table mappings are shown in [Table 1-108](#).

**Table 1-108. Interrupt Vector Table Mapping**

Vector MAPS	Vectors Fetched From	Address Range	VMAP <sup>(1)</sup>	M0M1MAP <sup>(1)</sup>	ENPIE <sup>(1)</sup>
M1 Vector <sup>(2)</sup>	M1 SARAM Block	0x000000 - 0x00003F	0	0	X
M0 Vector <sup>(2)</sup>	M0 SARAM Block	0x000000 - 0x00003F	0	1	X
BROM Vector	Boot ROM Block	0x3FFFC0 - 0x3FFFFFF	1	X	0
PIE Vector	PIE Block	0x000D00 - 0x000DFF	1	X	1

(1) On the 280x devices, the VMAP and M0M1MAP modes are set to 1 on reset. The ENPIE mode is forced to 0 on reset.

(2) Vector map M0 and M1 Vector is a reserved mode only. On the 28x devices these are used as SARAM.

The M1 and M0 vector table mapping are reserved for TI testing only. When using other vector mappings, the M0 and M1 memory blocks are treated as SARAM blocks and can be used freely without any restrictions.

After a device reset operation, the vector table is mapped as shown in [Table 1-109](#).

**Table 1-109. Vector Table Mapping After Reset Operation**

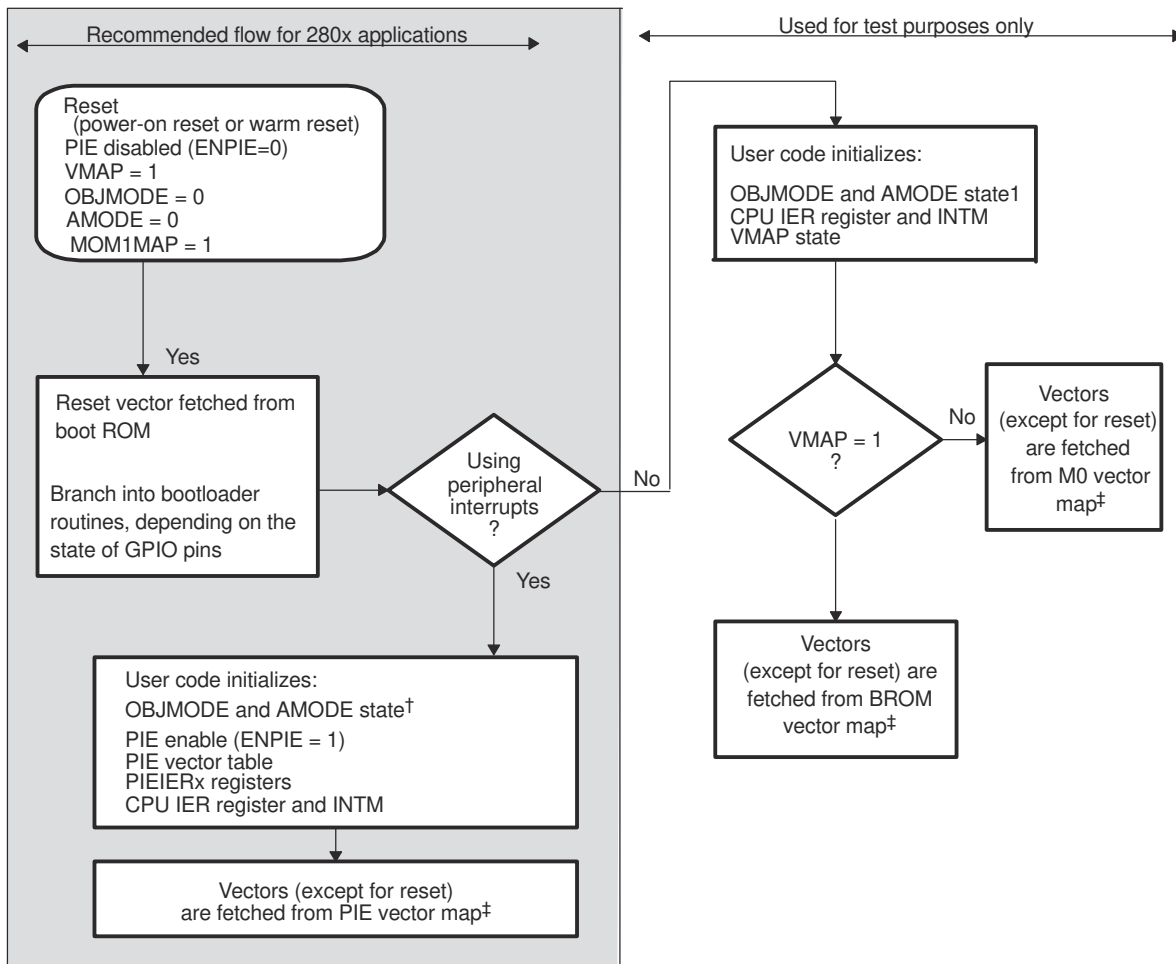
Vector MAPS	Reset Fetched From	Address Range	VMAP <sup>(1)</sup>	M0M1MAP <sup>(1)</sup>	ENPIE <sup>(1)</sup>
BROM Vector <sup>(2)</sup>	Boot ROM Block	0x3FFFC0 - 0x3FFFFFF	1	1	0

(1) On the 28x devices, the VMAP and M0M1MAP modes are set to 1 on reset. The ENPIE mode is forced to 0 on reset.

(2) The reset vector is always fetched from the boot ROM.

After the reset and boot is complete, the PIE vector table should be initialized by the user's code. Then the application enables the PIE vector table. From that point on the interrupt vectors are fetched from the PIE vector table. Note: when a reset occurs, the reset vector is always fetched from the vector table as shown in [Table 1-109](#). After a reset the PIE vector table is always disabled.

[Figure 1-86](#) illustrates the process by which the vector table mapping is selected.



A. The compatibility operating mode of the 28x CPU is determined by a combination of the OBJMODE and AMODE bits in Status Register 1 (ST1):

Operating Mode	OBJMODE	AMODE	
C28x Mode	1	0	
24x/240xA Source-Compatible	1	1	
C27x Object-Compatible	0	0	(Default at reset)

B. The reset vector is always fetched from the boot ROM.

Figure 1-86. Reset Flow Diagram

### 1.6.3 Interrupt Sources

Figure 1-87 shows how the various interrupt sources are multiplexed within the devices. This multiplexing (MUX) scheme may not be exactly the same on all 28x devices. See the data manual of your particular device for details.

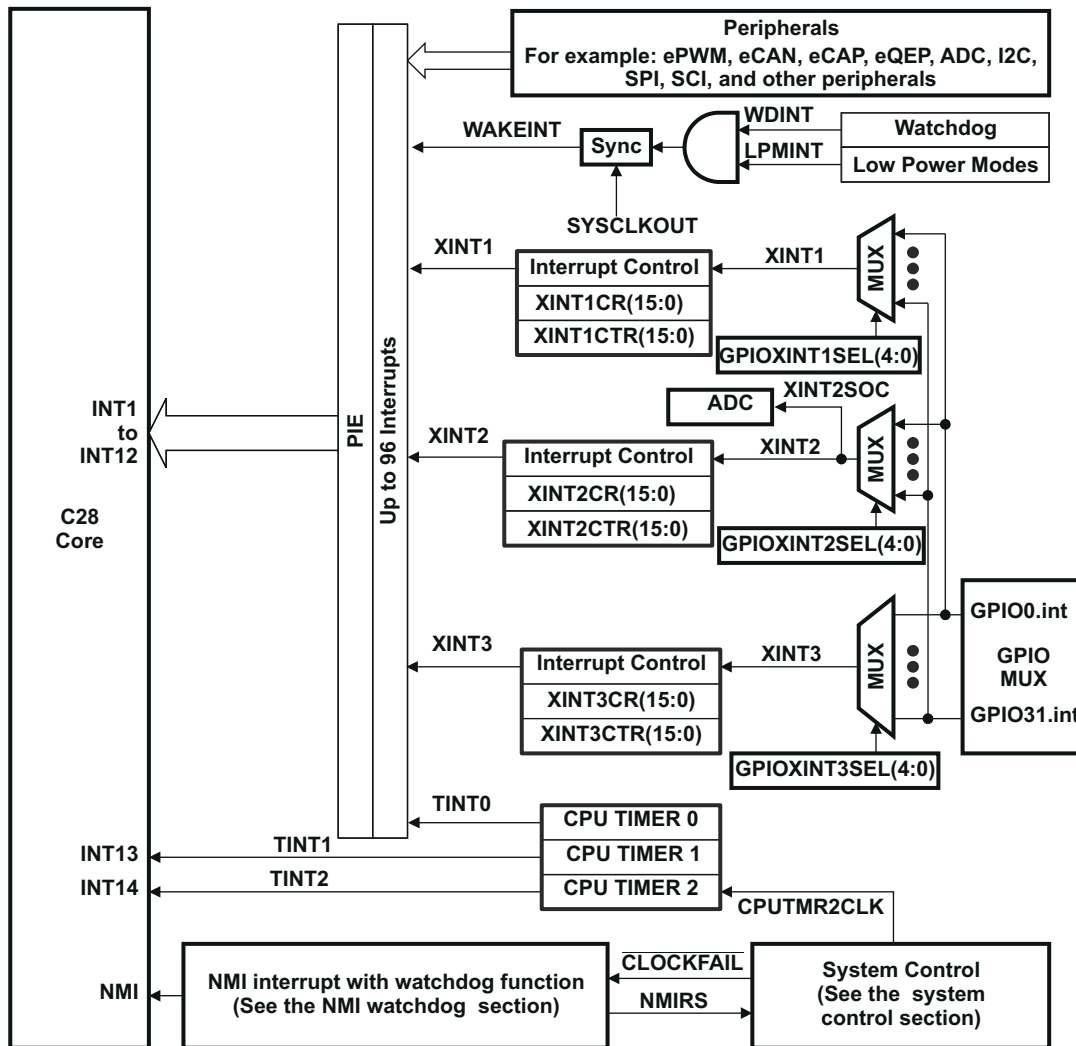


Figure 1-87. PIE Interrupt Sources and External Interrupts XINT1/XINT2/XINT3

### 1.6.3.1 Procedure for Handling Multiplexed Interrupts

The PIE module multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 - PIE group 12. Each group has an associated enable PIEIER and flag PIEIFR register. These registers are used to control the flow of interrupts to the CPU. The PIE module also uses the PIEIER and PIEIFR registers to decode to which interrupt service routine the CPU should branch.

There are three main rules that should be followed when clearing bits within the PIEIFR and the PIEIER registers:

#### Rule 1: Never clear a PIEIFR bit by software

An incoming interrupt may be lost while a write or a read-modify-write operation to the PIEIFR register takes place. To clear a PIEIFR bit, the pending interrupt must be serviced. If you want to clear the PIEIFR bit without executing the normal service routine, use the following procedure:

1. Set the EALLOW bit to allow modification to the PIE vector table.
2. Modify the PIE vector table so that the vector for the peripheral's service routine points to a temporary ISR. This temporary ISR will only perform a return from interrupt (IRET) operation.
3. Enable the interrupt so that the interrupt will be serviced by the temporary ISR.
4. After the temporary interrupt routine is serviced, the PIEIFR bit will be clear
5. Modify the PIE vector table to re-map the peripheral's service routine to the proper service routine.
6. Clear the EALLOW bit.

#### Rule 2: Procedure for software-prioritizing interrupts

Use the method found in the C2000Ware example for software prioritization of interrupts (SPRC530).

1. Use the CPU IER register as a global priority and the individual PIEIER registers for group priorities. In this case the PIEIER register is only modified within an interrupt. In addition, only the PIEIER for the same group as the interrupt being serviced is modified. This modification is done while the PIEACK bit holds additional interrupts back from the CPU.
2. Never disable a PIEIER bit for a group when servicing an interrupt from an unrelated group.

#### Rule 3: Disabling interrupts using PIEIER

If the PIEIER registers are used to enable and then later disable an interrupt, the procedure described in [Section 1.6.3.2](#) must be followed.

### 1.6.3.2 Procedures for Enabling And Disabling Multiplexed Peripheral Interrupts

The proper procedure for enabling or disabling an interrupt is by using the peripheral interrupt enable/disable flags. The primary purpose of the PIEIER and CPU IER registers is for software prioritization of interrupts within the same PIE interrupt group. Use the method found in the C2000Ware example for software prioritization of interrupts (SPRC530).

Should bits within the PIEIER registers need to be cleared outside of this context, one of the following two procedures should be followed. The first method preserves the associated PIE flag register so that interrupts are not lost. The second method clears the associated PIE flag register.

#### **Method 1: Use the PIEIERx register to disable the interrupt and preserve the associated PIEIFRx flags.**

To clear bits within a PIEIERx register while preserving the associated flags in the PIEIFRx register, the following procedure should be followed:

1. Disable global interrupts (INTM = 1).
2. Clear the PIEIERx.y bit to disable the interrupt for a given peripheral. This can be done for one or more peripherals within the same group.
3. Wait 5 cycles. This delay is required to be sure that any interrupt that was incoming to the CPU has been flagged within the CPU IFR register.
4. Clear the CPU IFRx bit for the peripheral group. This is a safe operation on the CPU IFR register.
5. Clear the PIEACKx bit for the peripheral group.
6. Enable global interrupts (INTM = 0).

#### **Method 2: Use the PIEIERx register to disable the interrupt and clear the associated PIEIFRx flags.**

To perform a software reset of a peripheral interrupt and clear the associated flag in the PIEIFRx register and CPU IFR register, the following procedure should be followed:

1. Disable global interrupts (INTM = 1).
2. Set the EALLOW bit.
3. Modify the PIE vector table to temporarily map the vector of the specific peripheral interrupt to a empty interrupt service routine (ISR). This empty ISR will only perform a return from interrupt (IRET) instruction. This is the safe way to clear a single PIEIFRx.y bit without losing any interrupts from other peripherals within the group.
4. Disable the peripheral interrupt at the peripheral register.
5. Enable global interrupts (INTM = 0).
6. Wait for any pending interrupt from the peripheral to be serviced by the empty ISR routine.
7. Disable global interrupts (INTM = 1).
8. Modify the PIE vector table to map the peripheral vector back to its original ISR.
9. Clear the EALLOW bit.
10. Disable the PIEIER bit for given peripheral.
11. Clear the IFR bit for given peripheral group (this is safe operation on CPU IFR register).
12. Clear the PIEACK bit for the PIE group.
13. Enable global interrupts.





- b. If no flagged interrupts within the group are enabled, the PIE will respond with the vector for the highest priority interrupt within that group. That is the branch address used for INTx.1. This behavior corresponds to the 28x TRAP or INT instructions.

---

#### Note

Because the PIEIERx register is used to determine which vector will be used for the branch, you must take care when clearing bits within the PIEIERx register. The proper procedure for clearing bits within a PIEIERx register is described in [Section 1.6.3.2](#). Failure to follow these steps can result in changes occurring to the PIEIERx register after an interrupt has been passed to the CPU at Step 5 in [Figure 1-88](#). In this case, the PIE will respond as if a TRAP or INT instruction was executed unless there are other interrupts both pending and enabled.

At this point, the PIEIFRx.y bit is cleared and the CPU branches to the vector of the interrupt fetched from the PIE.

---

#### 1.6.3.4 PIE Vector Table

The PIE vector table (see [Table 1-111](#)) consists of a 256 x 16 SARAM block that can also be used as RAM (in data space only) if the PIE block is not in use. The PIE vector table contents are undefined on reset. The CPU fixes interrupt priority for INT1 to INT12. The PIE controls priority for each group of eight interrupts. For example if INT1.1 should occur simultaneously with INT1.8, both interrupts are presented to the CPU simultaneously by the PIE block, and the CPU services INT1.1 first. If INT1.1 should occur simultaneously with INT1.8, then INT1.1 is sent to the CPU first and then INT1.8 follows. Interrupt prioritization is performed during the vector fetch portion of the interrupt processing.

When the PIE is enabled, a TRAP #1 through TRAP #12 or an INTR INT1 to INTR INT12 instruction transfers program control to the interrupt service routine corresponding to the first vector within the PIE group. For example: TRAP #1 fetches the vector from INT1.1, TRAP #2 fetches the vector from INT2.1 and so forth. Similarly an OR IFR, #16-bit operation causes the vector to be fetched from INTR1.1 to INTR12.1 locations, if the respective interrupt flag is set. All other TRAP, INTR, OR IFR, #16-bit operations fetch the vector from the respective table location. The vector table is EALLOW protected.

Out of the 96 possible MUXed interrupts in [Table 1-110](#), 43 interrupts are currently used. The remaining interrupts are reserved for future devices. These reserved interrupts can be used as software interrupts if they are enabled at the PIEIFRx level, provided none of the interrupts within the group is being used by a peripheral. Otherwise, interrupts coming from peripherals may be lost by accidentally clearing their flags when modifying the PIEIFR.

To summarize, there are two safe cases when the reserved interrupts can be used as software interrupts:

1. No peripheral within the group is asserting interrupts.
2. No peripheral interrupts are assigned to the group. For example, PIE group 11 and 12 do not have any peripherals attached to them.

The interrupt grouping for peripherals and external interrupts connected to the PIE module is shown in [Table 1-110](#). Each row in the table shows the 8 interrupts multiplexed into a particular CPU interrupt. The entire PIE vector table, including both MUXed and non-MUXed interrupts, is shown in [Table 1-111](#).

**Table 1-110. PIE MUXed Peripheral Interrupt Vector Table**

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
<b>INT1.y</b>	WAKEINT (LPM/WD) 0xD4E	TINT0 (TIMER 0) 0xD4C	ADCINT9 (ADC) 0xD4A	XINT2 Ext. int. 2 0xD48	XINT1 Ext. int. 1 0xD46	Reserved - 0xD44	ADCINT2 (ADC) 0xD42	ADCINT1 (ADC) 0xD40
<b>INT2.y</b>	Reserved - 0xD5E	Reserved - 0xD5C	Reserved - 0xD5A	Reserved - 0xD58	EPWM4_TZINT (ePWM4) 0xD56	EPWM3_TZINT (ePWM3) 0xD54	EPWM2_TZINT (ePWM2) 0xD52	EPWM1_TZINT (ePWM1) 0xD50
<b>INT3.y</b>	Reserved - 0xD6E	Reserved - 0xD6C	Reserved - 0xD6A	Reserved - 0xD68	EPWM4_INT (ePWM4) 0xD66	EPWM3_INT (ePWM3) 0xD64	EPWM2_INT (ePWM2) 0xD62	EPWM1_INT (ePWM1) 0xD60
<b>INT4.y</b>	Reserved - 0xD7E	Reserved - 0xD7C	Reserved - 0xD7A	Reserved - 0xD78	Reserved - 0xD76	Reserved - 0xD74	Reserved - 0xD72	ECAP1_INT (eCAP1) 0xD70
<b>INT5.y</b>	Reserved - 0xD8E	Reserved - 0xD8C	Reserved - 0xD8A	Reserved - 0xD88	Reserved - 0xD86	Reserved - 0xD84	Reserved - 0xD82	Reserved - 0xD80
<b>INT6.y</b>	Reserved - 0xD9E	Reserved - 0xD9C	Reserved - 0xD9A	Reserved - 0xD98	Reserved - 0xD96	Reserved - 0xD94	SPITXINTA (SPI-A) 0xD92	SPIRXINTA (SPI-A) 0xD90
<b>INT7.y</b>	Reserved - 0xDAE	Reserved - 0xDAC	Reserved - 0xDAA	Reserved - 0xDA8	Reserved - 0xDA6	Reserved - 0xDA4	Reserved - 0xDA2	Reserved - 0xDA0
<b>INT8.y</b>	Reserved - 0xDBE	Reserved - 0xDBC	Reserved - 0xDBA	Reserved - 0xDB8	Reserved - 0xDB6	Reserved - 0xDB4	I2CINT2A (I <sup>2</sup> C-A) 0xDB2	I2CINT1A (I <sup>2</sup> C-A) 0xDB0
<b>INT9.y</b>	Reserved - 0xDCE	Reserved - 0xDCC	Reserved - 0xDCA	Reserved - 0xDC8	Reserved - 0xDC6	Reserved - 0xDC4	SCITXINTA (SCI-A) 0xDC2	SCIRXINTA (SCI-A) 0xDC0
<b>INT10.y</b>	ADCINT8 (ADC) 0xDDE	ADCINT7 (ADC) 0xDDC	ADCINT6 (ADC) 0xDDA	ADCINT5 (ADC) 0xDD8	ADCINT4 (ADC) 0xDD6	ADCINT3 (ADC) 0xDD4	ADCINT2 (ADC) 0xDD2	ADCINT1 (ADC) 0xDD0
<b>INT11.y</b>	Reserved - 0xDEE	Reserved - 0xDEC	Reserved - 0xDEA	Reserved - 0xDE8	Reserved - 0xDE6	Reserved - 0xDE4	Reserved - 0xDE2	Reserved - 0xDE0
<b>INT12.y</b>	Reserved - 0xDFE	Reserved - 0xDFC	Reserved - 0xDFA	Reserved - 0xDF8	Reserved - 0xDF6	Reserved - 0xDF4	Reserved - 0xDF2	XINT3 Ext. Int. 3 0xDF0

**Table 1-111. PIE Vector Table**

Name	VECTOR ID	Address <sup>(1)</sup>	Size (x16)	Description <sup>(2)</sup>	CPU Priority	PIE Group Priority
Reset	0	0x0000 0D00	2	Reset is always fetched from location 0x003F FFC0 in Boot ROM.	1 (highest)	-
INT1	1	0x0000 0D02	2	Not used. See PIE Group 1	5	-
INT2	2	0x0000 0D04	2	Not used. See PIE Group 2	6	-
INT3	3	0x0000 0D06	2	Not used. See PIE Group 3	7	-
INT4	4	0x0000 0D08	2	Not used. See PIE Group 4	8	-
INT5	5	0x0000 0D0A	2	Not used. See PIE Group 5	9	-
INT6	6	0x0000 0D0C	2	Not used. See PIE Group 6	10	-
INT7	7	0x0000 0D0E	2	Not used. See PIE Group 7	11	-
INT8	8	0x0000 0D10	2	Not used. See PIE Group 8	12	-
INT9	9	0x0000 0D12	2	Not used. See PIE Group 9	13	-
INT10	10	0x0000 0D14	2	Not used. See PIE Group 10	14	-
INT11	11	0x0000 0D16	2	Not used. See PIE Group 11	15	-
INT12	12	0x0000 0D18	2	Not used. See PIE Group 12	16	-
INT13	13	0x0000 0D1A	2	CPU-Timer1	17	-
INT14	14	0x0000 0D1C	2	CPU-Timer2	18	-
DATALOG	15	0x0000 0D1E	2	CPU Data Logging Interrupt	19 (lowest)	-
RTOSINT	16	0x0000 0D20	2	CPU Real-Time OS Interrupt	4	-
EMUINT	17	0x0000 0D22	2	CPU Emulation Interrupt	2	-
NMI	18	0x0000 0D24	2	External Non-Maskable Interrupt	3	-
ILLEGAL	19	0x0000 0D26	2	Illegal Operation	-	-
USER1	20	0x0000 0D28	2	User-Defined Trap	-	-
USER2	21	0x0000 0D2A	2	User Defined Trap	-	-
USER3	22	0x0000 0D2C	2	User Defined Trap	-	-
USER4	23	0x0000 0D2E	2	User Defined Trap	-	-
USER5	24	0x0000 0D30	2	User Defined Trap	-	-
USER6	25	0x0000 0D32	2	User Defined Trap	-	-
USER7	26	0x0000 0D34	2	User Defined Trap	-	-
USER8	27	0x0000 0D36	2	User Defined Trap	-	-
USER9	28	0x0000 0D38	2	User Defined Trap	-	-
USER10	29	0x0000 0D3A	2	User Defined Trap	-	-
USER11	30	0x0000 0D3C	2	User Defined Trap	-	-
USER12	31	0x0000 0D3E	2	User Defined Trap	-	-
<b>PIE Group 1 Vectors - MUXed into CPU INT1</b>						
INT1.1	32	0x0000 0D40	2	ADCINT1 (ADC)	5	1 (highest)
INT1.2	33	0x0000 0D42	2	ADCINT2 (ADC)	5	2
INT1.3	34	0x0000 0D44	2	Reserved -	5	3
INT1.4	35	0x0000 0D46	2	XINT1	5	4
INT1.5	36	0x0000 0D48	2	XINT2	5	5
INT1.6	37	0x0000 0D4A	2	ADCINT9 (ADC)	5	6
INT1.7	38	0x0000 0D4C	2	TINT0 (CPU-Timer0)	5	7
INT1.8	39	0x0000 0D4E	2	WAKEINT (LPM/WD)	5	8 (lowest)

**Table 1-111. PIE Vector Table (continued)**

Name	VECTOR ID	Address <sup>(1)</sup>	Size (x16)	Description <sup>(2)</sup>		CPU Priority	PIE Group Priority
<b>PIE Group 2 Vectors - MUXed into CPU INT2</b>							
INT2.1	40	0x0000 0D50	2	EPWM1_TZINT (EPWM1)		6	1 (highest)
INT2.2	41	0x0000 0D52	2	EPWM2_TZINT (EPWM2)		6	2
INT2.3	42	0x0000 0D54	2	EPWM3_TZINT (EPWM3)		6	3
INT2.4	43	0x0000 0D56	2	EPWM4_TZINT (EPWM4)		6	4
INT2.5	44	0x0000 0D58	2	Reserved -		6	5
INT2.6	45	0x0000 0D5A	2	Reserved -		6	6
INT2.7	46	0x0000 0D5C	2	Reserved -		6	7
INT2.8	47	0x0000 0D5E	2	Reserved -		6	8 (lowest)
<b>PIE Group 3 Vectors - MUXed into CPU INT3</b>							
INT3.1	48	0x0000 0D60	2	EPWM1_INT (EPWM1)		7	1 (highest)
INT3.2	49	0x0000 0D62	2	EPWM2_INT (EPWM2)		7	2
INT3.3	50	0x0000 0D64	2	EPWM3_INT (EPWM3)		7	3
INT3.4	51	0x0000 0D66	2	EPWM4_INT (EPWM4)		7	4
INT3.5	52	0x0000 0D68	2	Reserved -		7	5
INT3.6	53	0x0000 0D6A	2	Reserved -		7	6
INT3.7	54	0x0000 0D6C	2	Reserved -		7	7
INT3.8	55	0x0000 0D6E	2	Reserved -		7	8 (lowest)
<b>PIE Group 4 Vectors - MUXed into CPU INT4</b>							
INT4.1	56	0x0000 0D70	2	ECAP1_INT (ECAP1)		8	1 (highest)
INT4.2	57	0x0000 0D72	2	Reserved -		8	2
INT4.3	58	0x0000 0D74	2	Reserved -		8	3
INT4.4	59	0x0000 0D76	2	Reserved -		8	4
INT4.5	60	0x0000 0D78	2	Reserved -		8	5
INT4.6	61	0x0000 0D7A	2	Reserved -		8	6
INT4.7	62	0x0000 0D7C	2	Reserved -		8	7
INT4.8	63	0x0000 0D7E	2	Reserved -		8	8 (lowest)
<b>PIE Group 5 Vectors - MUXed into CPU INT5</b>							
INT5.1	64	0x0000 0D80	2	EQEP1_INT (EQEP1)		9	1 (highest)
INT5.2	65	0x0000 0D82	2	Reserved (EQEP2)		9	2
INT5.3	66	0x0000 0D84	2	Reserved -		9	3
INT5.4	67	0x0000 0D86	2	Reserved -		9	4
INT5.5	68	0x0000 0D88	2	Reserved -		9	5
INT5.6	69	0x0000 0D8A	2	Reserved -		9	6
INT5.7	70	0x0000 0D8C	2	Reserved -		9	7
INT5.8	71	0x0000 0D8E	2	Reserved -		9	8 (lowest)
<b>PIE Group 6 Vectors - MUXed into CPU INT6</b>							
INT6.1	72	0x0000 0D90	2	SPIRXINTA (SPI-A)		10	1 (highest)
INT6.2	73	0x0000 0D92	2	SPITXINTA (SPI-A)		10	2
INT6.3	74	0x0000 0D94	2	Reserved -		10	3
INT6.4	75	0x0000 0D96	2	Reserved -		10	4
INT6.5	76	0x0000 0D98	2	Reserved -		10	5
INT6.6	77	0x0000 0D9A	2	Reserved -		10	6
INT6.7	78	0x0000 0D9C	2	Reserved -		10	7
INT6.8	79	0x0000 0D9E	2	Reserved -		10	8 (lowest)

**Table 1-111. PIE Vector Table (continued)**

Name	VECTOR ID	Address <sup>(1)</sup>	Size (x16)	Description <sup>(2)</sup>		CPU Priority	PIE Group Priority
<b>PIE Group 7 Vectors - MUXed into CPU INT7</b>							
INT7.1	80	0x0000 0DA0	2	Reserved	-	11	1 (highest)
INT7.2	81	0x0000 0DA2	2	Reserved	-	11	2
INT7.3	82	0x0000 0DA4	2	Reserved	-	11	3
INT7.4	83	0x0000 0DA6	2	Reserved	-	11	4
INT7.5	84	0x0000 0DA8	2	Reserved	-	11	5
INT7.6	85	0x0000 0DAA	2	Reserved	-	11	6
INT7.7	86	0x0000 0DAC	2	Reserved	-	11	7
INT7.8	87	0x0000 0DAE	2	Reserved	-	11	8 (lowest)
<b>PIE Group 8 Vectors - MUXed into CPU INT8</b>							
INT8.1	88	0x0000 0DB0	2	I2CINT1A	(I <sup>2</sup> C-A)	12	1 (highest)
INT8.2	89	0x0000 0DB2	2	I2CINT2A	(I <sup>2</sup> C-A)	12	2
INT8.3	90	0x0000 0DB4	2	Reserved	-	12	3
INT8.4	91	0x0000 0DB6	2	Reserved	-	12	4
INT8.5	92	0x0000 0DB8	2	Reserved	-	12	5
INT8.6	93	0x0000 0DBA	2	Reserved	-	12	6
INT8.7	94	0x0000 0DBC	2	Reserved	-	12	7
INT8.8	95	0x0000 0DBE	2	Reserved	-	12	8 (lowest)
<b>PIE Group 9 Vectors - MUXed into CPU INT9</b>							
INT9.1	96	0x0000 0DC0	2	SCIRXINTA	(SCI-A)	13	1 (highest)
INT9.2	97	0x0000 0DC2	2	SCITXINTA	(SCI-A)	13	2
INT9.3	98	0x0000 0DC4	2	Reserved	-	13	3
INT9.4	99	0x0000 0DC6	2	Reserved	-	13	4
INT9.5	100	0x0000 0DC8	2	Reserved	-	13	5
INT9.6	101	0x0000 0DCA	2	Reserved	-	13	6
INT9.7	102	0x0000 0DCC	2	Reserved	-	13	7
INT9.8	103	0x0000 0DCE	2	Reserved	-	13	8 (lowest)
<b>PIE Group 10 Vectors - MUXed into CPU INT10</b>							
INT10.1	104	0x0000 0DD0	2	ADCINT1	(ADC)	14	1 (highest)
INT10.2	105	0x0000 0DD2	2	ADCINT2	(ADC)	14	2
INT10.3	106	0x0000 0DD4	2	ADCINT3	(ADC)	14	3
INT10.4	107	0x0000 0DD6	2	ADCINT4	(ADC)	14	4
INT10.5	108	0x0000 0DD8	2	ADCINT5	(ADC)	14	5
INT10.6	109	0x0000 0DDA	2	ADCINT6	(ADC)	14	6
INT10.7	110	0x0000 0DDC	2	ADCINT7	(ADC)	14	7
INT10.8	111	0x0000 0DDE	2	ADCINT8	(ADC)	14	8 (lowest)
<b>PIE Group 11 Vectors - MUXed into CPU INT11</b>							
INT11.1	112	0x0000 0DE0	2	Reserved	-	15	1 (highest)
INT11.2	113	0x0000 0DE2	2	Reserved	-	15	2
INT11.3	114	0x0000 0DE4	2	Reserved	-	15	3
INT11.4	115	0x0000 0DE6	2	Reserved	-	15	4
INT11.5	116	0x0000 0DE8	2	Reserved	-	15	5
INT11.6	117	0x0000 0DEA	2	Reserved	-	15	6
INT11.7	118	0x0000 0DEC	2	Reserved	-	15	7
INT11.8	119	0x0000 0DEE	2	Reserved	-	15	8 (lowest)

**Table 1-111. PIE Vector Table (continued)**

Name	VECTOR ID	Address <sup>(1)</sup>	Size (x16)	Description <sup>(2)</sup>		CPU Priority	PIE Group Priority
<b>PIE Group 12 Vectors - MUXed into CPU INT12</b>							
INT12.1	120	0x0000 0DF0	2	XINT3	-	16	1 (highest)
INT12.2	121	0x0000 0DF2	2	Reserved	-	16	2
INT12.3	122	0x0000 0DF4	2	Reserved	-	16	3
INT12.4	123	0x0000 0DF6	2	Reserved	-	16	4
INT12.5	124	0x0000 0DF8	2	Reserved	-	16	5
INT12.6	125	0x0000 0DFA	2	Reserved	-	16	6
INT12.7	126	0x0000 0DFC	2	Reserved	-	16	7
INT12.8	127	0x0000 0DFE	2	Reserved	-	16	8 (lowest)

(1) Reset is always fetched from location 0x003F FFC0 in Boot ROM.

(2) All the locations within the PIE vector table are EALLOW protected.

### 1.6.4 PIE Configuration Registers

The registers controlling the functionality of the PIE block are shown in [Table 1-112](#).

**Table 1-112. PIE Configuration and Control Registers**

Name	Address	Size (x16)	Description	Bit Description
PIECTR1	0x0000 - 0CE0	1	PIE, Control Register	<a href="#">Section 1.6.4.1</a>
PIEACK	0x0000 - 0CE1	1	PIE, Acknowledge Register	<a href="#">Section 1.6.4.2</a>
PIEIER1	0x0000 - 0CE2	1	PIE, INT1 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR1	0x0000 - 0CE3	1	PIE, INT1 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER2	0x0000 - 0CE4	1	PIE, INT2 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR2	0x0000 - 0CE5	1	PIE, INT2 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER3	0x0000 - 0CE6	1	PIE, INT3 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR3	0x0000 - 0CE7	1	PIE, INT3 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER4	0x0000 - 0CE8	1	PIE, INT4 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR4	0x0000 - 0CE9	1	PIE, INT4 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER5	0x0000 - 0CEA	1	PIE, INT5 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR5	0x0000 - 0CEB	1	PIE, INT5 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER6	0x0000 - 0CEC	1	PIE, INT6 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR6	0x0000 - 0CED	1	PIE, INT6 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER7	0x0000 - 0CEE	1	PIE, INT7 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR7	0x0000 - 0CEF	1	PIE, INT7 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER8	0x0000 - 0CF0	1	PIE, INT8 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR8	0x0000 - 0CF1	1	PIE, INT8 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER9	0x0000 - 0CF2	1	PIE, INT9 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR9	0x0000 - 0CF3	1	PIE, INT9 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER10	0x0000 - 0CF4	1	PIE, INT10 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR10	0x0000 - 0CF5	1	PIE, INT10 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER11	0x0000 - 0CF6	1	PIE, INT11 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR11	0x0000 - 0CF7	1	PIE, INT11 Group Flag Register	<a href="#">Section 1.6.4.4</a>
PIEIER12	0x0000 - 0CF8	1	PIE, INT12 Group Enable Register	<a href="#">Section 1.6.4.3</a>
PIEIFR12	0x0000 - 0CF9	1	PIE, INT12 Group Flag Register	<a href="#">Section 1.6.4.4</a>

### 1.6.4.1 PIE Control (PIECTRL) Register

Figure 1-89. PIE Control (PIECTRL) Register

15	PIEVECT	1	0
		ENPIE	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 1-113. PIE Control (PIECTRL) Register Field Descriptions

Bits	Field	Value	Description
15-1	PIEVECT		These bits indicate the address within the PIE vector table from which the vector was fetched. The least significant bit of the address is ignored and only bits 1 to 15 of the address is shown. You can read the vector value to determine which interrupt generated the vector fetch. <b>For Example:</b> If PIECTRL = 0x0D27 then the vector from address 0x0D26 (illegal operation) was fetched.
0	ENPIE		Enable vector fetching from PIE vector table. <b>Note:</b> The reset vector is never fetched from the PIE, even when it is enabled. This vector is always fetched from boot ROM.
		0	If this bit is set to 0, the PIE block is disabled and vectors are fetched from the CPU vector table in boot ROM. All PIE block registers (PIEACK, PIEIFR, PIEIER) can be accessed even when the PIE block is disabled.
		1	When ENPIE is set to 1, all vectors, except for reset, are fetched from the PIE vector table. The reset vector is always fetched from the boot ROM.

### 1.6.4.2 PIE Interrupt Acknowledge (PIEACK) Register

Figure 1-90. PIE Interrupt Acknowledge (PIEACK) Register

15	12	11	0
Reserved		PIEACK	
R-0		R/W1C-0	

LEGEND: R/W1C = Read/Write 1 to clear; R = Read only; -n = value after reset

Table 1-114. PIE Interrupt Acknowledge (PIEACK) Register Field Descriptions

Bits	Field	Value	Description
15-12	Reserved		Reserved
11-0	PIEACK		Each bit in PIEACK refers to a specific PIE group. Bit 0 refers to interrupts in PIE group 1 that are MUXed into $\overline{INT1}$ up to Bit 11, which refers to PIE group 12 which is MUXed into CPU $\overline{INT12}$
		bit x = 0 <sup>(1)</sup>	If a bit reads as a 0, it indicates that the PIE can send an interrupt from the respective group to the CPU. Writes of 0 are ignored.
		bit x = 1	Reading a 1 indicates if an interrupt from the respective group has been sent to the CPU and all other interrupts from the group are currently blocked. Writing a 1 to the respective interrupt bit clears the bit and enables the PIE block to drive a pulse into the CPU interrupt input if an interrupt is pending for that group.

(1) bit x = PIEACK bit 0 to PIEACK bit 11. Bit 0 refers to CPU  $\overline{INT1}$  up to bit 11 that refers to CPU  $\overline{INT12}$ .

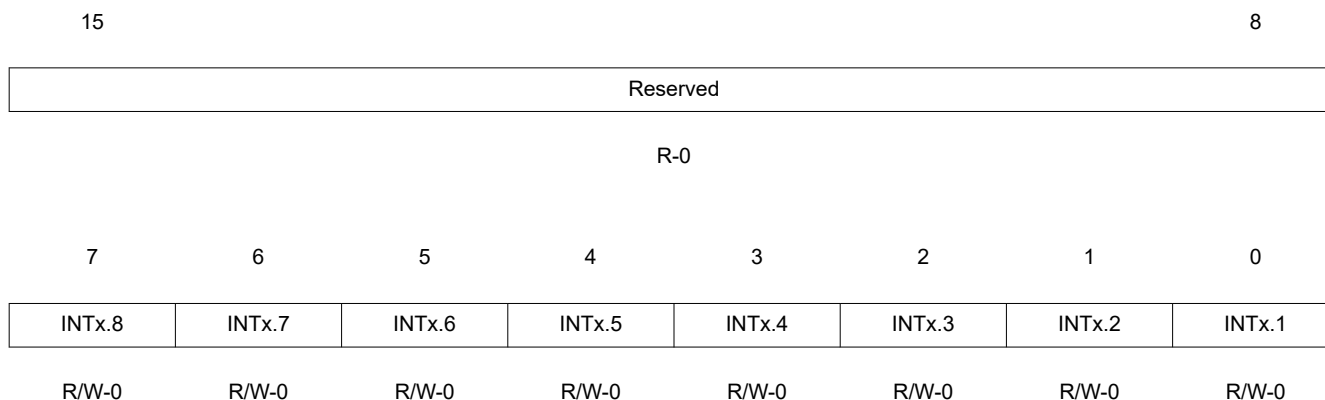
### 1.6.4.3 PIE Interrupt Enable (PIEIERn) Registers

There are twelve PIEIERn registers, one for each CPU interrupt used by the PIE module (INT1-INT12).

#### Note

Take care when clearing the PIEIERn bits during normal operation. See [Section 1.6.3.2](#) for the proper procedure for handling these bits.

**Figure 1-91. PIE Interrupt Enable (PIEIERn) Registers (n = 1 to 12)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-115. PIE Interrupt Enable (PIEIERn) Registers Field Descriptions**

Bits	Field	Description
15-8	Reserved	Any writes to these (bits) must always have a value of 0.
7	INTx.8	These register bits individually enable an interrupt within a group and behave very much like the core interrupt enable register. x = 1 to 12. INTx means CPU INT1 to INT12.  Setting a bit to 0 disables the servicing of the interrupt.  Setting a bit to 1 enables the servicing of the respective interrupt.
6	INTx.7	
5	INTx.6	
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	



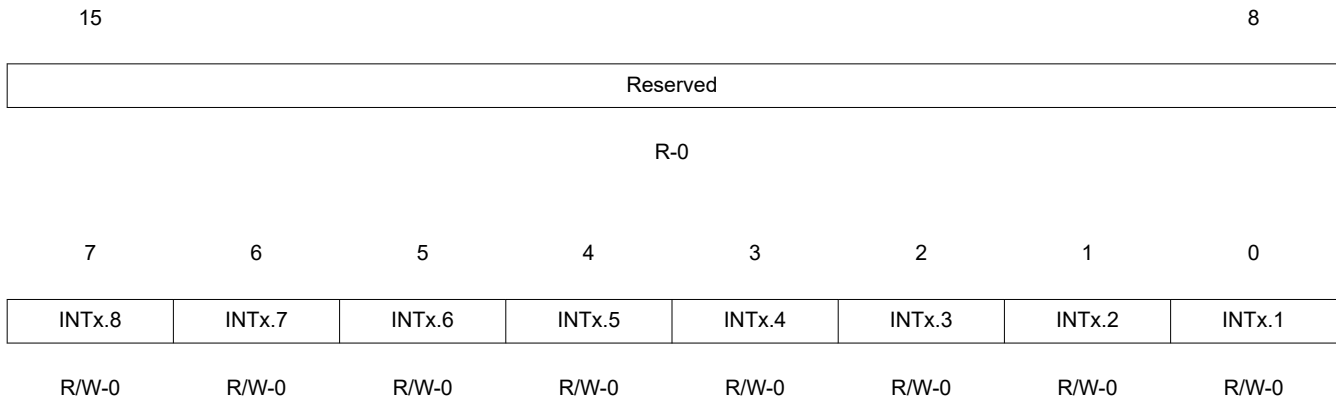
### 1.6.4.4 PIE Interrupt Flag (PIEIFRn) Registers

There are twelve PIEIFRn registers, one for each CPU interrupt used by the PIE module (INT1-INT12).

**CAUTION**

Never clear a PIEIFR bit. An interrupt may be lost during the read-modify-write operation. See [Section 1.6.3.1](#) for the method to clear flagged interrupts.

**Figure 1-92. PIE Interrupt Flag (PIEIFRn) Registers (n = 1 to 12)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-116. PIE Interrupt Flag (PIEIFRn) Registers Field Descriptions**

Bits	Field	Description
15-8	Reserved	Reserved
7	INTx.8	These register bits indicate whether an interrupt is currently active. They behave very much like the CPU interrupt flag register. When an interrupt is active, the respective register bit is set. The bit is cleared when the interrupt is serviced or by writing a 0 to the register bit. This register can also be read to determine which interrupts are active or pending. x = 1 to 12. INTx means CPU INT1 to INT12.
6	INTx.7	The PIEIFR register bit is cleared during the interrupt vector fetch portion of the interrupt processing.
5	INTx.6	Hardware has priority over CPU accesses to the PIEIFR registers.
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

### 1.6.4.5 CPU Interrupt Flag Register (IFR)

The CPU interrupt flag register (IFR), is a 16-bit, CPU register and is used to identify and clear pending interrupts. The IFR contains flag bits for all the maskable interrupts at the CPU level (INT1-INT14, DLOGINT and RTOSINT). When the PIE is enabled, the PIE module multiplexes interrupt sources for INT1-INT12.

When a maskable interrupt is requested, the flag bit in the corresponding peripheral control register is set to 1. If the corresponding mask bit is also 1, the interrupt request is sent to the CPU, setting the corresponding flag in the IFR. This indicates that the interrupt is pending or waiting for acknowledgment.

To identify pending interrupts, use the PUSH IFR instruction and then test the value on the stack. Use the OR IFR instruction to set IFR bits and use the AND IFR instruction to manually clear pending interrupts. All pending interrupts are cleared with the AND IFR #0 instruction or by a hardware reset.

The following events also clear an IFR flag:

- The CPU acknowledges the interrupt.
- The 28x device is reset.

#### Note

1. To clear a CPU IFR bit, you must write a zero to it, not a one.
2. When a maskable interrupt is acknowledged, only the IFR bit is cleared automatically. The flag bit in the corresponding peripheral control register is not cleared. If an application requires that the control register flag be cleared, the bit must be cleared by software.
3. When an interrupt is requested by an INTR instruction and the corresponding IFR bit is set, the CPU does not clear the bit automatically. If an application requires that the IFR bit be cleared, the bit must be cleared by software.
4. IMR and IFR registers pertain to core-level interrupts. All peripherals have their own interrupt mask and flag bits in their respective control/configuration registers. Note that several peripheral interrupts are grouped under one core-level interrupt.

**Figure 1-93. Interrupt Flag Register (IFR) — CPU Register**

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-117. Interrupt Flag Register (IFR) — CPU Register Field Descriptions**

Bits	Field	Value	Description
15	RTOSINT	0	Real-time operating system flag. RTOSINT is the flag for RTOS interrupts. No RTOS interrupt is pending
		1	At least one RTOS interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

**Table 1-117. Interrupt Flag Register (IFR) — CPU Register Field Descriptions (continued)**

Bits	Field	Value	Description
14	DLOGINT	0	Data logging interrupt flag. DLOGINT is the flag for data logging interrupts. No DLOGINT is pending
		1	At least one DLOGINT interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
13	INT14	0	Interrupt 14 flag. INT14 is the flag for interrupts connected to CPU interrupt level INT14. No INT14 interrupt is pending
		1	At least one INT14 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
12	INT13	0	Interrupt 13 flag. INT13 is the flag for interrupts connected to CPU interrupt level INT13. No INT13 interrupt is pending
		1	At least one INT13 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
11	INT12	0	Interrupt 12 flag. INT12 is the flag for interrupts connected to CPU interrupt level INT12. No INT12 interrupt is pending
		1	At least one INT12 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
10	INT11	0	Interrupt 11 flag. INT11 is the flag for interrupts connected to CPU interrupt level INT11. No INT11 interrupt is pending
		1	At least one INT11 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
9	INT10	0	Interrupt 10 flag. INT10 is the flag for interrupts connected to CPU interrupt level INT10. No INT10 interrupt is pending
		1	At least one INT6 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
8	INT9	0	Interrupt 9 flag. INT9 is the flag for interrupts connected to CPU interrupt level INT6. No INT9 interrupt is pending
		1	At least one INT9 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
7	INT8	0	Interrupt 8 flag. INT8 is the flag for interrupts connected to CPU interrupt level INT6. No INT8 interrupt is pending
		1	At least one INT8 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
6	INT7	0	Interrupt 7 flag. INT7 is the flag for interrupts connected to CPU interrupt level INT7. No INT7 interrupt is pending
		1	At least one INT7 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
5	INT6	0	Interrupt 6 flag. INT6 is the flag for interrupts connected to CPU interrupt level INT6. No INT6 interrupt is pending
		1	At least one INT6 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
4	INT5	0	Interrupt 5 flag. INT5 is the flag for interrupts connected to CPU interrupt level INT5. No INT5 interrupt is pending
		1	At least one INT5 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
3	INT4	0	Interrupt 4 flag. INT4 is the flag for interrupts connected to CPU interrupt level INT4. No INT4 interrupt is pending
		1	At least one INT4 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

**Table 1-117. Interrupt Flag Register (IFR) — CPU Register Field Descriptions (continued)**

Bits	Field	Value	Description
2	INT3	0	Interrupt 3 flag. INT3 is the flag for interrupts connected to CPU interrupt level INT3. No INT3 interrupt is pending
		1	At least one INT3 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
1	INT2	0	Interrupt 2 flag. INT2 is the flag for interrupts connected to CPU interrupt level INT2. No INT2 interrupt is pending
		1	At least one INT2 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
0	INT1	0	Interrupt 1 flag. INT1 is the flag for interrupts connected to CPU interrupt level INT1. No INT1 interrupt is pending
		1	At least one INT1 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

### 1.6.4.6 Interrupt Enable Register (IER)

The IER is a 16-bit CPU register. The IER contains enable bits for all the maskable CPU interrupt levels (INT1-INT14, RTOSINT and DLOGINT). Neither NMI nor XRS is included in the IER; thus, IER has no effect on these interrupts.

You can read the IER to identify enabled or disabled interrupt levels, and you can write to the IER to enable or disable interrupt levels. To enable an interrupt level, set its corresponding IER bit to one using the OR IER instruction. To disable an interrupt level, set its corresponding IER bit to zero using the AND IER instruction. When an interrupt is disabled, it is not acknowledged, regardless of the value of the INTM bit. When an interrupt is enabled, it is acknowledged if the corresponding IFR bit is one and the INTM bit is zero.

When using the OR IER and AND IER instructions to modify IER bits, make sure they do not modify the state of bit 15 (RTOSINT) unless a real-time operating system is present.

When a hardware interrupt is serviced or an INTR instruction is executed, the corresponding IER bit is cleared automatically. When an interrupt is requested by the TRAP instruction the IER bit is not cleared automatically. In the case of the TRAP instruction if the bit needs to be cleared it must be done by the interrupt service routine.

At reset, all the IER bits are cleared to 0, disabling all maskable CPU level interrupts.

**Figure 1-94. Interrupt Enable Register (IER) — CPU Register**

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-118. Interrupt Enable Register (IER) Field Descriptions**

Bits	Field	Value	Description
15	RTOSINT	0 1	Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt. RTOSINT is disabled RTOSINT is enabled
14	DLOGINT	0 1	Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt. DLOGINT is disabled DLOGINT is enabled
13	INT14	0 1	Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14. Level INT14 is disabled Level INT14 is enabled
12	INT13	0 1	Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13. Level INT13 is disabled Level INT13 is enabled

**Table 1-118. Interrupt Enable Register (IER) Field Descriptions (continued)**

Bits	Field	Value	Description
11	INT12		Interrupt 12 enable. INT12 enables or disables CPU interrupt level INT12.
		0	Level INT12 is disabled
		1	Level INT12 is enabled
		10	INT11
0	Level INT11 is disabled		
		1	Level INT11 is enabled
		9	INT10
0	Level INT10 is disabled		
		1	Level INT10 is enabled
		8	INT9
0	Level INT9 is disabled		
		1	Level INT9 is enabled
		7	INT8
0	Level INT8 is disabled		
		1	Level INT8 is enabled
		6	INT7
0	Level INT7 is disabled		
		1	Level INT7 is enabled
		5	INT6
0	Level INT6 is disabled		
		1	Level INT6 is enabled
		4	INT5
0	Level INT5 is disabled		
		1	Level INT5 is enabled
		3	INT4
0	Level INT4 is disabled		
		1	Level INT4 is enabled
		2	INT3
0	Level INT3 is disabled		
		1	Level INT3 is enabled
		1	INT2
0	Level INT2 is disabled		
		1	Level INT2 is enabled
		0	INT1
0	Level INT1 is disabled		
		1	Level INT1 is enabled

### 1.6.4.7 Debug Interrupt Enable Register (DBGIER)

The Debug Interrupt Enable Register (DBGIER) is used only when the CPU is halted in real-time emulation mode. An interrupt enabled in the DBGIER is defined as a time-critical interrupt. When the CPU is halted in real-time mode, the only interrupts that are serviced are time-critical interrupts that are also enabled in the IER. If the CPU is running in real-time emulation mode, the standard interrupt-handling process is used and the DBGIER is ignored.

As with the IER, you can read the DBGIER to identify enabled or disabled interrupts and write to the DBGIER to enable or disable interrupts. To enable an interrupt, set its corresponding bit to 1. To disable an interrupt, set its corresponding bit to 0. Use the PUSH DBGIER instruction to read from the DBGIER and POP DBGIER to write to the DBGIER register. At reset, all the DBGIER bits are set to 0.

**Figure 1-95. Debug Interrupt Enable Register (DBGIER) — CPU Register**

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 1-119. Debug Interrupt Enable Register (DBGIER) Field Descriptions**

Bits	Field	Value	Description
15	RTOSINT	0 1	Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt. RTOSINT is disabled RTOSINT is enabled
14	DLOGINT	0 1	Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt. DLOGINT is disabled DLOGINT is enabled
13	INT14	0 1	Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14. Level INT14 is disabled Level INT14 is enabled
12	INT13	0 1	Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13. Level INT13 is disabled Level INT13 is enabled
11	INT12	0 1	Interrupt 12 enable. INT12 enables or disables CPU interrupt level INT12. Level INT12 is disabled Level INT12 is enabled
10	INT11	0 1	Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11. Level INT11 is disabled Level INT11 is enabled

**Table 1-119. Debug Interrupt Enable Register (DBGIER) Field Descriptions (continued)**

Bits	Field	Value	Description
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
8	INT9		Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0	Level INT9 is disabled
7	INT8		Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0	Level INT8 is disabled
6	INT7		Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT7.
		0	Level INT7 is disabled
5	INT6		Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0	Level INT6 is disabled
4	INT5		Interrupt 5 enable. INT5 enables or disables CPU interrupt level INT5.
		0	Level INT5 is disabled
3	INT4		Interrupt 4 enable. INT4 enables or disables CPU interrupt level INT4.
		0	Level INT4 is disabled
2	INT3		Interrupt 3 enable. INT3 enables or disables CPU interrupt level INT3.
		0	Level INT3 is disabled
1	INT2		Interrupt 2 enable. INT2 enables or disables CPU interrupt level INT2.
		0	Level INT2 is disabled
0	INT1		Interrupt 1 enable. INT1 enables or disables CPU interrupt level INT1.
		0	Level INT1 is disabled
		1	Level INT1 is enabled



## 1.6.5 External Interrupt Configuration Registers

Three external interrupts, XINT1 – XINT3, are supported. Each of these external interrupts can be selected for negative or positive edge triggered and can also be enabled or disabled. The masked interrupts also contain a 16-bit free running up counter that is reset to zero when a valid interrupt edge is detected. This counter can be used to accurately time stamp the interrupt.

**Table 1-120. Interrupt Control and Counter Registers (not EALLOW Protected)**

Name	Address Range	Size (x16)	Description	Bit Description
XINT1CR	0x0000 7070	1	XINT1 control register	<a href="#">Section 1.6.5.1</a>
XINT2CR	0x0000 7071	1	XINT2 control register	<a href="#">Section 1.6.5.1</a>
XINT3CR	0x0000 7072	1	XINT3 control register	<a href="#">Section 1.6.5.1</a>
reserved	0x0000 7073 - 0x0000 7077	5		
XINT1CTR	0x0000 7078	1	XINT1 counter register	<a href="#">Section 1.6.5.2</a>
XINT2CTR	0x0000 7079	1	XINT2 counter register	<a href="#">Section 1.6.5.2</a>
XINT3CTR	0x0000 707A	1	XINT3 counter register	<a href="#">Section 1.6.5.2</a>
reserved	0x0000 707B - 0x0000 707E	5		

### 1.6.5.1 External Interrupt Control Registers (XINTnCR)

XINT1CR through XINT3CR are identical except for the interrupt number; therefore, [Figure 1-96](#) and [Table 1-121](#) represent registers for external interrupts 1 through 3 as XINTnCR, where  $n$  = the interrupt number.

**Figure 1-96. External Interrupt Control Registers (XINTnCR) (n = 1 to 3)**

15	4	3	2	1	0
Reserved		POLARITY		Reserved	ENABLE
R-0		R/W-0		R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-121. External Interrupt Control Registers (XINTnCR) Field Descriptions**

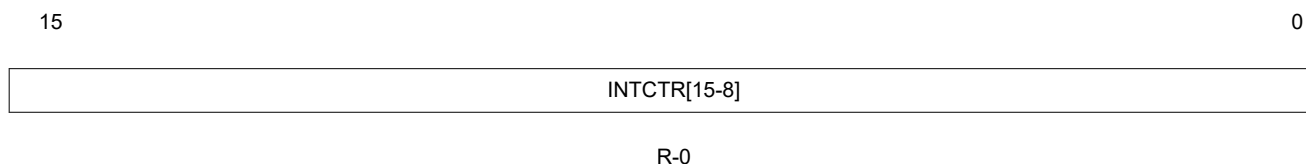
Bits	Field	Value	Description
15-4	Reserved		Any writes to these (bits) must always have a value of 0.
3-2	POLARITY	00 01 10 11	This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin. Interrupt generated on a falling edge (high-to-low transition) Interrupt generated on a rising edge (low-to-high transition) Interrupt generated on a falling edge (high-to-low transition) Interrupt generated on both a falling edge and a rising edge (high-to-low and low-to-high transition)
1	Reserved		Any writes to these (bits) must always have a value of 0.
0	ENABLE	0 1	This read/write bit enables or disables external interrupt XINTn. Disable interrupt Enable interrupt

### 1.6.5.2 External Interrupt n Counter Registers (XINTnCTR)

For XINT1/XINT2/XINT3, there is also a 16-bit counter that is reset to 0x000 whenever an interrupt edge is detected. These counters can be used to accurately time stamp an occurrence of the interrupt.

XINT1CTR through XINT3CTR are identical except for the interrupt number; therefore, [Figure 1-97](#) and [Table 1-122](#) represent registers for the external interrupts as XINTnCTR, where n = the interrupt number.

**Figure 1-97. External Interrupt n Counter Registers (XINTnCTR)**



LEGEND: R = Read only; -n = value after reset

**Table 1-122. External Interrupt n Counter Registers (XINTnCTR)Field Descriptions**

Bits	Field	Description
15-0	INTCTR	This is a free running 16-bit up-counter that is clocked at the SYSCLKOUT rate. The counter value is reset to 0x0000 when a valid interrupt edge is detected and then continues counting until the next valid interrupt edge is detected. When the interrupt is disabled, the counter stops. The counter is a free-running counter and wraps around to zero when the maximum value is reached. The counter is a read only register and can only be reset to zero by a valid interrupt edge or by reset.

## 1.7 VREG/BOR/POR

Although the core and I/O circuitry operate on two different voltages, these devices have an on-chip voltage regulator (VREG) to generate the  $V_{DD}$  voltage from the  $V_{DDIO}$  supply. This eliminates the cost and area of a second external regulator on an application board. Additionally, internal power-on reset (POR) and brown-out reset (BOR) circuits monitor both the  $V_{DD}$  and  $V_{DDIO}$  rails during power-up and run mode, eliminating a need for any external voltage supervisory circuits.

The  $V_{DD}$  BOR is only valid when the VREG is enabled. If VREG is disabled, and external LDO is used for 1.8V, then there is no BOR function on  $V_{DD}$ .

### 1.7.1 On-Chip Voltage Regulator (VREG)

An on-chip voltage regulator facilitates the powering of the device without adding the cost or board space of a second external regulator. This linear regulator generates the core  $V_{DD}$  voltage from the  $V_{DDIO}$  supply. Therefore, although capacitors are required on each  $V_{DD}$  pin to stabilize the generated voltage, power need not be supplied to these pins to operate the device. Conversely, the VREG can be bypassed or overdriven, should power or redundancy be the primary concern of the application.

#### 1.7.1.1 Using the On-Chip VREG

To utilize the on-chip VREG, the VREGENZ pin should be pulled low and the appropriate recommended operating voltage should be supplied to the  $V_{DDIO}$  and  $V_{DDA}$  pins. In this case, the  $V_{DD}$  voltage needed by the core logic will be generated by the VREG. Each  $V_{DD}$  pin requires on the order of 1.2  $\mu\text{F}$  capacitance for proper regulation of the VREG. These capacitors should be located as close as possible to the device pins. See the [TMS320F2802x Microcontrollers Data Manual](#) for the acceptable range of capacitance.

#### 1.7.1.2 Bypassing the On-Chip VREG

To conserve power, it is also possible to bypass the on-chip VREG and supply the core logic voltage to the  $V_{DD}$  pins with an external regulator. To enable this option, the VREGENZ pin must be pulled high. Refer to the [TMS320F2802x Microcontrollers Data Manual](#) for the acceptable range of voltage that must be supplied to the  $V_{DD}$  pins.

### 1.7.2 On-chip Power-On Reset (POR) and Brown-Out Reset (BOR) Circuit

Two on-chip supervisory circuits, the power-on reset (POR) and the brown-out reset (BOR) remove the burden of monitoring the  $V_{DD}$  and  $V_{DDIO}$  supply rails from the application board. The purpose of the POR is to create a clean reset throughout the device during the entire power-up procedure. The trip point is a looser, lower trip point than the BOR, which watches for dips in the  $V_{DD}$  or  $V_{DDIO}$  rail during device operation. The POR function is present on both  $V_{DD}$  and  $V_{DDIO}$  rails at all times. After initial device power-up, the BOR function is present on  $V_{DDIO}$  at all times, and on  $V_{DD}$  when the internal VREG is enabled (VREGENZ pin is pulled low). Both functions pull the  $\overline{XRS}$  pin low when one of the voltages is below their respective trip point. Additionally, when monitoring the  $V_{DD}$  rail, the BOR pulls  $\overline{XRS}$  low when  $V_{DD}$  is above its over-voltage trip point. See the device data sheet for the various trip points as well as the delay time from the removal of the fault condition to the release of the  $\overline{XRS}$  pin.

A bit is provided in the BORCFG register (address 0x985) to disable both the VDD and VDDIO BOR functions. The default state of this bit is to enable the BOR function. When the BOR functions are disabled, the POR functions will remain enabled. The BORCFG register is only reset by the XRS signal.

**Figure 1-98. BOR Configuration (BORCFG) Register**

15	3	2	1	0
Reserved		Reserved	Reserved	BORENZ
R-0		R-1	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 1-123. BOR Configuration (BORCFG) Field Descriptions**

Bits	Field	Value	Description
15-3	Reserved	0	Any writes to these bits must always have a value of 0.
2	Reserved	1	Reads always return a one. Writes have no effect.
1	Reserved	0	Any writes to these bits must always have a value of 0.
0	BORENZ	0	BOR enable active low bit. BOR functions are enabled.
		1	BOR functions are disabled.

The boot ROM is a block of read-only memory that is factory programmed. This chapter explains the boot procedure, the available boot modes, and the various details of the ROM code including memory maps, initializations, reset handling, and status information. The ROM source code is available under \libraries\boot\_rom directory in [C2000Ware](#).

<b>2.1 Boot ROM Memory Map</b> .....	<b>166</b>
<b>2.2 Bootloader Features</b> .....	<b>170</b>
<b>2.3 Building the Boot Table</b> .....	<b>201</b>
<b>2.4 Bootloader Code Overview</b> .....	<b>205</b>

## 2.1 Boot ROM Memory Map

The boot ROM is an 8K x 16 block of read-only memory located at addresses 0x3F E000 - 0x3F FFFF.

The on-chip boot ROM is factory programmed with bootload routines and math tables. These are for use with the [C28x IQMath Library - A Virtual Floating Point Engine](#).

This document describes the following items:

- Bootloader functions
- Version number, release date and checksum
- Reset vector
- Illegal trap vector (ITRAP)
- CPU vector table (used for test purposes only)
- IQmath Tables
- Selected IQmath functions
- Flash API library

[Figure 2-1](#) shows the memory map of the on-chip boot ROM. The memory block is 8Kx16 in size and is located at 0x3F E000 - 0x3F FFFF in both program and data space.

Data space	Program space	Address
IQ math tables		3F E000
IQmath functions		3F EC86
Boot loader functions		3F F4B0
Flash API library		3F F8D2
ROM version ROM checksum		3F FFB9
Reset vector CPU vector table		3F FFC0
		3F FFFF

**Figure 2-1. Memory Map of On-Chip ROM**

### 2.1.1 On-Chip Boot ROM IQmath Tables

The fixed-point math tables and functions included in the boot ROM are used by the [C28x IQMath Library - A Virtual Floating Point Engine](#)). The 28x IQmath Library is a collection of highly optimized and high precision mathematical functions for C/C++ programmers to seamlessly port a floating-point algorithm into fixed-point code on 28x devices.

These routines are typically used in computationally-intensive, real-time applications where optimal execution speed and high accuracy is critical. By using these routines, you can achieve execution speeds that are considerably faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use high precision functions, the TI IQmath Library can significantly shorten the development time.

The IQmath library accesses the tables through the IQmathTables and the IQmathTablesRam linker sections. Both of these sections are completely included in the boot ROM.

If you do not wish to load a copy of these tables already included in the ROM into the device, use the boot ROM memory addresses and label the sections as “NOLOAD” as shown in [Example 2-1](#). This facilitates referencing the lookup tables without actually loading the section to the target.

The preferred alternative to using the linker command file is to use the IQmath boot ROM symbol library. If this library is linked in the project before the IQmath library, and the linker-priority option is used, then any math tables and IQmath functions within the boot ROM will be used first. Refer to the IQMath library documentation for more information.

The following math tables are included in the boot ROM:

- **Sine/Cosine table, IQ Math Table**

- Table size: 1282 words
- Q format: Q30
- Contents: 32-bit samples for one and a quarter period sine wave

This is useful for accurate sine wave generation and 32-bit FFTs. This can also be used for 16-bit math; just skip over every second value

- **Normalized Inverse Table, IQ Math Table**

- Table size: 528 words
- Q format: Q29
- Contents: 32-bit normalized inverse samples plus saturation limits

This table is used as an initial estimate in the Newton-Raphson inverse algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.

- **Normalized Square Root Table, IQ Math Table**

- Table size: 274 words
- Q format: Q30
- Contents: 32-bit normalized inverse square root samples plus saturation

This table is used as an initial estimate in the Newton-Raphson square-root algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.

- **Normalized Arctan Table, IQ Math Table**

- Table size: 452 words
- Q format: Q30
- Contents 32-bit second order coefficients for line of best fit plus normalization table

This table is used as an initial estimate in the Arctan iterative algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.

- **Rounding and Saturation Table, IQ Math Table**

- Table size: 360 words
- Q format: Q30
- Contents: 32-bit rounding and saturation limits for various Q values

- **Exp Min/Max Table, IQMath Table**
  - Table size: 120 words
  - Q format: Q1 - Q30
  - Contents: 32-bit Min and Max values for each Q value
- **Exp Coefficient Table, IQMath Table**
  - Table size: 20 words
  - Q format: Q31
  - Contents: 32-bit coefficients for calculating exp (X) using a Taylor series
- **Inverse Sin/Cos Table, IQ Math Table**
  - Table size: 85 x 16
  - Q format: Q29
  - Contents: Coefficient table to calculate the formula  $f(x) = c4*x^4 + c3*x^3 + c2*x^2 + c1*x + c0$ .

### Example 2-1. Linker Command File to Access IQ Tables

```

MEMORY
{
    PAGE 0 :
    ...
    IQTABLES : origin = 0x3FE000, length = 0x000b50
    IQTABLES2 : origin = 0x3FEB50, length = 0x00008c
    IQTABLES3 : origin = 0x3FEBDC, length = 0x0000AA
    ...
}
SECTIONS
{
    ...
    IQmathTables : load = IQTABLES, type = NOLOAD, PAGE = 0
    IQmathTables2 > IQTABLES2, type = NOLOAD, PAGE = 0
    {
        IQmath.lib<IQNexpTable.obj> (IQmathTablesRam)
    }
    IQmathTables3 : load = IQTABLES3, PAGE = 0
    {
        IQNasinTable.obj (IQmathTablesRam)
    }
    ...
}
    
```

### 2.1.2 On-Chip Boot ROM IQmath Functions

The following IQmath functions are included in the Boot ROM:

- IQNatan2 N= 15, 20, 24, 29
- IQNcos N= 15, 20, 24, 29
- IQNdiv N= 15, 20, 24, 29
- IQisqrt N= 15, 20, 24, 29
- IQNmag N= 15, 20, 24, 29
- IQNsin N= 15, 20, 24, 29
- IQNsqrtn N= 15, 20, 24, 29

These functions can be accessed using the IQmath boot ROM symbol library included with the boot ROM source. If this library is linked in the project before the IQmath library, and the linker-priority option is used, then any math tables and IQmath functions within the boot ROM will be used first. Refer to the IQMath Library documentation for more information.

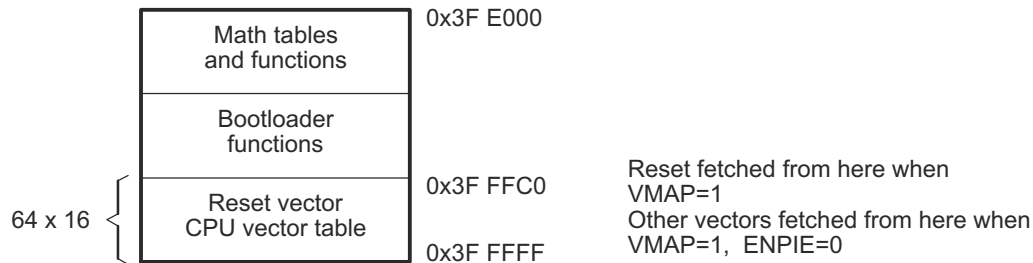
### 2.1.3 On-Chip Flash API

The boot ROM contains the API to program and erase the flash. This flash API can be accessed using the boot ROM flash API symbol library released with the boot ROM source. Refer to the Flash API Library documentation in [C2000Ware](#) for information on how to use the symbol library.



### 2.1.4 CPU Vector Table

A CPU vector table, [Figure 2-2](#), resides in boot ROM memory from address 0x3F E000 - 0x3F FFFF. This vector table is active after reset when VMAP = 1, ENPIE = 0 (PIE vector table disabled).



- The VMAP bit is located in Status Register 1 (ST1). VMAP is always 1 on reset. It can be changed after reset by software, however the normal operating mode will be to leave VMAP = 1.
- The ENPIE bit is located in the PIECTRL register. The default state of this bit at reset is 0, which disables the Peripheral Interrupt Expansion block (PIE).

**Figure 2-2. Vector Table Map**

The only vector that will normally be handled from the internal boot ROM memory is the reset vector located at 0x3F FFC0. The reset vector is factory programmed to point to the InitBoot function stored in the boot ROM. This function starts the bootload process. A series of checking operations is performed on  $\overline{\text{TRST}}$  and select general-purpose I/O (GPIO) pins to determine which boot mode to use. This boot mode selection is described in [Section 2.2.9](#) of this document.

The remaining vectors in the boot ROM are not used during normal operation. After the boot process is complete, you should initialize the Peripheral Interrupt Expansion (PIE) vector table and enable the PIE block. From that point on, all vectors, except reset, will be fetched from the PIE module and not the CPU vector table shown in [Table 2-1](#).

For TI silicon debug and test purposes the vectors located in the boot ROM memory point to locations in the M0 SARAM block as shown in [Table 2-1](#). During silicon debug, you can program the specified locations in M0 with branch instructions to catch any vectors fetched from boot ROM. This is not required for normal device operation.

**Table 2-1. Vector Locations**

Vector	Location in Boot ROM	Contents (that is, points to)	Vector	Location in Boot ROM	Contents (that is, points to)
RESET	0x3F FFC0	InitBoot	RTOSINT	0x3F FFE0	0x00 0060
INT1	0x3F FFC2	0x00 0042	Reserved	0x3F FFE2	0x00 0062
INT2	0x3F FFC4	0x00 0044	NMI	0x3F FFE4	0x00 0064
INT3	0x3F FFC6	0x00 0046	ILLEGAL	0x3F FFE6	ITRAPIsr
INT4	0x3F FFC8	0x00 0048	USER1	0x3F FFE8	0x00 0068
INT5	0x3F FFCA	0x00 004A	USER2	0x3F FFEA	0x00 006A
INT6	0x3F FFCC	0x00 004C	USER3	0x3F FFEC	0x00 006C
INT7	0x3F FFCE	0x00 004E	USER4	0x3F FFEE	0x00 006E
INT8	0x3F FFD0	0x00 0050	USER5	0x3F FFF0	0x00 0070
INT9	0x3F FFD2	0x00 0052	USER6	0x3F FFF2	0x00 0072
INT10	0x3F FFD4	0x00 0054	USER7	0x3F FFF4	0x00 0074
INT11	0x3F FFD6	0x00 0056	USER8	0x3F FFF6	0x00 0076
INT12	0x3F FFD8	0x00 0058	USER9	0x3F FFF8	0x00 0078
INT13	0x3F FFDA	0x00 005A	USER10	0x3F FFFA	0x00 007A
INT14	0x3F FFDC	0x00 005C	USER11	0x3F FFFC	0x00 007C
DLOGINT	0x3F FFDE	0x00 005E	USER12	0x3F FFFE	0x00 007E

## 2.2 Bootloader Features

This section describes in detail the boot mode selection process, as well as the specifics of the bootloader operation.

### 2.2.1 Bootloader Functional Operation

The bootloader is the program located in the on-chip boot ROM that is executed following a reset.

The bootloader provides a variety of different ways to download code to accommodate different system requirements. The bootloader uses the state of  $\overline{\text{TRST}}$  and two GPIO signals to determine which boot mode to use. The boot mode selection process and the specifics of each bootloader are described in the remainder of this document. [Figure 2-3](#) shows the basic bootloader flow:

The reset vector in boot ROM redirects program execution to the InitBoot function. After performing device initialization the bootloader will check the state of the  $\overline{\text{TRST}}$  pin to determine if an emulation pod is connected.

- **Debugger Boot (Debugger is connected and  $\overline{\text{TRST}} = 1$ )**

In debugger boot, the boot ROM will check two SARAM locations called EMU\_KEY and EMU\_BMODE for a boot mode. If the contents of either location are invalid, then the "wait" boot mode is used. All boot mode options can be accessed by modifying the value of EMU\_BMODE through the debugger when performing an debugger boot.

- **Standalone Boot (  $\overline{\text{TRST}} = 0$ )**

If the device is in standalone boot mode, then the state of two GPIO pins is used to determine which boot mode will execute. Options include: GetMode, wait, SCI, and parallel I/O. Each of the modes is described in [Section 2.2.9](#). The GetMode option by default boots to flash but can be customized by programming two values into OTP to select another boot loader.

After the selection process and if the required bootloading is complete, the processor will continue execution at an entry point determined by the boot mode selected. If a bootloader was called, then the input stream loaded by the peripheral determines this entry address. This data stream is described in [Section 2.2.11](#). If, instead, you choose to boot directly to Flash, OTP, or SARAM, the entry address is predefined for each of these memory blocks.

The following sections discuss in detail the different boot modes available and the process used for loading data code into the device.

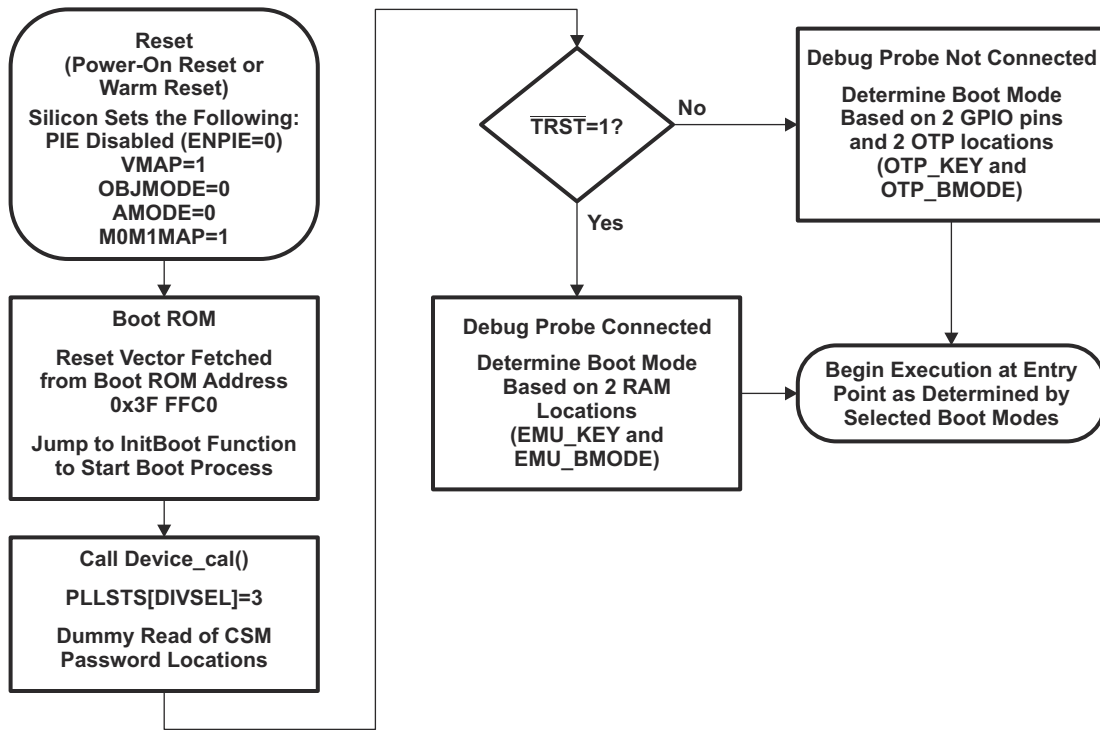


Figure 2-3. Bootloader Flow Diagram

## 2.2.2 Bootloader Device Configuration

At reset, the device is in C27x object-compatible mode. It is up to the application to place the device in the proper operating mode before execution proceeds.

When booting from the internal boot ROM, the device is configured for C28x operating mode by the boot ROM software. You are responsible for any additional configuration required.

For example, if your application includes C2xLP device source, then you are responsible for configuring the device for C2xLP source compatibility prior to execution of code generated from C2xLP source.

The configuration required for each operating mode is summarized in [Table 2-2](#).

**Table 2-2. Configuration for Device Modes**

	C27x Mode (Reset) <sup>(2)</sup>	C28x Mode	C2xLP Source Compatible Mode <sup>(2)</sup>
OBJMODE	0	1	1
AMODE	0	0	1
PAGE0	0	0	0
M0M1MAP <sup>(1)</sup>	1	1	1
Other Settings			SXM = 1, C = 1, SPM = 0

(1) Normally for C27x compatibility, the M0M1MAP would be 0. On these devices, however, it is tied off high internally; therefore, at reset, M0M1MAP is always configured for C28x mode.

(2) C27x refers to the TMS320C27x family of processors. C2xLP refers to the TMS320F24x/TMS320LF240xA family of devices that incorporate the C2xLP core. The information in the table above is for reference only and is not applicable for the typical user development. For more information on the C2xLP core, refer to the [TMS320C28x DSP CPU and Instruction Set Reference Guide](#).

## 2.2.3 PLL Multiplier and DIVSEL Selection

The Boot ROM changes the PLL multiplier (PLLCR) and divider (PLLSTS[DIVSEL]) bits as follows:

- **All boot modes:**
  - PLLCR is not modified. PLLSTS[DIVSEL] is set to 3 for SYSCLKOUT = CLKIN/1 . This increases the speed of the loaders.

### Note

The PLL multiplier (PLLCR) and divider (PLLSTS[DIVSEL]) are not affected by a reset from the debugger. Therefore, a boot that is initialized from a reset from the Code Composer Studio™ IDE may be at a different speed than booting by pulling the external reset line ( $\overline{XRS}$ ) low.

The reset value of PLLSTS[DIVSEL] is 0. This configures the device for SYSCLKOUT = CLKIN/4 . The boot ROM will change this to SYSCLKOUT = CLKIN/1 to improve performance of the loaders. PLLSTS[DIVSEL] is left in this state when the boot ROM exits and it is up to the application to change it before configuring the PLLCR register.

### 2.2.4 Watchdog Module

When branching directly to Flash, OTP, or M0 single-access RAM (SARAM) the watchdog is not touched. In the other boot modes, the watchdog is disabled before booting and then re-enabled and cleared before branching to the final destination address. In the case of an incorrect key value passed to the loader, the watchdog will be enabled and the device will boot to flash.

### 2.2.5 Taking an ITRAP Interrupt

If an illegal opcode is fetched, the device will take an ITRAP (illegal trap) interrupt. During the boot process, the interrupt vector used by the ITRAP is within the CPU vector table of the boot ROM. The ITRAP vector points to an interrupt service routine (ISR) within the boot ROM named ITRAPISR(). This interrupt service routine attempts to enable the watchdog and then loops forever until the processor is reset. This ISR will be used for any ITRAP until the user's application initializes and enables the peripheral interrupt expansion (PIE) block. Once the PIE is enabled, the ITRAP vector located within the PIE vector table will be used.

### 2.2.6 Internal Pullup Circuit

Each GPIO pin has an internal pullup circuit that can be enabled or disabled in software. The pins that are read by the boot mode selection code to determine the boot mode selection have pull-ups enabled after reset by default. In noisy conditions it is still recommended to configure each of the boot mode selection pins externally.

The peripheral bootloaders all enable the pullup circuit for the pins that are used for control and data transfer. The bootloader leaves the circuit enabled for these pins when it exits. For example, the SCI-A bootloader enables the pullup circuit on the SCITXA and SCIRXA pins. It is the user's responsibility to disable them, if desired, after the bootloader exits.

### 2.2.7 PIE Configuration

The boot modes do not enable the PIE. It is left in its default state, which is disabled.

The boot ROM does, however, use the first six locations within the PIE vector table for emulation boot mode information and Flash API variables. These locations are not used by the PIE itself and not used by typical applications.

---

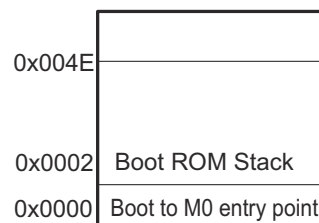
**Note**

If you are porting code from another 28x processor, check to see if the code initializes the first 6 locations in the PIE vector table to some default value. If it does, consider modifying the code to not write to these locations so the EMU boot mode will not be overwritten during debug.

---

### 2.2.8 Reserved Memory

The M0 memory block address range 0x0002 - 0x004E is reserved for the stack and .ebss code sections during the boot-load process. If code is bootloaded into this region there is no error checking to prevent it from corrupting the boot ROM stack. Address 0x0000-0x0001 is the boot to M0 entry point. This should be loaded with a branch instruction to the start of the main application when using "boot to SARAM" mode.



Boot ROM loaders on older C28x devices had the stack in M1 memory.

**Figure 2-4. Boot ROM Stack**

---

**Note**

If code or data is bootloaded into the address range address range 0x0002 - 0x004E, there is no error checking to prevent it from corrupting the boot ROM stack.

---

In addition, the first six locations of the PIE vector table are used by the boot ROM. These locations are not used by the PIE itself and not used by typical applications. These locations are used as SARAM by the boot ROM and will not affect the behavior of the PIE.

---

**Note**

Some example code from previous devices may initialize these locations. This will overwrite any boot mode you have populated. These locations are listed in [Table 2-3](#).

---

**Table 2-3. PIE Vector SARAM Locations Used by the Boot ROM**

Location	Name	Note
0x0D00 x 16	EMU_KEY	Used for emulation boot
0x0D01 x 16	EMU_BMODE	Used for emulation boot
0x0D02 x 32	Flash_CallbackPtr	Used by the flash API
0x0D04 x 32	Flash_CPUScaleFactor	Used by the flash API

## 2.2.9 Bootloader Modes

To accommodate different system requirements, the boot ROM offers a variety of boot modes. This section describes the different boot modes and gives a brief summary of their functional operation. The states of  $\overline{\text{TRST}}$  and two GPIO pins are used to determine the desired boot mode as shown in [Table 2-4](#) and [Figure 2-5](#).

**Table 2-4. Boot Mode Selection**

	GPIO37/TDO	GPIO34/ CMP2OUT	$\overline{\text{TRST}}$	
Mode EMU	x	x	1	Emulation Boot
Mode 0	0	0	0	Parallel I/O
Mode 1	0	1	0	SCI
Mode 2	1	0	0	Wait
Mode 3	1	1	0	GetMode

---

**Note**

The default behavior of the GetMode option on unprogrammed devices is to boot to flash. This behavior can be changed by programming two locations in the OTP as shown in [Table 2-6](#). In addition, if these locations are used by an application, then GetMode will jump to flash as long as `OTP_KEY != 0x55AA` and/or `OTP_BMODE` is not a valid value.

This device does not support the hardware wait-in-reset mode that is available on other C2000 parts. The "wait" boot mode can be used to emulate a wait-in-reset mode. The "wait" mode is very important for debugging devices with the CSM password programmed (that is, secured). When the device is powered up, the CPU will start running and may execute an instruction that performs an access to an emulation code security logic (ECSL) protected area. If this happens, the ECSL will trip and cause the JTAG debug probe connection to be broken. The "wait" mode keeps this from happening by looping within the boot ROM until a JTAG debug probe is connected.

---

[Figure 2-5](#) shows an overview of the boot process. Each step is described in greater detail in following sections.

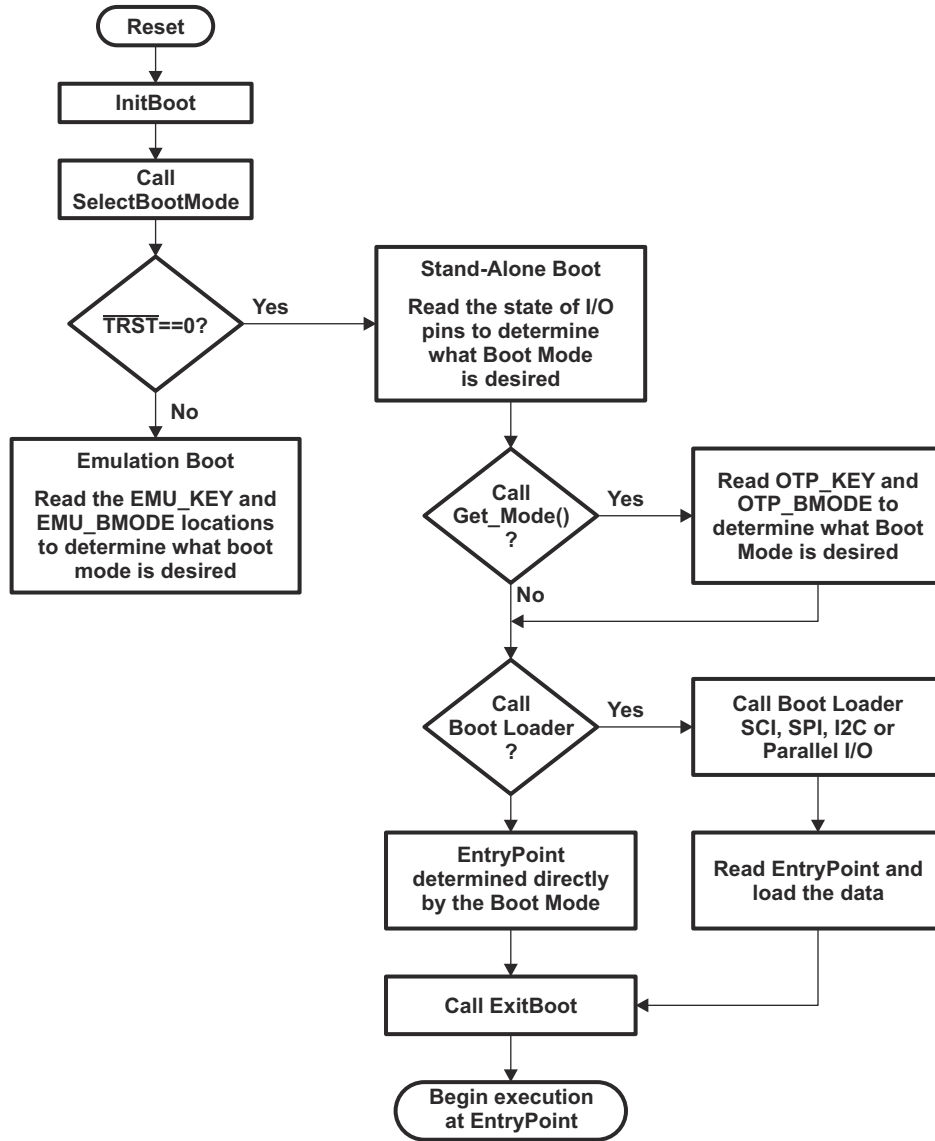


Figure 2-5. Boot ROM Function Overview

The following boot mode is used when an JTAG debug probe is connected:

- **Emulation Boot**

In this case, an emulation pod is connected to the device ( $\overline{\text{TRST}} = 1$ ) and the boot ROM derives the boot mode from the first two locations in the PIE vector table. These locations, called EMU\_KEY and EMU\_BMODE, are not used by the PIE module and not typically used by applications. Valid values for EMU\_KEY and EMU\_BMODE are shown in [Table 2-5](#).

An EMU\_KEY value of 0x55AA indicates the EMU\_BMODE is valid. An invalid key or invalid mode will result in calling the wait boot mode. EMU\_BMODE and EMU\_KEY are automatically populated by the boot ROM when powering up with  $\overline{\text{TRST}} = 0$ . EMU\_BMODE can also be initialized manually through the debugger.

**Table 2-5. Valid EMU\_KEY and EMU\_BMODE Values**

Address	Name	Value
0x0D00	EMU_KEY	if TRST == 1 and EMU_KEY == 0x55AA, then check EMU_BMODE for the boot mode, else { Invalid EMU_KEY Boot mode = WAIT_BOOT }
0x0D01	EMU_BMODE	0x0000 Boot mode = PARALLEL_BOOT 0x0001 Boot mode = SCI_BOOT 0x0002 Boot mode = WAIT_BOOT 0x0003 Boot mode = GET_BOOT (GetMode from OTP_KEY/OTP_BMODE) 0x0004 Boot mode = SPI_BOOT 0x0005 Boot mode = I2C_BOOT <sup>(1)</sup> 0x0006 Boot mode = OTP_BOOT 0x000A Boot mode = RAM_BOOT 0x000B Boot mode = FLASH_BOOT Other Boot mode = WAIT_BOOT

(1) I2C boot uses GPIO32 and GPIO33, which are not available on all packages.

[Table 2-7](#) shows the expanded emulation boot mode table.

Following are two examples of an emulation boot.

### 2.2.9.1 Example 1: Debug an application that loads through the SCI at boot.

To debug an application that loads through the SCI at boot, follow these steps:

- Configure the pins for mode 1, SCI, and initiate a power-on-reset.
- The boot ROM will detect  $\overline{\text{TRST}} = 0$  and will use the two pins to determine SCI boot.
- The boot ROM populates EMU\_KEY with 0x55AA and EMU\_BMODE with SCI\_BOOT.
- The boot ROM waits in the SCI loader for data.
- Connect the debugger.  $\overline{\text{TRST}}$  will go high.
- Perform a debugger reset and run. The boot loader will use the EMU\_BMODE and boot to SCI.



### 2.2.9.2 Example 2: You want to connect your JTAG debug probe, but do not want application code to start executing before the JTAG debug probe connects.

To connect your JTAG debug probe, but keep application code from executing before the JTAG debug probe connects:

- Configure GPIO37 and GPIO34 pins for mode 2, WAIT, and initiate a power-on-reset.
- The boot ROM will detect  $\overline{\text{TRST}} = 0$  and will use the two pins to determine wait boot.
- The boot ROM populates EMU\_KEY with 0x55AA and EMU\_BMODE with WAIT\_BOOT.
- The boot ROM waits in the wait routine.
- Connect the debugger;  $\overline{\text{TRST}}$  will go high.
- Modify the EMU\_BMODE via the debugger to boot to FLASH or other desired boot mode.
- Perform a debugger reset and run. The boot loader will use the EMU\_BMODE and boot to the desired loader or location.

---

#### Note

**The behavior of JTAG debug probes with regards to  $\overline{\text{TRST}}$  differs.** Some JTAG debug probes pull  $\overline{\text{TRST}}$  high only when Code Composer Studio is in a connected state. For these JTAG debug probes, if CCS is disconnected  $\overline{\text{TRST}}$  will return to a low state. With CCS disconnected, GPIO34 and GPIO37 will be used to determine the boot mode. For these JTAG debug probes, this is true even if the JTAG debug probe pod is physically connected.

Some JTAG debug probes pull  $\overline{\text{TRST}}$  high when CCS connects and leave it high as long as the power sense pin is active.  $\overline{\text{TRST}}$  will remain high even after CCS disconnects. For these JTAG debug probes, the EMU mode stored in RAM will be used unless the target is power cycled, causing the state of  $\overline{\text{TRST}}$  to reset back to a low state.

---

The following boot modes are invoked by the state of the boot mode pins if an JTAG debug probe is not connected:

- **Wait**

This device does not support the hardware wait-in-reset mode that is available on other C2000 parts. The "wait" boot mode can be used to emulate a wait-in-reset mode. The "wait" mode is very important for debugging devices with the CSM password programmed (that is, secured). When the device is powered up, the CPU will start running and may execute an instruction that performs an access to a emulation code security logic (ECSL) protected area. If this happens, the ECSL will trip and cause the JTAG debug probe connection to be broken. The "wait" mode keeps this from happening by looping within the boot ROM until a JTAG debug probe is connected

This mode writes WAIT\_BOOT to EMU\_BMODE. Once the JTAG debug probe is connected you can then manually populate the EMU\_BMODE with the appropriate boot mode for the debug session.

- **SCI**

In this mode, the boot ROM will load code to be executed into on-chip memory via the SCI-A port. When invoked as a stand-alone mode, the boot ROM writes SCI\_BOOT to EMU\_BMODE.

- **Parallel I/O 8-bit**

The parallel I/O boot mode is typically used only by production flash programmers.

- **GetMode**

The GetMode option uses two locations within the OTP to determine the boot mode. On an unprogrammed device this mode will always boot to flash. On a programmed device, you can choose to program these locations to change the behavior. If either of these locations is not an expected value, then boot to flash will be used.

The values used by the Get\_Mode() function are shown in [Table 2-6](#).

**Table 2-6. OTP Values for GetMode**

Address	Name	Value
0x3D 7BFE	OTP_KEY	GetMode will be entered if one of the two conditions is true: Case 1: $\overline{\text{TRST}} == 0$ , GPIO34 == 1 and GPIO37 == 1 Case 2: $\overline{\text{TRST}} == 1$ , EMU_KEY == 0x55AA and EMU_BMODE == GET_BOOT GetMode first checks the value of OTP_KEY: if OTP_KEY == 0x55AA, then check OTP_BMODE for the boot mode else { Invalid key: Boot mode = FLASH_BOOT }
0x3D 7BFF	OTP_BMODE	0x0001      Boot mode = SCI_BOOT 0x0004      Boot mode = SPI_BOOT 0x0005      Boot mode = I2C_BOOT <sup>(1)</sup> 0x0006      Boot mode = OTP_BOOT Other        Boot mode = FLASH_BOOT

(1) The I2C boot loader uses GPIO32 and GPIO33, which are not available on all packages.

The following boot modes are available through the emulation boot option. Some are also available as a programmed get mode option.

- **Jump to M0 SARAM**

This mode is only available in emulation boot. The boot ROM software configures the device for 28x operation and branches directly to address 0x000000. This is the first address in the M0 memory block.

- **Jump to branch instruction in Flash memory.**

Jump to Flash is the default behavior of the Get Mode boot option. Jump to flash is also available as an emulation boot option.

In this mode, the boot ROM software configures the device for 28x operation and branches directly to location 0x3F 7FF6. This location is just before the 128-bit code security module (CSM) password locations. You are required to have previously programmed a branch instruction at location 0x3F 7FF6 that will redirect code execution to either a custom boot-loader or the application code.

- **SPI EEPROM or Flash boot mode (SPI-A)**

Jump to SPI is available in stand-alone mode as a programmed Get Mode option. That is, to configure a device for SPI boot in stand-alone mode, the OTP\_KEY and OTP\_BMODE locations must be programmed for SPI\_BOOT and the boot mode pins configured for the Get Mode boot option.

SPI boot is also available as an emulation boot option.

In this mode, the boot ROM will load code and data into on-chip memory from an external SPI EEPROM or SPI flash via the SPI-A port.

- **I2C-A boot mode (I2C-A)**

Jump to I2C is available in stand-alone mode as a programmed Get mode option. That is, to configure a device for I2C boot in stand-alone mode, the OTP\_KEY and OTP\_BMODE locations must be programmed for I2C\_BOOT and the boot mode pins configured for the Get Mode boot option.

I2C boot is also available as an emulation boot option.

In this mode, the boot ROM will load code and data into on-chip memory from an external serial EEPROM or flash at address 0x50 on the I2C-A bus.

**Table 2-7. Emulation Boot Modes (TRST = 1)**

TRST	GPIO37/ TDO	GPIO34	EMU KEY	EMU BMODE	OTP KEY	OTP BMODE	Boot Mode Selected <sup>(1)</sup>	EMU KEY	EMU BMODE			
			Read from 0x0D00	Read from 0x0D01	Read from 0x3D7BFE	Read from 0x3D7BFF		Written to 0x0D00	Written to 0x0D01			
1	x <sup>(2)</sup>	x	!=0x55AA	x	x	x	Wait	-	-			
			0x55AA	0x0000	x	x	Parallel I/O	-	-			
				0x0001	x	x	SCI	-	-			
				0x0002	x	x	Wait	-	-			
				0x0003	!= 0x55AA	0X55AA	x	x	GetMode: Flash	-	-	
							0x0001	x	x	GetMode: SCI	-	-
							0x0003	x	x	GetMode: Flash	-	-
							0x0004	x	x	GetMode: SPI	-	-
							0x0005	x	x	GetMode: I2C <sup>(3)</sup>	-	-
							0x0006	x	x	GetMode: OTP	-	-
							Other	x	x	GetMode: Flash	-	-
				0x0004	x	x	SPI	-	-			
				0x0005	x	x	I2C <sup>(3)</sup>	-	-			
				0x0006	x	x	OTP	-	-			
				0x000A	x	x	Boot to RAM	-	-			
0x000B	x	x	Boot to FLASH	-	-							
Other	x	x	Wait	-	-							

- (1) Get Mode indicated the boot mode was derived from the values programmed in the OTP\_KEY and OTP\_BMODE locations.  
(2) x = don't care.  
(3) I2C uses GPIO32 and GPIO33, which are not available on all packages.

**Table 2-8. Standalone Boot Modes (TRST = 0)**

TRST	GPIO37/ TDO	GPIO34	EMU KEY	EMU BMODE	OTP KEY	OTP BMODE	Boot Mode Selected <sup>(1)</sup>	EMU KEY <sup>(2)</sup>	EMU BMODE <sup>(2)</sup>	
			Read from 0x0D00	Read from 0x0D01	Read from 0x3D7BFE	Read from 0x3D7BFF		Written to 0x0D00	Written to 0x0D01	
0	0	0	x <sup>(3)</sup>	x	x	x	Parallel I/O	0x55AA	0x0000	
0	0	1	x	x	x	x	SCI	0x55AA	0x0001	
0	1	0	x	x	x	x	Wait	0x55AA	0x0002	
0	1	1	x	x	!=0x55AA	0x55AA	x	0x55AA	0x0003	
							0x0001			GetMode: SCI
							0x0003			GetMode: Flash
							0x0004			GetMode: SPI
							0x0005			GetMode: I2C
							0x0006			GetMode: OTP
							Other			GetMode: Flash

- (1) Get Mode indicated the boot mode was derived from the values programmed in the OTP\_KEY and OTP\_BMODE locations.  
(2) The boot ROM will write this value to EMU\_KEY and EMU\_BMODE. This value can be used or overwritten by the user if a debugger is connected.  
(3) x = don't care.

### 2.2.10 Device\_Cal

The *Device\_cal()* routine is programmed into TI reserved memory by the factory. The boot ROM automatically calls the *Device\_cal()* routine to calibrate the internal oscillators and ADC with device specific calibration data. During normal operation, this process occurs automatically and no action is required by the user.

If the boot ROM is bypassed by Code Composer Studio™ IDE during the development process, then the calibration must be initialized by the application. For working examples, see the system initialization function *InitSysCtrl()*, in [C2000Ware](#).

---

#### Note

Failure to initialize these registers will cause the oscillators and ADC to function out of specification. The following three steps describe how to call the *Device\_cal* routine from an application.

---

Step 1: Create a pointer to the *Device\_cal()* function as shown in [Example 2-2](#). This #define is included in C2000Ware.

Step 2: Call the function pointed to by *Device\_cal* as shown in [Example 2-2](#). The ADC clocks must be enabled before making this call.

#### Example 2-2. Calling the *Device\_cal()* function

```
//Device call is a pointer to a function
//that begins at the address shown
# define Device_cal (void(*) (void)) 0x3D7C80
... ..
EALLOW;
SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
(*Device_cal) ();
SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0;
EDIS;
... ..
```

### 2.2.11 Bootloader Data Stream Structure

The basic structure is the same for all the bootloaders and is based on the C54x source data stream generated by the C54x hex utility. The C28x hex utility (hex2000.exe) has been updated to support this structure. The hex2000.exe utility is included with the C2000 code generation tools. All values in the data stream structure are in hex.

The first 16-bit word in the data stream is known as the key value, which is 0x08AA. If a bootloader receives an invalid key value, then the load is aborted.

The next eight words are used to initialize register values or otherwise enhance the bootloader by passing values to it. If a bootloader does not use these values then they are reserved for future use and the bootloader simply reads the value and then discards it. Only the SPI and I2C bootloaders use these words to initialize registers.

The tenth and eleventh words comprise the 22-bit entry point address. This address is used to initialize the PC after the boot load is complete. This address is most likely the entry point of the program downloaded by the bootloader.

The twelfth word in the data stream is the size of the first data block to be transferred. The size of the block is defined for 8-bit data stream format as the number of 16-bit words in the block. For example, to transfer a block of twenty 8-bit data values from an 8-bit data stream, the block size would be 0x000A to indicate ten 16-bit words.

The next two words indicate to the loader the destination address of the block of data. Following the size and address will be the 16-bit words that form that block of data.

This pattern of block size/destination address repeats for each block of data to be transferred. Once all the blocks have been transferred, a block size of 0x0000 signals to the loader that the transfer is complete. At this point the loader will return the entry point address to the calling routine which in turn will cleanup and exit. Execution will then continue at the entry point address as determined by the input data stream contents.

In 8-bit mode, the least significant byte (LSB) of the word is sent first followed by the most significant byte (MSB). For 32-bit values, such as a destination address, the most significant word (MSW) is loaded first, followed by the least significant word (LSW). The bootloaders take this into account when loading an 8-bit data stream. [Table 2-9](#) and [Example 2-3](#) show the structure of the incoming data stream to the bootloader.

**Table 2-9. LSB/MSB Loading Sequence in 8-bit Data Stream**

		Contents	
Byte		LSB (First Byte of 2)	MSB (Second Byte of 2)
1	2	LSB: AA (KeyValue for memory width = 8 bits)	MSB: 08h (KeyValue for memory width = 8 bits)
3	4	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
5	6	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
7	8	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
...	...	...	...
17	18	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
19	20	LSB: Upper half of Entry point PC[23:16]	MSB: Upper half of entry point PC[31:24] (Always 0x00)
21	22	LSB: Lower half of Entry point PC[7:0]	MSB: Lower half of Entry point PC[15:8]
23	24	LSB: Block size in words of the first block to load. If the block size is 0, this indicates the end of the source program. Otherwise another block follows. For example, a block size of 0x000A would indicate 10 words or 20 bytes in the block.	MSB: block size
25	26	LSB: MSW destination address, first block Addr[23:16]	MSB: MSW destination address, first block Addr[31:24]
27	28	LSB: LSW destination address, first block Addr[7:0]	MSB: LSW destination address, first block Addr[15:8]
29	30	LSB: First word of the first block being loaded	MSB: First word of the first block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the first block to load	MSB: Last word of the first block to load
.	.	LSB: Block size of the second block	MSB: Block size of the second block
.	.	LSB: MSW destination address, second block Addr[23:16]	MSB: MSW destination address, second block Addr[31:24]
.	.	LSB: LSW destination address, second block Addr[7:0]	MSB: LSW destination address, second block Addr[15:8]
.	.	LSB: First word of the second block being loaded	MSB: First word of the second block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the second block	MSB: Last word of the second block
.	.	LSB: Block size of the last block	MSB: Block size of the last block
.	.	LSB: MSW of destination address of last block Addr[23:16]	MSB: MSW destination address, last block Addr[31:24]
.	.	LSB: LSW destination address, last block Addr[7:0]	MSB: LSW destination address, last block Addr[15:8]
.	.	LSB: First word of the last block being loaded	MSB: First word of the last block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the last block	MSB: Last word of the last block
n	n+1	LSB: 00h	MSB: 00h - indicates the end of the source

**Example 2-3. Data Stream Structure 8-bit**

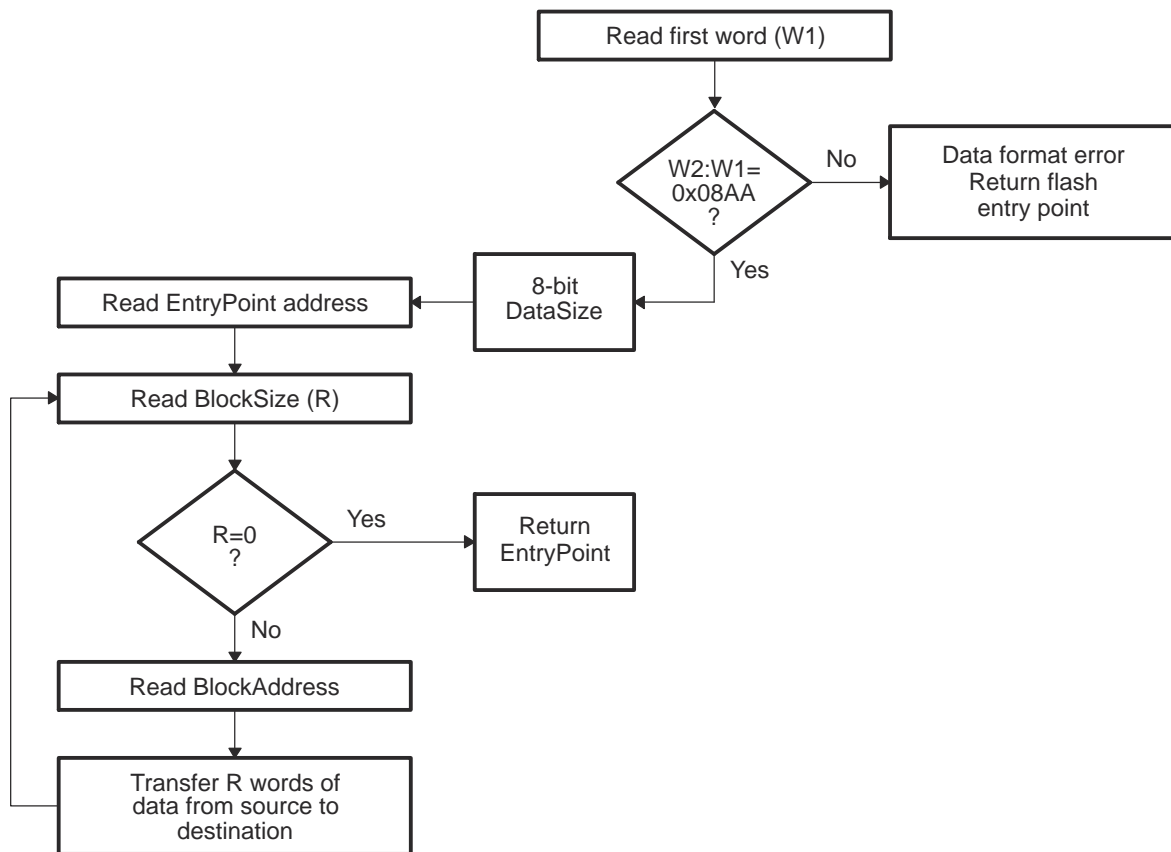
```

AA 08          ; 0x08AA 8-bit key value
00 00 00 00   ; 8 reserved words
00 00 00 00
00 00 00 00
00 00 00 00
3F 00 00 80   ; 0x003F8000 EntryAddr, starting point after boot load completes
05 00        ; 0x0005 - First block consists of 5 16-bit words
3F 00 10 90   ; 0x003F9010 - First block will be loaded starting at 0x3F9010
01 00        ; Data loaded = 0x0001 0x0002 0x0003 0x0004 0x0005
02 00
03 00
04 00
05 00
02 00        ; 0x0002 - 2nd block consists of 2 16-bit words
3F 00 00 80   ; 0x003F8000 - 2nd block will be loaded starting at 0x3F8000
00 77        ; Data loaded = 0x7700 0x7625
25 76
00 00        ; 0x0000 - Size of 0 indicates end of data stream
After load has completed the following memory values will have been initialized as follows:
Location      Value
0x3F9010     0x0001
0x3F9011     0x0002
0x3F9012     0x0003
0x3F9013     0x0004
0x3F9014     0x0005
0x3F8000     0x7700
0x3F8001     0x7625
PC Begins execution at 0x3F8000
    
```

### 2.2.12 Basic Transfer Procedure

Figure 2-6 illustrates the basic process a bootloader uses to transfer data and start program execution. This process occurs after the bootloader determines the valid boot mode selected by the state of the  $\overline{\text{TRST}}$  and GPIO pins.

The bootloader compares the first word sent by the host against the key value of 0x08AA. If the first word matches the key value, the loader continuously fetches data until it transfers all the hex words and then starts program execution. But, if the key does not match the key value, then the loader aborts and the CPU jumps to the default flash entry point.



**Figure 2-6. Bootloader Basic Transfer Procedure**

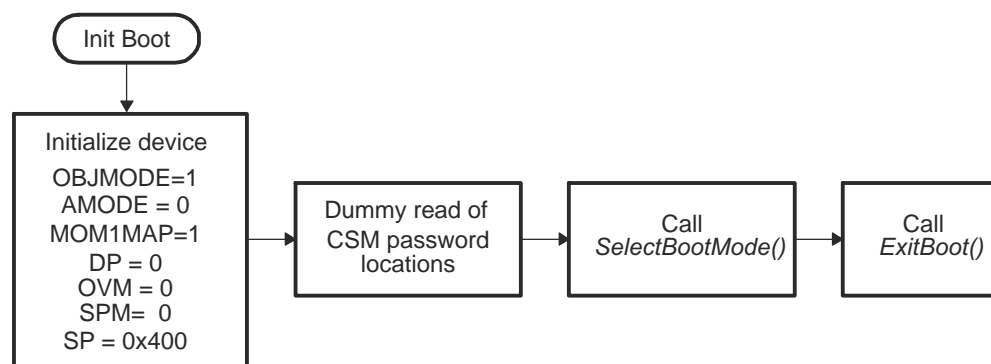
**Note**

See the info specific to a particular bootloader for any limitations. In 8-bit mode, the LSB of the 16-bit word is read first followed by the MSB.

### 2.2.13 InitBoot Assembly Routine

The first routine called after reset is the *InitBoot* assembly routine. This routine initializes the device for operation in C28x object mode. *InitBoot* also performs a dummy read of the Code Security Module (CSM) password locations. If the CSM passwords are erased (all 0xFFFFs) then this has the effect of unlocking the device. Otherwise the device will remain locked and this dummy read of the password locations will have no effect. This can be useful if you have a new device that you want to boot load.

After the dummy read of the CSM password locations, the *InitBoot* routine calls the *SelectBootMode* function. This function determines the type of boot mode desired by the state of  $\overline{\text{TRST}}$  and certain GPIO pins. This process is described in [Section 2.2.14](#). Once the boot is complete, the *SelectBootMode* function passes back the entry point address (*EntryAddr*) to the *InitBoot* function. The *EntryAddr* is the location where code execution will begin after the bootloader exits. *InitBoot* then calls the *ExitBoot* routine that then restores CPU registers to their reset state and exits to the *EntryAddr* that was determined by the boot mode.



**Figure 2-7. Overview of InitBoot Assembly Function**

### 2.2.14 SelectBootMode Function

To determine the desired boot mode, the *SelectBootMode* function examines the state of  $\overline{\text{TRST}}$  and 2 GPIO pins as shown in [Table 2-4](#).

For a boot mode to be selected, the pins corresponding to the desired boot mode have to be pulled low or high until the selection process completes. Note that the state of the selection pins is not latched at reset; they are sampled some cycles later in the *SelectBootMode* function. The internal pullup resistors are enabled at reset for the boot mode selection pins. It is still suggested that the boot mode configuration be made externally to avoid the effect of any noise on these pins.

---

#### Note

The *SelectBootMode* routine disables the watchdog before calling the SCI, I2C, SPI, or parallel bootloaders. The bootloaders do not service the watchdog and assume that it is disabled. Before exiting, the *SelectBootMode* routine will re-enable the watchdog and reset its timer.

If a bootloader is not going to be called, then the watchdog is left untouched.

---

When selecting a boot mode, the pins should be pulled low or high through a weak pulldown or weak pull-up such that the device can drive them to a new state when required.



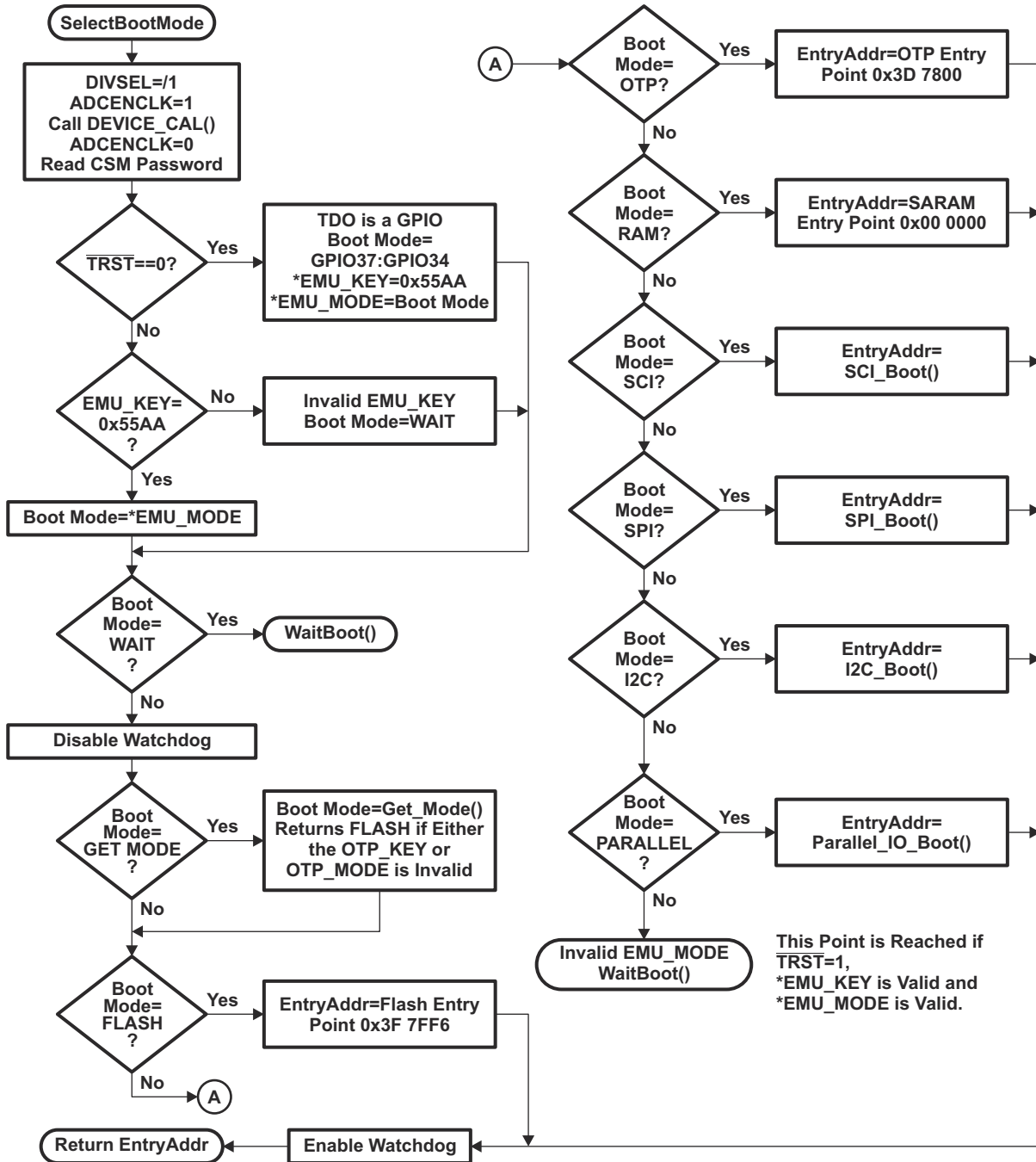


Figure 2-8. Overview of the SelectBootMode Function

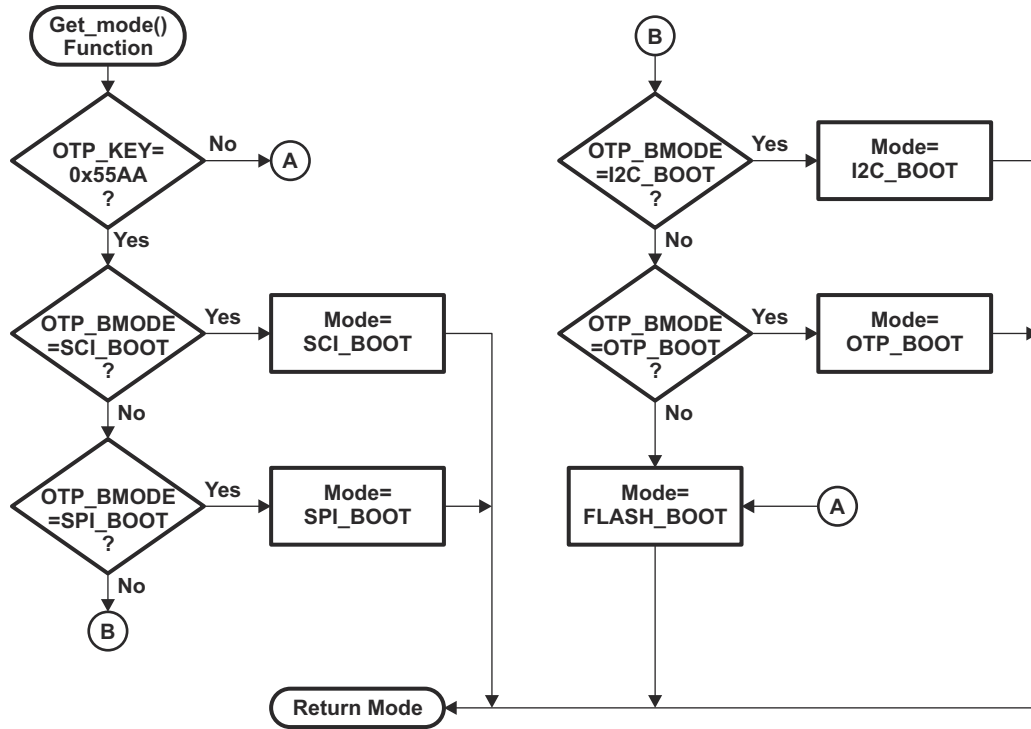


Figure 2-9. Overview of Get\_mode() Function

### 2.2.15 CopyData Function

All bootloaders use the same function to copy data from the port to the device's SARAM. This function is the *CopyData()* function. This function uses a pointer to a *GetWordData* function that is initialized by each of the loaders to properly read data from that port. For example, when the SPI loader is evoked, the *GetWordData* function pointer is initialized to point to the *SPI-specific SPI\_GetWordData* function. Thus when the *CopyData()* function is called, the correct port is accessed. The flow of the *CopyData* function is shown in [Figure 2-10](#).

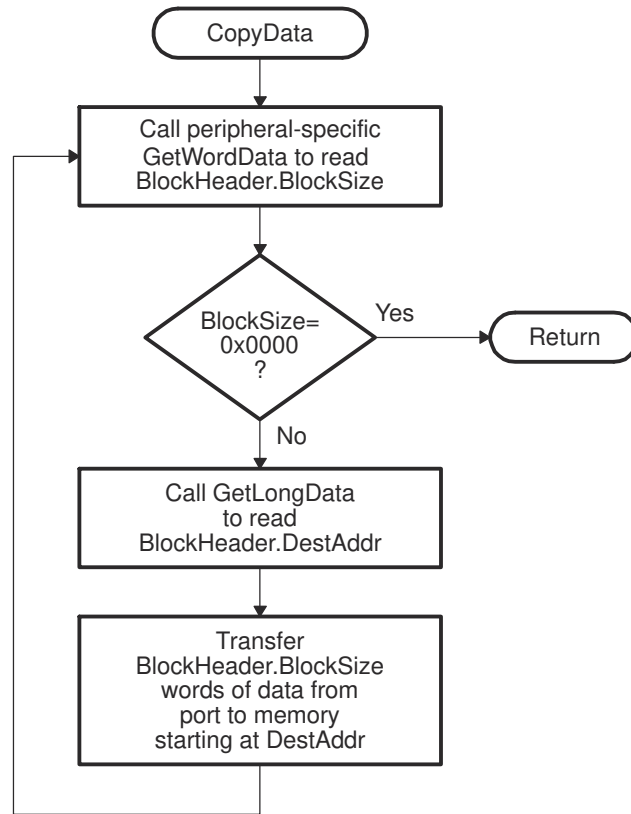


Figure 2-10. Overview of CopyData Function

### 2.2.16 SCI\_Boot Function

The SCI boot mode asynchronously transfers code from SCI-A to internal memory. This boot mode only supports an incoming 8-bit data stream and follows the same data flow as outlined in [Example 2-3](#).

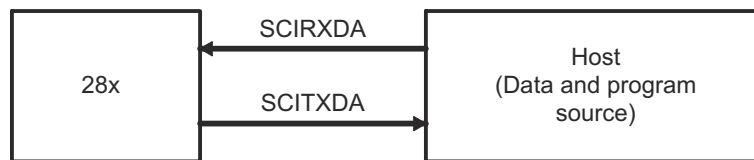


Figure 2-11. Overview of SCI Bootloader Operation

The SCI-A loader uses following pins:

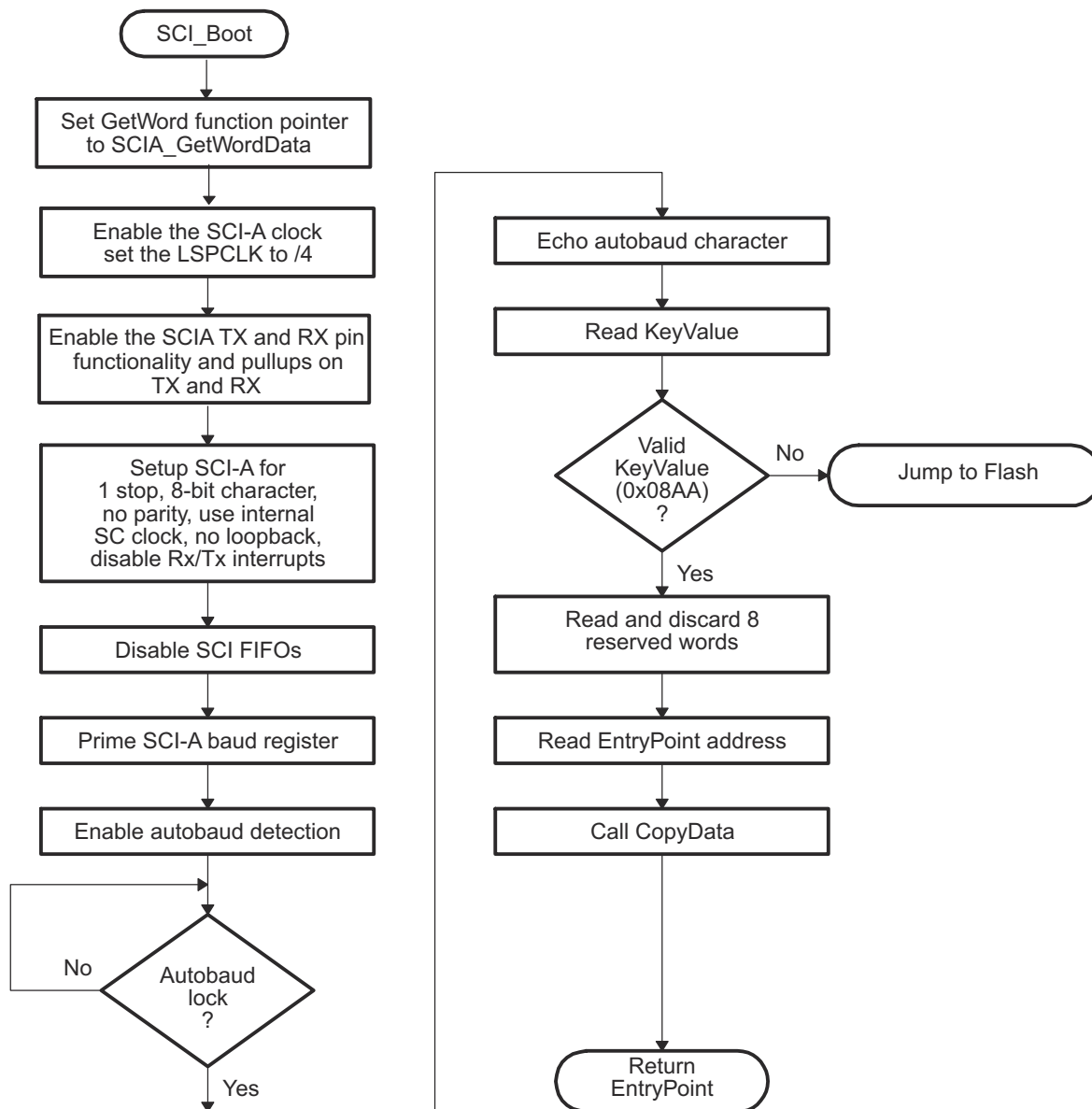
- SCIRXDA on GPIO28
- SCITXDA on GPIO29

The 28x device communicates with the external host device through the SCI-A peripheral. The autobaud feature of the SCI port is used to lock baud rates with the host. For this reason the SCI loader is very flexible and you can use a number of different baud rates to communicate with the device.

After each data transfer, the 28x will echo back the 8-bit character received to the host. In this manner, the host can perform checks that each character was received by the 28x.

At higher baud rates, the slew rate of the incoming data bits can be affected by transceiver and connector performance. While normal serial communications may work well, this slew rate may limit reliable auto-baud detection at higher baud rates (typically beyond 100kbaud) and cause the auto-baud lock feature to fail. To avoid this, the following is recommended:

1. Achieve a baud-lock between the host and 28x SCI bootloader using a lower baud rate.
2. Load the incoming 28x application or custom loader at this lower baud rate.
3. The host may then handshake with the loaded 28x application to set the SCI baud rate register to the desired high baud rate.



**Figure 2-12. Overview of SCI\_Boot Function**

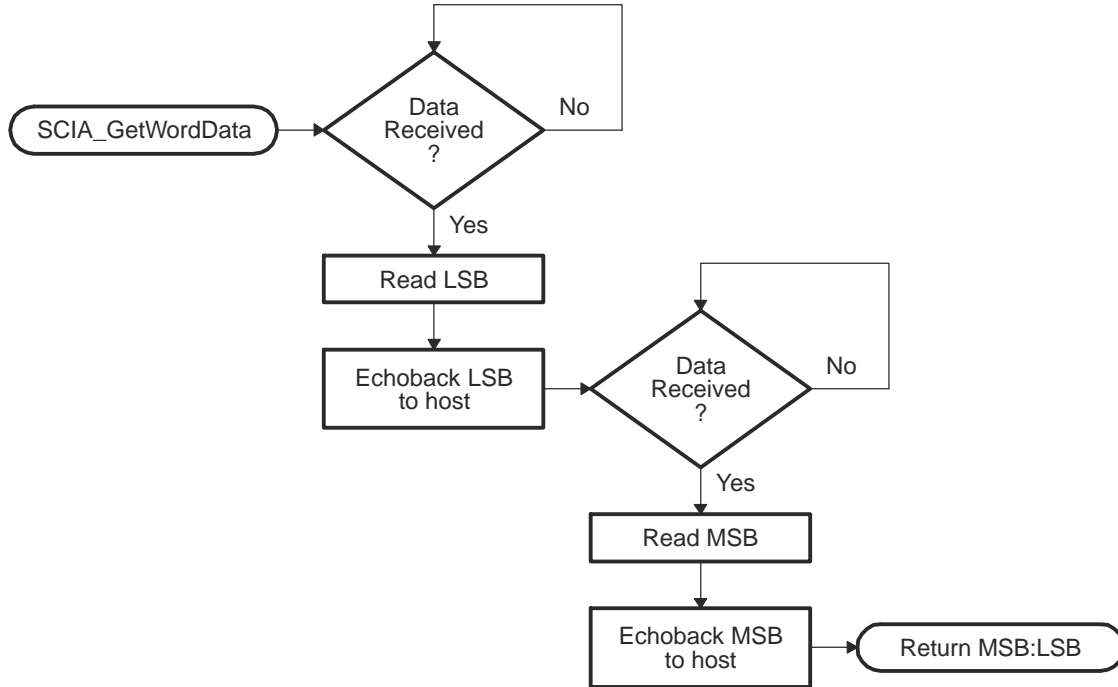


Figure 2-13. Overview of SCI\_GetWordData Function

### 2.2.17 Parallel\_Boot Function (GPIO)

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from GPIO0 -GPIO7 to internal memory. Each value is 8 bits long and follows the same data flow as outlined in Section 2.2.11.

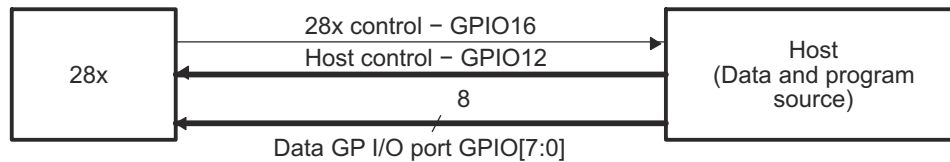


Figure 2-14. Overview of Parallel GPIO Bootloader Operation

The parallel GPIO loader uses following pins:

- Data on GPIO[7:0]
- 28x Control on GPIO16
- Host Control on GPIO12

The 28x communicates with the external host device by polling/driving the GPIO12 and GPIO16 lines. The handshake protocol shown in Figure 2-15 must be used to successfully transfer each word via GPIO [7:0]. This protocol is very robust and allows for a slower or faster host to communicate with the 28x .

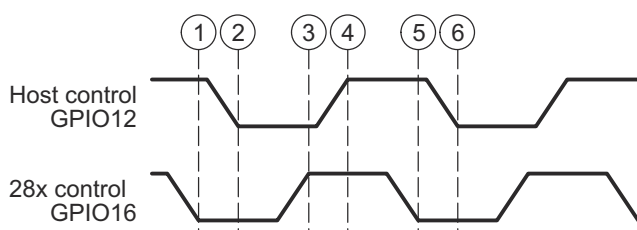
Two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of GPIO[7:0] ignoring the higher byte.

The 8-bit data stream is shown in Table 2-10.

**Table 2-10. Parallel GPIO Boot 8-Bit Data Stream**

Bytes	GPIO[7 :0] (Byte 1 of 2)	GPIO[7 :0] (Byte 2 of 2)	Description
1 2	AA	08	0x08AA (KeyValue for memory width = bits)
3 4	00	00	8 reserved words (words 2 - 9)
...	...	...	...
17 18	00	00	Last reserved word
19 20	BB	00	Entry point PC[22:16]
21 22	DD	CC	Entry point PC[15:0] (PC = 0x00BBCCDD)
23 24	NN	MM	Block size of the first block of data to load = 0xMMNN words
25 26	BB	AA	Destination address of first block Addr[31:16]
27 28	DD	CC	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
29 30	BB	AA	First word of the first block in the source being loaded = 0xAABB
...	...	...	...
...	...	...	Data for this section.
...	...	...	...
.	BB	AA	Last word of the first block of the source being loaded = 0xAABB
.	NN	MM	Block size of the 2nd block to load = 0xMMNN words
.	BB	AA	Destination address of second block Addr[31:16]
.	DD	CC	Destination address of second block Addr[15:0]
.	BB	AA	First word of the second block in the source being loaded
.	...	...	...
n n+1	BB	AA	Last word of the last block of the source being loaded (More sections if required)
n+2 n+3	00	00	Block size of 0000h - indicates end of the source program

The 28x device first signals the host that it is ready to begin data transfer by pulling the GPIO16 pin low. The host load then initiates the data transfer by pulling the GPIO12 pin low. The complete protocol is shown in [Figure 2-15](#).


**Figure 2-15. Parallel GPIO Boot Loader Handshake Protocol**

1. The 28x device indicates it is ready to start receiving data by pulling the GPIO16 pin low.
2. The bootloader waits until the host puts data on GPIO [7:0]. The host signals to the 28x device that data is ready by pulling the GPIO12 pin low.
3. The 28x device reads the data and signals the host that the read is complete by pulling GPIO16 high.
4. The bootloader waits until the host acknowledges the 28x device by pulling GPIO12 high.
5. The 28x device again indicates it is ready for more data by pulling the GPIO16 pin low.

This process is repeated for each data value to be sent.

[Figure 2-16](#) shows an overview of the Parallel GPIO bootloader flow.

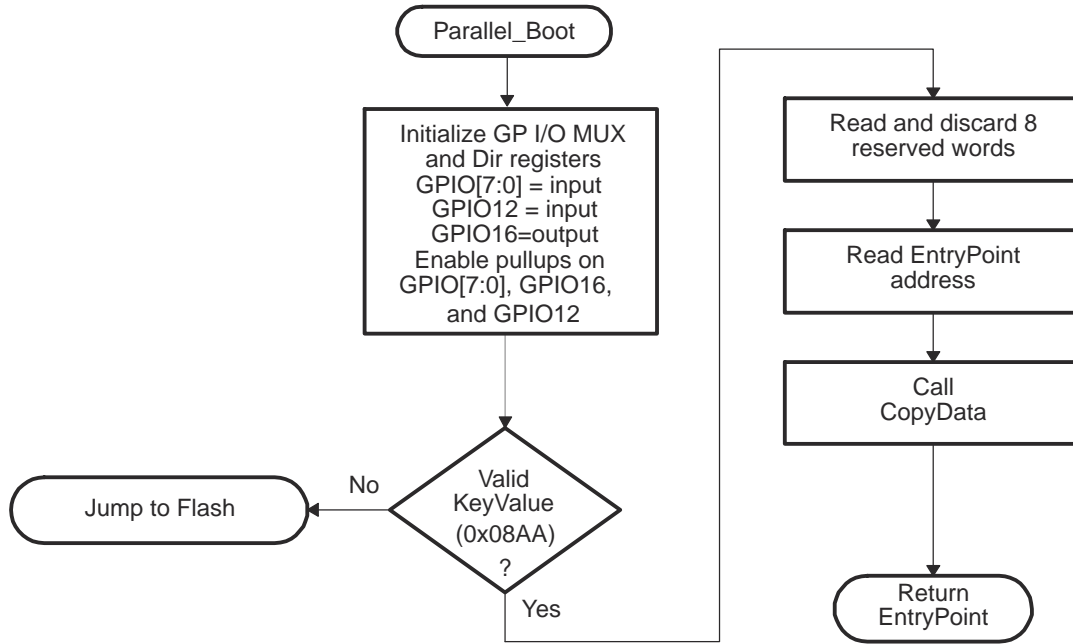
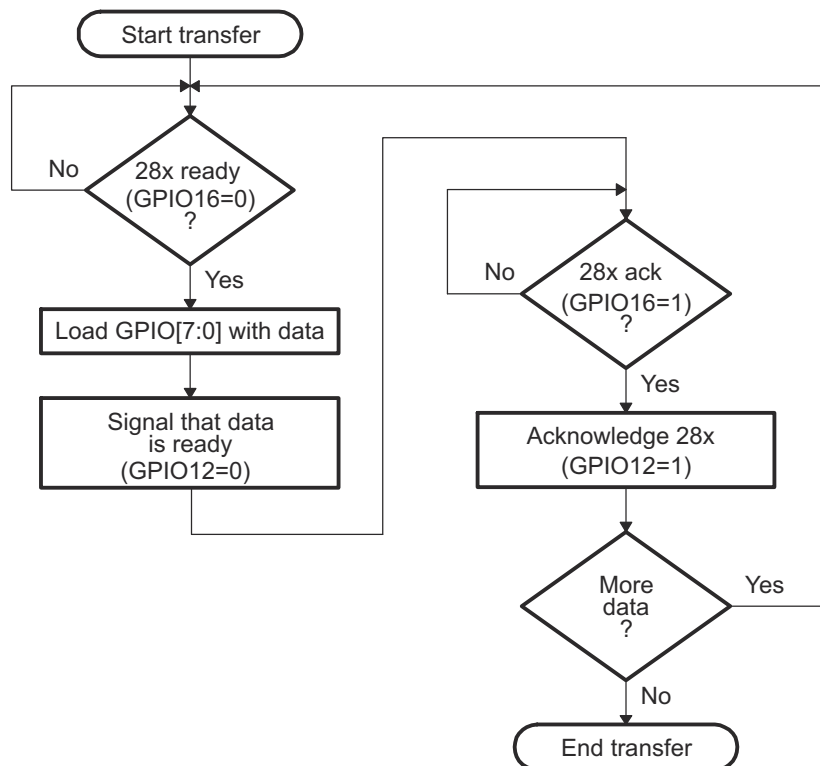


Figure 2-16. Parallel GPIO Mode Overview

Figure 2-17 shows the transfer flow from the host side. The operating speed of the CPU and host are not critical in this mode as the host will wait for the 28x device, and the 28x device will in turn wait for the host. In this manner the protocol will work with both a host running faster and a host running slower than the 28x device.



**Figure 2-17. Parallel GPIO Mode - Host Transfer Flow**

Figure 2-18 show the flow used to read a single word of data from the parallel port.

- **8-bit data stream**

The 8-bit routine, shown in Figure 2-18, discards the upper 8 bits of the first read from the port and treats the lower 8 bits as the least significant byte (LSB) of the word to be fetched. The routine will then perform a second read to fetch the most significant byte (MSB). It then combines the MSB and LSB into a single 16-bit value to be passed back to the calling routine.



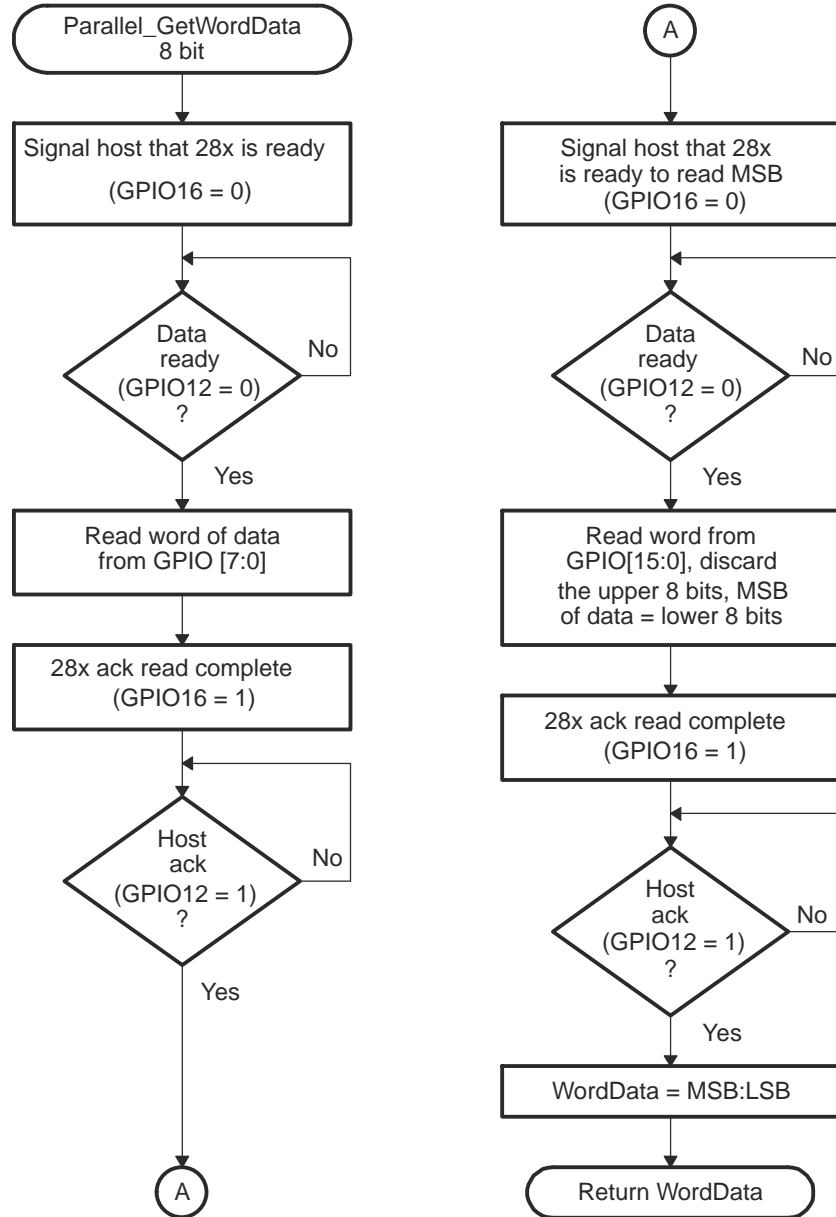
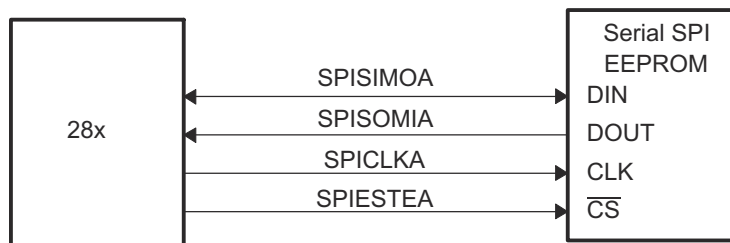


Figure 2-18. 8-Bit Parallel GetWord Function

## 2.2.18 SPI\_Boot Function

The SPI boot ROM loader initializes the SPI module to interface to a SPI-compatible, 16-bit or 24-bit addressable serial EEPROM or flash device.

It expects such a device to be present on the SPI-A pins as indicated in [Figure 2-19](#). The SPI bootloader supports an 8-bit data stream. It does not support a 16-bit data stream.



**Figure 2-19. SPI Loader**

The SPI-A loader uses following pins:

- SPISIMOA on GPIO16
- SPISOMIA on GPIO17
- SPICLKA on GPIO18
- SPISTEA on GPIO19

The SPI boot ROM loader initializes the SPI with the following settings: FIFO enabled, 8-bit character, internal SPICLK master mode and talk mode, clock phase = 1, polarity = 0, using the slowest baud rate.

If the download is to be performed from an SPI port on another device, then that device must be setup to operate in the slave mode and mimic a serial SPI EEPROM. Immediately after entering the SPI\_Boot function, the pin functions for the SPI pins are set to primary and the SPI is initialized. The initialization is done at the slowest speed possible. Once the SPI is initialized and the key value read, you could specify a change in baud rate or low speed peripheral clock.

**Table 2-11. SPI 8-Bit Data Stream**

Byte	Contents
1	LSB: AA (KeyValue for memory width = 8-bits)
2	MSB: 08h (KeyValue for memory width = 8-bits)
3	LSB: LOSPCP
4	MSB: SPIBRR
5	LSB: reserved for future use
6	MSB: reserved for future use
...	...
...	Data for this section.
...	...
17	LSB: reserved for future use
18	MSB: reserved for future use
19	LSB: Upper half (MSW) of Entry point PC[23:16]
20	MSB: Upper half (MSW) of Entry point PC[31:24] (Note: Always 0x00)
21	LSB: Lower half (LSW) of Entry point PC[7:0]
22	MSB: Lower half (LSW) of Entry point PC[15:8]
...	....
...	Data for this section.
...	...
...	Blocks of data in the format size/destination address/data as shown in the generic data stream description

**Table 2-11. SPI 8-Bit Data Stream (continued)**

Byte	Contents
...	...
...	Data for this section.
...	...
n	LSB: 00h
n+1	MSB: 00h - indicates the end of the source

The data transfer is done in "burst" mode from the serial SPI EEPROM. The transfer is carried out entirely in byte mode (SPI at 8 bits/character). A step-by-step description of the sequence follows:

1. The SPI-A port is initialized
2. The GPIO19 (SPISTE) pin is used as a chip-select for the serial SPI EEPROM or flash
3. The SPI-A outputs a read command for the serial SPI EEPROM or flash
4. The SPI-A sends the serial SPI EEPROM an address 0x0000; that is, the host requires that the EEPROM or flash must have the downloadable packet starting at address 0x0000 in the EEPROM or flash. The loader is compatible with both 16-bit addresses and 24-bit addresses.
5. The next word fetched must match the key value for an 8-bit data stream (0x08AA). The least significant byte of this word is the byte read first and the most significant byte is the next byte fetched. This is true of all word transfers on the SPI. If the key value does not match, then the load is aborted and the device will branch to the flash entry point address.
6. The next two bytes fetched can be used to change the value of the low speed peripheral clock register (LOSPCP) and the SPI baud rate register (SPIBRR). The first byte read is the LOSPCP value and the second byte read is the SPIBRR value. The next 7 words are reserved for future enhancements. The SPI bootloader reads these 7 words and discards them.
7. The next two words makeup the 32-bit entry point address where execution will continue after the boot load process is complete. This is typically the entry point for the program being downloaded through the SPI port.
8. Multiple blocks of code and data are then copied into memory from the external serial SPI EEPROM through the SPI port. The blocks of code are organized in the standard data stream structure presented earlier. This is done until a block size of 0x0000 is encountered. At that point in time the entry point address is returned to the calling routine that then exits the bootloader and resumes execution at the address specified.

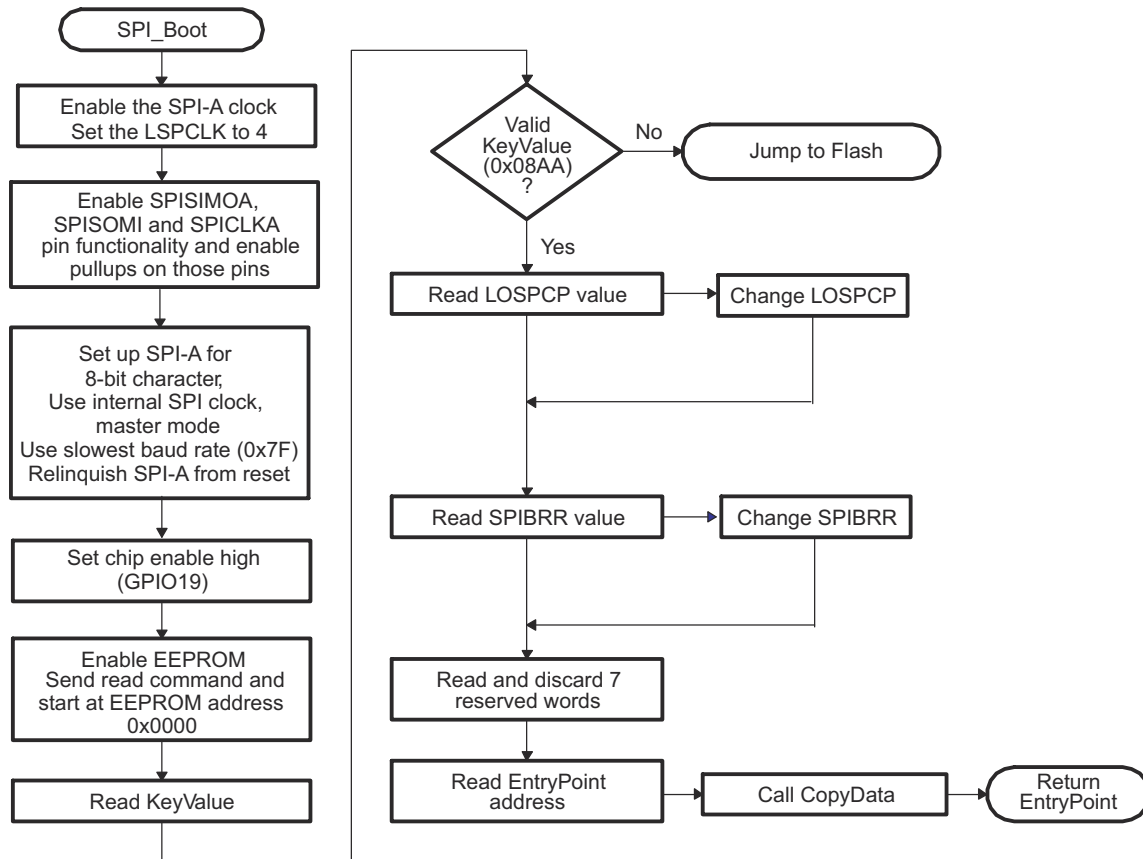


Figure 2-20. Data Transfer From EEPROM Flow

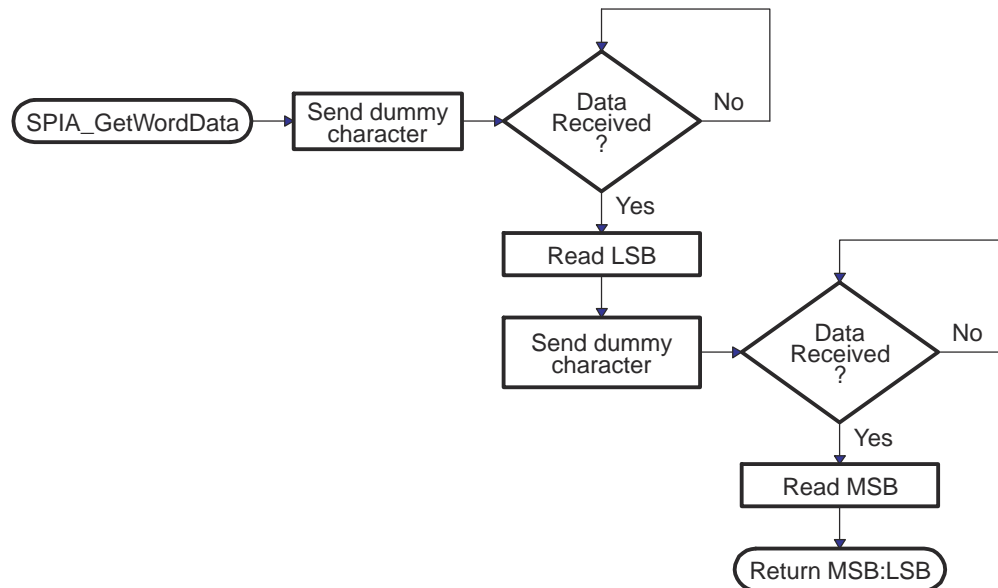
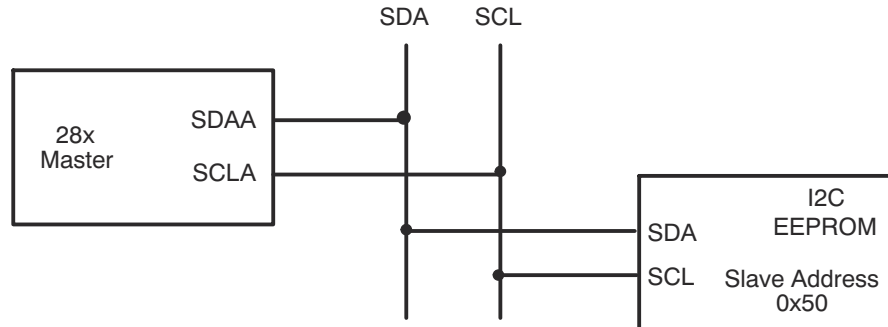


Figure 2-21. Overview of SPIA\_GetWordData Function

### 2.2.19 I2C Boot Function

The I2C bootloader expects an 8-bit wide I2C-compatible EEPROM device to be present at address 0x50 on the I2C-A bus as indicated in [Figure 2-22](#). The EEPROM must adhere to conventional I2C EEPROM protocol, as described in this section, with a 16-bit base address architecture.



**Figure 2-22. EEPROM Device at Address 0x50**

The I2C loader uses following pins:

- SDAA on GPIO 32
- SCLA on GPIO 33

If the download is to be performed from a device other than an EEPROM, then that device must be set up to operate in the slave mode and mimic the I2C EEPROM. Immediately after entering the I2C boot function, the GPIO pins are configured for I2C-A operation and the I2C is initialized. The following requirements must be met when booting from the I2C module:

- The input frequency to the device must be in the appropriate range.
- The EEPROM must be at slave address 0x50.

The bit-period prescalers (I2CCLKH and I2CCLKL) are configured by the bootloader to run the I2C at a 50 percent duty cycle at 100-kHz bit rate (standard I2C mode) when the system clock is 10 MHz. These registers can be modified after receiving the first few bytes from the EEPROM. This allows the communication to be increased up to a 400-kHz bit rate (fast I2C mode) during the remaining data reads.

Arbitration, bus busy, and slave signals are not checked. Therefore, no other master is allowed to control the bus during this initialization phase. If the application requires another master during I2C boot mode, that master must be configured to hold off sending any I2C messages until the application software signals that it is past the bootloader portion of initialization.

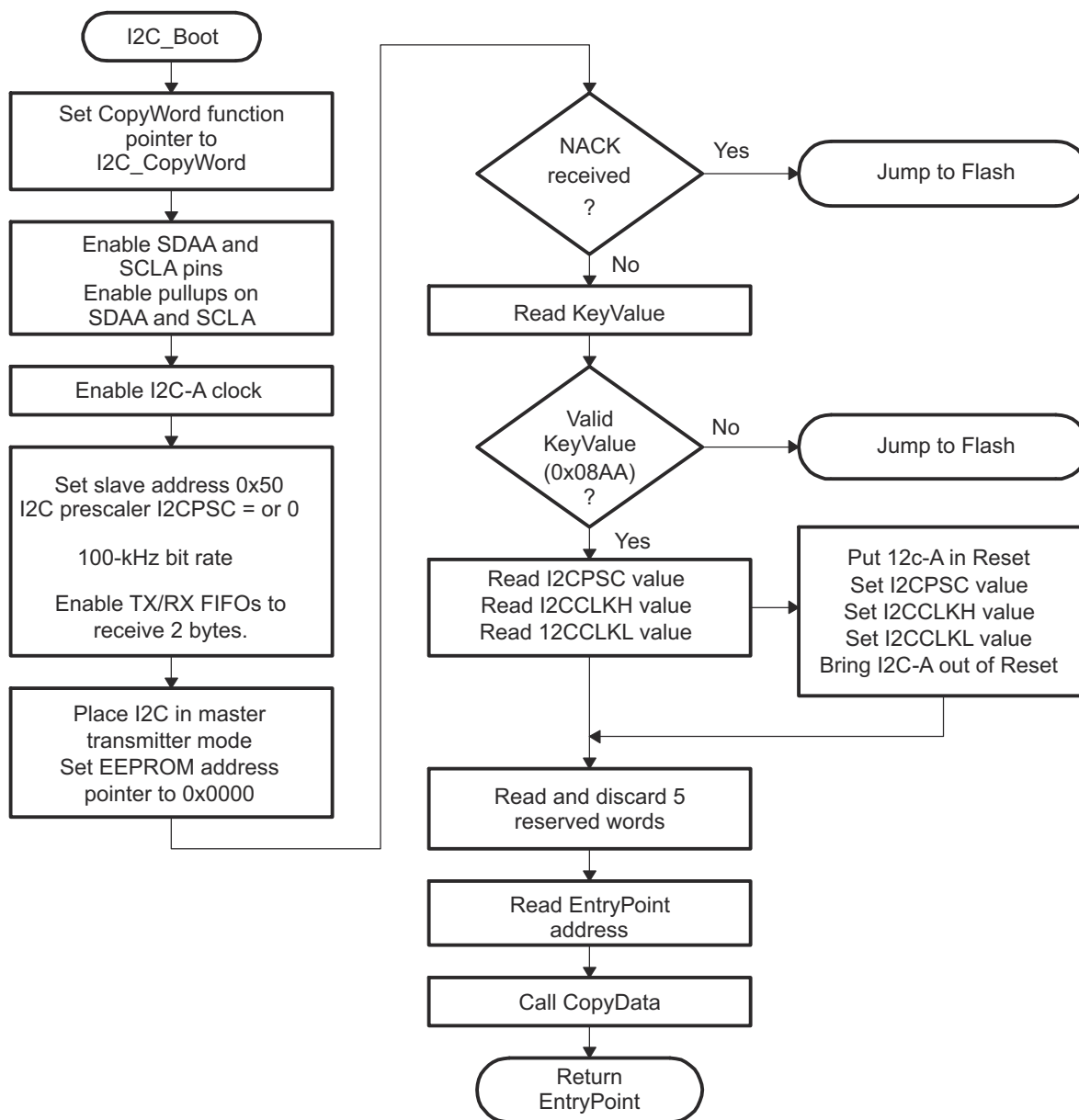


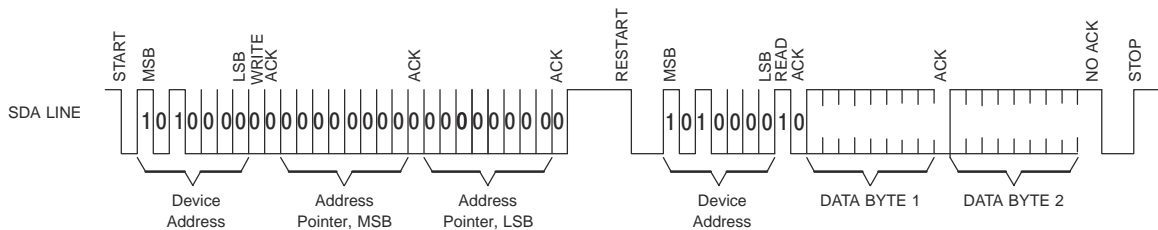
Figure 2-23. Overview of I2C\_Boot Function

The nonacknowledgment bit is checked only during the first message sent to initialize the EEPROM base address. This is to make sure that an EEPROM is present at address 0x50 before continuing. If an EEPROM is not present, code will The nonacknowledgment bit is not checked during the address phase of the data read messages (I2C\_Get Word). If a non acknowledgment is received during the data read messages, the I2C bus will hang. Table 2-12 shows the 8-bit data stream used by the I2C.

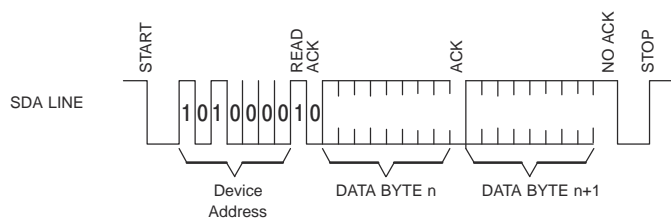
**Table 2-12. I2C 8-Bit Data Stream**

Byte	Contents
1	LSB: AA (KeyValue for memory width = 8 bits)
2	MSB: 08h (KeyValue for memory width = 8 bits)
3	LSB: I2CPSC[7:0]
4	reserved
5	LSB: I2CCLKH[7:0]
6	MSB: I2CCLKH[15:8]
7	LSB: I2CCLKL[7:0]
8	MSB: I2CCLKL[15:8]
...	...
...	Data for this section.
17	LSB: Reserved for future use
18	MSB: Reserved for future use
19	LSB: Upper half of entry point PC
20	MSB: Upper half of entry point PC[22:16] (Note: Always 0x00)
21	LSB: Lower half of entry point PC[15:8]
22	MSB: Lower half of entry point PC[7:0]
...	...
...	Data for this section.
...	...
...	Blocks of data in the format size/destination address/data as shown in the generic data stream description.
...	...
...	Data for this section.
LSB: 00h	
n+1	MSB: 00h - indicates the end of the source

The I2C EEPROM protocol required by the I2C bootloader is shown in [Figure 2-24](#) and [Figure 2-25](#). The first communication, which sets the EEPROM address pointer to 0x0000 and reads the KeyValue (0x08AA) from it, is shown in [Figure 2-24](#). All subsequent reads are shown in [Figure 2-25](#) and are read two bytes at a time.



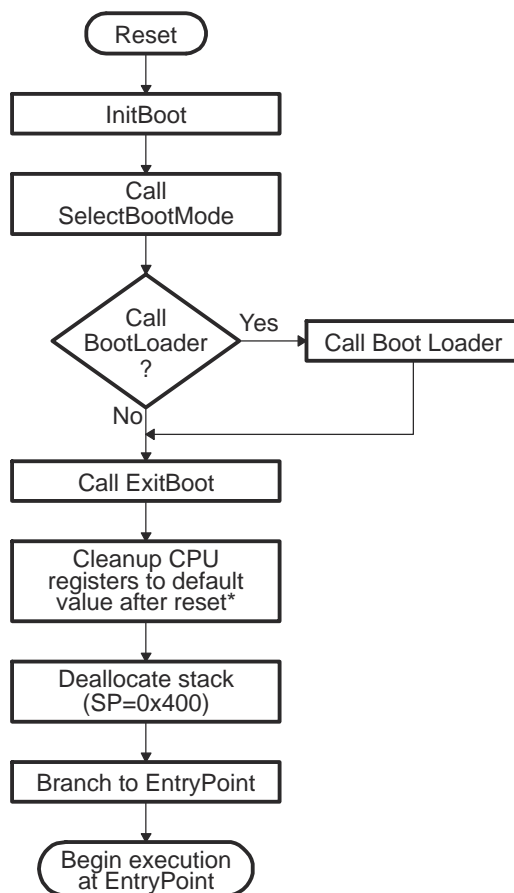
**Figure 2-24. Random Read**



**Figure 2-25. Sequential Read**

## 2.2.20 ExitBoot Assembly Routine

The Boot ROM includes an ExitBoot routine that restores the CPU registers to their default state at reset. This is performed on all registers with one exception. The OBJMODE bit in ST1 is left set so that the device remains configured for C28x operation. This flow is detailed in [Figure 2-26](#):



**Figure 2-26. ExitBoot Procedure Flow**

The following CPU registers are restored to their default values:

- ACC = 0x0000 0000
- RPC = 0x0000 0000
- P = 0x0000 0000
- XT = 0x0000 0000
- ST0 = 0x0000
- ST1 = 0x0A0B
- XAR0 = XAR7 = 0x0000 0000

After the ExitBoot routine completes and the program flow is redirected to the entry point address, the CPU registers will have the following values as shown in [Table 2-13](#).



**Table 2-13. CPU Register Restored Values**

Register	Value	Register	Value
ACC	0x0000 0000	P	0x0000 0000
XT	0x0000 0000	RPC	0x00 0000
XAR0-XAR7	0x0000 0000	DP	0x0000
ST0	0x0000	ST1	0x0A0B
	15:10 OVC = 0		15:13 ARP = 0
	9:7 PM = 0		12 XF = 0
	6 V = 0		11 MOM1MAP = 1
	5 N = 0		10 reserved
	4 Z = 0		9 OBJMODE = 1
	3 C = 0		8 AMODE = 0
	2 TC = 0		7 IDLESTAT = 0
	1 OVM = 0		6 EALLOW = 0
	0 SXM = 0		5 LOOP = 0
			4 SPA = 0
			3 VMAP = 1
			2 PAGE0 = 0
			1 DBGM = 1
			0 INTM = 1

## 2.3 Building the Boot Table

This chapter explains how to generate the data stream and boot table required for the bootloader.

### 2.3.1 The C2000 Hex Utility

To use the features of the bootloader, you must generate a data stream and boot table as described in [Section 2.2.11](#). The hex conversion utility tool, included with the 28x code generation tools, can generate the required data stream including the required boot table. This section describes the hex2000 utility. An example of a file conversion performed by hex2000 is described in [Section 2.3.2](#).

The hex utility supports creation of the boot table required for the SCI, SPI, I2C, and parallel I/O loaders. That is, the hex utility adds the required information to the file such as the key value, reserved bits, entry point, address, block start address, block length and terminating value. The contents of the boot table vary slightly depending on the boot mode and the options selected when running the hex conversion utility. The actual file format required by the host (ASCII, binary, or hex) will differ from one specific application to another and some additional conversion may be required.

To build the boot table, follow these steps:

- 1. Assemble or compile the code.**

This creates the object files that will then be used by the linker to create a single output file.

- 2. Link the file.**

The linker combines all of the object files into a single output file in common object file format (COFF). The specified linker command file is used by the linker to allocate the code sections to different memory blocks. Each block of the boot table data corresponds to an initialized section in the COFF file. Uninitialized sections are not converted by the hex conversion utility. The following options may be useful:

The linker `-m` option can be used to generate a map file. This map file will show all of the sections that were created, their location in memory and their length. It can be useful to check this file to make sure that the initialized sections are where you expect them to be.

The linker `-w` option is also very useful. This option will tell you if the linker has assigned a section to a memory region on its own. For example, if you have a section in your code called `ramfuncs`.

### 3. Run the hex conversion utility.

Choose the appropriate options for the desired boot mode and run the hex conversion utility to convert the COFF file produced by the linker to a boot table.

Table 2-14 summarizes the hex conversion utility options available for the bootloader. See the [TMS320C28x Assembly Language Tools v18.1.0.LTS User's Guide](#) for more information about the compiling and linking process, and for a detailed description of the hex2000 operations used to generate a boot table. Updates will be made to support the I2C boot. See the Codegen release notes for the latest information.

**Table 2-14. Bootloader Options**

Option	Description
-boot	Convert all sections into bootable form (use instead of a SECTIONS directive)
-sci8	Specify the source of the bootloader table as the SCI-A port, 8-bit mode
-spi8	Specify the source of the bootloader table as the SPI-A port, 8-bit mode
-gpio8	Specify the source of the bootloader table as the GPIO port, 8-bit mode
-bootorg value	Specify the source address of the bootloader table
-lospcp value	Specify the initial value for the LOSPCP register. This value is used only for the spi8 boot table format and ignored for all other formats. If the value is greater than 0x7F, the value is truncated to 0x7F.
-spibrr value	Specify the initial value for the SPIBRR register. This value is used only for the spi8 boot table format and ignored for all other formats. If the value is greater than 0x7F, the value is truncated to 0x7F.
-e value	Specify the entry point at which to begin execution after boot loading. The value can be an address or a global symbol. This value is optional. The entry point can be defined at compile time using the linker -e option to assign the entry point to a global symbol. The entry point for a C program is normally <code>_c_int00</code> unless defined otherwise by the -e linker option.
-i2c8	Specify the source of the bootloader table as the I2C-A port, 8-bit
-i2cpsc value	Specify the value for the I2CPSC register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM. This value will be truncated to the least significant eight bits and should be set to maintain an I2C module clock of 7-12 MHz.
-i2cclk value	Specify the value for the I2CCLKH register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM.
-i2cclk value	Specify the value for the I2CCLKL register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM.

#### 2.3.2 Example: Preparing a COFF File for SCI Bootloading

This section shows how to convert a COFF file into a format suitable for SCI- based bootloading. This example assumes that the host sending the data stream is capable of reading an ASCII hex format file. An example COFF file named GPIO34TOG.out has been used for the conversion.

Build the project and link using the -m linker option to generate a map file. Examine the .map file produced by the linker. The information shown in [Example 2-4](#) has been copied from the example map file (GPIO34TOG.map). This shows the section allocation map for the code. The map file includes the following information:

- **Output Section**

This is the name of the output section specified with the SECTIONS directive in the linker command file.

- **Origin**

The first origin listed for each output section is the starting address of that entire output section. The following origin values are the starting address of that portion of the output section.

- **Length**

The first length listed for each output section is the length for that entire output section. The following length values are the lengths associated with that portion of the output section.

- **Attributes/input sections**

This lists the input files that are part of the section or any value associated with an output section.

See the [TMS320C28x Assembly Language Tools v21.6.0.LTS User's Guide](#) for detailed information on generating a linker command file and a memory map.

All sections shown in [Example 2-4](#) that are initialized need to be loaded into the device in order for the code to execute properly. In this case, the `codestart`, `ramfuncs`, `.cinit`, `myreset` and `.text` sections need to be loaded. The other sections are uninitialized and will not be included in the loading process. The map file also indicates the size of each section and the starting address. For example, the `.text` section has 0x155 words and starts at 0x3FA000.

#### Example 2-4. GPIO34TOG Map File

output section	page	origin	length	attributes/ input sections
<code>codestart</code>	0	00000000	00000002	
		00000000	00000002	<code>device280x_CodeStartBranch.obj (codestart)</code>
<code>.pinit</code>	0	00000002	00000000	
<code>.switch</code>	0	00000002	00000000	UNINITIALIZED
<code>ramfuncs</code>	0	00000002	00000016	
		00000002	00000016	<code>device280x_SysCtrl.obj (ramfuncs)</code>
<code>.cinit</code>	0	00000018	00000019	
		00000018	0000000e	<code>rts2800_ml.lib : exit.obj (.cinit)</code>
		00000026	0000000a	<code>: _lock.obj (.cinit)</code>
		00000030	00000001	--HOLE-- [fill = 0]
<code>myreset</code>	0	00000032	00000002	
		00000032	00000002	<code>device280x_CodeStartBranch.obj (myreset)</code>
<code>IQmath</code>	0	003fa000	00000000	UNINITIALIZED
<code>.text</code>	0	003fa000	00000155	
		003fa000	00000046	<code>rts2800_ml.lib : boot.obj (.text)</code>

To load the code using the SCI bootloader, the host must send the data in the format that the bootloader understands. That is, the data must be sent as blocks of data with a size, starting address followed by the data. A block size of 0 indicates the end of the data. The `HEX2000.exe` utility can be used to convert the COFF file into a format that includes this boot information. The following command syntax has been used to convert the application into an ASCII hex format file that includes all of the required information for the bootloader:

#### Example 2-5. HEX2000.exe Command Syntax

```
C: HEX2000 GPIO34TOG.OUT -boot -gpio8 -a
Where:
- boot   Convert all sections into bootable form.
- gpio8  Use the GPIO in 8-bit mode data format. The SCI
         uses the same data format as the GPIO in 8-bit mode.
- a      Select ASCII-Hex as the output format.
```

The command line shown in [Example 2-5](#) will generate an ASCII-Hex output file called `GPIO34TOG.a00`, whose contents are explained in [Example 2-6](#). This example assumes that the host will be able to read an ASCII hex format file. The format may differ for your application. Each section of data loaded can be tied back to the map file described in [Example 2-4](#). After the data stream is loaded, the boot ROM will jump to the Entry point address that was read as part of the data stream. In this case, execution will begin at 0x3FA000.

**Example 2-6. GPIO34TOG Data Stream**

```

AA 08 ;Keyvalue
00 00 00 00 00 00 00 00 ;8 reserved words
00 00 00 00 00 00 00 00
3F 00 00 A0 ;Entrypoint 0x003FA000
02 00 ;Load 2 words - codestart section
00 00 00 00 ;Load block starting at 0x000000
7F 00 9A A0 ;Data block 0x007F, 0xA09A
16 00 ;Load 0x0016 words - ramfuncs section
00 00 02 00 ;Load block starting at 0x000002
22 76 1F 76 2A 00 00 1A 01 00 06 CC F0 ;Data = 0x7522, 0x761F etc...
FF 05 50 06 96 06 CC FF F0 A9 1A 00 05
06 96 04 1A FF 00 05 1A FF 00 1A 76 07
F6 00 77 06 00
55 01 ;Load 0x0155 words - .text section
3F 00 00 A0 ;Load block starting at 0x003FA000
AD 28 00 04 69 FF 1F 56 16 56 1A 56 40 ;Data = 0x28AD, 0x4000 etc...
29 1F 76 00 00 02 29 1B 76 22 76 A9 28
18 00 A8 28 00 00 01 09 1D 61 C0 76 18
00 04 29 0F 6F 00 9B A9 24 01 DF 04 6C
04 29 A8 24 01 DF A6 1E A1 F7 86 24 A7
06 .. ..
.. .. ..
.. .. ..
FC 63 E6 6F
19 00 ;Load 0x0019 words - .cinit section
00 00 18 00 ;Load block starting at 0x000018
FF FF 00 B0 3F 00 00 00 FE FF 02 B0 3F ;Data = 0xFFFF, 0xB000 etc...
00 00 00 00 00 FE FF 04 B0 3F 00 00 00
00 00 FE FF .. .. ..
.. .. ..
3F 00 00 00
02 00 ;Load 0x0002 words - myreset section
00 00 32 00 ;Load block starting at 0x000032
00 00 00 00 ;Data = 0x0000, 0x0000
00 00 ;Block size of 0 - end of data

```

## 2.4 Bootloader Code Overview

This chapter contains information on the Boot ROM version, checksum, and code.

### 2.4.1 Boot ROM Version and Checksum Information

The boot ROM contains its own version number located at address 0x3F FFBA. This version number starts at 1 and will be incremented any time the boot ROM code is modified. The next address, 0x3F FFBB contains the month and year (MM/YY in decimal) that the boot code was released. The next four memory locations contain a checksum value for the boot ROM. The checksum is intended for TI internal use and will change based on the silicon and boot ROM version.

**Table 2-15. Bootloader Revision and Checksum Information**

Address	Contents
0x3F FFBA	Boot ROM Version Number
0x3F FFBB	MM/YY of release (in decimal)
0x3F FFBC	Least significant word of checksum
0x3F FFBD	...
0x3F FFBE	...
0x3F FFBF	Most significant word of checksum

Table 2-16 shows the boot ROM revision details.

**Table 2-16. Bootloader Revision Per Device**

Device(s)	Silicon REVID (Address 0x883)	Boot ROM Revision
2802x	0 (First silicon)	Version 1a
2802x	1	Version 2
2802x	2	Version 2

### 2.4.2 Bootloader Code Revision History

The associated boot ROM source code can be found under the \libraries\boot-rom directory in [C2000Ware](#).

This page intentionally left blank.

The enhanced pulse width modulator (ePWM) module is a key element in controlling many of the power electronic systems found in both commercial and industrial equipments. These systems include digital motor control, switch mode power supply control, uninterruptible power supplies (UPS), and other forms of power conversion. The ePWM module performs a digital-to-analog (DAC) function, where the duty cycle is equivalent to a DAC analog value; it is sometimes referred to as a Power DAC.

This chapter is applicable for ePWM type 1. See the [C2000 Real-Time Control Peripheral Reference Guide](#) for a list of all devices with an ePWM module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

This chapter includes an overview of the module and information about each of the submodules:

- Time-Base Module
- Counter-Compare Module
- Action-Qualifier Module
- Dead-Band Generator Module
- PWM-Chopper (PC) Module
- Trip-Zone Module
- Event-Trigger Module

ePWM Type 1 is fully compatible to the Type 0 module. Type 1 has the following enhancements in addition to the Type 0 module:

- **Increased Dead-Band Resolution:** The dead-band clocking has been enhanced to allow half-cycle clocking to double resolution.
- **Enhanced Interrupt and SOC Generation:** Interrupts and ADC start-of-conversion can now be generated on both the TBCTR == zero and TBCTR == period events. This feature enables dual edge PWM control. Additionally, the ADC start-of-conversion can be generated from an event defined in the digital compare sub-module.
- **High-Resolution Period Capability:** Provides the ability to enable high-resolution period. This is discussed in more detail in [Chapter 4](#).
- **Digital Compare Sub-module:** The digital compare sub-module enhances the event triggering and trip zone sub-modules by providing filtering, blanking and improved trip functionality to digital compare signals. Such features are essential for peak current mode control and for support of analog comparators.

<b>3.1 Introduction</b> .....	<b>208</b>
<b>3.2 ePWM Submodules</b> .....	<b>215</b>
<b>3.3 Applications to Power Topologies</b> .....	<b>268</b>
<b>3.4 Registers</b> .....	<b>294</b>

## 3.1 Introduction

An effective PWM peripheral must be able to generate complex pulse width waveforms with minimal CPU overhead or intervention. It needs to be highly programmable and very flexible while being easy to understand and use. The ePWM unit described here addresses these requirements by allocating all needed timing and control resources on a per PWM channel basis. Cross coupling or sharing of resources has been avoided; instead, the ePWM is built up from smaller single channel modules with separate resources that can operate together as required to form a system. This modular approach results in an orthogonal architecture and provides a more transparent view of the peripheral structure, helping users to understand its operation quickly.

In this document the letter x within a signal or module name is used to indicate a generic ePWM instance on a device. For example output signals EPWMxA and EPWMxB refer to the output signals from the ePWMx instance. Thus, EPWM1A and EPWM1B belong to ePWM1 and likewise EPWM4A and EPWM4B belong to ePWM4.

### 3.1.1 EPWM Related Collateral

#### Foundational Materials

- [Real-Time Control Reference Guide](#)
  - Refer to the EPWM section

#### Getting Started Materials

- [Flexible PWMs Enable Multi-Axis Drives, Multi-Level Inverters Application Report](#)
- [Getting Started with the C2000 ePWM Module \(Video\)](#)
- [Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Control Application Report](#)
  - Chapters 1 to 6 are Fundamental material, derivations, and explanations that are useful for learning about how PWM can be used to implement a DAC. Subsequent chapters are Getting Started and Expert material for implementing in a system.
- [Using the Enhanced Pulse Width Modulator \(ePWM\) Module Application Report](#)

#### Expert Materials

- [C2000 real-time microcontrollers - Reference designs](#)
  - See TI designs related to specific end applications used.

### 3.1.2 Submodule Overview

The ePWM module represents one complete PWM channel composed of two PWM outputs: EPWMxA and EPWMxB. Multiple ePWM modules are instanced within a device as shown in [Figure 3-1](#). Each ePWM instance is identical with one exception. Some instances include a hardware extension that allows more precise control of the PWM outputs. This extension is the high-resolution pulse width modulator (HRPWM) and is described in [Chapter 4](#). See your device-specific data sheet to determine which ePWM instances include this feature. Each ePWM module is indicated by a numerical value starting with 1. For example, ePWM1 is the first instance and ePWM3 is the third instance in the system and ePWMx indicates any instance.

The ePWM modules are chained together via a clock synchronization scheme that allows them to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral modules (eCAP). The number of modules is device-dependent and based on target application needs. Modules can also operate stand-alone.

Each ePWM module supports the following features:

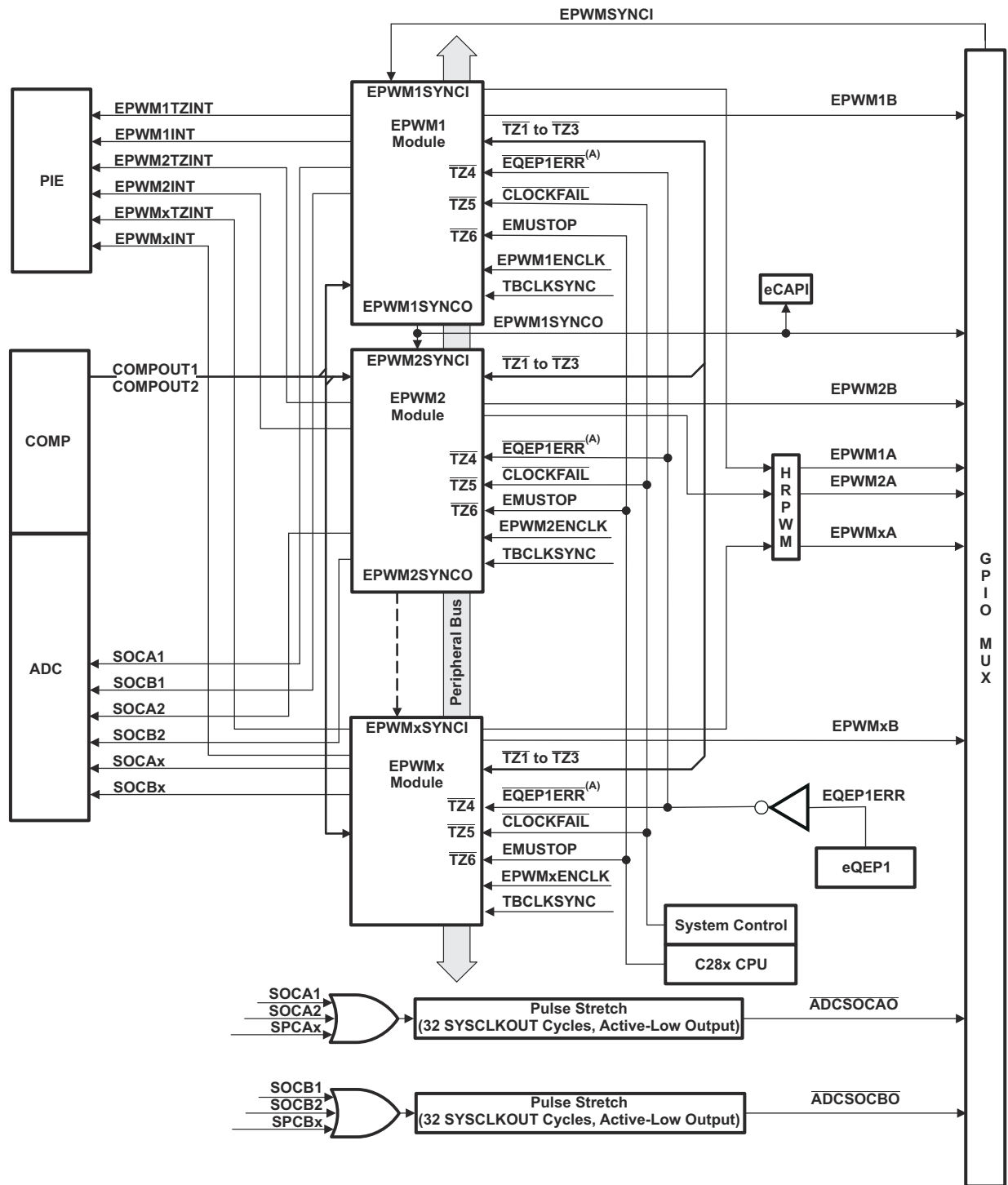
- Dedicated 16-bit time-base counter with period and frequency control.
- Two PWM outputs (EPWMxA and EPWMxB) that can be used in the following configurations:
  - Two independent PWM outputs with single-edge operation
  - Two independent PWM outputs with dual-edge symmetric operation
  - One independent PWM output with dual-edge asymmetric operation



- Asynchronous override control of PWM signals through software.
- Programmable phase-control support for lag or lead operation relative to other ePWM modules.
- Hardware-locked (synchronized) phase relationship on a cycle-by-cycle basis.
- Dead-band generation with independent rising and falling edge delay control.
- Programmable trip zone allocation of both cycle-by-cycle trip and one-shot trip on fault conditions.
- A trip condition can force either high, low, or high-impedance state logic levels at PWM outputs.
- Comparator module outputs and trip zone inputs can generate events, filtered events, or trip conditions.
- All events can trigger both CPU interrupts and ADC start of conversion (SOC).
- Programmable event prescaling minimizes CPU overhead on interrupts.
- PWM chopping by high-frequency carrier signal, useful for pulse transformer gate drives.

Each ePWM module is connected to the input/output signals shown in [Figure 3-1](#). The signals are described in detail in subsequent sections.

The order in which the ePWM modules are connected may differ from what is shown in [Figure 3-1](#). See [Section 3.2.2.3.3](#) for the synchronization scheme for a particular device. Each ePWM module consists of eight submodules and is connected within a system with the signals shown in [Figure 3-2](#).



A. This signal exists only on devices with an eQEP1 module.

**Figure 3-1. Multiple ePWM Modules**

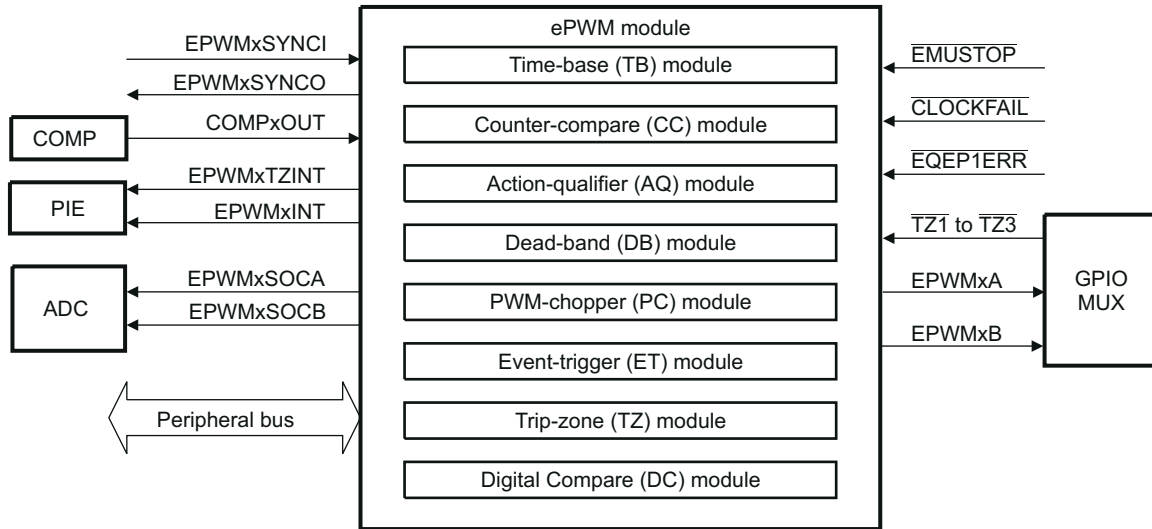
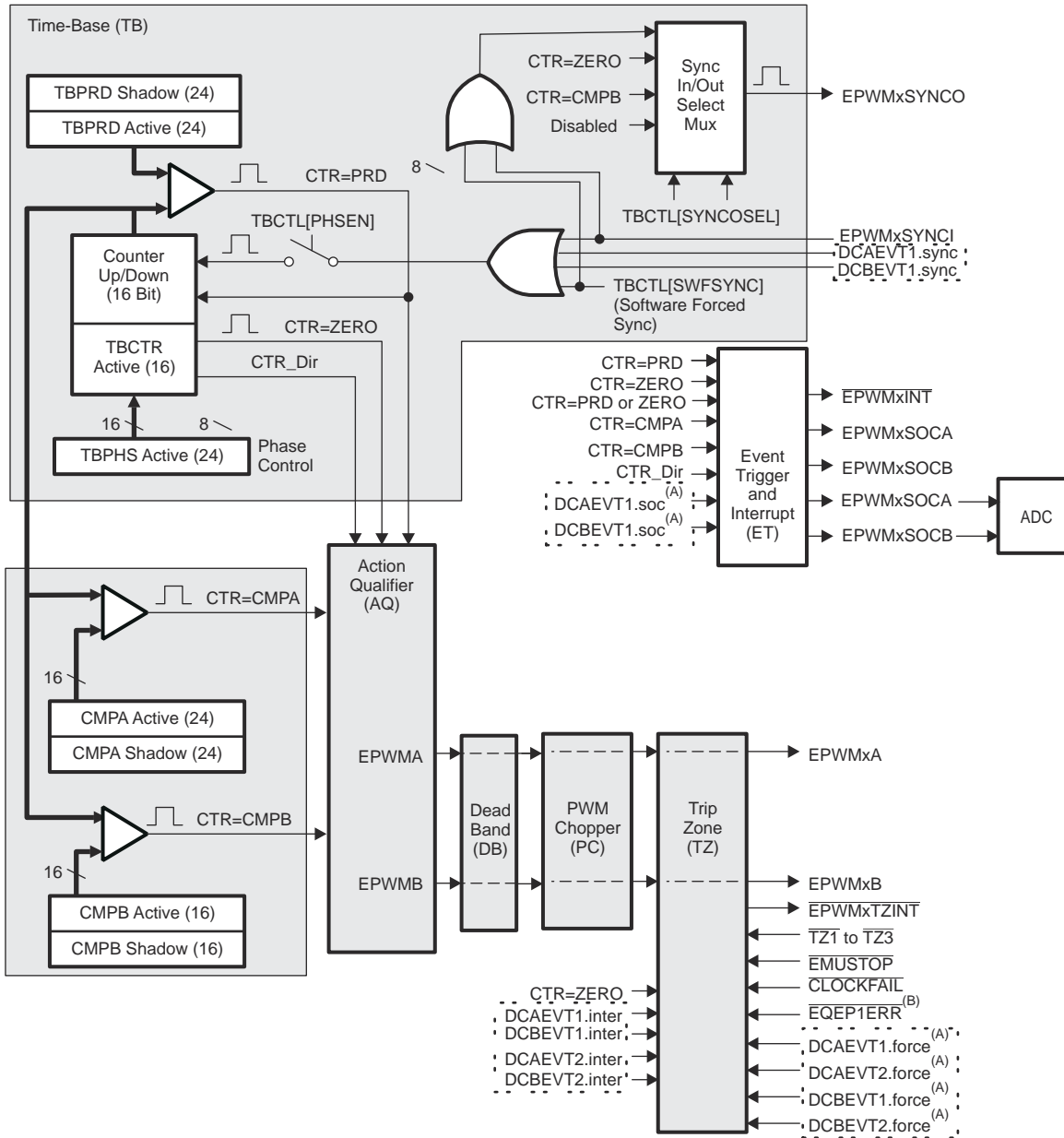


Figure 3-2. Submodules and Signal Connections for an ePWM Module

Figure 3-3 shows more internal details of a single ePWM module. The main signals used by the ePWM module are:

- **PWM output signals (EPWMxA and EPWMxB):** The PWM output signals are made available external to the device through the GPIO peripheral described in the system control and interrupts guide for your device.
- **Trip-zone signals (TZ1 to TZ6):** These input signals alert the ePWM module of fault conditions external to the ePWM module. Each module on a device can be configured to either use or ignore any of the trip-zone signals. The TZ1 to TZ3 trip-zone signals can be configured as asynchronous inputs through the GPIO peripheral. TZ4 is connected to an inverted EQEP1 error signal (EQEP1ERR) from the EQEP1 module (for those devices with an EQEP1 module). TZ5 is connected to the system clock fail logic, and TZ6 is connected to the EMUSTOP output from the CPU. This allows you to configure a trip action when the clock fails or the CPU halts.
- **Time-base synchronization input (EPWMxSYNCl) and output (EPWMxSYNCO) signals:** The synchronization signals daisy chain the ePWM modules together. Each module can be configured to either use or ignore its synchronization input. The clock synchronization input and output signal are brought out to pins only for ePWM1 (ePWM module #1). The synchronization output for ePWM1 (EPWM1SYNCO) is also connected to the SYNCl of the first enhanced capture module (eCAP1).
- **ADC start-of-conversion signals (EPWMxSOCA and EPWMxSOCB):** Each ePWM module has two ADC start of conversion signals. Any ePWM module can trigger a start of conversion. Whichever event triggers the start of conversion is configured in the Event-Trigger submodule of the ePWM.
- **Comparator output signals (COMPxOUT):** Output signals from the comparator module in conjunction with the trip zone signals can generate digital compare events.
- **Peripheral Bus:** The peripheral bus is 32-bits wide and allows both 16-bit and 32-bit writes to the ePWM register file.

Figure 3-3 also shows the key internal submodule interconnect signals. Each submodule is described in detail in its respective section.



- A. These events are generated by the type 1 ePWM digital compare (DC) submodule based on the levels of the COMPxOUT and TZ signals.
- B. This signal exists only on devices with in eQEP1 module.

**Figure 3-3. ePWM Submodules and Critical Internal Signal Interconnects**

### 3.1.3 Register Mapping

The complete ePWM module control and status register set is grouped by submodule as shown in [Table 3-1](#). Each register set is duplicated for each instance of the ePWM module. The start address for each ePWM register file instance on a device is specified in the appropriate data manual.

**Table 3-1. ePWM Module Control and Status Register Set Grouped by Submodule**

Name	Offset <sup>(1)</sup>	Size (x16)	Shadow	EALLOW	Description
<b>Time-Base Submodule Registers</b>					
TBCTL	0x0000	1			Time-Base Control Register
TBSTS	0x0001	1			Time-Base Status Register
TBPHSHR	0x0002	1			Extension for HRPWM Phase Register <sup>(2)</sup>
TBPHS	0x0003	1			Time-Base Phase Register
TBCTR	0x0004	1			Time-Base Counter Register
TBPRD	0x0005	1	Yes		Time-Base Period Register
TBPRDHR	0x0006	1	Yes		Time Base Period High Resolution Register <sup>(3)</sup>
<b>Counter-Compare Submodule Registers</b>					
CMPCTL	0x0007	1			Counter-Compare Control Register
CMPAHR	0x0008	1	Yes		Extension for HRPWM Counter-Compare A Register <sup>(2)</sup>
CMPA	0x0009	1	Yes		Counter-Compare A Register
CMPB	0x000A	1	Yes		Counter-Compare B Register
<b>Action-Qualifier Submodule Registers</b>					
AQCTLA	0x000B	1			Action-Qualifier Control Register for Output A (EPWMxA)
AQCTLB	0x000C	1			Action-Qualifier Control Register for Output B (EPWMxB)
AQSFRC	0x000D	1			Action-Qualifier Software Force Register
AQCSFRC	0x000E	1	Yes		Action-Qualifier Continuous S/W Force Register Set
<b>Dead-Band Generator Submodule Registers</b>					
DBCTL	0x000F	1			Dead-Band Generator Control Register
DBRED	0x0010	1			Dead-Band Generator Rising Edge Delay Count Register
DBFED	0x0011	1			Dead-Band Generator Falling Edge Delay Count Register
<b>Trip-Zone Submodule Registers</b>					
TZSEL	0x0012	1		Yes	Trip-Zone Select Register
TZDCSEL	0x0013	1		Yes	Trip Zone Digital Compare Select Register
TZCTL	0x0014	1		Yes	Trip-Zone Control Register <sup>(3)</sup>
TZEINT	0x0015	1		Yes	Trip-Zone Enable Interrupt Register <sup>(3)</sup>
TZFLG	0x0016	1			Trip-Zone Flag Register <sup>(3)</sup>
TZCLR	0x0017	1		Yes	Trip-Zone Clear Register <sup>(3)</sup>
TZFRC	0x0018	1		Yes	Trip-Zone Force Register <sup>(3)</sup>
<b>Event-Trigger Submodule Registers</b>					
ETSEL	0x0019	1			Event-Trigger Selection Register
ETPS	0x001A	1			Event-Trigger Pre-Scale Register
ETFLG	0x001B	1			Event-Trigger Flag Register
ETCLR	0x001C	1			Event-Trigger Clear Register
ETFRC	0x001D	1			Event-Trigger Force Register
<b>PWM-Chopper Submodule Registers</b>					
PCCTL	0x001E	1			PWM-Chopper Control Register
<b>High-Resolution Pulse Width Modulator (HRPWM) Extension Registers</b>					
HRCNFG	0x0020	1		Yes	HRPWM Configuration Register <sup>(2)</sup> <sup>(3)</sup>
HRPWR	0x0021	1		Yes	HRPWM Power Register <sup>(3)</sup> <sup>(4)</sup>

**Table 3-1. ePWM Module Control and Status Register Set Grouped by Submodule (continued)**

Name	Offset <sup>(1)</sup>	Size (x16)	Shadow	EALLOW	Description
HRMSTEP	0x0026	1		Yes	HRPWM MEP Step Register <sup>(3) (4)</sup>
HRPCTL	0x0028	1		Yes	High Resolution Period Control Register <sup>(3)</sup>
TBPRDHRM	0x002A	1	Writes		Time Base Period High Resolution Register Mirror <sup>(3)</sup>
TBPRDM	0x002B	1	Writes		Time Base Period Register Mirror
CMPAHRM	0x002C	1	Writes		Compare A High Resolution Register Mirror <sup>(3)</sup>
CMPAM	0x002D	1	Writes		Compare A Register Mirror
Digital Compare Event Registers					
DCTRISEL	0x0030	1		Yes	Digital Compare Trip Select Register
DCACTL	0x0031	1		Yes	Digital Compare A Control Register
DCBCTL	0x0032	1		Yes	Digital Compare B Control Register
DCFCTL	0x0033	1		Yes	Digital Compare Filter Control Register
DCCAPCTL	0x0034	1		Yes	Digital Compare Capture Control Register
DCFOFFSET	0x0035	1	Writes		Digital Compare Filter Offset Register
DCFOFFSETCNT	0x0036	1			Digital Compare Filter Offset Counter Register
DCFWINDOW	0x0037	1			Digital Compare Filter Window Register
DCFWINDOWCNT	0x0038	1			Digital Compare Filter Window Counter Register
DCCAP	0x0039	1	Yes		Digital Compare Counter Capture Register

(1) Locations not shown are reserved.

(2) These registers are only available on ePWM instances that include the high-resolution PWM extension. Otherwise these locations are reserved. These registers are also described in [Chapter 4](#). See your device-specific data sheet to determine which instances include the HRPWM.

(3) EALLOW protected registers as described in the *System Control and Interrupts* chapter.

(4) These registers only exist in the ePWM1 register space. They cannot be accessed from any other ePWM module's register space.

The CMPA, CMPAHR, TBPRD, and TBPRDHR registers are mirrored in the register map (Mirror registers include an "-M" suffix - CMPAM, CMPAHRM, TBPRDM, and TBPRDHRM). Note in the tables below, that in both Immediate mode and Shadow mode, reads from these mirror registers result in the active value of the register or a TI internal test value.

In Immediate Mode:

Register	Offset	Write	Read	Register	Offset	Write	Read
TBPRDHR	0x06	Active	Active	TBPRDHRM	0x2A	Active	TI_Internal
TBPRD	0x05	Active	Active	TBPRDM	0x2B	Active	Active
CMPAHR	0x08	Active	Active	CMPAHRM	0x2C	Active	TI_Internal
CMPA	0x09	Active	Active	CMPAM	0x2D	Active	Active

In Shadow Mode:

Register	Offset	Write	Read	Register	Offset	Write	Read
TBPRDHR	0x06	Shadow	Shadow	TBPRDHRM	0x2A	Shadow	TI_Internal
TBPRD	0x05	Shadow	Shadow	TBPRDM	0x2B	Shadow	Active
CMPAHR	0x08	Shadow	Shadow	CMPAHRM	0x2C	Shadow	TI_Internal
CMPA	0x09	Shadow	Shadow	CMPAM	0x2D	Shadow	Active

## 3.2 ePWM Submodules

Eight submodules are included in every ePWM peripheral. Each of these submodules performs specific tasks that can be configured by software.

### 3.2.1 Overview

[Table 3-2](#) lists the eight key submodules together with a list of their main configuration parameters. For example, if you need to adjust or control the duty cycle of a PWM waveform, then you should see the counter-compare submodule in [Section 3.2.3](#) for relevant details.

**Table 3-2. Submodule Configuration Parameters**

Submodule	Configuration Parameter or Option
Time-base (TB)	<ul style="list-style-type: none"> <li>• Scale the time-base clock (TBCLK) relative to the system clock (SYSCLKOUT).</li> <li>• Configure the PWM time-base counter (TBCTR) frequency or period.</li> <li>• Set the mode for the time-base counter:               <ul style="list-style-type: none"> <li>– count-up mode: used for asymmetric PWM</li> <li>– count-down mode: used for asymmetric PWM</li> <li>– count-up-and-down mode: used for symmetric PWM</li> </ul> </li> <li>• Configure the time-base phase relative to another ePWM module.</li> <li>• Synchronize the time-base counter between modules through hardware or software.</li> <li>• Configure the direction (up or down) of the time-base counter after a synchronization event.</li> <li>• Configure how the time-base counter will behave when the device is halted by a debug probe.</li> <li>• Specify the source for the synchronization output of the ePWM module:               <ul style="list-style-type: none"> <li>– Synchronization input signal</li> <li>– Time-base counter equal to zero</li> <li>– Time-base counter equal to counter-compare B (CMPB)</li> <li>– No output synchronization signal generated.</li> </ul> </li> </ul>
Counter-compare (CC)	<ul style="list-style-type: none"> <li>• Specify the PWM duty cycle for output EPWMxA and/or output EPWMxB</li> <li>• Specify the time at which switching events occur on the EPWMxA or EPWMxB output</li> </ul>
Action-qualifier (AQ)	<ul style="list-style-type: none"> <li>• Specify the type of action taken when a time-base or counter-compare submodule event occurs:               <ul style="list-style-type: none"> <li>– No action taken</li> <li>– Output EPWMxA and/or EPWMxB switched high</li> <li>– Output EPWMxA and/or EPWMxB switched low</li> <li>– Output EPWMxA and/or EPWMxB toggled</li> </ul> </li> <li>• Force the PWM output state through software control</li> <li>• Configure and control the PWM dead-band through software</li> </ul>
Dead-band (DB)	<ul style="list-style-type: none"> <li>• Control of traditional complementary dead-band relationship between upper and lower switches</li> <li>• Specify the output rising-edge-delay value</li> <li>• Specify the output falling-edge delay value</li> <li>• Bypass the dead-band module entirely. In this case the PWM waveform is passed through without modification.</li> <li>• Option to enable half-cycle clocking for double resolution.</li> </ul>
PWM-chopper (PC)	<ul style="list-style-type: none"> <li>• Create a chopping (carrier) frequency.</li> <li>• Pulse width of the first pulse in the chopped pulse train.</li> <li>• Duty cycle of the second and subsequent pulses.</li> <li>• Bypass the PWM-chopper module entirely. In this case the PWM waveform is passed through without modification.</li> </ul>

**Table 3-2. Submodule Configuration Parameters (continued)**

Submodule	Configuration Parameter or Option
Trip-zone (TZ)	<ul style="list-style-type: none"> <li>• Configure the ePWM module to react to one, all, or none of the trip-zone signals or digital compare events.</li> <li>• Specify the tripping action taken when a fault occurs:                             <ul style="list-style-type: none"> <li>– Force EPWMxA and/or EPWMxB high</li> <li>– Force EPWMxA and/or EPWMxB low</li> <li>– Force EPWMxA and/or EPWMxB to a high-impedance state</li> <li>– Configure EPWMxA and/or EPWMxB to ignore any trip condition.</li> </ul> </li> <li>• Configure how often the ePWM will react to each trip-zone signal:                             <ul style="list-style-type: none"> <li>– One-shot</li> <li>– Cycle-by-cycle</li> </ul> </li> <li>• Enable the trip-zone to initiate an interrupt.</li> <li>• Bypass the trip-zone module entirely.</li> </ul>
Event-trigger (ET)	<ul style="list-style-type: none"> <li>• Enable the ePWM events that will trigger an interrupt.</li> <li>• Enable ePWM events that will trigger an ADC start-of-conversion event.</li> <li>• Specify the rate at which events cause triggers (every occurrence or every second or third occurrence)</li> <li>• Poll, set, or clear event flags</li> </ul>
Digital-compare (DC)	<ul style="list-style-type: none"> <li>• Enables comparator (COMP) module outputs and trip zone signals to create events and filtered events</li> <li>• Specify event-filtering options to capture TBCTR counter or generate blanking window</li> </ul>

Code examples are provided in the remainder of this document that show how to implement various ePWM module configurations. These examples use the constant definitions in the device *EPwm\_defines.h* file in the device-specific header file and peripheral examples software package.



### 3.2.2 Time-Base (TB) Submodule

Each ePWM module has its own time-base submodule that determines all of the event timing for the ePWM module. Built-in synchronization logic allows the time-base of multiple ePWM modules to work together as a single system. Figure 3-4 illustrates the time-base module's place within the ePWM.

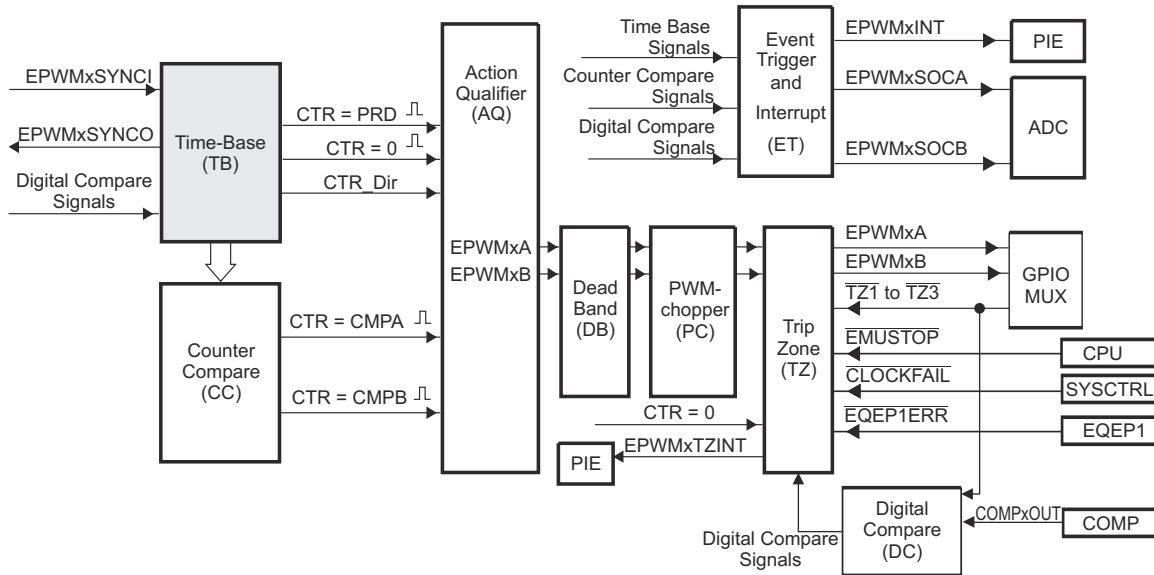


Figure 3-4. Time-Base Submodule Block Diagram

#### 3.2.2.1 Purpose of the Time-Base Submodule

You can configure the time-base submodule for the following:

- Specify the ePWM time-base counter (TBCTR) frequency or period to control how often events occur.
- Manage time-base synchronization with other ePWM modules.
- Maintain a phase relationship with other ePWM modules.
- Set the time-base counter to count-up, count-down, or count-up-and-down mode.
- Generate the following events:
  - CTR = PRD: Time-base counter equal to the specified period (TBCTR = TBPRD).
  - CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000).
- Configure the rate of the time-base clock; a prescaled version of the CPU system clock (SYSCLKOUT). This allows the time-base counter to increment/decrement at a slower rate.

### 3.2.2.2 Controlling and Monitoring the Time-base Submodule

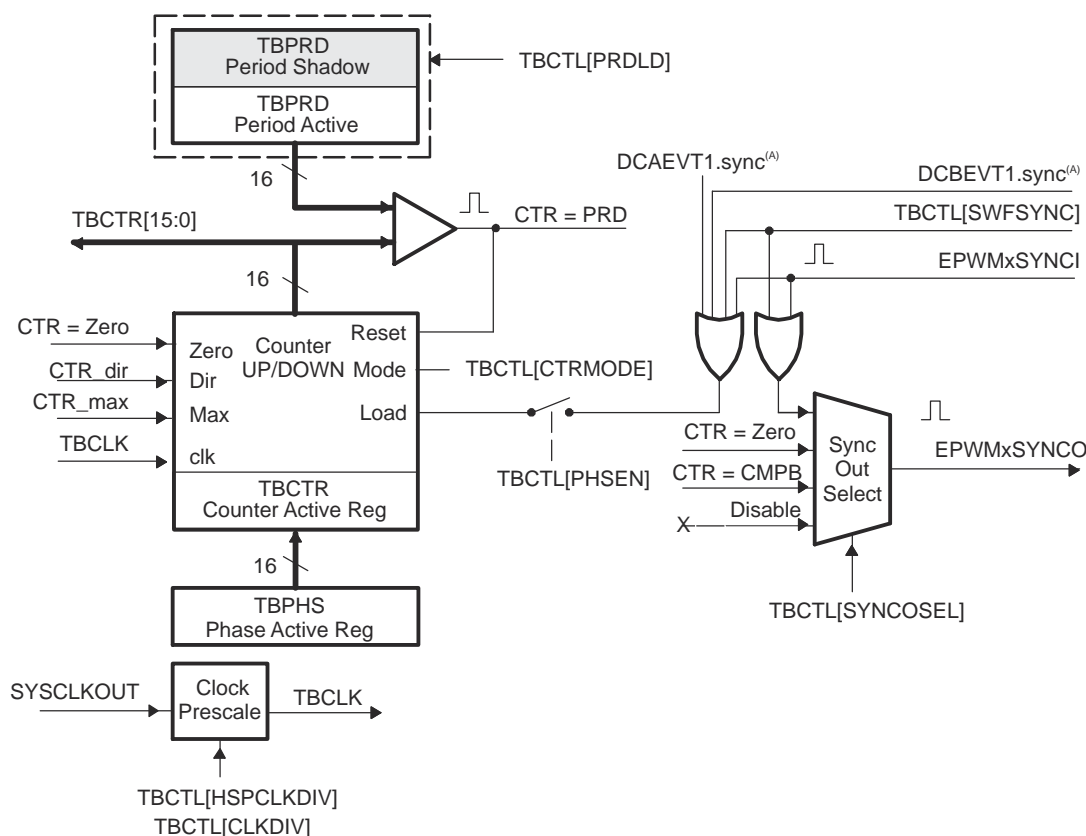
Table 3-3 shows the registers used to control and monitor the time-base submodule.

**Table 3-3. Time-Base Submodule Registers**

Register	Address Offset	Shadowed	Description	Bit Description
TBCTL	0x0000	No	Time-Base Control Register	Section 3.4.1.1
TBSTS	0x0001	No	Time-Base Status Register	Section 3.4.1.2
TBPHSHR	0x0002	No	HRPWM Extension Phase Register <sup>(1)</sup>	Section 3.4.1.3
TBPHS	0x0003	No	Time-Base Phase Register	Section 3.4.1.4
TBCTR	0x0004	No	Time-Base Counter Register	Section 3.4.1.5
TBPRD	0x0005	Yes	Time-Base Period Register	Section 3.4.1.6
TBPRDHR	0x0006	Yes	HRPWM Extension Period Register <sup>(1)</sup>	Section 3.4.1.7
TBPRDHRM	0x002A	Yes	HRPWM Time-Base Period Extension Mirror Register <sup>(1)</sup>	Section 3.4.1.8
TBPRDM	0x002B	Yes	HRPWM Extension Period Mirror Register <sup>(1)</sup>	Section 3.4.1.9

- (1) This register is available only on ePWM instances that include the high-resolution extension (HRPWM). On ePWM modules that do not include the HRPWM, this location is reserved. This register is also described in Chapter 4. See your device-specific data sheet to determine which ePWM instances include this feature.

The block diagram in Figure 3-5 shows the critical signals and registers of the time-base submodule. Table 3-4 provides descriptions of the key signals associated with the time-base submodule.



**Figure 3-5. Time-Base Submodule Signals and Registers**

**Table 3-4. Key Time-Base Signals**

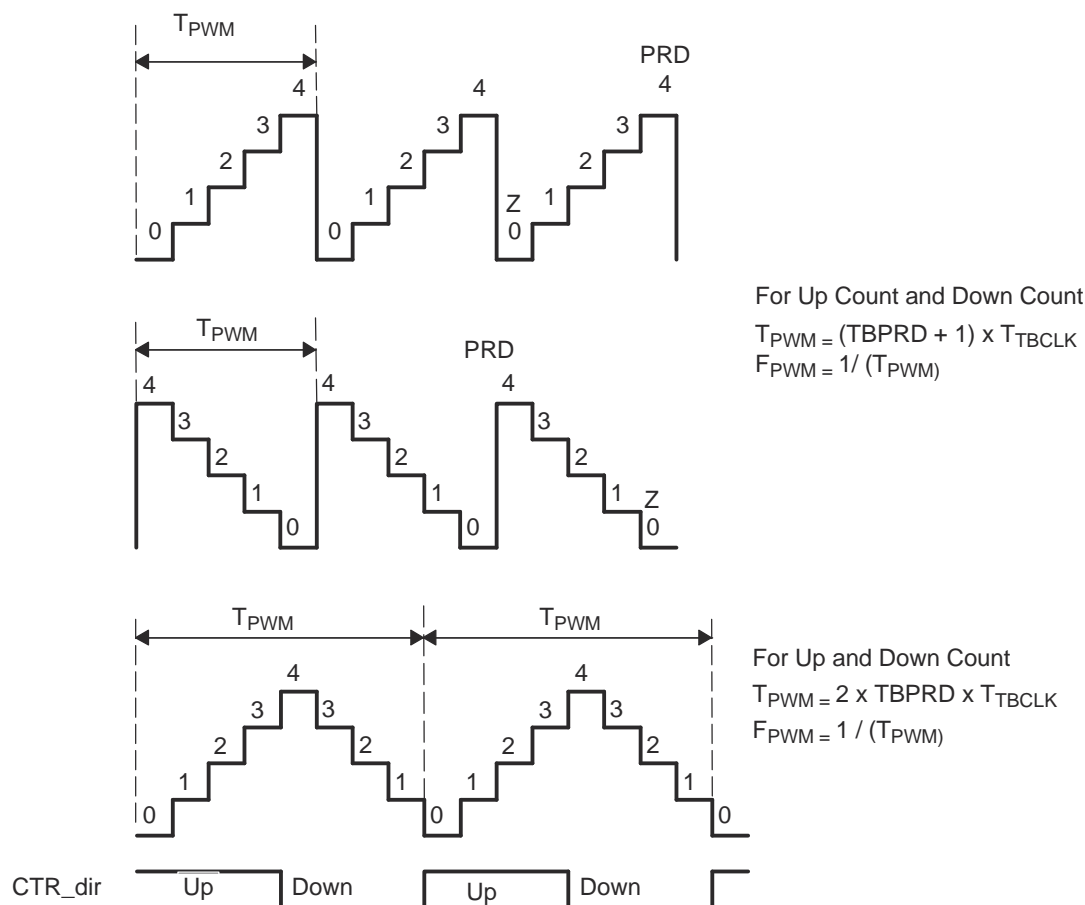
Signal	Description
EPWMxSYNCl	Time-base synchronization input.  Input pulse used to synchronize the time-base counter with the counter of ePWM module earlier in the synchronization chain. An ePWM peripheral can be configured to use or ignore this signal. For the first ePWM module (EPWM1) this signal comes from a device pin. For subsequent ePWM modules this signal is passed from another ePWM peripheral. For example, EPWM2SYNCl is generated by the ePWM1 peripheral, EPWM3SYNCl is generated by ePWM2 and so forth. See <a href="#">Section 3.2.2.3.3</a> for information on the synchronization order of a particular device.
EPWMxSYNCO	Time-base synchronization output.  This output pulse is used to synchronize the counter of an ePWM module later in the synchronization chain. The ePWM module generates this signal from one of three event sources: <ol style="list-style-type: none"> <li>1. EPWMxSYNCl (Synchronization input pulse)</li> <li>2. CTR = Zero: The time-base counter equal to zero (TBCTR = 0x0000).</li> <li>3. CTR = CMPB: The time-base counter equal to the counter-compare B (TBCTR = CMPB) register.</li> </ol>
CTR = PRD	Time-base counter equal to the specified period.  This signal is generated whenever the counter value is equal to the active period register value. That is when TBCTR = TBPRD.
CTR = Zero	Time-base counter equal to zero  This signal is generated whenever the counter value is zero. That is when TBCTR equals 0x0000.
CTR = CMPB	Time-base counter equal to active counter-compare B register (TBCTR = CMPB).  This event is generated by the counter-compare submodule and used by the synchronization out logic
CTR_dir	Time-base counter direction.  Indicates the current direction of the ePWM's time-base counter. This signal is high when the counter is increasing and low when it is decreasing.
CTR_max	Time-base counter equal max value. (TBCTR = 0xFFFF)  Generated event when the TBCTR value reaches its maximum value. This signal is only used only as a status bit
TBCLK	Time-base clock.  This is a prescaled version of the system clock (SYSCLKOUT) and is used by all submodules within the ePWM. This clock determines the rate at which time-base counter increments or decrements.

### 3.2.2.3 Calculating PWM Period and Frequency

The frequency of PWM events is controlled by the time-base period (TBPRD) register and the mode of the time-base counter. [Figure 3-6](#) shows the period ( $T_{pwm}$ ) and frequency ( $F_{pwm}$ ) relationships for the up-count, down-count, and up-down-count time-base counter modes when the period is set to 4 (TBPRD = 4). The time increment for each step is defined by the time-base clock (TBCLK) which is a prescaled version of the system clock (SYSCLKOUT).

The time-base counter has three modes of operation selected by the time-base control register (TBCTL):

- **Up-Down-Count Mode:** In up-down-count mode, the time-base counter starts from zero and increments until the period (TBPRD) value is reached. When the period value is reached, the time-base counter then decrements until it reaches zero. At this point the counter repeats the pattern and begins to increment.
- **Up-Count Mode:** In this mode, the time-base counter starts from zero and increments until it reaches the value in the period register (TBPRD). When the period value is reached, the time-base counter resets to zero and begins to increment once again.
- **Down-Count Mode:** In down-count mode, the time-base counter starts from the period (TBPRD) value and decrements until it reaches zero. When it reaches zero, the time-base counter is reset to the period value and it begins to decrement once again.


**Figure 3-6. Time-Base Frequency and Period**

### 3.2.2.3.1 Time-Base Period Shadow Register

The time-base period register (TBPRD) has a shadow register. Shadowing allows the register update to be synchronized with the hardware. The following definitions are used to describe all shadow registers in the ePWM module:

- **Active Register:** The active register controls the hardware and is responsible for actions that the hardware causes or invokes.
- **Shadow Register:** The shadow register buffers or provides a temporary holding location for the active register. It has no direct effect on any control hardware. At a strategic point in time the shadow register's content is transferred to the active register. This prevents corruption or spurious operation due to the register being asynchronously modified by software.

The memory address of the shadow period register is the same as the active register. Which register is written to or read from is determined by the TBCTL[PRDL] bit. This bit enables and disables the TBPRD shadow register as follows:

- **Time-Base Period Shadow Mode:** The TBPRD shadow register is enabled when TBCTL[PRDL] = 0. Reads from and writes to the TBPRD memory address go to the shadow register. The shadow register contents are transferred to the active register (TBPRD (Active) ← TBPRD (shadow)) when the time-base counter equals zero (TBCTR = 0x0000). By default the TBPRD shadow register is enabled.
- **Time-Base Period Immediate Load Mode:** If immediate load mode is selected (TBCTL[PRDL] = 1), then a read from or a write to the TBPRD memory address goes directly to the active register.

### 3.2.2.3.2 Time-Base Clock Synchronization

The TBCLKSYNC bit in the peripheral clock enable registers allows all users to globally synchronize all enabled ePWM modules to the time-base clock (TBCLK). When set, all enabled ePWM module clocks are started with the first rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescalers for each ePWM module must be set identically.

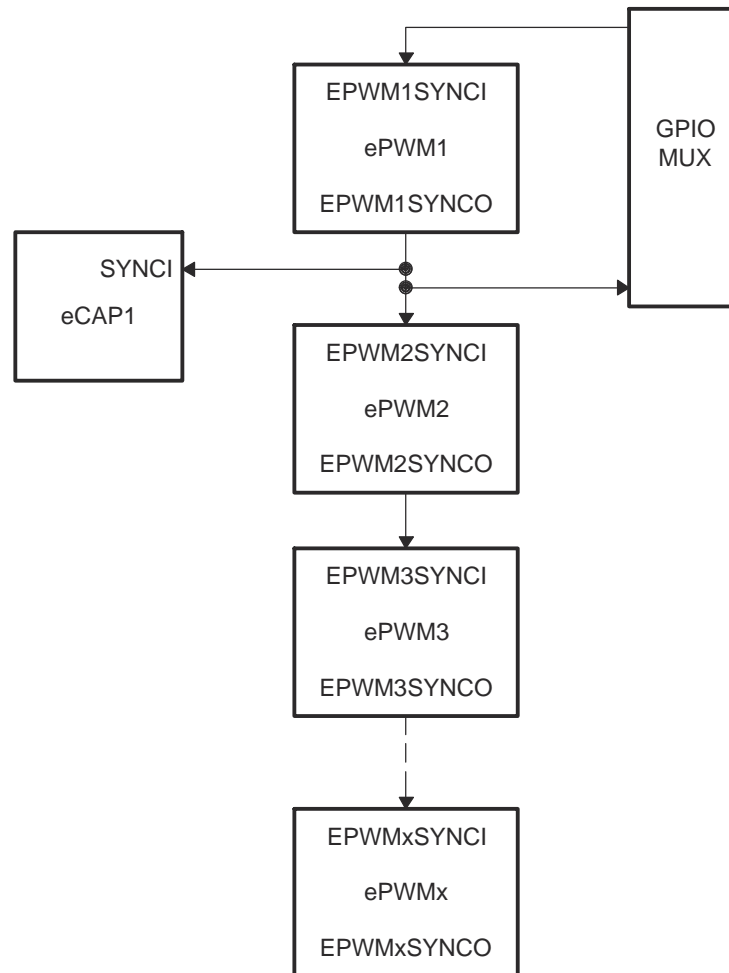
The proper procedure for enabling ePWM clocks is as follows:

1. Enable ePWM module clocks in the PCLKCRx register
2. Set TBCLKSYNC= 0
3. Configure ePWM modules
4. Set TBCLKSYNC=1

### 3.2.2.3.3 Time-Base Counter Synchronization

A time-base synchronization scheme connects all of the ePWM modules on a device. Each ePWM module has a synchronization input (EPWMxSYNCI) and a synchronization output (EPWMxSYNCO). The input synchronization for the first instance (ePWM1) comes from an external pin. The possible synchronization connections for the remaining ePWM modules are shown in [Figure 3-7](#).

Scheme 1 is shown in [Figure 3-7](#).



**Figure 3-7. Time-Base Counter Synchronization Scheme 1**

Each ePWM module can be configured to use or ignore the synchronization input. If the TBCTL[PHSEN] bit is set, then the time-base counter (TBCTR) of the ePWM module will be automatically loaded with the phase register (TBPHS) contents when one of the following conditions occur:

- **EPWMxSYNCI: Synchronization Input Pulse:** The value of the phase register is loaded into the counter register when an input synchronization pulse is detected (TBPHS → TBCTR). This operation occurs on the next valid time-base clock (TBCLK) edge. The source of the EPWMSYNCI pulse is selected by setting a GPIO to the EPWMSYNCI option per the GPIO Mux.

The delay from internal master module to slave modules is given by:

- if ( TBCLK = SYSCLKOUT): 2 x SYSCLKOUT
- if ( TBCLK != SYSCLKOUT): 1 TBCLK

- **Software Forced Synchronization Pulse:** Writing a 1 to the TBCTL[SWFSYNC] control bit invokes a software forced synchronization. This pulse is ORed with the synchronization input signal, and therefore has the same effect as a pulse on EPWMxSYNCI.
- **Digital Compare Event Synchronization Pulse:** DCAEVT1 and DCBEVT1 digital compare events can be configured to generate synchronization pulses which have the same affect as EPWMxSYNCI.

This feature enables the ePWM module to be automatically synchronized to the time base of another ePWM module. Lead or lag phase control can be added to the waveforms generated by different ePWM modules to synchronize them. In up-down-count mode, the TBCTL[PSHDIR] bit configures the direction of the time-base counter immediately after a synchronization event. The new direction is independent of the direction prior to the synchronization event. The PSHDIR bit is ignored in count-up or count-down modes. See [Figure 3-8](#) through [Figure 3-11](#) for examples.

Clearing the TBCTL[PHSEN] bit configures the ePWM to ignore the synchronization input pulse. The synchronization pulse can still be allowed to flow-through to the EPWMxSYNCO and be used to synchronize other ePWM modules. In this way, you can set up a master time-base (for example, ePWM1) and downstream modules (ePWM2 - ePWMx) may elect to run in synchronization with the master. See [Section 3.3](#) for more details on synchronization strategies.

### 3.2.2.4 Phase Locking the Time-Base Clocks of Multiple ePWM Modules

The TBCLKSYNC bit can be used to globally synchronize the time-base clocks of all enabled ePWM modules on a device. This bit is part of the device's clock enable registers and is described in the *System Control and Interrupts* section of this manual. When TBCLKSYNC = 0, the time-base clock of all ePWM modules is stopped (default). When TBCLKSYNC = 1, all ePWM time-base clocks are started with the rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescaler bits in the TBCTL register of each ePWM module must be set identically. The proper procedure for enabling the ePWM clocks is as follows:

1. Enable the individual ePWM module clocks. This is described in the device-specific version of the *System Control and Interrupts* section.
2. Set TBCLKSYNC = 0. This will stop the time-base clock within any enabled ePWM module.
3. Configure the prescaler values and desired ePWM modes.
4. Set TBCLKSYNC = 1.

### 3.2.2.5 Time-base Counter Modes and Timing Waveforms

The time-base counter operates in one of four modes:

- Up-count mode which is asymmetrical
- Down-count mode which is asymmetrical
- Up-down-count which is symmetrical
- Frozen where the time-base counter is held constant at the current value

To illustrate the operation of the first three modes, the following timing diagrams show when events are generated and how the time-base responds to an EPWMxSYNCl signal.

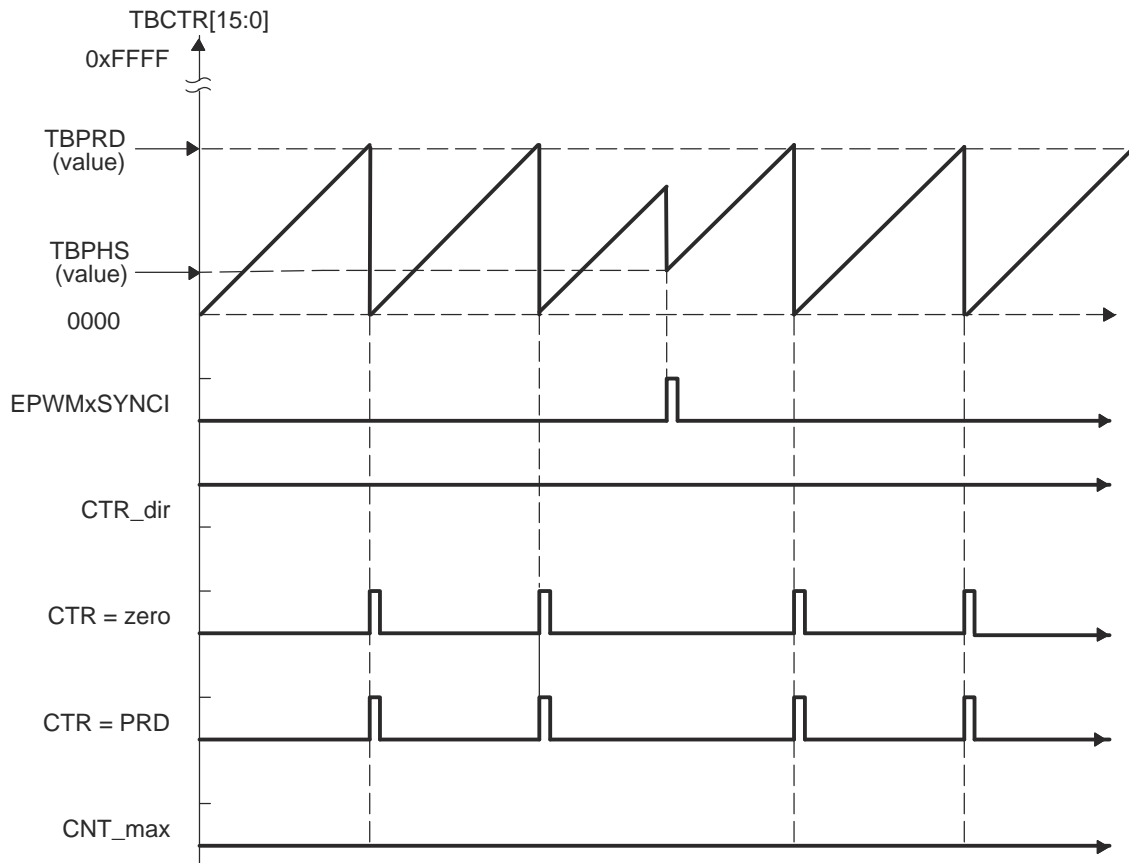


Figure 3-8. Time-Base Up-Count Mode Waveforms

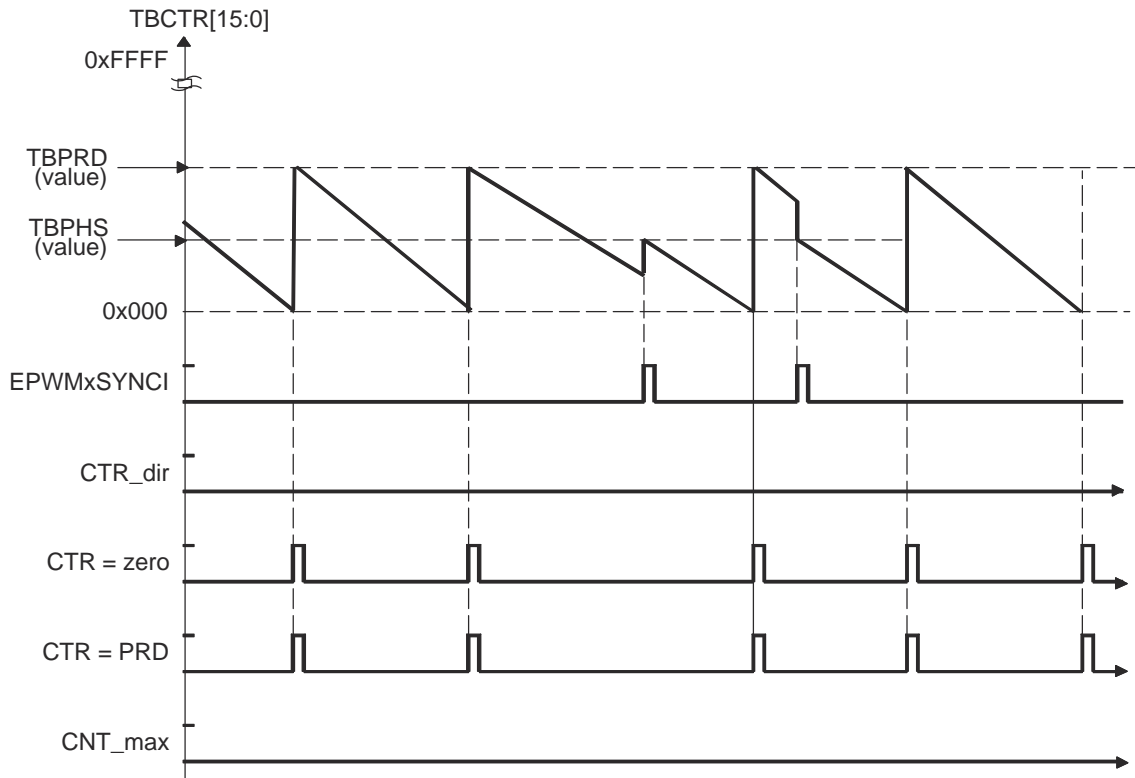


Figure 3-9. Time-Base Down-Count Mode Waveforms

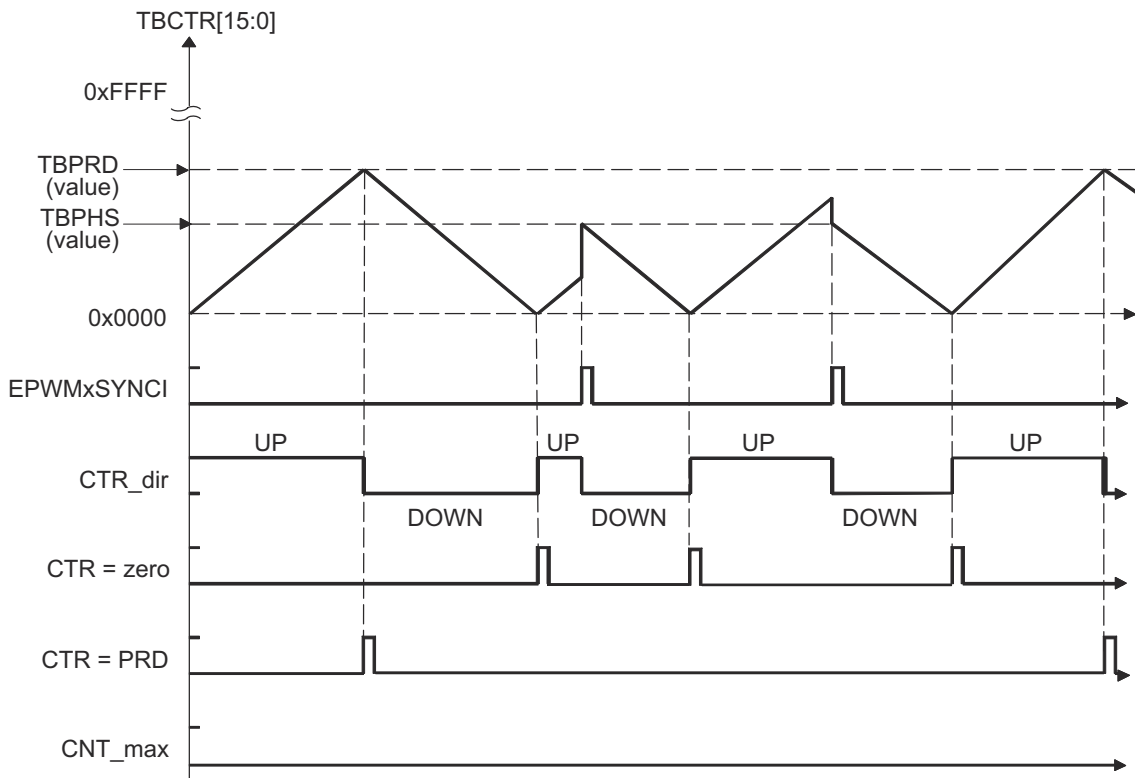


Figure 3-10. Time-Base Up-Down-Count Waveforms, TBCTL[PHSDIR = 0] Count Down On Synchronization Event



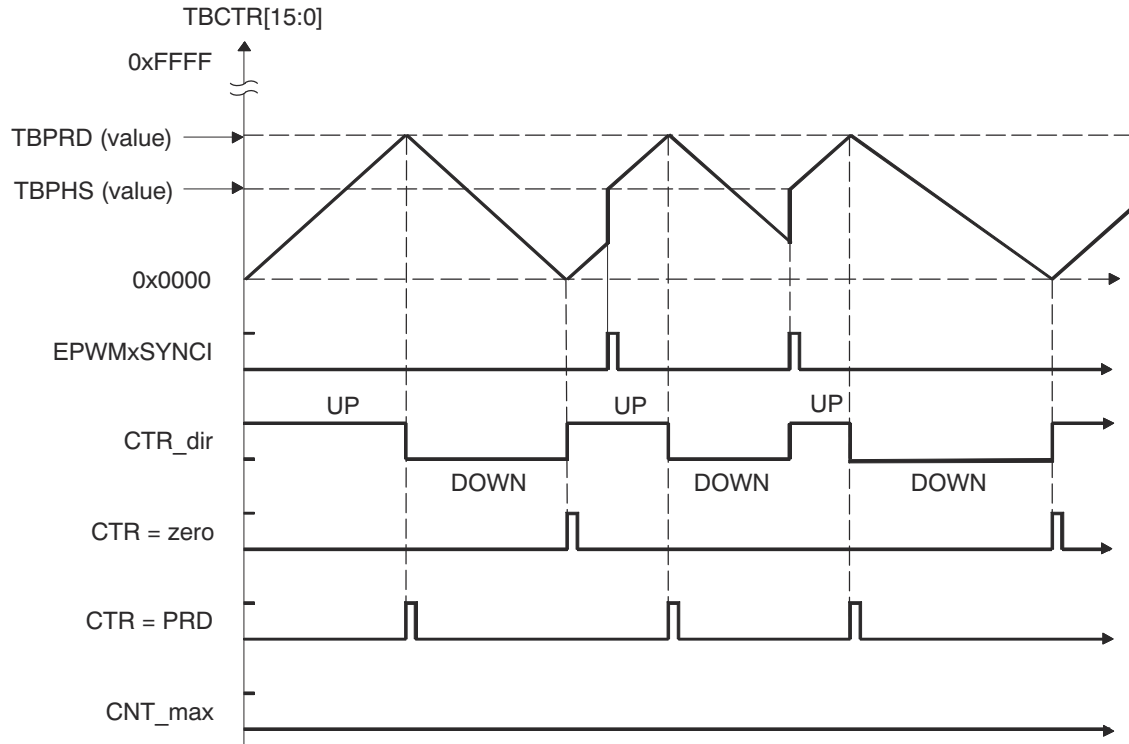


Figure 3-11. Time-Base Up-Down Count Waveforms, TBCTL[PHSDIR = 1] Count Up On Synchronization Event

### 3.2.3 Counter-Compare (CC) Submodule

Figure 3-12 shows the counter-compare submodule within the ePWM.

Figure 3-13 shows the basic structure of the counter-compare submodule.

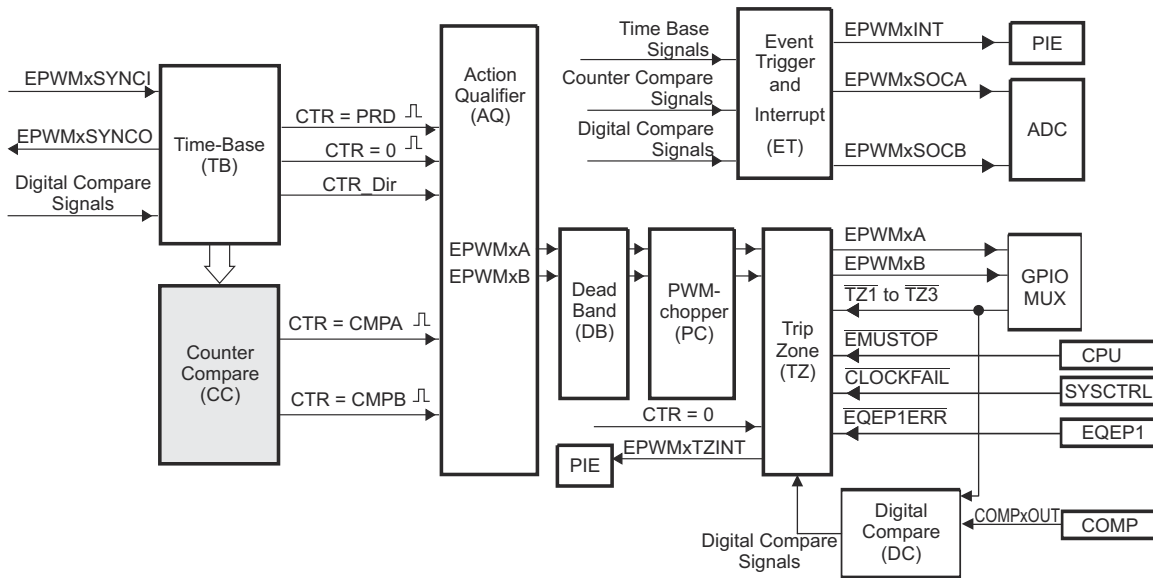


Figure 3-12. Counter-Compare Submodule

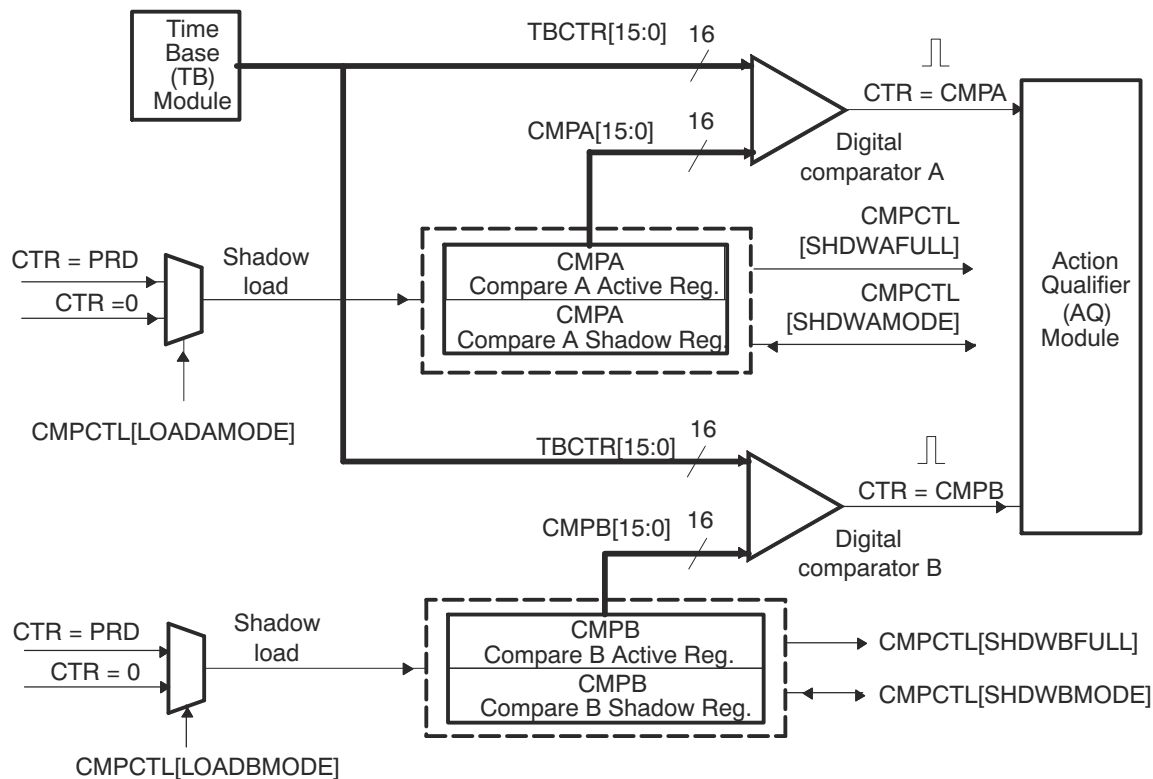


Figure 3-13. Detailed View of the Counter-Compare Submodule

### 3.2.3.1 Purpose of the Counter-Compare Submodule

The counter-compare submodule takes as input the time-base counter value. This value is continuously compared to the counter-compare A (CMPA) and counter-compare B (CMPB) registers. When the time-base counter is equal to one of the compare registers, the counter-compare unit generates an appropriate event.

The counter-compare:

- Generates events based on programmable time stamps using the CMPA and CMPB registers
  - CTR = CMPA: Time-base counter equals counter-compare A register (TBCTR = CMPA)
  - CTR = CMPB: Time-base counter equals counter-compare B register (TBCTR = CMPB)
- Controls the PWM duty cycle if the action-qualifier submodule is configured appropriately
- Shadows new compare values to prevent corruption or glitches during the active PWM cycle

### 3.2.3.2 Controlling and Monitoring the Counter-Compare Submodule

The counter-compare submodule operation is controlled and monitored by the registers shown in [Table 3-5](#).

**Table 3-5. Counter-Compare Submodule Registers**

Register	Address Offset	Shadowed	Description	Bit Description
CMPCTL	0x0007	No	Counter-Compare Control Register.	<a href="#">Section 3.4.2.1</a>
CMPAHR	0x0008	Yes	HRPWM Counter-Compare A Extension Register <sup>(1)</sup>	<a href="#">Section 3.4.2.2</a>
CMPA	0x0009	Yes	Counter-Compare A Register	<a href="#">Section 3.4.2.3</a>
CMPB	0x000A	Yes	Counter-Compare B Register	<a href="#">Section 3.4.2.4</a>
CMPAHRM	0x002C	Writes	HRPWM counter-compare A Extension Mirror Register <sup>(1)</sup>	<a href="#">Section 3.4.2.5</a>
CMPAM	0x002D	Writes	Counter-compare A mirror Register	<a href="#">Section 3.4.2.6</a>

- (1) This register is available only on ePWM modules with the high-resolution extension (HRPWM). On ePWM modules that do not include the HRPWM, this location is reserved. This register is also described in [Chapter 4](#). Refer to your device-specific data sheet to determine which ePWM instances include this feature.

The key signals associated with the counter-compare submodule are described in [Table 3-6](#).

**Table 3-6. Counter-Compare Submodule Key Signals**

Signal	Description of Event	Registers Compared
CTR = CMPA	Time-base counter equal to the active counter-compare A value	TBCTR = CMPA
CTR = CMPB	Time-base counter equal to the active counter-compare B value	TBCTR = CMPB
CTR = PRD	Time-base counter equal to the active period. Used to load active counter-compare A and B registers from the shadow register	TBCTR = TBPRD
CTR = ZERO	Time-base counter equal to zero. Used to load active counter-compare A and B registers from the shadow register	TBCTR = 0x0000

### 3.2.3.3 Operational Highlights for the Counter-Compare Submodule

The counter-compare submodule is responsible for generating two independent compare events based on two compare registers:

1. CTR = CMPA: Time-base counter equal to counter-compare A register (TBCTR = CMPA)
2. CTR = CMPB: Time-base counter equal to counter-compare B register (TBCTR = CMPB)

For up-count or down-count mode, each event occurs only once per cycle. For up-down-count mode each event occurs twice per cycle if the compare value is between 0x0000-TBPRD and once per cycle if the compare value is equal to 0x0000 or equal to TBPRD. These events are fed into the action-qualifier submodule where they are qualified by the counter direction and converted into actions if enabled. Refer to [Section 3.2.4.1](#) for more details.

The counter-compare registers CMPA and CMPB each have an associated shadow register. Shadowing provides a way to keep updates to the registers synchronized with the hardware. When shadowing is used, updates to the active registers only occur at strategic points. This prevents corruption or spurious operation due to the register being asynchronously modified by software. The memory address of the active register and the shadow register is identical. Which register is written to or read from is determined by the CMPCTL[SHDWAMODE] and CMPCTL[SHDWBMODE] bits. These bits enable and disable the CMPA shadow register and CMPB shadow register respectively. The behavior of the two load modes is:

**Shadow Mode:** The shadow mode for the CMPA is enabled by clearing the CMPCTL[SHDWAMODE] bit and the shadow register for CMPB is enabled by clearing the CMPCTL[SHDWBMODE] bit. Shadow mode is enabled by default for both CMPA and CMPB.

If the shadow register is enabled then the content of the shadow register is transferred to the active register on one of the following events as specified by the CMPCTL[LOADAMODE] and CMPCTL[LOADBMODE] register bits:

- CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD)
- CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000)
- Both CTR = PRD and CTR = Zero

Only the active register contents are used by the counter-compare submodule to generate events to be sent to the action-qualifier.

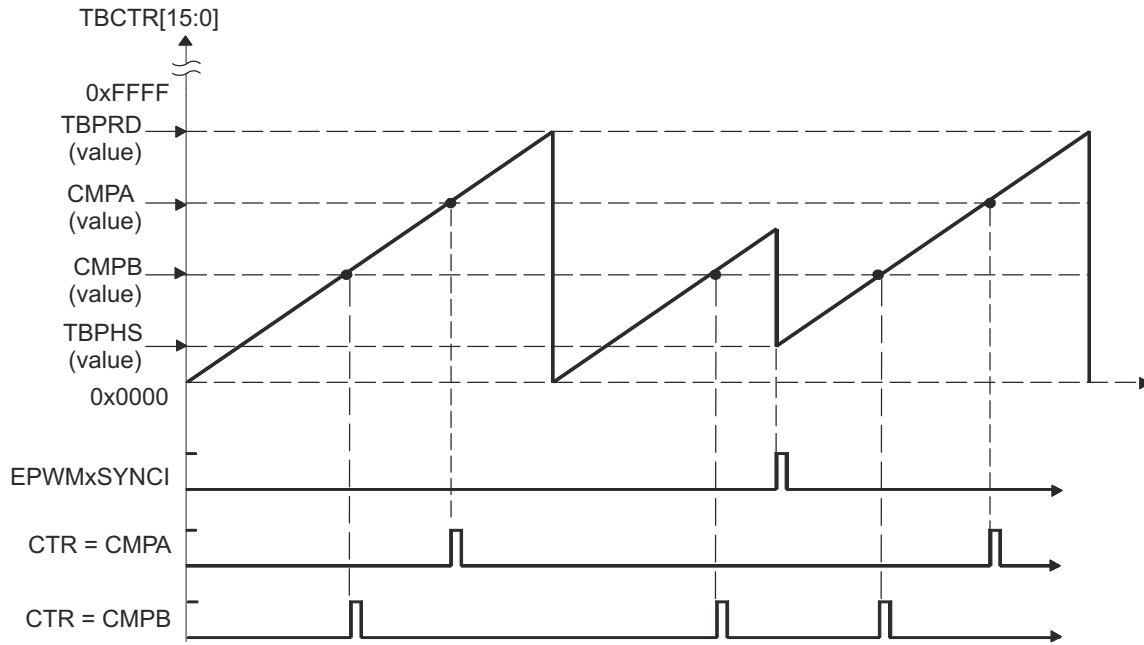
**Immediate Load Mode:** If immediate load mode is selected (TBCTL[SHADWAMODE] = 1 or TBCTL[SHADWBMODE] = 1), then a read from or a write to the register will go directly to the active register.

### 3.2.3.4 Count Mode Timing Waveforms

The counter-compare module can generate compare events in all three count modes:

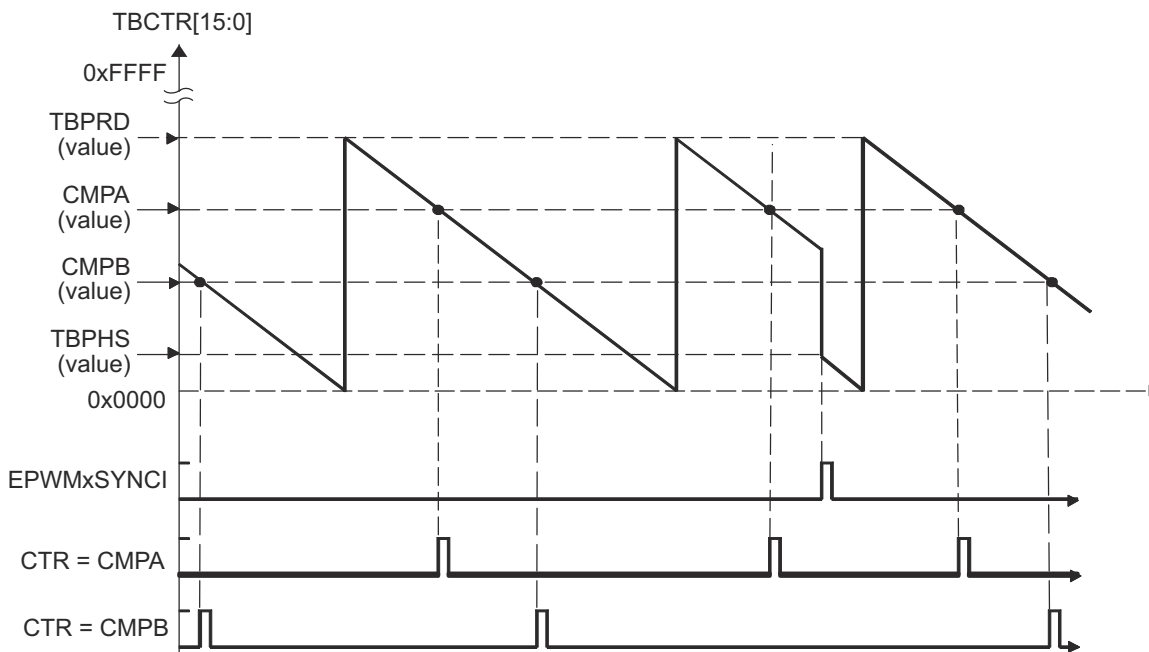
- Up-count mode: used to generate an asymmetrical PWM waveform
- Down-count mode: used to generate an asymmetrical PWM waveform
- Up-down-count mode: used to generate a symmetrical PWM waveform

To best illustrate the operation of the first three modes, the timing diagrams in [Figure 3-14](#) through [Figure 3-17](#) show when events are generated and how the EPWMxSYNCI signal interacts.



An EPWMxSYNCl external synchronization event can cause a discontinuity in the TBCTR count sequence. This can lead to a compare event being skipped. This skipping is considered normal operation and must be taken into account.

**Figure 3-14. Counter-Compare Event Waveforms in Up-Count Mode**



**Figure 3-15. Counter-Compare Events in Down-Count Mode**

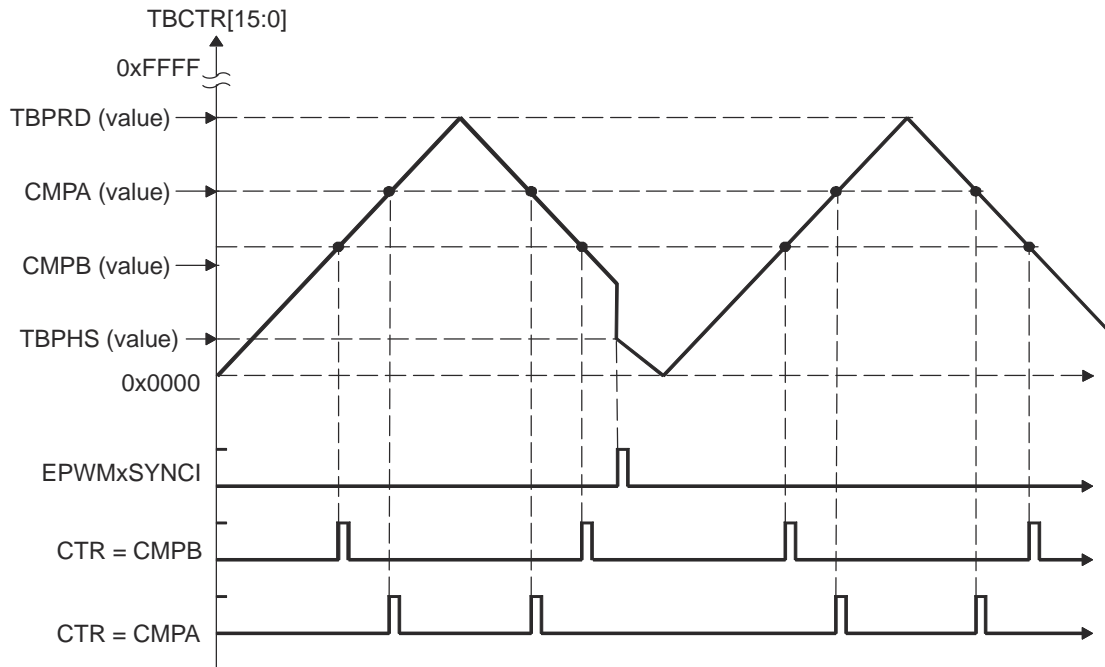


Figure 3-16. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

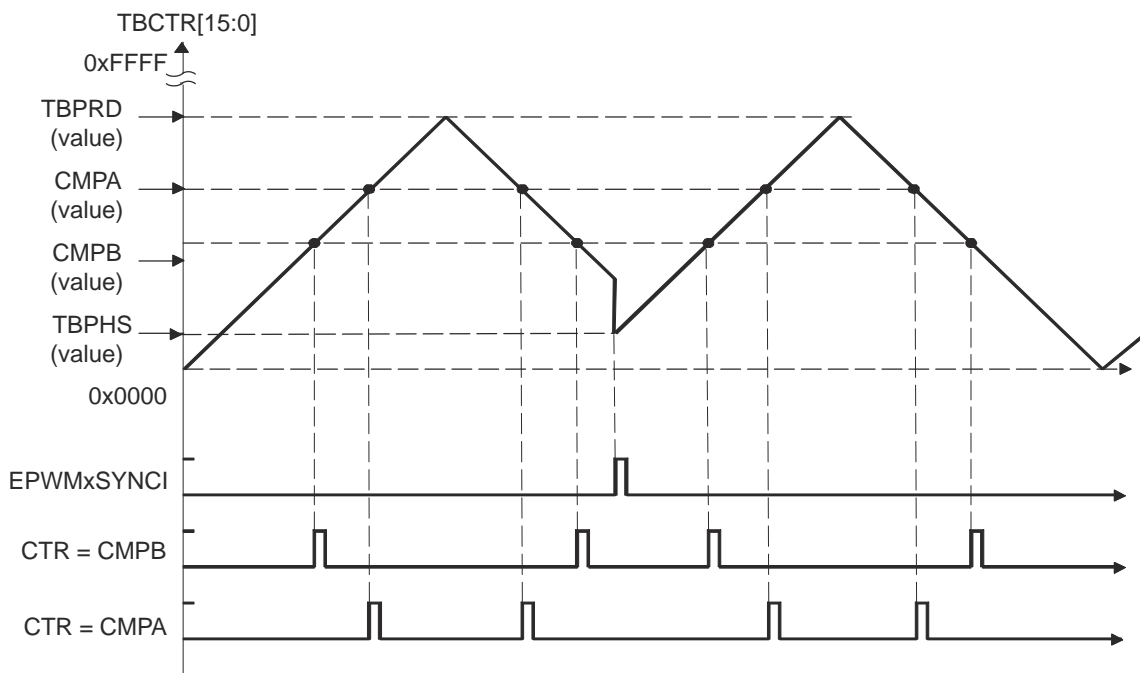


Figure 3-17. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 1] Count Up On Synchronization Event

### 3.2.4 Action-Qualifier (AQ) Submodule

Figure 3-18 shows the action-qualifier (AQ) submodule (see shaded block) in the ePWM system.

The action-qualifier submodule has the most important role in waveform construction and PWM generation. It decides which events are converted into various action types, thereby producing the required switched waveforms at the EPWMxA and EPWMxB outputs.

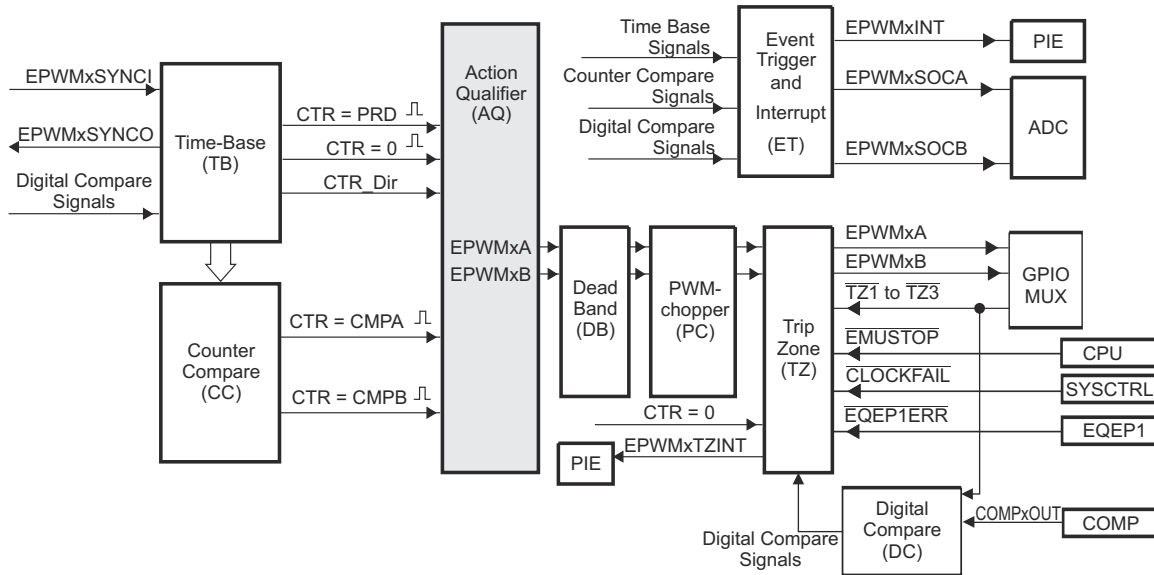


Figure 3-18. Action-Qualifier Submodule

#### 3.2.4.1 Purpose of the Action-Qualifier Submodule

The action-qualifier submodule is responsible for the following:

- Qualifying and generating actions (set, clear, toggle) based on the following events:
  - CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD).
  - CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000)
  - CTR = CMPA: Time-base counter equal to the counter-compare A register (TBCTR = CMPA)
  - CTR = CMPB: Time-base counter equal to the counter-compare B register (TBCTR = CMPB)
- Managing priority when these events occur concurrently
- Providing independent control of events when the time-base counter is increasing and when it is decreasing

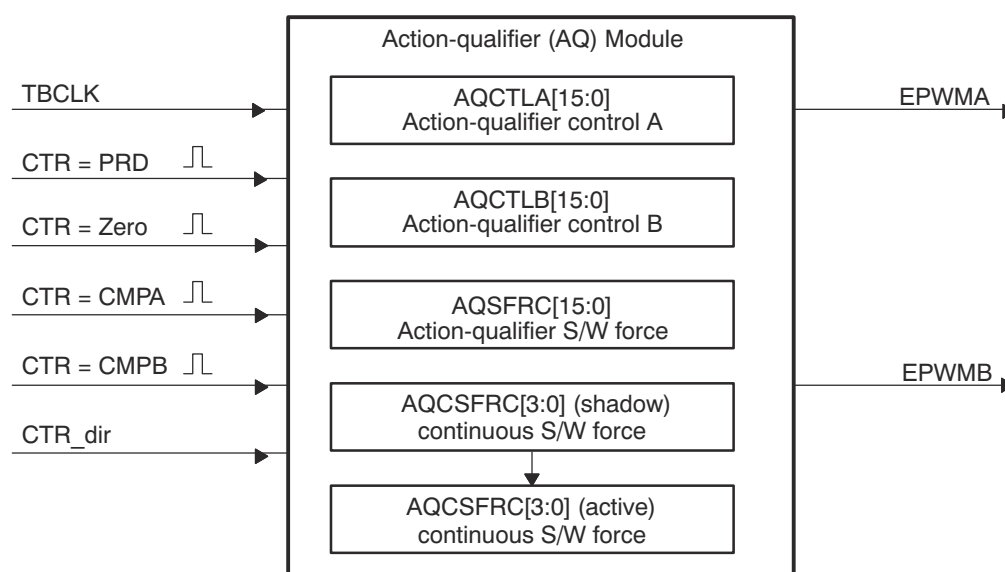
### 3.2.4.2 Action-Qualifier Submodule Control and Status Register Definitions

The action-qualifier submodule operation is controlled and monitored via the registers in [Table 3-7](#).

**Table 3-7. Action-Qualifier Submodule Registers**

Register	Address Offset	Shadowed	Description	Bit Description
AQCTLA	0x000B	No	Action-Qualifier Control Register for Output A (EPWMxA)	<a href="#">Section 3.4.3.1</a>
AQCTLB	0x000C	No	Action-Qualifier Control Register for Output B (EPWMxB)	<a href="#">Section 3.4.3.2</a>
AQSFRC	0x000D	No	Action-Qualifier Software Force Register	<a href="#">Section 3.4.3.3</a>
AQCSFRC	0x000E	Yes	Action-Qualifier Continuous Software Force	<a href="#">Section 3.4.3.4</a>

The action-qualifier submodule is based on event-driven logic. It can be thought of as a programmable cross switch with events at the input and actions at the output, all of which are software controlled via the set of registers shown in [Table 3-7](#).



**Figure 3-19. Action-Qualifier Submodule Inputs and Outputs**

For convenience, the possible input events are summarized again in [Table 3-8](#).

**Table 3-8. Action-Qualifier Submodule Possible Input Events**

Signal	Description	Registers Compared
CTR = PRD	Time-base counter equal to the period value	TBCTR = TBPRD
CTR = Zero	Time-base counter equal to zero	TBCTR = 0x0000
CTR = CMPA	Time-base counter equal to the counter-compare A	TBCTR = CMPA
CTR = CMPB	Time-base counter equal to the counter-compare B	TBCTR = CMPB
Software forced event	Asynchronous event initiated by software	

The software forced action is a useful asynchronous event. This control is handled by registers AQSFRC and AQCSFRC.

The action-qualifier submodule controls how the two outputs EPWMxA and EPWMxB behave when a particular event occurs. The event inputs to the action-qualifier submodule are further qualified by the counter direction (up or down). This allows for independent action on outputs on both the count-up and count-down phases.























The possible actions imposed on outputs EPWMxA and EPWMxB are:

- **Set High:** Set output EPWMxA or EPWMxB to a high level.
- **Clear Low:** Set output EPWMxA or EPWMxB to a low level.
- **Toggle:** If EPWMxA or EPWMxB is currently pulled high, then pull the output low. If EPWMxA or EPWMxB is currently pulled low, then pull the output high.
- **Do Nothing:** Keep outputs EPWMxA and EPWMxB at same level as currently set. Although the "Do Nothing" option prevents an event from causing an action on the EPWMxA and EPWMxB outputs, this event can still trigger interrupts and ADC start of conversion. See the Event-trigger Submodule description in [Section 3.2.8](#) for details.

Actions are specified independently for either output (EPWMxA or EPWMxB). Any or all events can be configured to generate actions on a given output. For example, both CTR = CMPA and CTR = CMPB can operate on output EPWMxA. All qualifier actions are configured via the control registers found at the end of this section.

For clarity, the drawings in this document use a set of symbolic actions. These symbols are summarized in [Figure 3-20](#). Each symbol represents an action as a marker in time. Some actions are fixed in time (zero and period) while the CMPA and CMPB actions are moveable and their time positions are programmed via the counter-compare A and B registers, respectively. To turn off or disable an action, use the "Do Nothing option"; it is the default at reset.

S/W force	TB Counter equals:				Actions
	Zero	Comp A	Comp B	Period	
					Do Nothing
					Clear Low
					Set High
					Toggle

**Figure 3-20. Possible Action-Qualifier Actions for EPWMxA and EPWMxB Outputs**

### 3.2.4.3 Action-Qualifier Event Priority

It is possible for the ePWM action qualifier to receive more than one event at the same time. In this case events are assigned a priority by the hardware. The general rule is events occurring later in time have a higher priority and software forced events always have the highest priority. The event priority levels for up-down-count mode are shown in [Table 3-9](#). A priority level of 1 is the highest priority and level 7 is the lowest. The priority changes slightly depending on the direction of TBCTR.

**Table 3-9. Action-Qualifier Event Priority for Up-Down-Count Mode**

Priority Level	Event If TBCTR is Incrementing TBCTR = Zero up to TBCTR = TBPRD	Event If TBCTR is Decrementing TBCTR = TBPRD down to TBCTR = 1
1 (Highest)	Software forced event	Software forced event
2	Counter equals CMPB on up-count (CBU)	Counter equals CMPB on down-count (CBD)
3	Counter equals CMPA on up-count (CAU)	Counter equals CMPA on down-count (CAD)
4	Counter equals zero	Counter equals period (TBPRD)
5	Counter equals CMPB on down-count (CBD)	Counter equals CMPB on up-count (CBU)
6 (Lowest)	Counter equals CMPA on down-count (CAD)	Counter equals CMPA on up-count (CBU)

[Table 3-10](#) shows the action-qualifier priority for up-count mode. In this case, the counter direction is always defined as up and thus down-count events will never be taken.

**Table 3-10. Action-Qualifier Event Priority for Up-Count Mode**

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to period (TBPRD)
3	Counter equal to CMPB on up-count (CBU)
4	Counter equal to CMPA on up-count (CAU)
5 (Lowest)	Counter equal to Zero

[Table 3-11](#) shows the action-qualifier priority for down-count mode. In this case, the counter direction is always defined as down and thus up-count events will never be taken.

**Table 3-11. Action-Qualifier Event Priority for Down-Count Mode**

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to Zero
3	Counter equal to CMPB on down-count (CBD)
4	Counter equal to CMPA on down-count (CAD)
5 (Lowest)	Counter equal to period (TBPRD)

It is possible to set the compare value greater than the period. In this case the action will take place as shown in [Table 3-12](#).

**Table 3-12. Behavior if CMPA/CMPB is Greater than the Period**

Counter Mode	Compare on Up-Count Event CAU/CBU	Compare on Down-Count Event CAD/CBD
Up-Count Mode	If $CMPA/CMPB \leq TBPRD$ period, then the event occurs on a compare match (TBCTR=CMPA or CMPB). If $CMPA/CMPB > TBPRD$ , then the event will not occur.	Never occurs.

**Table 3-12. Behavior if CMPA/CMPB is Greater than the Period (continued)**

Counter Mode	Compare on Up-Count Event CAU/CBU	Compare on Down-Count Event CAD/CBD
Down-Count Mode	Never occurs.	If CMPA/CMPB < TBPRD, the event will occur on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event will occur on a period match (TBCTR=TBPRD).
Up-Down-Count Mode	If CMPA/CMPB < TBPRD and the counter is incrementing, the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB is ≥ TBPRD, the event will occur on a period match (TBCTR = TBPRD).	If CMPA/CMPB < TBPRD and the counter is decrementing, the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event occurs on a period match (TBCTR=TBPRD).

### 3.2.4.4 Waveforms for Common Configurations

#### Note

The waveforms in this document show the ePWMs behavior for a static compare register value. In a running system, the active compare registers (CMPA and CMPB) are typically updated from their respective shadow registers once every period. The user specifies when the update will take place; either when the time-base counter reaches zero or when the time-base counter reaches period. There are some cases when the action based on the new value can be delayed by one period or the action based on the old value can take effect for an extra period. Some PWM configurations avoid this situation. These include, but are not limited to, the following:

#### Use up-down-count mode to generate a symmetric PWM:

- If you load CMPA/CMPB on zero, then use CMPA/CMPB values greater than or equal to 1.
- If you load CMPA/CMPB on period, then use CMPA/CMPB values less than or equal to TBPRD-1.

This means there will always be a pulse of at least one TBCLK cycle in a PWM period which, when very short, tend to be ignored by the system.

#### Use up-down-count mode to generate an asymmetric PWM:

- To achieve 50%-0% asymmetric PWM use the following configuration: Load CMPA/CMPB on period and use the period action to clear the PWM and a compare-up action to set the PWM. Modulate the compare value from 0 to TBPRD to achieve 50%-0% PWM duty.

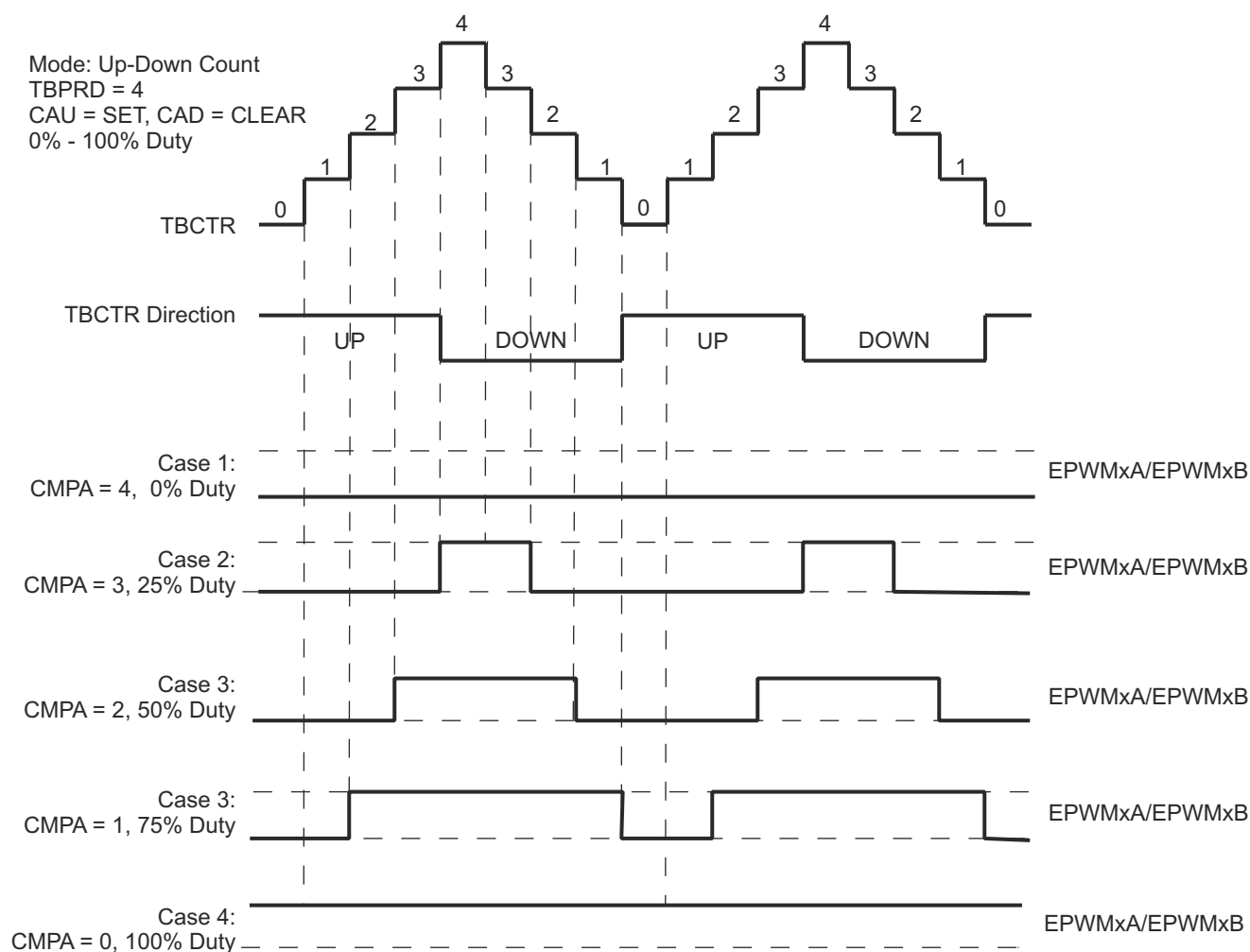
#### When using up-count mode to generate an asymmetric PWM:

- To achieve 0-100% asymmetric PWM use the following configuration: Load CMPA/CMPB on TBPRD. Use the Zero action to set the PWM and a compare-up action to clear the PWM. Modulate the compare value from 0 to TBPRD+1 to achieve 0-100% PWM duty.

See [Using the ePWM Module for 0% - 100% Duty Cycle Control](#). The software configurations described in this application report are not applicable when changing compare registers from a non-zero value to zero. However, they still apply when changing from a compare value of 0 to a non-zero value.

Figure 3-21 shows how a symmetric PWM waveform can be generated using the up-down-count mode of the TBCTR. In this mode 0%-100% DC modulation is achieved by using equal compare matches on the up count and down count portions of the waveform. In the example shown, CMPA is used to make the comparison. When the counter is incrementing the CMPA match will pull the PWM output high. Likewise, when the counter is decrementing the compare match will pull the PWM signal low. When CMPA = TBPRD, the PWM signal is low for the entire period giving the 0% duty waveform. When CMPA = 0, the PWM signal is high achieving 100% duty.

When using this configuration in practice, if you load CMPA/CMPB on zero, then use CMPA/CMPB values greater than or equal to 1. If you load CMPA/CMPB on period, then use CMPA/CMPB values less than or equal to TBPRD-1. This means there will always be a pulse of at least one TBCLK cycle in a PWM period which, when very short, tend to be ignored by the system.

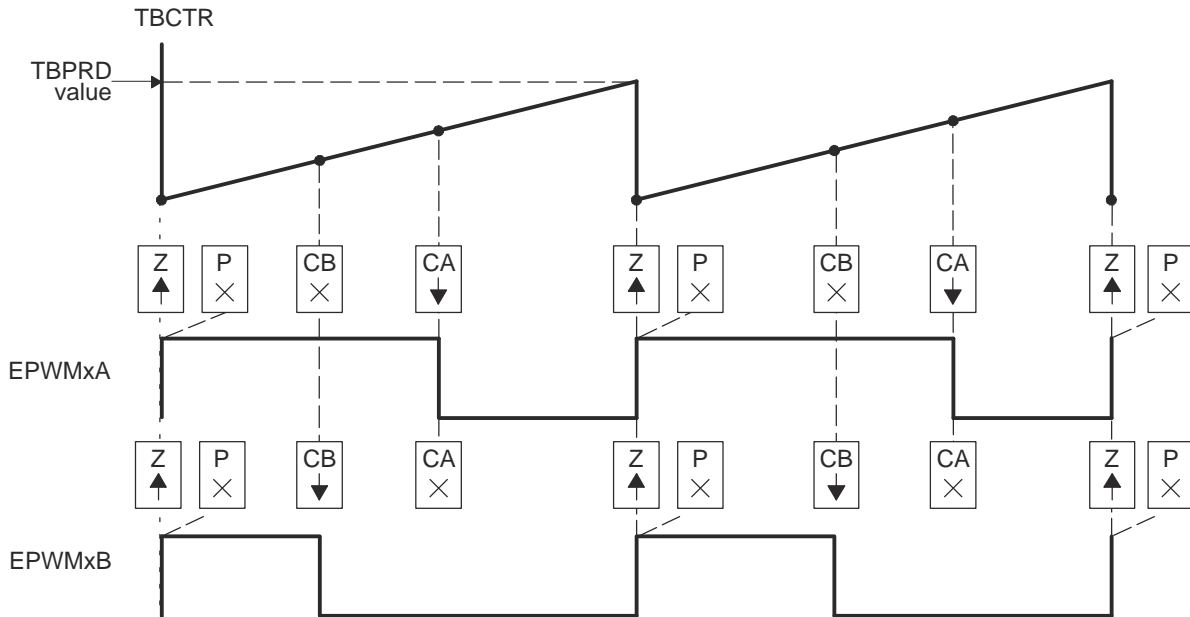


**Figure 3-21. Up-Down-Count Mode Symmetrical Waveform**

The PWM waveforms in [Figure 3-22](#) through [Figure 3-27](#) show some common action-qualifier configurations. The C-code samples in [Example 3-1](#) through [Example 3-6](#) shows how to configure an ePWM module for each case. Some conventions used in the figures and examples are as follows:

- TBPRD, CMPA, and CMPB refer to the value written in their respective registers. The active register, not the shadow register, is used by the hardware.
- CMPx, refers to either CMPA or CMPB.
- EPWMxA and EPWMxB refer to the output signals from ePWMx
- Up-Down means Count-up-and-down mode, Up means up-count mode and Dwn means down-count mode
- Sym = Symmetric, Asym = Asymmetric

Example 3-1 contains a code sample showing initialization and run time for the waveforms in Figure 3-22.



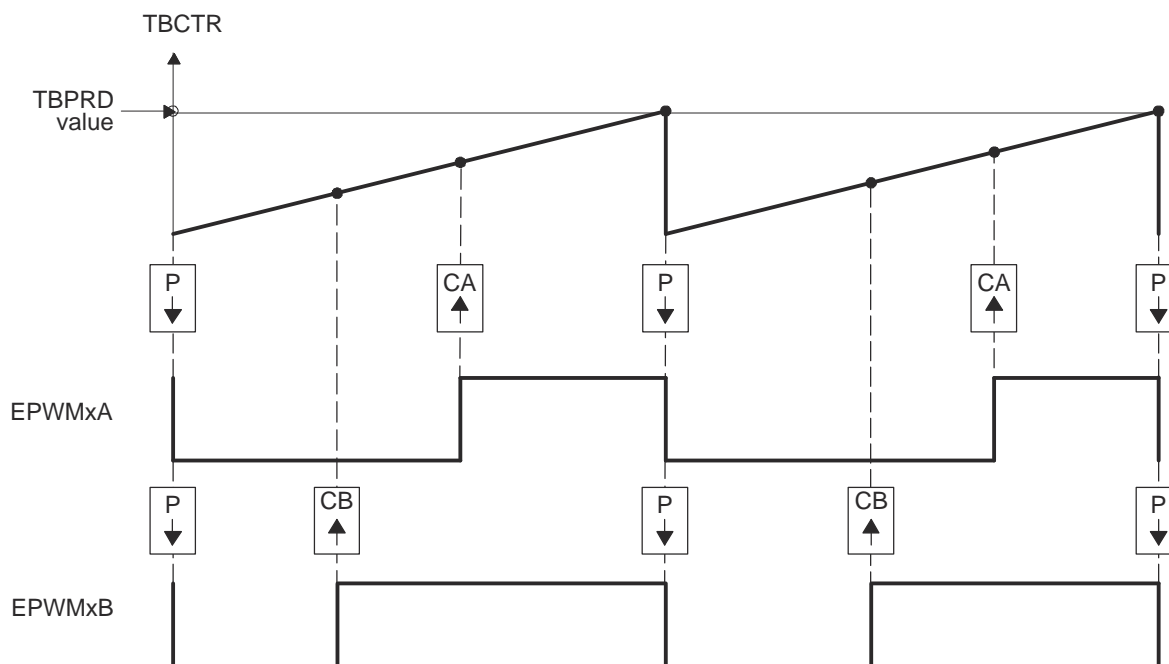
- A. PWM period = (TBPRD + 1) × T<sub>TBCLK</sub>
- B. Duty modulation for EPWMxA is set by CMPA, and is active high (that is, high time duty proportional to CMPA).
- C. Duty modulation for EPWMxB is set by CMPB and is active high (that is, high time duty proportional to CMPB).
- D. The "Do Nothing" actions (X) are shown for completeness, but will not be shown on subsequent diagrams.
- E. Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

**Figure 3-22. Up, Single Edge Asymmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB—Active High**

**Example 3-1. Code Sample for Figure 3-22**

```
// Initialization Time
// =====
EPwm1Regs.TBPRD = 600; // Period = 601 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350; // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 200; // Compare B = 200 TBCLK counts
EPwm1Regs.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTR = 0; // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // TBCLK = SYSCLK
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;
EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
//
// Run Time
// =====
EPwm1Regs.CMPA.half.CMPA = Duty1A; // adjust duty for output EPWM1A
EPwm1Regs.CMPB = Duty1B; // adjust duty for output EPWM1B
```

Example 3-2 contains a code sample showing initialization and run time for the waveforms in Figure 3-23.



- PWM period =  $(TBPRD + 1) \times T_{TBCLK}$
- Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
- Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
- Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

**Figure 3-23. Up, Single Edge Asymmetric Waveform With Independent Modulation on EPWMxA and EPWMxB—Active Low**

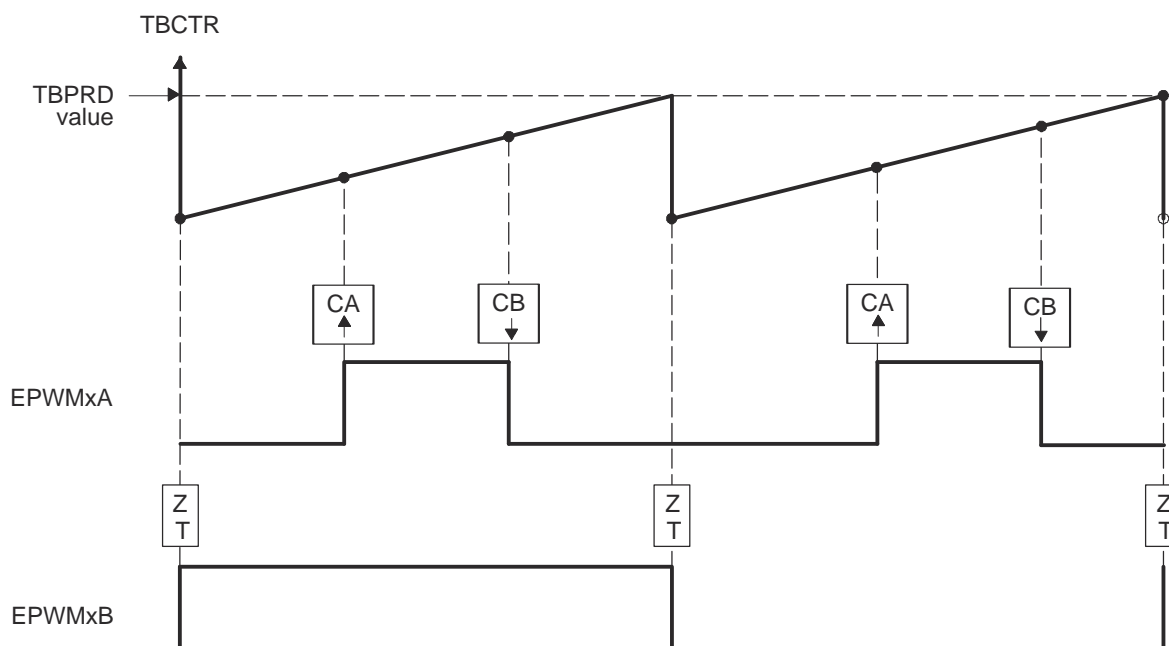
**Example 3-2. Code Sample for Figure 3-23**

```

// Initialization Time
// =====
EPwm1Regs.TBPRD = 600; // Period = 601 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350; // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 200; // Compare B = 200 TBCLK counts
EPwm1Regs.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTR = 0; // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // TBCLK = SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on TBCTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on TBCTR = Zero
EPwm1Regs.AQCTLA.bit.PR = AQ_CLEAR;
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
EPwm1Regs.AQCTLB.bit.PR = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.CBU = AQ_SET;
//
// Run Time
// =====
EPwm1Regs.CMPA.half.CMPA = Duty1A; // adjust duty for output EPWM1A
EPwm1Regs.CMPB = Duty1B; // adjust duty for output EPWM1B

```

Example 3-3 contains a code sample showing initialization and run time for the waveforms Figure 3-24. Use the constant definitions in the device-specific header file.



- A.  $PWM\ frequency = 1 / ((TBPRD + 1) \times T_{TBCLK})$
- B. Pulse can be placed anywhere within the PWM cycle (0000 - TBPRD)
- C. High time duty proportional to  $(CMPB - CMPA)$
- D. EPWMxB can be used to generate a 50% duty square wave with frequency =  $\frac{1}{2} \times ((TBPRD + 1) \times TBCLK)$

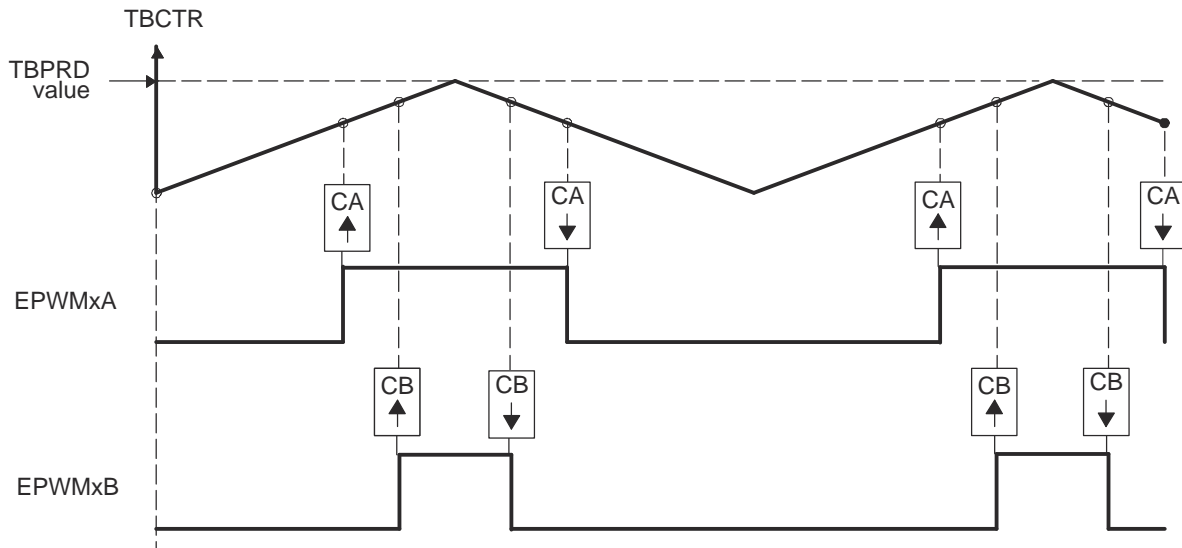
**Figure 3-24. Up-Count, Pulse Placement Asymmetric Waveform With Independent Modulation on EPWMxA**

### Example 3-3. Code Sample for Figure 3-24

```
// Initialization Time
// =====
EPwm1Regs.TBPRD = 600;           // Period = 601 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 200; // Compare A = 200 TBCLK counts
EPwm1Regs.CMPB = 400;          // Compare B = 400 TBCLK counts
EPwm1Regs.TBPHS = 0;           // Set Phase register to zero
EPwm1Regs.TBCTR = 0;           // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // TBCLK = SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on TBCTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on TBCTR = Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
EPwm1Regs.AQCTLA.bit.CBU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_TOGGLE;
//
// Run Time
// =====
EPwm1Regs.CMPA.half.CMPA = EdgePosA; // adjust duty for output EPWM1A only
EPwm1Regs.CMPB = EdgePosB;
```



Example 3-4 contains a code sample showing initialization and run time for the waveforms in Figure 3-25. Use the constant definitions in the device-specific header file.



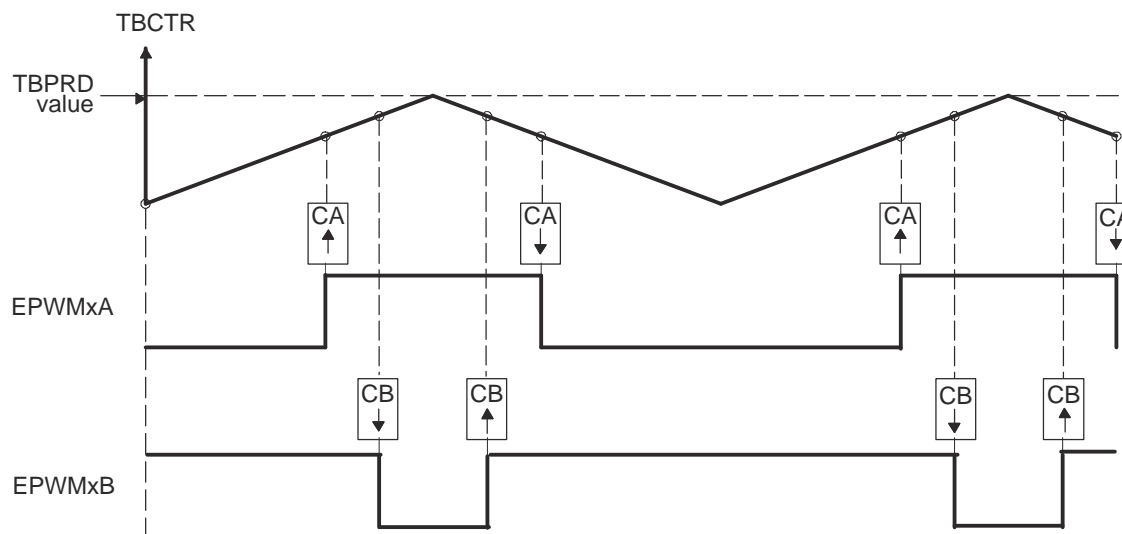
- PWM period =  $2 \times \text{TBPRD} \times T_{\text{TBCLK}}$
- Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
- Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
- Outputs EPWMxA and EPWMxB can drive independent power switches

**Figure 3-25. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Active Low**

#### Example 3-4. Code Sample for Figure 3-25

```
// Initialization Time
// =====
EPwm1Regs.TBPRD = 600; // Period = 2'600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 400; // Compare A = 400 TBCLK counts
EPwm1Regs.CMPB = 500; // Compare B = 500 TBCLK counts
EPwm1Regs.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTR = 0; // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetric
xEPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
xEPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // TBCLK = SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.CBU = AQ_SET;
EPwm1Regs.AQCTLB.bit.CBD = AQ_CLEAR;
//
// Run Time
// =====
EPwm1Regs.CMPA.half.CMPA = Duty1A; // adjust duty for output EPWM1A
EPwm1Regs.CMPB = Duty1B; // adjust duty for output EPWM1B
```

**Example 3-5** contains a code sample showing initialization and run time for the waveforms in [Figure 3-26](#). Use the constant definitions in the device-specific header file.



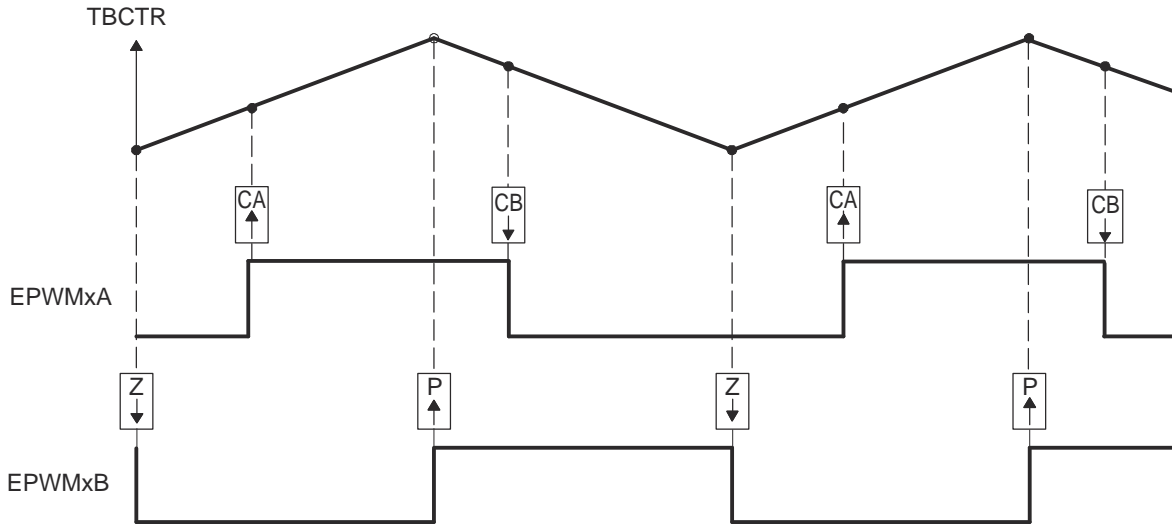
- PWM period =  $2 \times \text{TBPRD} \times T_{\text{TBCLK}}$
- Duty modulation for EPWMxA is set by CMPA, and is active low, that is, low time duty proportional to CMPA
- Duty modulation for EPWMxB is set by CMPB and is active high, that is, high time duty proportional to CMPB
- Outputs EPWMx can drive upper/lower (complementary) power switches
- Dead-band =  $\text{CMPB} - \text{CMPA}$  (fully programmable edge placement by software). Note the dead-band module is also available if the more classical edge delay method is required.

**Figure 3-26. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Complementary**

**Example 3-5. Code Sample for [Figure 3-26](#)**

```
// Initialization Time
// =====
EPwm1Regs.TBPRD = 600; // Period = 2'600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350; // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 400; // Compare B = 400 TBCLK counts
EPwm1Regs.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTR = 0; // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetric
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // TBCLK = SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.CBD = AQ_SET;
// Run Time
// =====
EPwm1Regs.CMPA.half.CMPA = Duty1A; // adjust duty for output EPWM1A
EPwm1Regs.CMPB = Duty1B; // adjust duty for output EPWM1B
```

Example 3-6 contains a code sample showing initialization and run time for the waveforms in Figure 3-27. Use the constant definitions in the device-specific header file.



- A. PWM period =  $2 \times \text{TBPRD} \times \text{TBCLK}$
- B. Rising edge and falling edge can be asymmetrically positioned within a PWM cycle. This allows for pulse placement techniques.
- C. Duty modulation for EPWMxA is set by CMPA and CMPB.
- D. Low time duty for EPWMxA is proportional to  $(\text{CMPA} + \text{CMPB})$ .
- E. To change this example to active high, CMPA and CMPB actions need to be inverted (that is, Set ! Clear and Clear Set).
- F. Duty modulation for EPWMxB is fixed at 50% (utilizes spare action resources for EPWMxB)

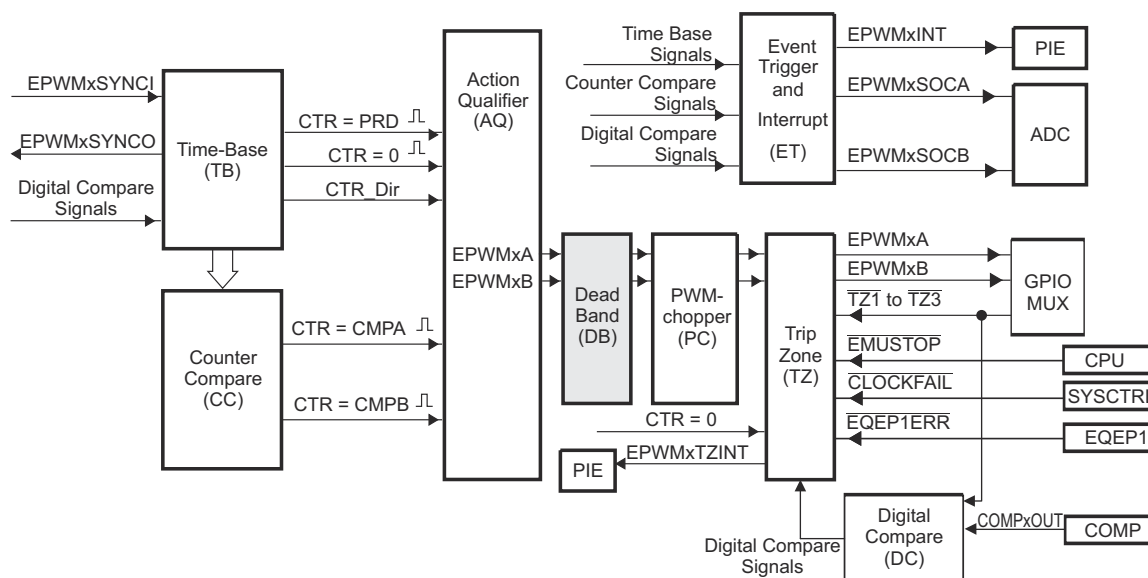
**Figure 3-27. Up-Down-Count, Dual Edge Asymmetric Waveform, With Independent Modulation on EPWMxA—Active Low**

**Example 3-6. Code Sample for Figure 3-27**

```
// Initialization Time
// =====
EPwm1Regs.TBPRD = 600; // Period = 2 ^ 600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 250; // Compare A = 250 TBCLK counts
EPwm1Regs.CMPB = 450; // Compare B = 450 TBCLK counts
EPwm1Regs.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTR = 0; // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetric
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // TBCLK = SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
EPwm1Regs.AQCTLA.bit.CBD = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.PRD = AQ_SET;
// Run Time
// =====
EPwm1Regs.CMPA.half.CMPA = EdgePosA; // adjust duty for output EPWM1A only
EPwm1Regs.CMPB = EdgePosB;
```

### 3.2.5 Dead-Band Generator (DB) Submodule

Figure 3-28 illustrates the dead-band submodule within the ePWM module.



**Figure 3-28. Dead-Band Submodule**

#### 3.2.5.1 Purpose of the Dead-Band Submodule

The "Action-qualifier (AQ) Module" section discussed how it is possible to generate the required dead-band by having full control over edge placement using both the CMPA and CMPB resources of the ePWM module. However, if the more classical edge delay-based dead-band with polarity control is required, then the dead-band submodule described here should be used.

The key functions of the dead-band module are:

- Generating appropriate signal pairs (EPWMxA and EPWMxB) with dead-band relationship from a single EPWMxA input
- Programming signal pairs for:
  - Active high (AH)
  - Active low (AL)
  - Active high complementary (AHC)
  - Active low complementary (ALC)
- Adding programmable delay to rising edges (RED)
- Adding programmable delay to falling edges (FED)
- Can be totally bypassed from the signal path (note dotted lines in diagram)

#### 3.2.5.2 Controlling and Monitoring the Dead-Band Submodule

The dead-band submodule operation is controlled and monitored via the following registers:

**Table 3-13. Dead-Band Generator Submodule Registers**

Register	Address Offset	Shadowed	Description	Bit Description
DBCTL	0x000F	No	Dead-Band Control Register	<a href="#">Section 3.4.4.1</a>
DBRED	0x0010	No	Dead-Band Rising Edge Delay Count Register	<a href="#">Section 3.4.4.2</a>
DBFED	0x0011	No	Dead-Band Falling Edge Delay Count Register	<a href="#">Section 3.4.4.3</a>

### 3.2.5.3 Operational Highlights for the Dead-Band Submodule

The following sections provide the operational highlights. The dead-band submodule has two groups of independent selection options as shown in Figure 3-29.

- Input Source Selection:** The input signals to the dead-band module are the EPWMxA and EPWMxB output signals from the action-qualifier. In this section they will be referred to as EPWMxA In and EPWMxB In. Using the DBCTL[IN\_MODE] control bits, the signal source for each delay, falling-edge or rising-edge, can be selected:
  - EPWMxA In is the source for both falling-edge and rising-edge delay. This is the default mode.
  - EPWMxA In is the source for falling-edge delay, EPWMxB In is the source for rising-edge delay.
  - EPWMxA In is the source for rising edge delay, EPWMxB In is the source for falling-edge delay.
  - EPWMxB In is the source for both falling-edge and rising-edge delay.
- Half Cycle Clocking:** The dead-band submodule can be clocked using half cycle clocking to double the resolution (that is, counter clocked at  $2 \times$  TBCLK).
- Output Mode Control:** The output mode is configured by way of the DBCTL[OUT\_MODE] bits. These bits determine if the falling-edge delay, rising-edge delay, neither, or both are applied to the input signals.
- Polarity Control:** The polarity control (DBCTL[POLSEL]) allows you to specify whether the rising-edge delayed signal and/or the falling-edge delayed signal is to be inverted before being sent out of the dead-band submodule.

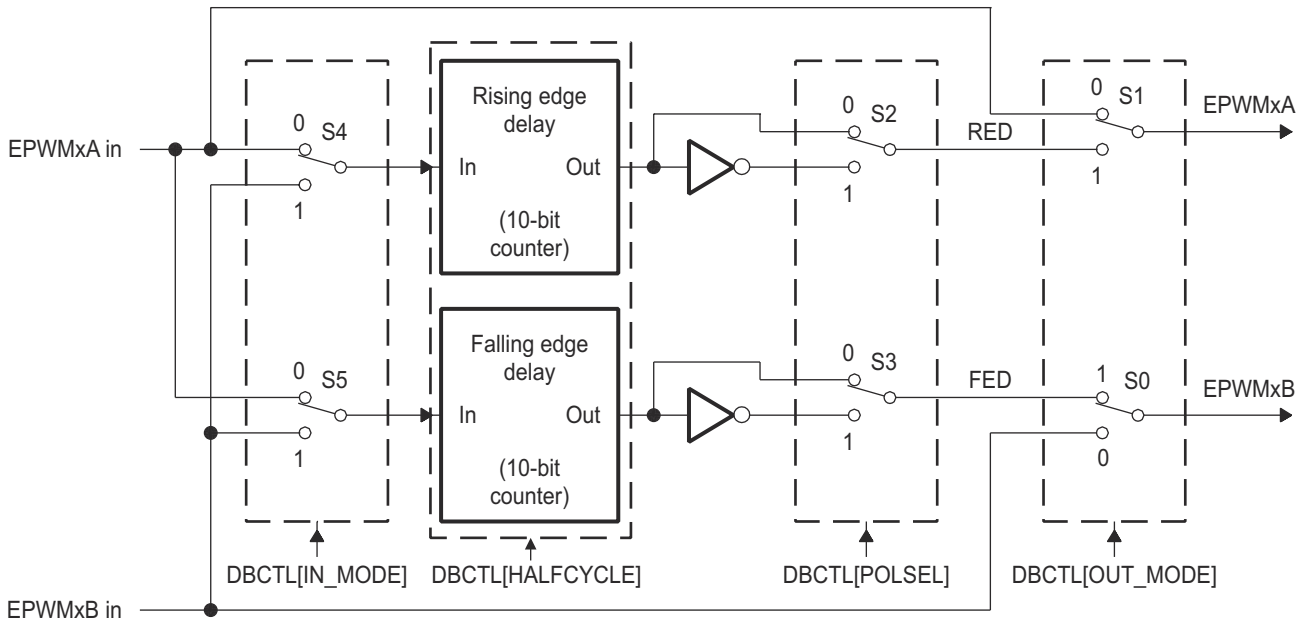


Figure 3-29. Configuration Options for the Dead-Band Submodule

Although all combinations are supported, not all are typical usage modes. Table 3-14 documents some classical dead-band configurations. These modes assume that the DBCTL[IN\_MODE] is configured such that EPWMxA In is the source for both falling-edge and rising-edge delay. Enhanced, or non-traditional modes can be achieved by changing the input signal source. The modes shown in Table 3-14 fall into the following categories:

- Mode 1: Bypass both falling-edge delay (FED) and rising-edge delay (RED).** Allows you to fully disable the dead-band submodule from the PWM signal path.
- Mode 2-5: Classical Dead-Band Polarity Settings.** These represent typical polarity configurations that should address all the active high/low modes required by available industry power switch gate drivers. The waveforms for these typical cases are shown in Figure 3-30. Note that to generate equivalent waveforms to Figure 3-30, configure the action-qualifier submodule to generate the signal as shown for EPWMxA.

- **Mode 6: Bypass rising-edge-delay and Mode 7: Bypass falling-edge-delay.** Finally the last two entries in [Table 3-14](#) show combinations where either the falling-edge-delay (FED) or rising-edge-delay (RED) blocks are bypassed.

**Table 3-14. Classical Dead-Band Operating Modes**

Mode	Mode Description	DBCTL[POLSEL]		DBCTL[OUT_MODE]	
		S3	S2	S1	S0
1	EPWMxA and EPWMxB Passed Through (No Delay)	X	X	0	0
2	Active High Complementary (AHC)	1	0	1	1
3	Active Low Complementary (ALC)	0	1	1	1
4	Active High (AH)	0	0	1	1
5	Active Low (AL)	1	1	1	1
6	EPWMxA Out = EPWMxA In (No Delay)	0 or 1	0 or 1	0	1
	EPWMxB Out = EPWMxA In with Falling Edge Delay				
7	EPWMxA Out = EPWMxA In with Rising Edge Delay	0 or 1	0 or 1	1	0
	EPWMxB Out = EPWMxB In with No Delay				

The dead-band submodule supports independent values for rising-edge (RED) and falling-edge (FED) delays. The amount of delay is programmed using the DBRED and DBFED registers. These are 10-bit registers and their value represents the number of time-base clock, TBCLK, periods a signal edge is delayed by. For example, the formulas to calculate falling-edge-delay and rising-edge-delay are:

$$\text{FED} = \text{DBFED} \times T_{\text{TBCLK}}$$

$$\text{RED} = \text{DBRED} \times T_{\text{TBCLK}}$$

Where  $T_{\text{TBCLK}}$  is the period of TBCLK, the prescaled version of SYSCLKOUT.

When half-cycle clocking is enabled, the formula to calculate the falling-edge-delay and rising-edge-delay becomes:

$$\text{FED} = \text{DBFED} \times T_{\text{TBCLK}}/2$$

$$\text{RED} = \text{DBRED} \times T_{\text{TBCLK}}/2$$

[Figure 3-30](#) shows waveforms for typical cases where  $0\% < \text{duty} < 100\%$ .

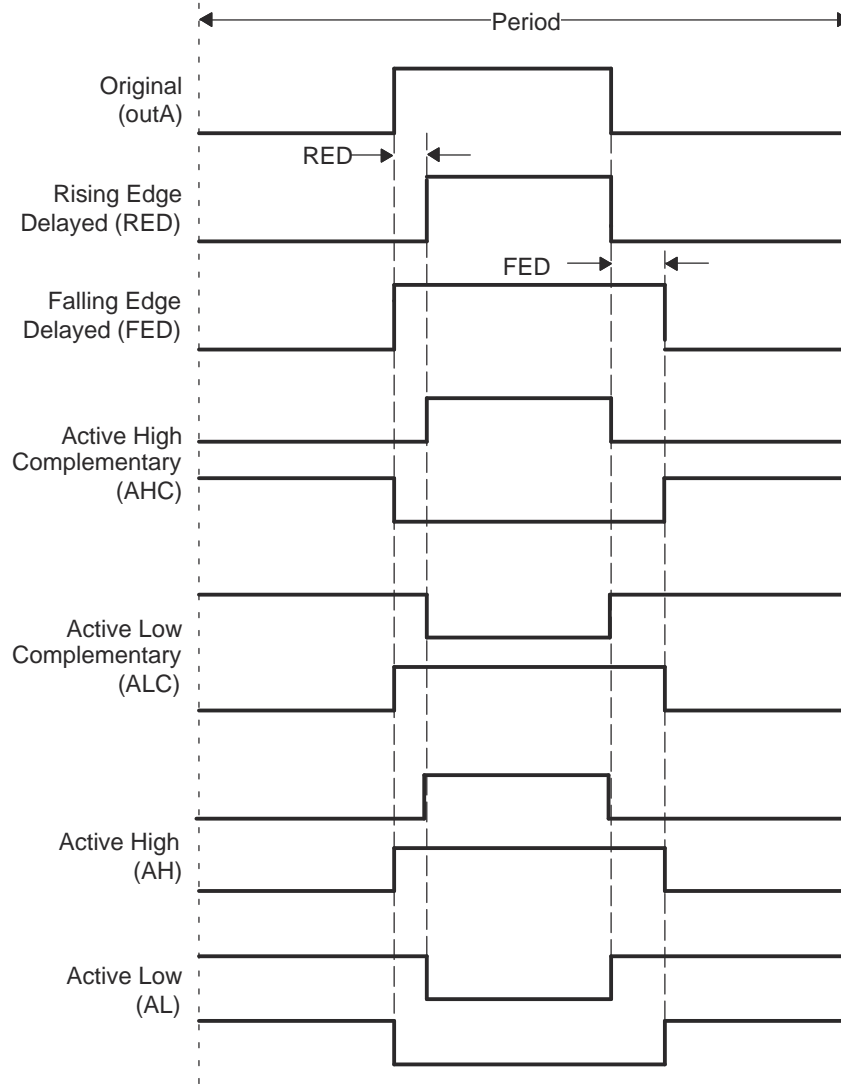
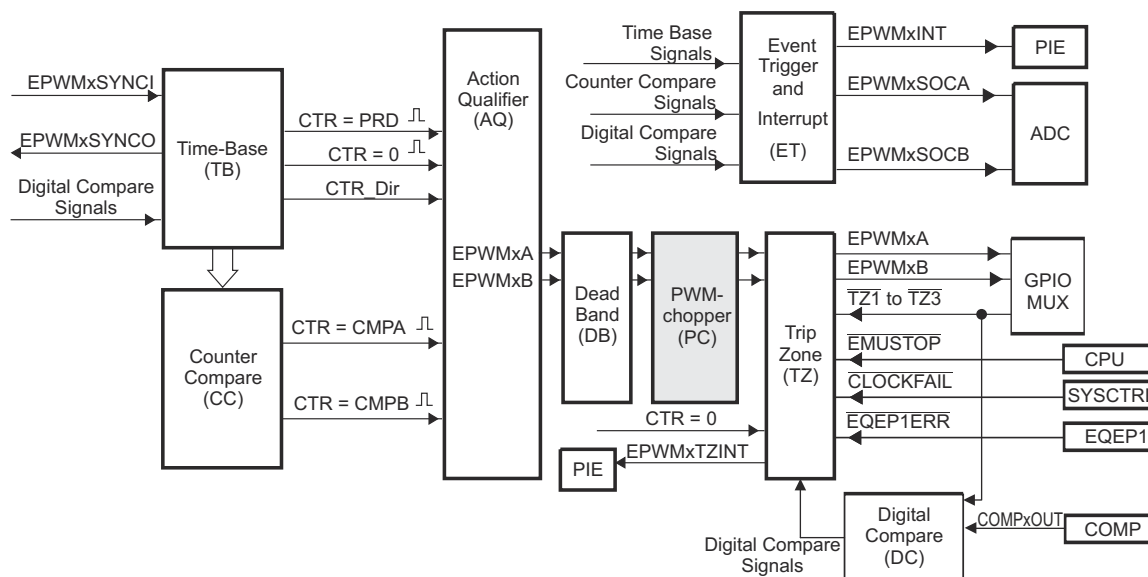


Figure 3-30. Dead-Band Waveforms for Typical Cases (0% < Duty < 100%)

### 3.2.6 PWM-Chopper (PC) Submodule

Figure 3-31 shows the PWM-chopper (PC) submodule within the ePWM module.

The PWM-chopper submodule allows a high-frequency carrier signal to modulate the PWM waveform generated by the action-qualifier and dead-band submodules. This capability is important if you need pulse transformer-based gate drivers to control the power switching elements.



**Figure 3-31. PWM-Chopper Submodule**

#### 3.2.6.1 Purpose of the PWM-Chopper Submodule

The key functions of the PWM-chopper submodule are:

- Programmable chopping (carrier) frequency
- Programmable pulse width of first pulse
- Programmable duty cycle of second and subsequent pulses
- Can be fully bypassed if not required

#### 3.2.6.2 Controlling the PWM-Chopper Submodule

The PWM-chopper submodule operation is controlled via the registers in Table 3-15.

**Table 3-15. PWM-Chopper Submodule Registers**

Register	Address Offset	Shadowed	Description	Bit Description
PCCTL	0x001E	No	PWM-chopper Control Register	<a href="#">Section 3.4.7.1</a>

#### 3.2.6.3 Operational Highlights for the PWM-Chopper Submodule

Figure 3-32 shows the operational details of the PWM-chopper submodule. The carrier clock is derived from SYSCLKOUT. Its frequency and duty cycle are controlled via the CHPFREQ and CHPDUTY bits in the PCCTL register. The one-shot block is a feature that provides a high energy first pulse to ensure hard and fast power switch turn on, while the subsequent pulses sustain pulses, ensuring the power switch remains on. The one-shot width is programmed via the OSHTWTH bits. The PWM-chopper submodule can be fully disabled (bypassed) via the CHPEN bit.



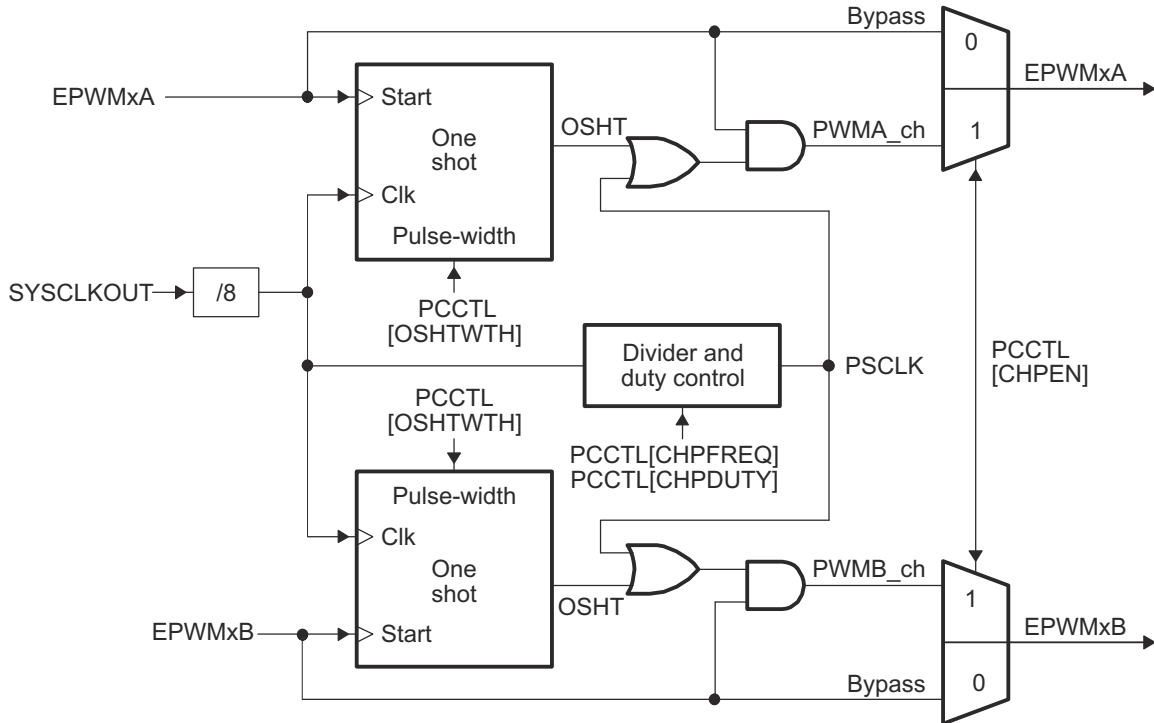


Figure 3-32. PWM-Chopper Submodule Operational Details

### 3.2.6.4 Waveforms

Figure 3-33 shows simplified waveforms of the chopping action only; one-shot and duty-cycle control are not shown. Details of the one-shot and duty-cycle control are discussed in the following sections.

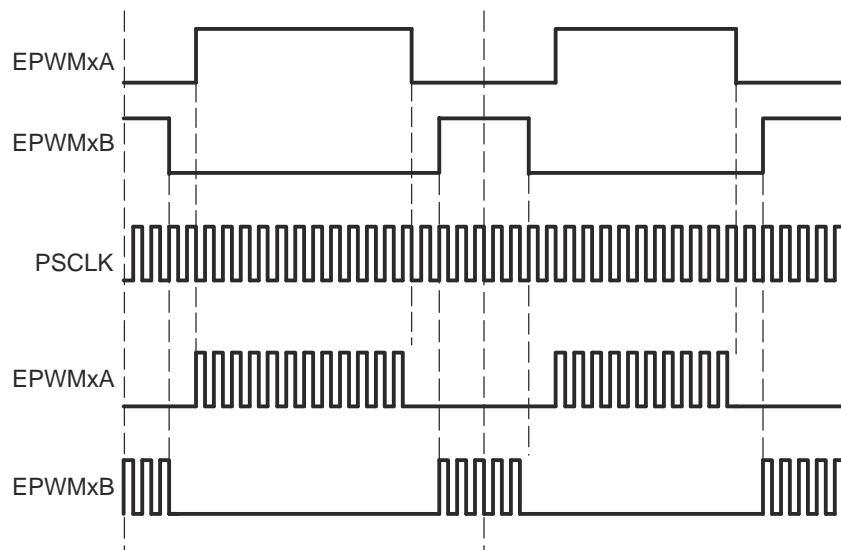


Figure 3-33. Simple PWM-Chopper Submodule Waveforms Showing Chopping Action Only

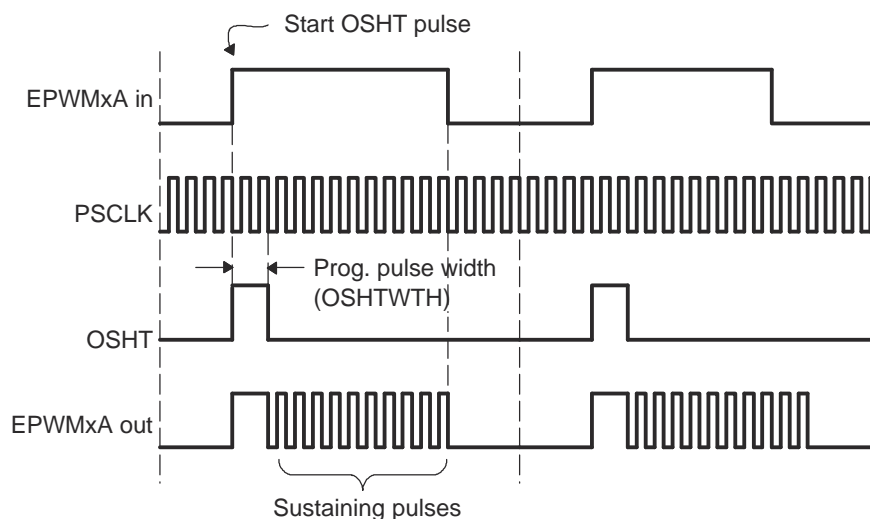
### 3.2.6.4.1 One-Shot Pulse

The width of the first pulse can be programmed to any of 16 possible pulse width values. The width or period of the first pulse is given by:

$$T_{1\text{stpulse}} = T_{\text{SYSCLKOUT}} \times 8 \times \text{OSHTWTH}$$

Where  $T_{\text{SYSCLKOUT}}$  is the period of the system clock (SYSCLKOUT) and OSHTWTH is the four control bits (value from 1 to 16).

Figure 3-34 shows the first and subsequent sustaining pulses and Table 3-16 gives the possible pulse width values for a SYSCLKOUT = 60 MHz.



**Figure 3-34. PWM-Chopper Submodule Waveforms Showing the First Pulse and Subsequent Sustaining Pulses**

**Table 3-16. Possible Pulse Width Values for  
SYSCLKOUT = 60 MHz**

OSHTWTHz (hex)	Pulse Width (nS)
0	133
1	267
2	400
3	533
4	667
5	800
6	933
7	1067
8	1200
9	1333
A	1467
B	1600
C	1733
D	1867
E	2000
F	2133

### 3.2.6.4.2 Duty Cycle Control

Pulse transformer-based gate drive designs need to comprehend the magnetic properties or characteristics of the transformer and associated circuitry. Saturation is one such consideration. To assist the gate drive designer, the duty cycles of the second and subsequent pulses have been made programmable. These sustaining pulses ensure the correct drive strength and polarity is maintained on the power switch gate during the on period, and hence a programmable duty cycle allows a design to be tuned or optimized via software control.

Figure 3-35 shows the duty cycle control that is possible by programming the CHPDUTY bits. One of seven possible duty ratios can be selected ranging from 12.5% to 87.5%.

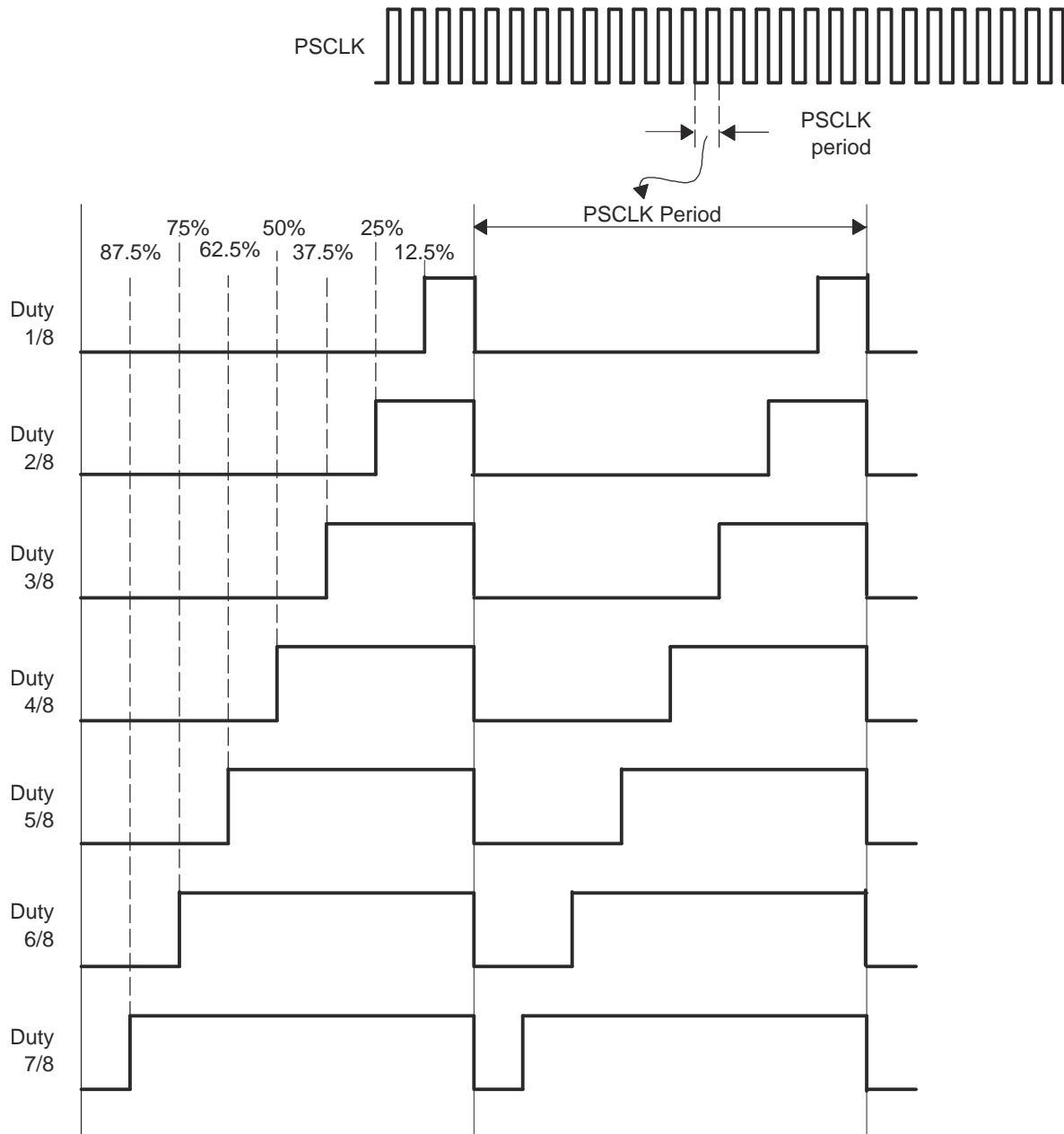
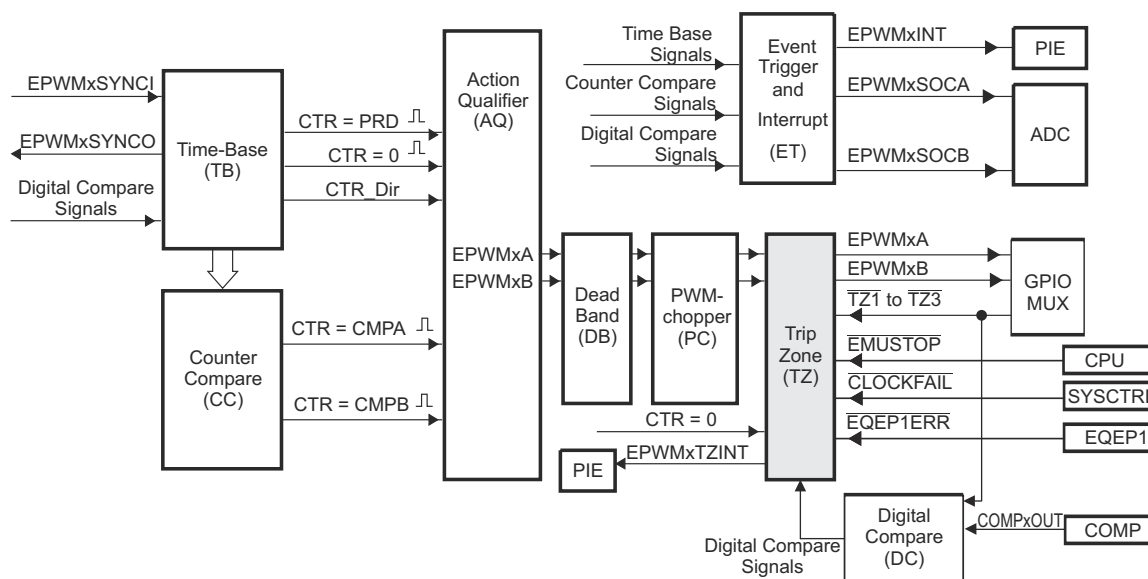


Figure 3-35. PWM-Chopper Submodule Waveforms Showing the Pulse Width (Duty Cycle) Control of Sustaining Pulses

### 3.2.7 Trip-Zone (TZ) Submodule

Figure 3-36 shows how the trip-zone (TZ) submodule fits within the ePWM module.

Each ePWM module is connected to six  $\overline{TZn}$  signals ( $\overline{TZ1}$  to  $\overline{TZ6}$ ).  $\overline{TZ1}$  to  $\overline{TZ3}$  are sourced from the GPIO mux.  $\overline{TZ4}$  is sourced from an inverted EQEP1ERR signal on those devices with an EQEP1 module.  $\overline{TZ5}$  is connected to the system clock fail logic, and  $\overline{TZ6}$  is sourced from the EMUSTOP output from the CPU. These signals indicate external fault or trip conditions, and the ePWM outputs can be programmed to respond accordingly when faults occur.



**Figure 3-36. Trip-Zone Submodule**

#### 3.2.7.1 Purpose of the Trip-Zone Submodule

The key functions of the Trip-Zone submodule are:

- Trip inputs  $\overline{TZ1}$  to  $\overline{TZ6}$  can be flexibly mapped to any ePWM module.
- Upon a fault condition, outputs EPWMxA and EPWMxB can be forced to one of the following:
  - High
  - Low
  - High-impedance
  - No action taken
- Support for one-shot trip (OSHT) for major short circuits or over-current conditions.
- Support for cycle-by-cycle tripping (CBC) for current limiting operation.
- Support for digital compare tripping (DC) based on state of on-chip analog comparator module outputs and/or  $\overline{TZ1}$  to  $\overline{TZ3}$  signals.
- Each trip-zone input and digital compare (DC) submodule DCAEVT1/2 or DCBEVT1/2 force event can be allocated to either one-shot or cycle-by-cycle operation.
- Interrupt generation is possible on any trip-zone input.
- Software-forced tripping is also supported.
- The trip-zone submodule can be fully bypassed if it is not required.

### 3.2.7.2 Controlling and Monitoring the Trip-Zone Submodule

The trip-zone submodule operation is controlled and monitored through the following registers:

**Table 3-17. Trip-Zone Submodule Registers**

Register	Address Offset	Shadowed	Description <sup>(1)</sup>	Bit Description
TZSEL	0x0012	No	Trip-Zone Select Register	<a href="#">Section 3.4.5.1</a>
TZDCSEL	0x0013	No	Trip-zone Digital Compare Select Register <sup>(2)</sup>	<a href="#">Section 3.4.5.2</a>
TZCTL	0x0014	No	Trip-Zone Control Register	<a href="#">Section 3.4.5.3</a>
TZEINT	0x0015	No	Trip-Zone Enable Interrupt Register	<a href="#">Section 3.4.5.4</a>
TZFLG	0x0016	No	Trip-Zone Flag Register	<a href="#">Section 3.4.5.5</a>
TZCLR	0x0017	No	Trip-Zone Clear Register	<a href="#">Section 3.4.5.6</a>
TZFRC	0x0018	No	Trip-Zone Force Register	<a href="#">Section 3.4.5.7</a>

- (1) All trip-zone registers are EALLOW protected and can be modified only after executing the EALLOW instruction. For more information, see the *System Control and Interrupts* chapter.
- (2) This register is discussed in more detail in [Section 3.2.9](#).

### 3.2.7.3 Operational Highlights for the Trip-Zone Submodule

The following sections describe the operational highlights and configuration options for the trip-zone submodule.

The trip-zone signals  $\overline{TZ1}$  to  $\overline{TZ6}$  (also collectively referred to as  $\overline{TZn}$ ) are active low input signals. When one of these signals goes low, or when a DCAEVT1/2 or DCBEVT1/2 force happens based on the TZDCSEL register event selection, it indicates that a trip event has occurred. Each ePWM module can be individually configured to ignore or use each of the trip-zone signals or DC events. Which trip-zone signals or DC events are used by a particular ePWM module is determined by the TZSEL register for that specific ePWM module. The trip-zone signals may or may not be synchronized to the system clock (SYSCLKOUT) and digitally filtered within the GPIO MUX block. A minimum of  $3 \cdot TBCLK$  low pulse width on  $\overline{TZn}$  inputs is sufficient to trigger a fault condition on the ePWM module. If the pulse width is less than this, the trip condition may not be latched by CBC or OST latches. The asynchronous trip makes sure that if clocks are missing for any reason, the outputs can still be tripped by a valid event present on  $\overline{TZn}$  inputs. The GPIOs or peripherals must be appropriately configured. For more information, see the *System Control and Interrupts* chapter.

Each  $\overline{TZn}$  input can be individually configured to provide either a cycle-by-cycle or one-shot trip event for an ePWM module. DCAEVT1 and DCBEVT1 events can be configured to directly trip an ePWM module or provide a one-shot trip event to the module. Likewise, DCAEVT2 and DCBEVT2 events can also be configured to directly trip an ePWM module or provide a cycle-by-cycle trip event to the module. This configuration is determined by the TZSEL[DCAEVT1/2], TZSEL[DCBEVT1/2], TZSEL[CBCn], and TZSEL[OSHTn] control bits (where n corresponds to the trip input) respectively.

- **Cycle-by-Cycle (CBC):** When a cycle-by-cycle trip event occurs, the action specified in the TZCTL[TZA] and TZCTL[TZB] bits is carried out immediately on the EPWMxA and/or EPWMxB output. [Table 3-18](#) lists the possible actions. In addition, the cycle-by-cycle trip event flag (TZFLG[CBC]) is set and a EPWMx\_TZINT interrupt is generated if it is enabled in the TZEINT register and PIE peripheral.

If the CBC interrupt is enabled via the TZEINT register, and DCAEVT2 or DCBEVT2 are selected as CBC trip sources via the TZSEL register, it is not necessary to also enable the DCAEVT2 or DCBEVT2 interrupts in the TZEINT register, as the DC events trigger interrupts through the CBC mechanism.

The specified condition on the inputs is automatically cleared when the ePWM time-base counter reaches zero (TBCTR = 0x0000) if the trip event is no longer present. Therefore, in this mode, the trip event is cleared or reset every PWM cycle. The TZFLG[CBC] flag bit will remain set until it is manually cleared by writing to the TZCLR[CBC] bit. If the cycle-by-cycle trip event is still present when the TZFLG[CBC] bit is cleared, then it will again be immediately set.

- One-Shot (OSHT):** When a one-shot trip event occurs, the action specified in the TZCTL[TZA] and TZCTL[TZB] bits is carried out immediately on the EPWMxA and/or EPWMxB output. [Table 3-18](#) lists the possible actions. In addition, the one-shot trip event flag (TZFLG[OST]) is set and a EPWMx\_TZINT interrupt is generated if it is enabled in the TZEINT register and PIE peripheral. The one-shot trip condition must be cleared manually by writing to the TZCLR[OST] bit.

If the one-shot interrupt is enabled via the TZEINT register, and DCAEVT1 or DCBEVT1 are selected as OSHT trip sources via the TZSEL register, it is not necessary to also enable the DCAEVT1 or DCBEVT1 interrupts in the TZEINT register, as the DC events trigger interrupts through the OSHT mechanism.

- Digital Compare Events (DCAEVT1/2 and DCBEVT1/2):** A digital compare DCAEVT1/2 or DCBEVT1/2 event is generated based on a combination of the DCAH/DCAL and DCBH/DCBL signals as selected by the TZDCSEL register. The signals which source the DCAH/DCAL and DCBH/DCBL signals are selected via the DCTRISEL register and can be either trip zone input pins or analog comparator COMPxOUT signals. For more information on the digital compare submodule signals, see [Section 3.2.9](#).

When a digital compare event occurs, the action specified in the TZCTL[DCAEVT1/2] and TZCTL[DCBEVT1/2] bits is carried out immediately on the EPWMxA and/or EPWMxB output. [Table 3-18](#) lists the possible actions. In addition, the relevant DC trip event flag (TZFLG[DCAEVT1/2] / TZFLG[DCBEVT1/2]) is set and a EPWMx\_TZINT interrupt is generated if it is enabled in the TZEINT register and PIE peripheral.

The specified condition on the pins is automatically cleared when the DC trip event is no longer present. The TZFLG[DCAEVT1/2] or TZFLG[DCBEVT1/2] flag bit will remain set until it is manually cleared by writing to the TZCLR[DCAEVT1/2] or TZCLR[DCBEVT1/2] bit. If the DC trip event is still present when the TZFLG[DCAEVT1/2] or TZFLG[DCBEVT1/2] flag is cleared, then it will again be immediately set.

The action taken when a trip event occurs can be configured individually for each of the ePWM output pins by way of the TZCTL register bit fields. One of four possible actions, shown in [Table 3-18](#), can be taken on a trip event.

**Table 3-18. Possible Actions On a Trip Event**

TZCTL Register Bit Settings	EPWMxA and/or EPWMxB	Comment
0,0	High-Impedance	Tripped
0,1	Force to High State	Tripped
1,0	Force to Low State	Tripped
1,1	No Change	Do Nothing. No change is made to the output.

### Example 3-7. Trip-Zone Configurations

#### Scenario A:

A one-shot trip event on  $\overline{TZ1}$  pulls both EPWM1A, EPWM1B low and also forces EPWM2A and EPWM2B high.

- Configure the ePWM1 registers as follows:
  - TZSEL[OSHT1] = 1: enables  $\overline{TZ1}$  as a one-shot event source for ePWM1
  - TZCTL[TZA] = 2: EPWM1A will be forced low on a trip event.
  - TZCTL[TZB] = 2: EPWM1B will be forced low on a trip event.
- Configure the ePWM2 registers as follows:
  - TZSEL[OSHT1] = 1: enables  $\overline{TZ1}$  as a one-shot event source for ePWM2
  - TZCTL[TZA] = 1: EPWM2A will be forced high on a trip event.
  - TZCTL[TZB] = 1: EPWM2B will be forced high on a trip event.

#### Scenario B:

A cycle-by-cycle event on  $\overline{TZ5}$  pulls both EPWM1A, EPWM1B low.

A one-shot event on  $\overline{TZ1}$  or  $\overline{TZ6}$  puts EPWM2A into a high impedance state.

- Configure the ePWM1 registers as follows:
  - TZSEL[CBC5] = 1: enables  $\overline{TZ5}$  as a one-shot event source for ePWM1
  - TZCTL[TZA] = 2: EPWM1A will be forced low on a trip event.
  - TZCTL[TZB] = 2: EPWM1B will be forced low on a trip event.
- Configure the ePWM2 registers as follows:
  - TZSEL[OSHT1] = 1: enables  $\overline{TZ1}$  as a one-shot event source for ePWM2
  - TZSEL[OSHT6] = 1: enables  $\overline{TZ6}$  as a one-shot event source for ePWM2
  - TZCTL[TZA] = 0: EPWM2A will be put into a high-impedance state on a trip event.
  - TZCTL[TZB] = 3: EPWM2B will ignore the trip event.

### 3.2.7.4 Generating Trip Event Interrupts

Figure 3-37 and Figure 3-38 illustrate the trip-zone submodule control and interrupt logic, respectively. DCAEVT1/2 and DCBEVT1/2 signals are described in further detail in Section 3.2.9.

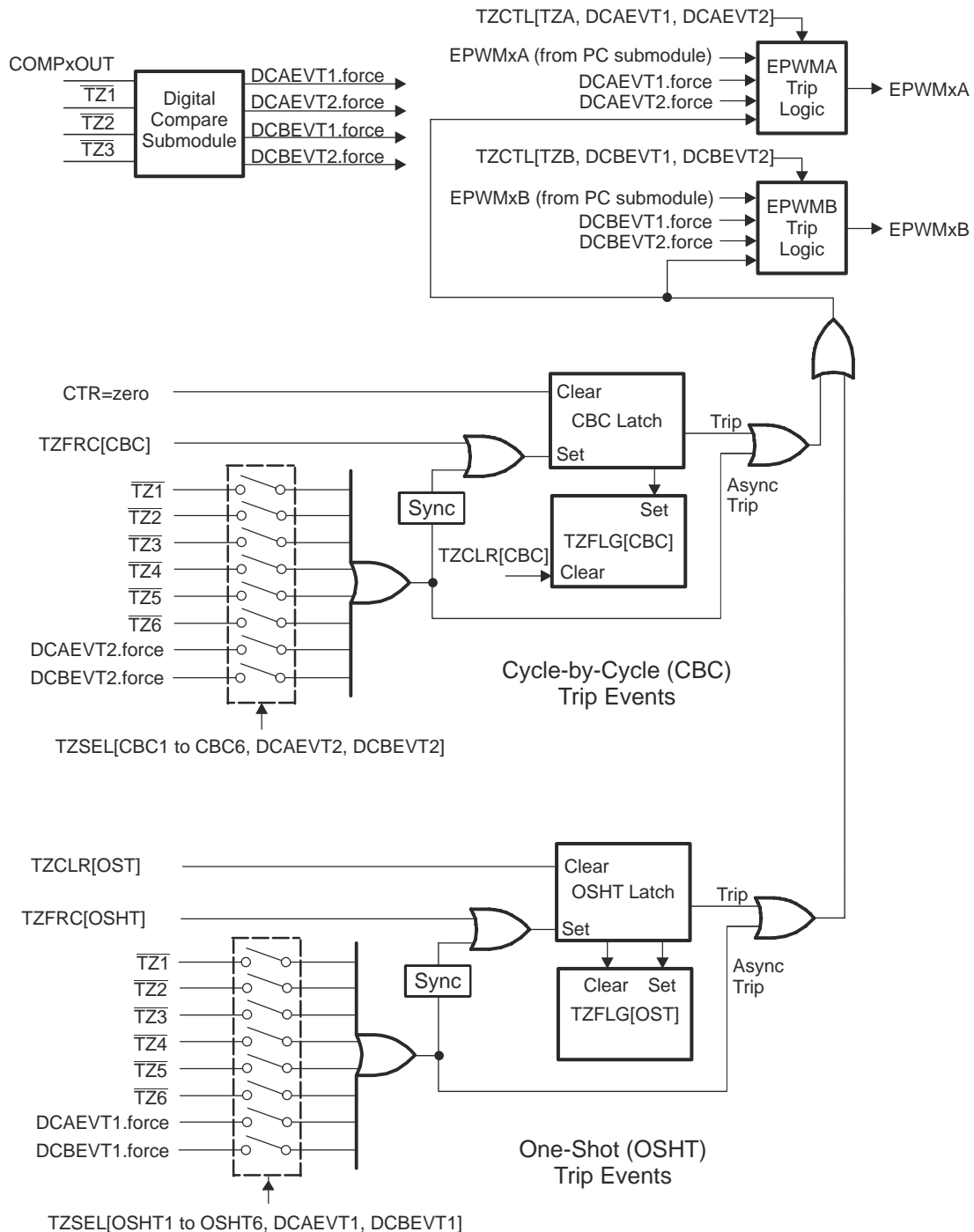


Figure 3-37. Trip-Zone Submodule Mode Control Logic



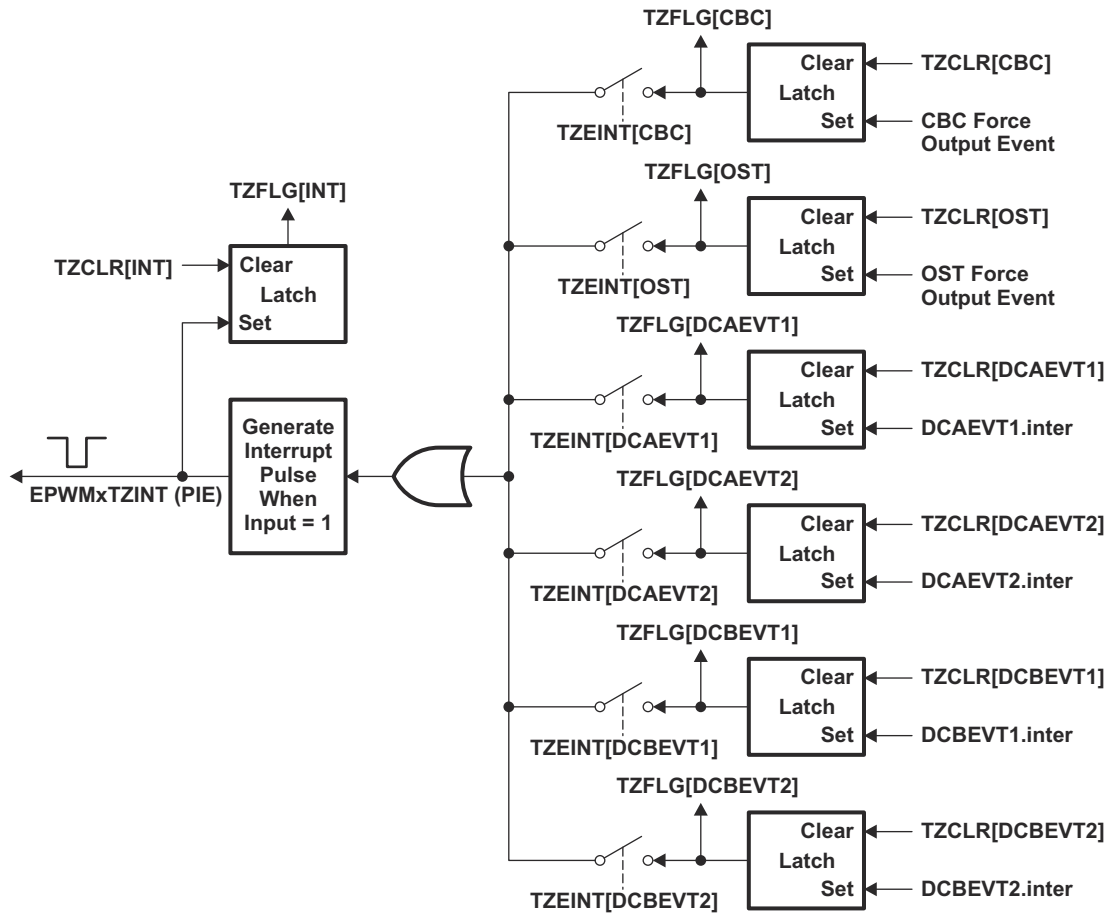


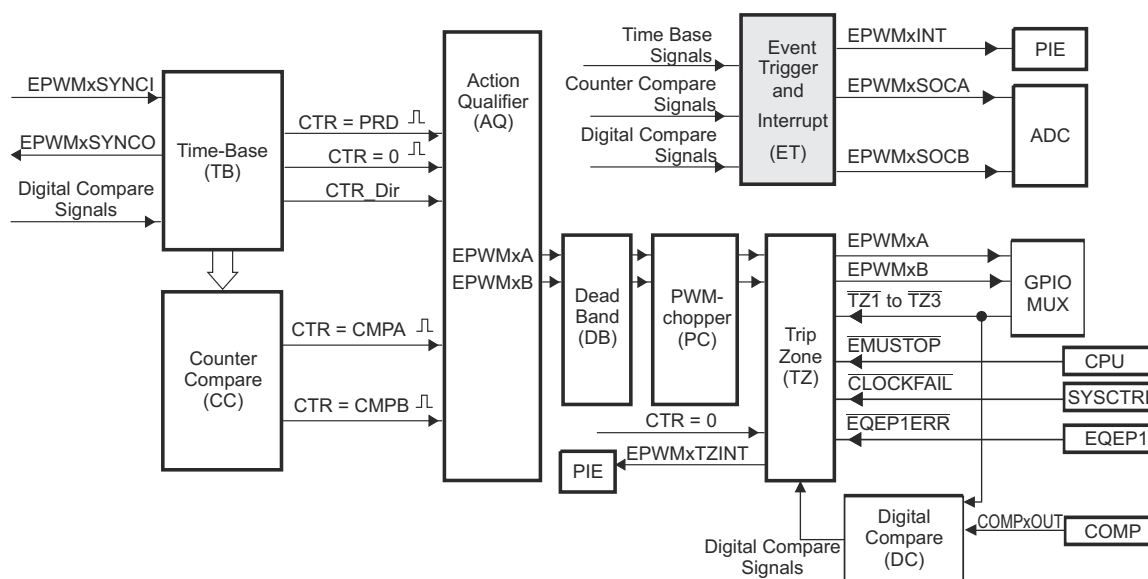
Figure 3-38. Trip-Zone Submodule Interrupt Logic

### 3.2.8 Event-Trigger (ET) Submodule

The key functions of the event-trigger submodule are:

- Receives event inputs generated by the time-base, counter-compare, and digital-compare submodules
- Uses the time-base direction information for up/down event qualification
- Uses prescaling logic to issue interrupt requests and ADC start of conversion at:
  - Every event
  - Every second event
  - Every third event
- Provides full visibility of event generation via event counters and flags
- Allows software forcing of Interrupts and ADC start of conversion

The event-trigger submodule manages the events generated by the time-base submodule, the counter-compare submodule, and the digital-compare submodule to generate an interrupt to the CPU and/or a start of conversion pulse to the ADC when a selected event occurs. Figure 3-39 illustrates where the event-trigger submodule fits within the ePWM system.



**Figure 3-39. Event-Trigger Submodule**

#### 3.2.8.1 Purpose of the Event-Trigger Submodule

The following sections describe the event-trigger submodule.

Each ePWM module has one interrupt request line connected to the PIE and two start of conversion signals connected to the ADC module. As shown in Figure 3-40, ADC start of conversion for all ePWM modules are connected to individual ADC trigger inputs to the ADC, and hence multiple modules can initiate an ADC start of conversion via the ADC trigger inputs.

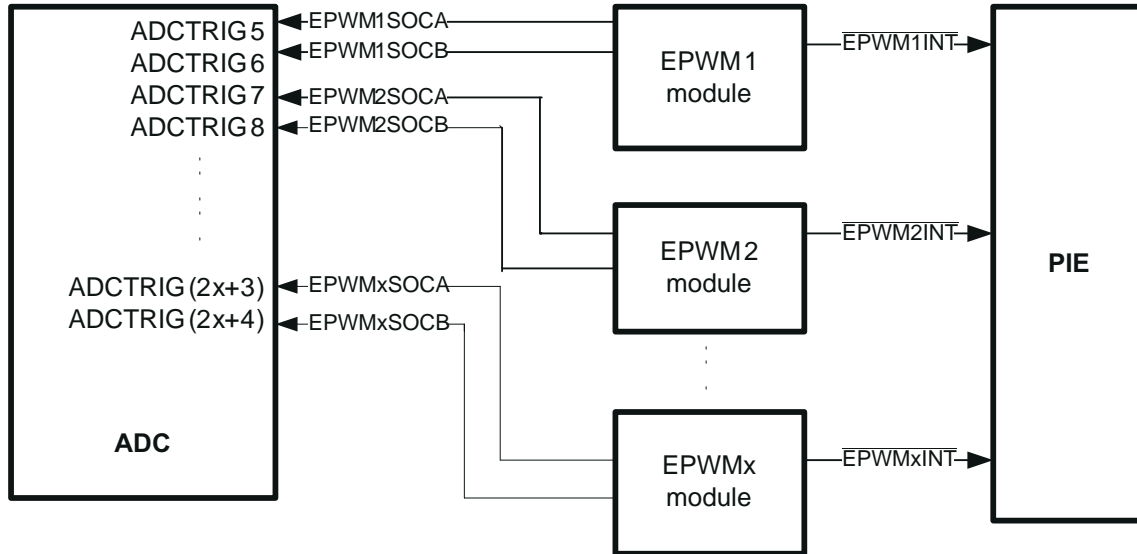


Figure 3-40. Event-Trigger Submodule Inter-Connectivity of ADC Start of Conversion

The event-trigger submodule monitors various event conditions (the left side inputs to event-trigger submodule shown in Figure 3-41) and can be configured to prescale these events before issuing an Interrupt request or an ADC start of conversion. The event-trigger prescaling logic can issue Interrupt requests and ADC start of conversion at:

- Every event
- Every second event
- Every third event

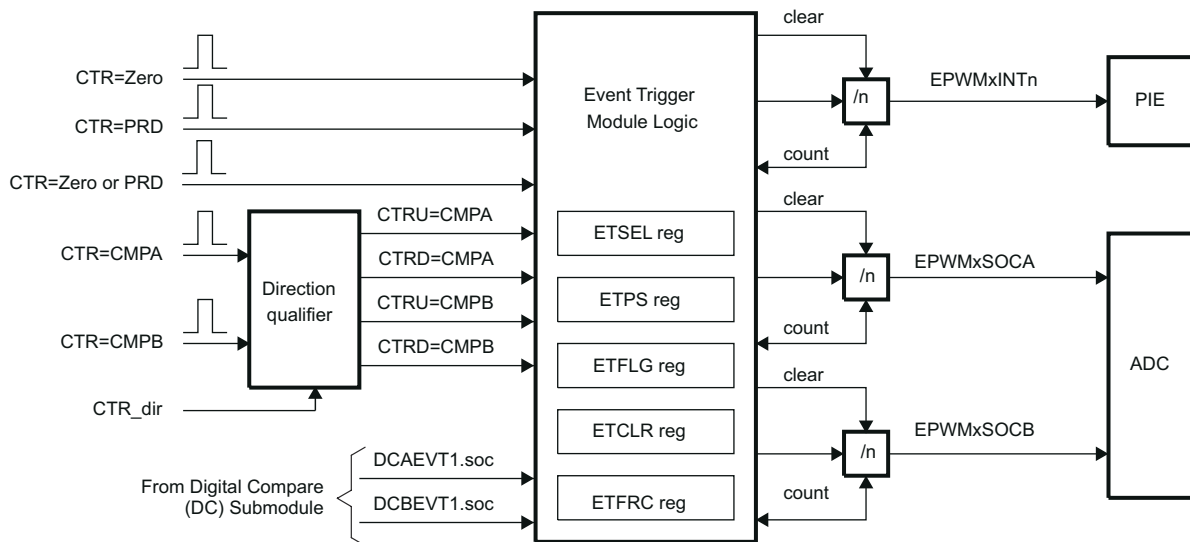


Figure 3-41. Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs

### 3.2.8.2 Controlling and Monitoring the Event-Trigger Submodule

The key registers used to configure the event-trigger submodule are shown in [Table 3-19](#).

**Table 3-19. Event-Trigger Submodule Registers**

Register	Address Offset	Shadowed	Description	Bit Description
ETSEL	0x0019	No	Event-trigger Selection Register	<a href="#">Section 3.4.6.1</a>
ETPS	0x001A	No	Event-trigger Prescale Register	<a href="#">Section 3.4.6.2</a>
ETFLG	0x001B	No	Event-trigger Flag Register	<a href="#">Section 3.4.6.3</a>
ETCLR	0x001C	No	Event-trigger Clear Register	<a href="#">Section 3.4.6.4</a>
ETFRC	0x001D	No	Event-trigger Force Register	<a href="#">Section 3.4.6.5</a>

- ETSEL—This selects which of the possible events will trigger an interrupt or start an ADC conversion
- ETPS—This programs the event prescaling options mentioned above.
- ETFLG—These are flag bits indicating status of the selected and prescaled events.
- ETCLR—These bits allow you to clear the flag bits in the ETFLG register via software.
- ETFRC—These bits allow software forcing of an event. Useful for debugging or s/w intervention.

### 3.2.8.3 Operational Highlights for the Event-Trigger Submodule

A more detailed look at how the various register bits interact with the Interrupt and ADC start of conversion logic are shown in [Figure 3-42](#), [Figure 3-43](#), and [Figure 3-44](#).

[Figure 3-42](#) shows the event-trigger's interrupt generation logic. The interrupt-period (ETPS[INTPRD]) bits specify the number of events required to cause an interrupt pulse to be generated. The choices available are:

- Do not generate an interrupt.
- Generate an interrupt on every event
- Generate an interrupt on every second event
- Generate an interrupt on every third event

Which event can cause an interrupt is configured by the interrupt selection (ETSEL[INTSEL]) bits. The event can be one of the following:

- Time-base counter equal to zero (TBCTR = 0x0000).
- Time-base counter equal to period (TBCTR = TBPRD).
- Time-base counter equal to zero or period (TBCTR = 0x0000 || TBCTR = TBPRD)
- Time-base counter equal to the compare A register (CMPA) when the timer is incrementing.
- Time-base counter equal to the compare A register (CMPA) when the timer is decrementing.
- Time-base counter equal to the compare B register (CMPB) when the timer is incrementing.
- Time-base counter equal to the compare B register (CMPB) when the timer is decrementing.

The number of events that have occurred can be read from the interrupt event counter (ETPS[INTCNT]) register bits. That is, when the specified event occurs the ETPS[INTCNT] bits are incremented until they reach the value specified by ETPS[INTPRD]. When ETPS[INTCNT] = ETPS[INTPRD] the counter stops counting and its output is set. The counter is only cleared when an interrupt is sent to the PIE.

When ETPS[INTCNT] reaches ETPS[INTPRD] the following behaviors will occur:

- If interrupts are enabled, ETSEL[INTEN] = 1 and the interrupt flag is clear, ETFLG[INT] = 0, then an interrupt pulse is generated and the interrupt flag is set, ETFLG[INT] = 1, and the event counter is cleared ETPS[INTCNT] = 0. The counter will begin counting events again.
- If interrupts are disabled, ETSEL[INTEN] = 0, or the interrupt flag is set, ETFLG[INT] = 1, the counter stops counting events when it reaches the period value ETPS[INTCNT] = ETPS[INTPRD].
- If interrupts are enabled, but the interrupt flag is already set, then the counter will hold its output high until the ETFLG[INT] flag is cleared. This allows for one interrupt to be pending while one is serviced.

When writing INTPRD values the following occur:

- Writing to the INTPRD bits will automatically clear the counter  $INTCNT = 0$  and the counter output will be reset (so no interrupts are generated).
- Writing an INTPRD value that is GREATER or equal to the current counter value will reset the  $INTCNT = 0$ .
- Writing an INTPRD value that is equal to the current counter value will trigger an interrupt if it is enabled and the status flag is cleared (and  $INTCNT$  will also be cleared to 0)
- Writing an INTPRD value that is LESS than the current counter value will result in undefined behavior (that is,  $INTCNT$  stops counting because  $INTPRD$  is below  $INTCNT$ , and interrupt will never fire).
- Writing a 1 to the ETFRC[INT] bit will increment the event counter  $INTCNT$ . The counter will behave as described above when  $INTCNT = INTPRD$ .
- When  $INTPRD = 0$ , the counter is disabled and hence no events will be detected and the ETFRC[INT] bit is also ignored.

The above definition means that you can generate an interrupt on every event, on every second event, or on every third event. An interrupt cannot be generated on every fourth or more events.

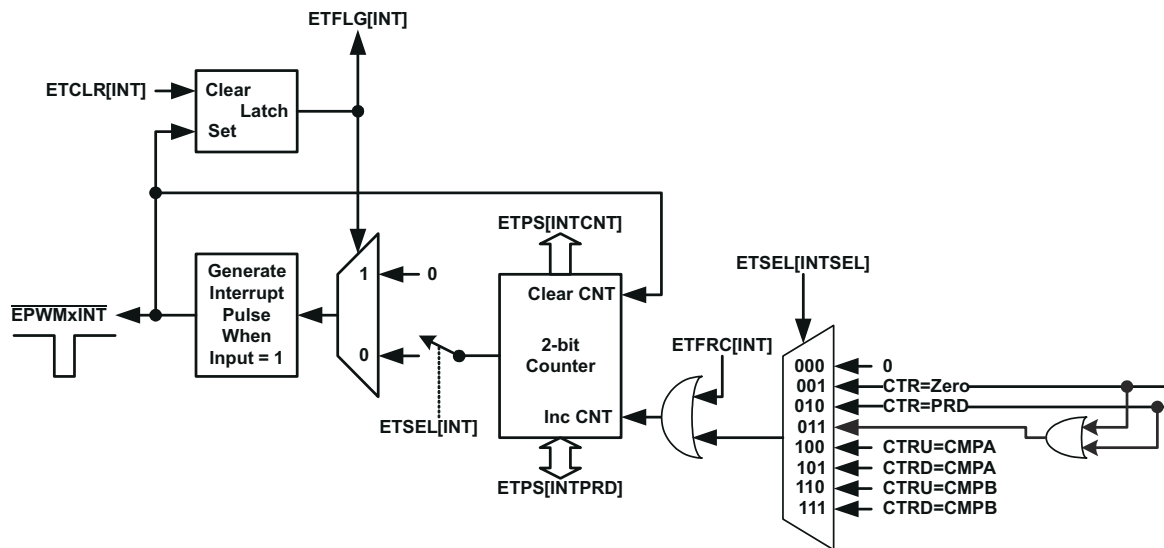
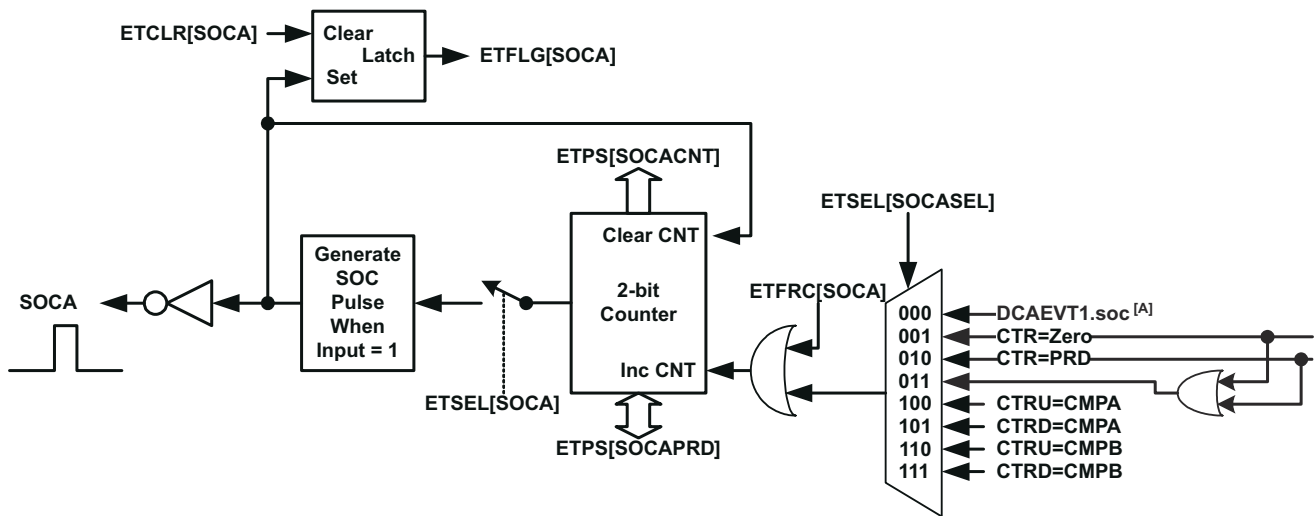


Figure 3-42. Event-Trigger Interrupt Generator

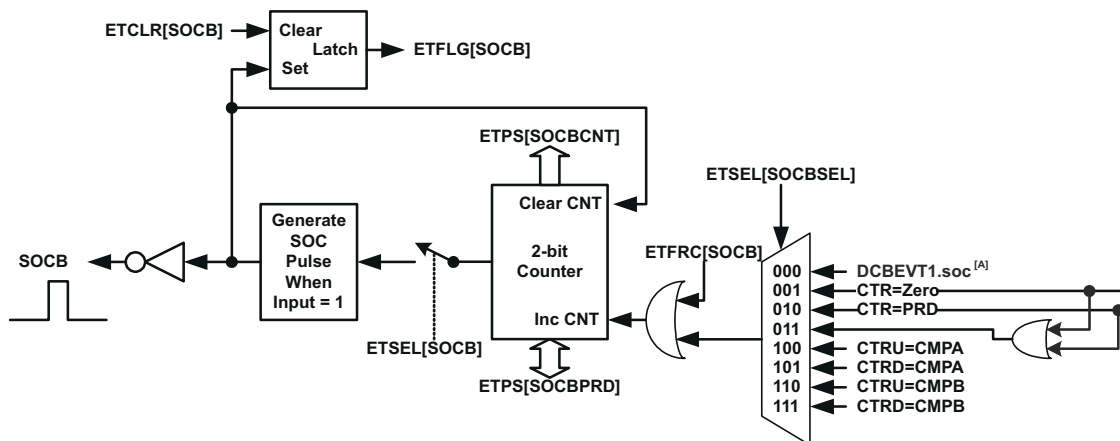
Figure 3-43 shows the operation of the event-trigger's start-of-conversion-A (SOCA) pulse generator. The ETPS[SOCACNT] counter and ETPS[SOCAPRD] period values behave similarly to the interrupt generator except that the pulses are continuously generated. That is, the pulse flag ETFLG[SOCA] is latched when a pulse is generated, but it does not stop further pulse generation. The enable/disable bit ETSEL[SOCAEN] stops pulse generation, but input events can still be counted until the period value is reached as with the interrupt generation logic. The event that will trigger an SOCA and SOCB pulse can be configured separately in the ETSEL[SOCASEL] and ETSEL[SOCBSEL] bits. The possible events are the same events that can be specified for the interrupt generation logic with the addition of the DCAEVT1.soc and DCBEVT1.soc event signals from the digital compare (DC) submodule.



A. The DCAEVT1.soc signals are signals generated by the Digital compare (DC) submodule described later in Section 3.2.9

Figure 3-43. Event-Trigger SOCA Pulse Generator

Figure 3-44 shows the operation of the event-trigger's start-of-conversion-B (SOCB) pulse generator. The event-trigger's SOCB pulse generator operates the same way as the SOCA.



A. The DCBEVT1.soc signals are signals generated by the Digital compare (DC) submodule described later in Section 3.2.9

Figure 3-44. Event-Trigger SOCB Pulse Generator

### 3.2.9 Digital Compare (DC) Submodule

Figure 3-45 illustrates where the digital compare (DC) submodule signals interface to other submodules in the ePWM system.

The digital compare (DC) submodule compares signals external to the ePWM module (for instance, COMPxOUT signals from the analog comparators) to directly generate PWM events/actions which then feed to the event-trigger, trip-zone, and time-base submodules. Additionally, blanking window functionality is supported to filter noise or unwanted pulses from the DC event signals.

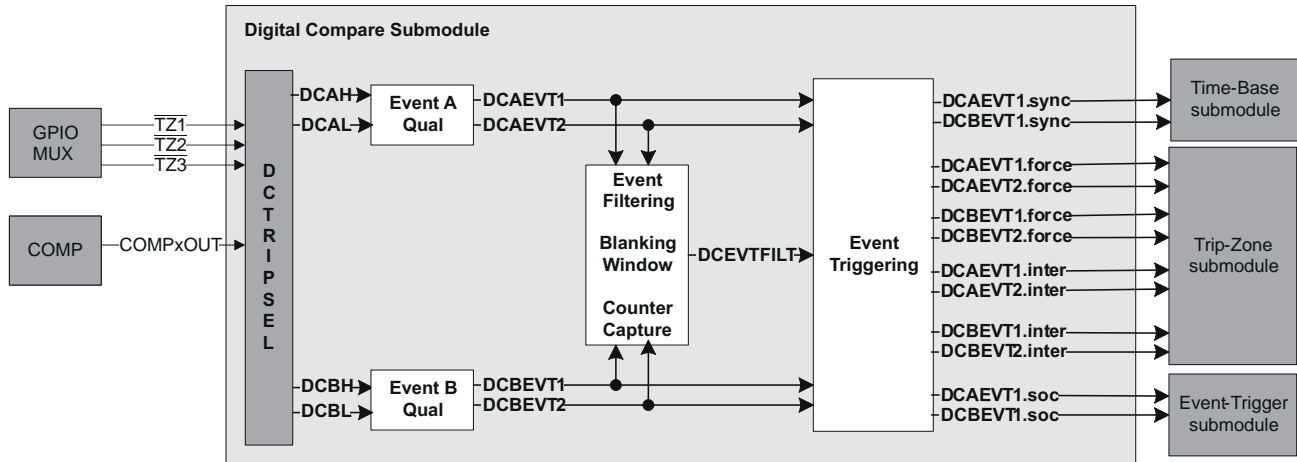


Figure 3-45. Digital-Compare Submodule High-Level Block Diagram

#### 3.2.9.1 Purpose of the Digital Compare Submodule

The key functions of the digital compare submodule are:

- Analog Comparator (COMP) module outputs and  $\overline{TZ1}$ ,  $\overline{TZ2}$ , and  $\overline{TZ3}$  inputs generate Digital Compare A High/Low (DCAH, DCAL) and Digital Compare B High/Low (DCBH, DCBL) signals.
- DCAH/L and DCBH/L signals trigger events which can then either be filtered or fed directly to the trip-zone, event-trigger, and time-base submodules to:
  - generate a trip zone interrupt
  - generate an ADC start of conversion
  - force an event
  - generate a synchronization event for synchronizing the ePWM module TBCTR.
- Event filtering (blanking window logic) can optionally blank the input signal to remove noise.

### 3.2.9.2 Controlling and Monitoring the Digital Compare Submodule

The digital compare submodule operation is controlled and monitored through the following registers.

**Table 3-20. Digital Compare Submodule Registers**

Register	Address Offset	Shadowed	Description	Bit Description
TZDCSEL <sup>(1) (2)</sup>	0x13	No	Trip Zone Digital Compare Select Register	<a href="#">Section 3.4.5.2</a>
DCTRISEL <sup>(2)</sup>	0x30	No	Digital Compare Trip Select Register	<a href="#">Section 3.4.8.1</a>
DCACTL <sup>(2)</sup>	0x31	No	Digital Compare A Control Register	<a href="#">Section 3.4.8.2</a>
DCBCTL <sup>(2)</sup>	0x32	No	Digital Compare B Control Register	<a href="#">Section 3.4.8.3</a>
DCFCTL <sup>(2)</sup>	0x33	No	Digital Compare Filter Control Register	<a href="#">Section 3.4.8.4</a>
DCCAPCTL <sup>(2)</sup>	0x34	No	Digital Compare Capture Control Register	<a href="#">Section 3.4.8.5</a>
DCOFFSET	0x35	Writes	Digital Compare Filter Offset Register	<a href="#">Section 3.4.8.6</a>
DCOFFSETCNT	0x36	No	Digital Compare Filter Offset Counter Register	<a href="#">Section 3.4.8.7</a>
DCFWINDOW	0x37	No	Digital Compare Filter Window Register	<a href="#">Section 3.4.8.8</a>
DCFWINDOWCNT	0x38	No	Digital Compare Filter Window Counter Register	<a href="#">Section 3.4.8.9</a>
DCCAP	0x39	Yes	Digital Compare Counter Capture Register	<a href="#">Section 3.4.8.10</a>

(1) The TZDCSEL register is part of the trip-zone submodule but is shown here because of its functional significance to the digital compare submodule.

(2) These registers are EALLOW protected and can be modified only after executing the EALLOW instruction. For more information, see the *System Control and Interrupts* chapter.

### 3.2.9.3 Operation Highlights of the Digital Compare Submodule

The following sections describe the operational highlights and configuration options for the digital compare submodule.

#### 3.2.9.3.1 Digital Compare Events

As shown in [Figure 3-45](#) earlier in this section, trip zone inputs ( $\overline{TZ1}$ ,  $\overline{TZ2}$ , and  $\overline{TZ3}$ ) and COMPxOUT signals from the analog comparator (COMP) module can be selected via the DCTRISEL bits to generate the Digital Compare A High and Low (DCAH/L) and Digital Compare B High and Low (DCBH/L) signals. Then, the configuration of the TZDCSEL register qualifies the actions on the selected DCAH/L and DCBH/L signals, which generate the DCAEVT1/2 and DCBEVT1/2 events (Event Qualification A and B).

#### Note

The  $\overline{TZn}$  signals, when used as a DCEVT tripping functions, are treated as a normal input signal and can be defined to be active high or active low inputs. EPWM outputs are asynchronously tripped when either the  $\overline{TZn}$ , DCAEVTx.force, or DCBEVTx.force signals are active. For the condition to remain latched, a minimum of  $3 \cdot TBCLK$  sync pulse width is required. If pulse width is  $< 3 \cdot TBCLK$  sync pulse width, the trip condition may or may not get latched by CBC or OST latches.

The DCAEVT1/2 and DCBEVT1/2 events can then be filtered to provide a filtered version of the event signals (DCEVTFILT) or the filtering can be bypassed. Filtering is discussed further in [Section 3.2.9.3.2](#). Either the DCAEVT1/2 and DCBEVT1/2 event signals or the filtered DCEVTFILT event signals can generate a force to the trip zone module, a TZ interrupt, an ADC SOC, or a PWM sync signal.

- **force signal:** DCAEVT1/2.force signals force trip zone conditions which either directly influence the output on the EPWMxA pin (via TZCTL[DCAEVT1 or DCAEVT2] configurations) or, if the DCAEVT1/2 signals are selected as one-shot or cycle-by-cycle trip sources (via the TZSEL register), the DCAEVT1/2.force signals can effect the trip action via the TZCTL[TZA] configuration. The DCBEVT1/2.force signals behaves similarly, but affect the EPWMxB output pin instead of the EPWMxA output pin.

The priority of conflicting actions on the TZCTL register is as follows (highest priority overrides lower priority):

- Output EPWMxA: TZA (highest) -> DCAEVT1 -> DCAEVT2 (lowest)
- Output EPWMxB: TzB (highest) -> DCBEVT1 -> DCBEVT2 (lowest)



- **interrupt signal:** DCAEVT1/2.interrupt signals generate trip zone interrupts to the PIE. To enable the interrupt, the user must set the DCAEVT1, DCAEVT2, DCBEVT1, or DCBEVT2 bits in the TZEINT register. Once one of these events occurs, an EPWMxTZINT interrupt is triggered, and the corresponding bit in the TZCLR register must be set in order to clear the interrupt.
- **soc signal:** The DCAEVT1.soc signal interfaces with the event-trigger submodule and can be selected as an event which generates an ADC start-of-conversion-A (SOCA) pulse via the ETSEL[SOCASEL] bit. Likewise, the DCBEVT1.soc signal can be selected as an event which generates an ADC start-of-conversion-B (SOCB) pulse via the ETSEL[SOCBSEL] bit.
- **sync signal:** The DCAEVT1.sync and DCBEVT1.sync events are ORed with the EPWMxSYNCl input signal and the TBCTL[SWFSYNC] signal to generate a synchronization pulse to the time-base counter.

Figure 3-46 and Figure 3-47 show how the DCAEVT1, DCAEVT2, and DCEVTFILT signals are processed to generate the digital compare A event force, interrupt, soc, and sync signals.

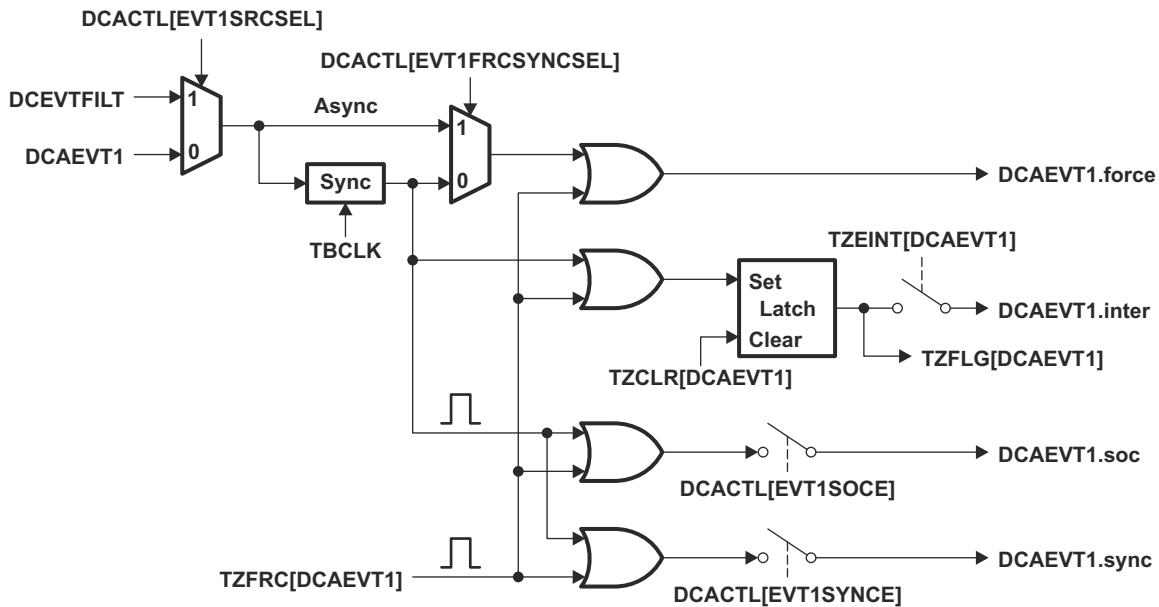


Figure 3-46. DCAEVT1 Event Triggering

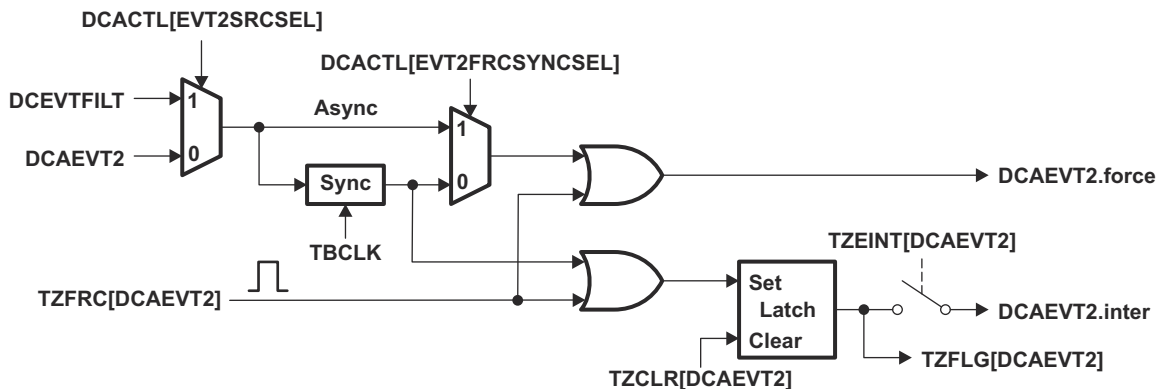


Figure 3-47. DCAEVT2 Event Triggering

Figure 3-48 and Figure 3-49 show how the DCBEVT1, DCBEVT2, and DCEVTFILT signals are processed to generate the digital compare B event force, interrupt, soc and sync signals.

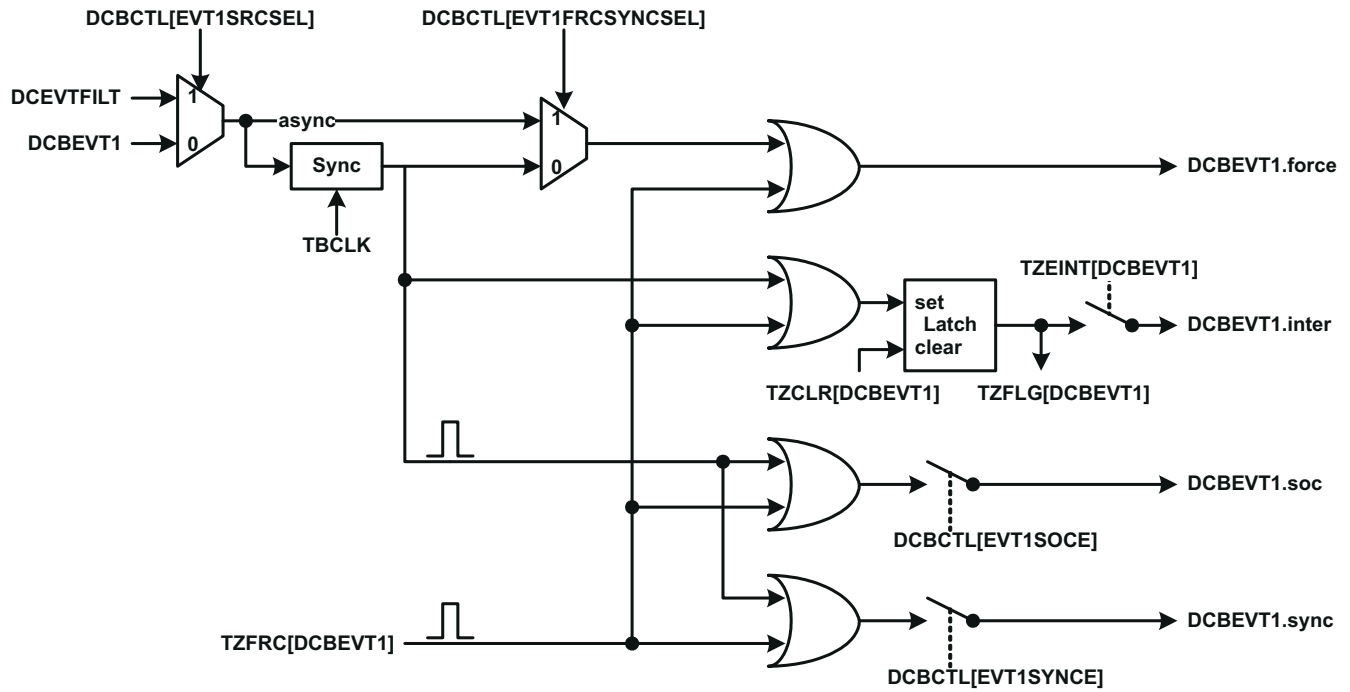


Figure 3-48. DCBEVT1 Event Triggering

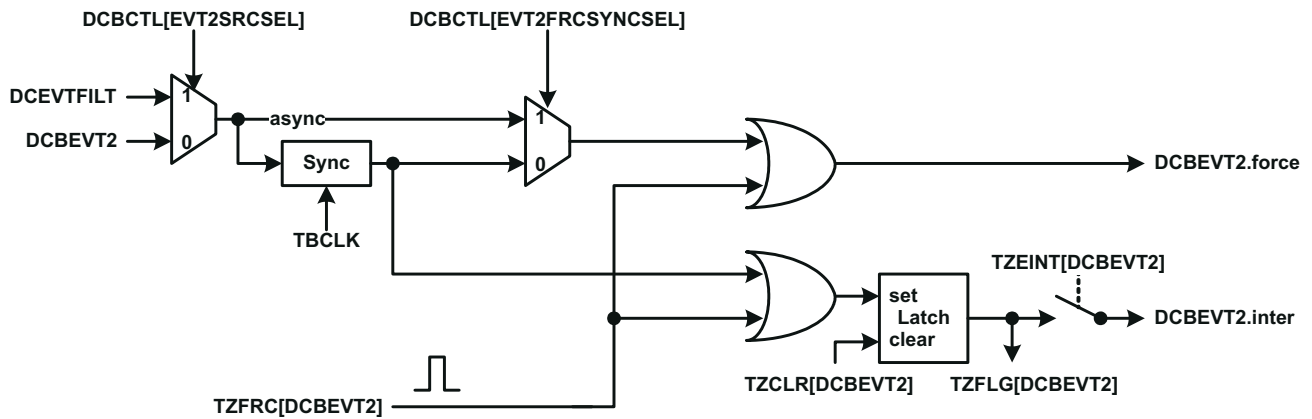


Figure 3-49. DCBEVT2 Event Triggering

### 3.2.9.3.2 Event Filtering

The DCAEVT1/2 and DCBEVT1/2 events can be filtered via event filtering logic to remove noise by optionally blanking events for a certain period of time. This is useful for cases where the analog comparator outputs may be selected to trigger DCAEVT1/2 and DCBEVT1/2 events, and the blanking logic is used to filter out potential noise on the signal prior to tripping the PWM outputs or generating an interrupt or ADC start-of-conversion. The event filtering can also capture the TBCTR value of the trip event. Figure 3-50 shows the details of the event filtering logic.

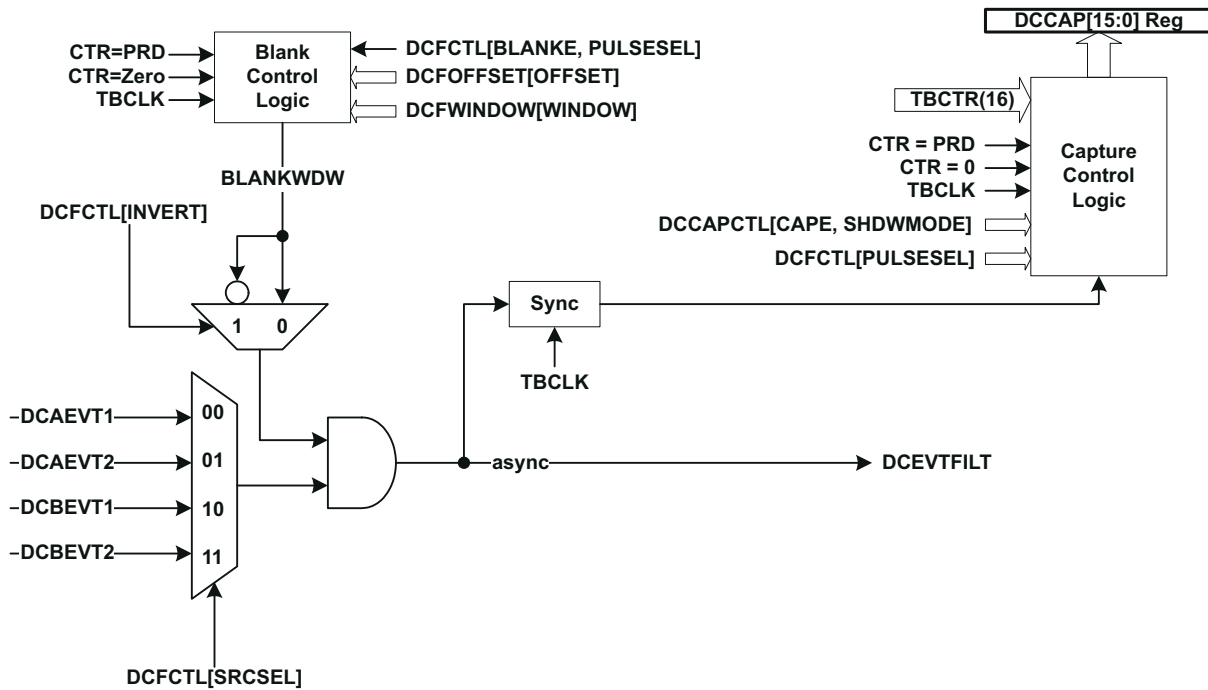


Figure 3-50. Event Filtering

If the blanking logic is enabled, one of the digital compare events – DCAEVT1, DCAEVT2, DCBEVT1, DCBEVT2 – is selected for filtering. The blanking window, which filters out all event occurrences on the signal while it is active, will be aligned to either a CTR = PRD pulse or a CTR = 0 pulse (configured by the DCFCTL[PULSESEL] bits). An offset value in TBCLK counts is programmed into the DCFOFFSET register, which determines at what point after the CTR = PRD or CTR = 0 pulse the blanking window starts. The duration of the blanking window, in number of TBCLK counts after the offset counter expires, is written to the DCFWINDOW register by the application. During the blanking window, all events are ignored. Before and after the blanking window ends, events can generate soc, sync, interrupt, and force signals as before.

Figure 3-51 illustrates several timing conditions for the offset and blanking window within an ePWM period. Notice that if the blanking window crosses the CTR = 0 or CTR = PRD boundary, the next window still starts at the same offset value after the CTR = 0 or CTR = PRD pulse.

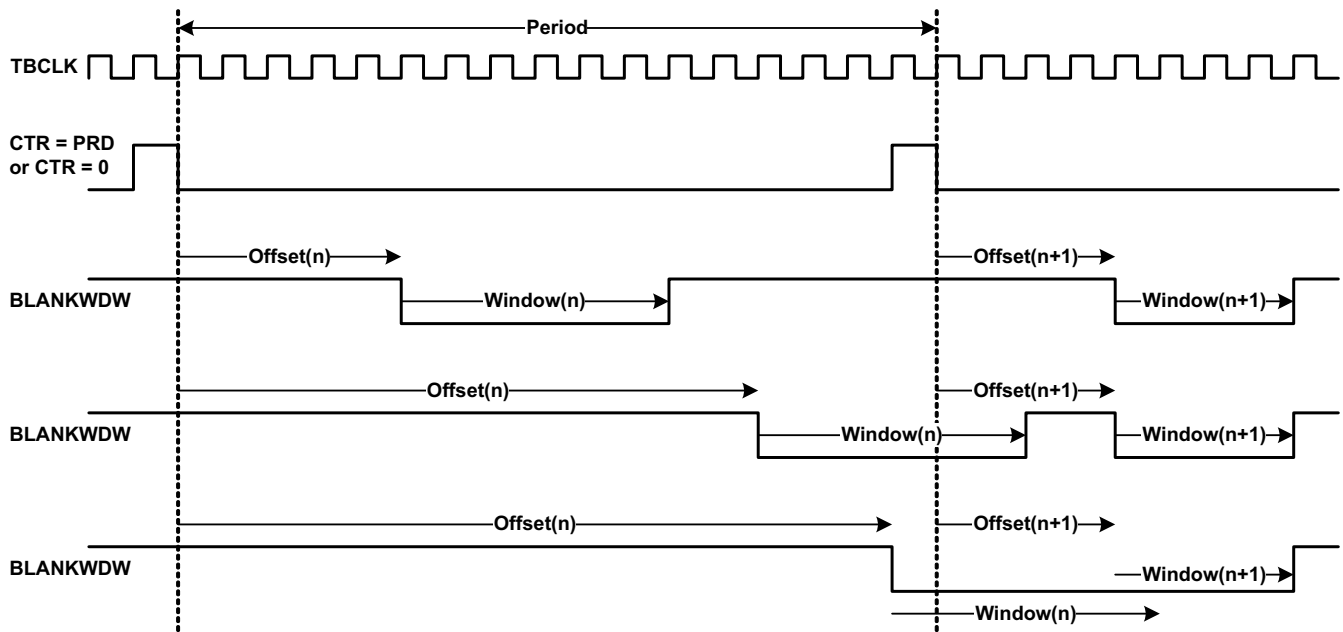


Figure 3-51. Blanking Window Timing Diagram

### 3.3 Applications to Power Topologies

An ePWM module has all the local resources necessary to operate completely as a standalone module or to operate in synchronization with other identical ePWM modules.

#### 3.3.1 Overview of Multiple Modules

Previously all discussions have described the operation of a single module. To facilitate the understanding of multiple modules working together in a system, the ePWM module described in reference is represented by the more simplified block diagram shown in Figure 3-52. This simplified ePWM block shows only the key resources needed to explain how a multiswitch power topology is controlled with multiple ePWM modules working together.

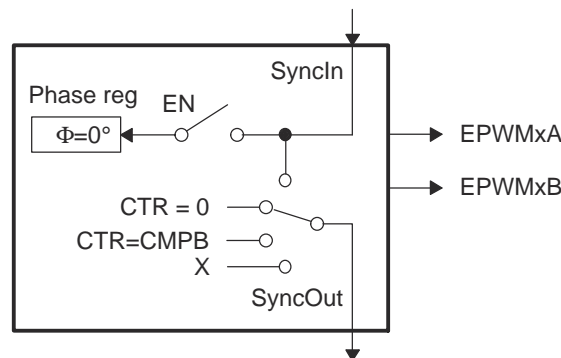


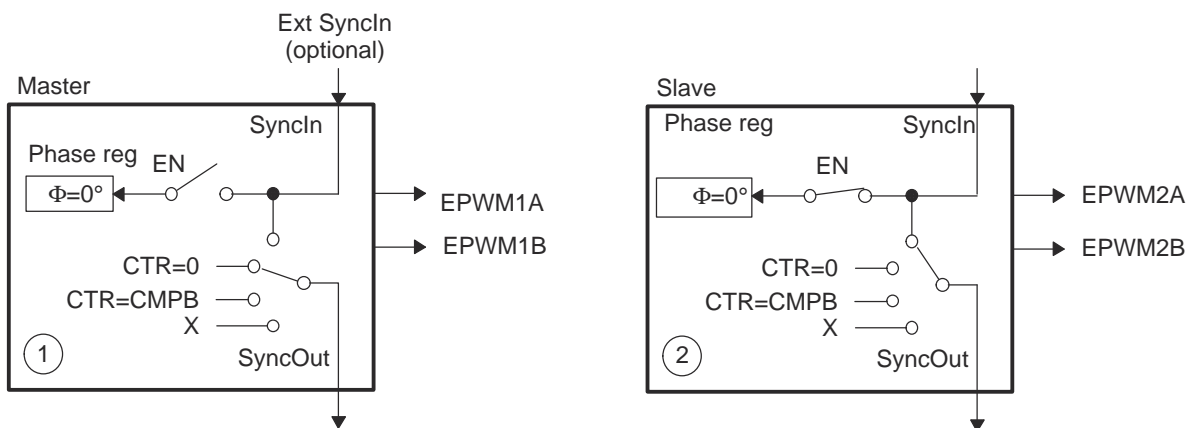
Figure 3-52. Simplified ePWM Module

### 3.3.2 Key Configuration Capabilities

The key configuration choices available to each module are as follows:

- Options for SyncIn
  - Load own counter with phase register on an incoming sync strobe—enable (EN) switch closed
  - Do nothing or ignore incoming sync strobe—enable switch open
  - Sync flow-through - SyncOut connected to SyncIn
  - Master mode, provides a sync at PWM boundaries—SyncOut connected to CTR = PRD
  - Master mode, provides a sync at any programmable point in time—SyncOut connected to CTR = CMPB
  - Module is in standalone mode and provides No sync to other modules—SyncOut connected to X (disabled)
- Options for SyncOut
  - Sync flow-through - SyncOut connected to SyncIn
  - Master mode, provides a sync at PWM boundaries—SyncOut connected to CTR = PRD
  - Master mode, provides a sync at any programmable point in time—SyncOut connected to CTR = CMPB
  - Module is in standalone mode and provides No sync to other modules—SyncOut connected to X (disabled)

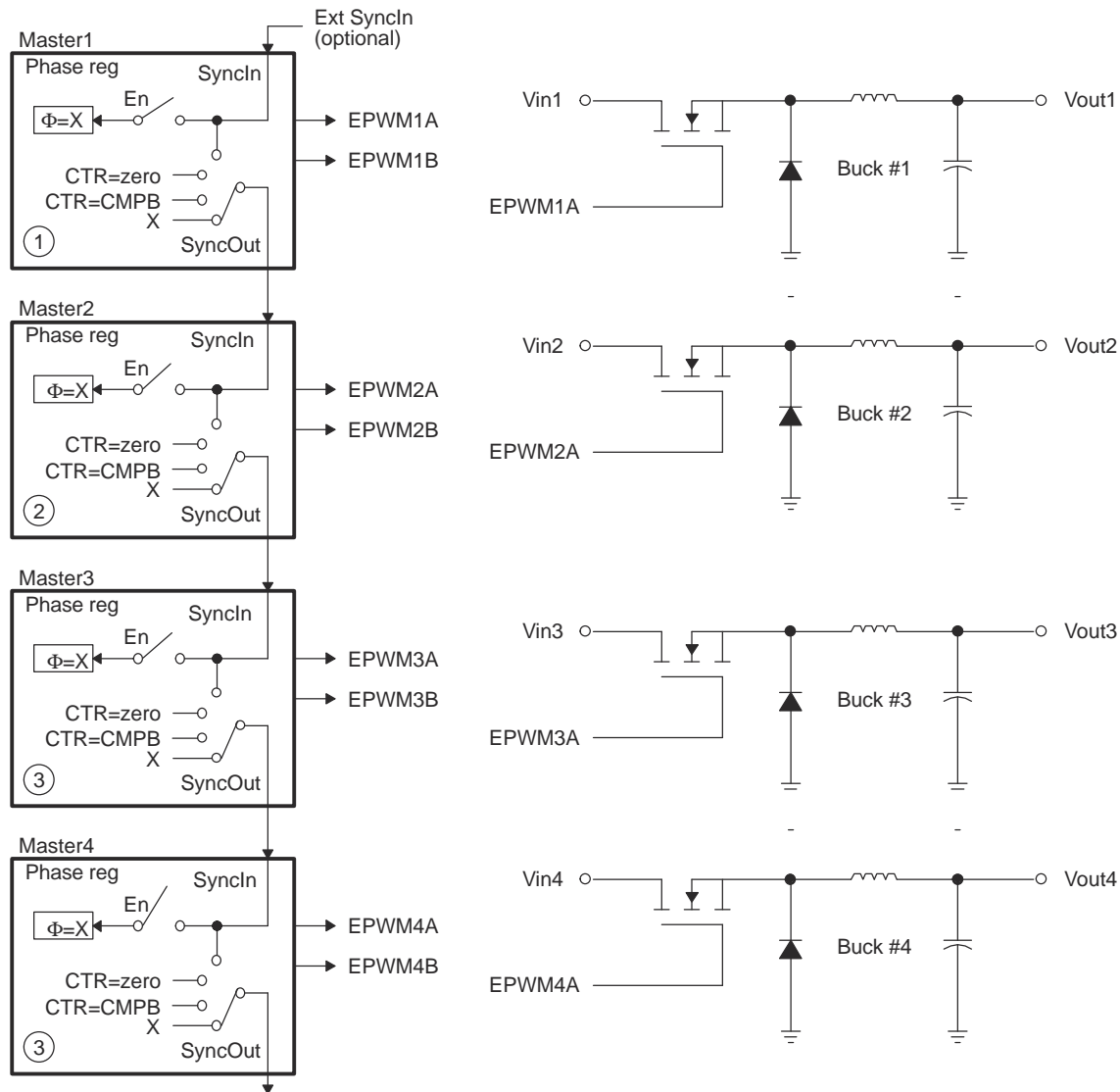
For each choice of SyncOut, a module may also choose to load its own counter with a new phase value on a SyncIn strobe input or choose to ignore it, that is, with the enable switch. Although various combinations are possible, the two most common—master module and slave module modes—are shown in [Figure 3-53](#).



**Figure 3-53. EPWM1 Configured as a Typical Master, EPWM2 Configured as a Slave**

### 3.3.3 Controlling Multiple Buck Converters With Independent Frequencies

One of the simplest power converter topologies is the buck. A single ePWM module configured as a master can control two buck stages with the same PWM frequency. If independent frequency control is required for each buck converter, then one ePWM module must be allocated for each converter stage. [Figure 3-54](#) shows four buck stages, each running at independent frequencies. In this case, all four ePWM modules are configured as masters and no synchronization is used. [Figure 3-55](#) shows the waveforms generated by the setup shown in [Figure 3-54](#); note that only three waveforms are shown, although there are four stages.



A.  $\Theta = X$  indicates value in phase register is a "don't care"

**Figure 3-54. Control of Four Buck Stages. Here  $F_{PWM1} \neq F_{PWM2} \neq F_{PWM3} \neq F_{PWM4}$**

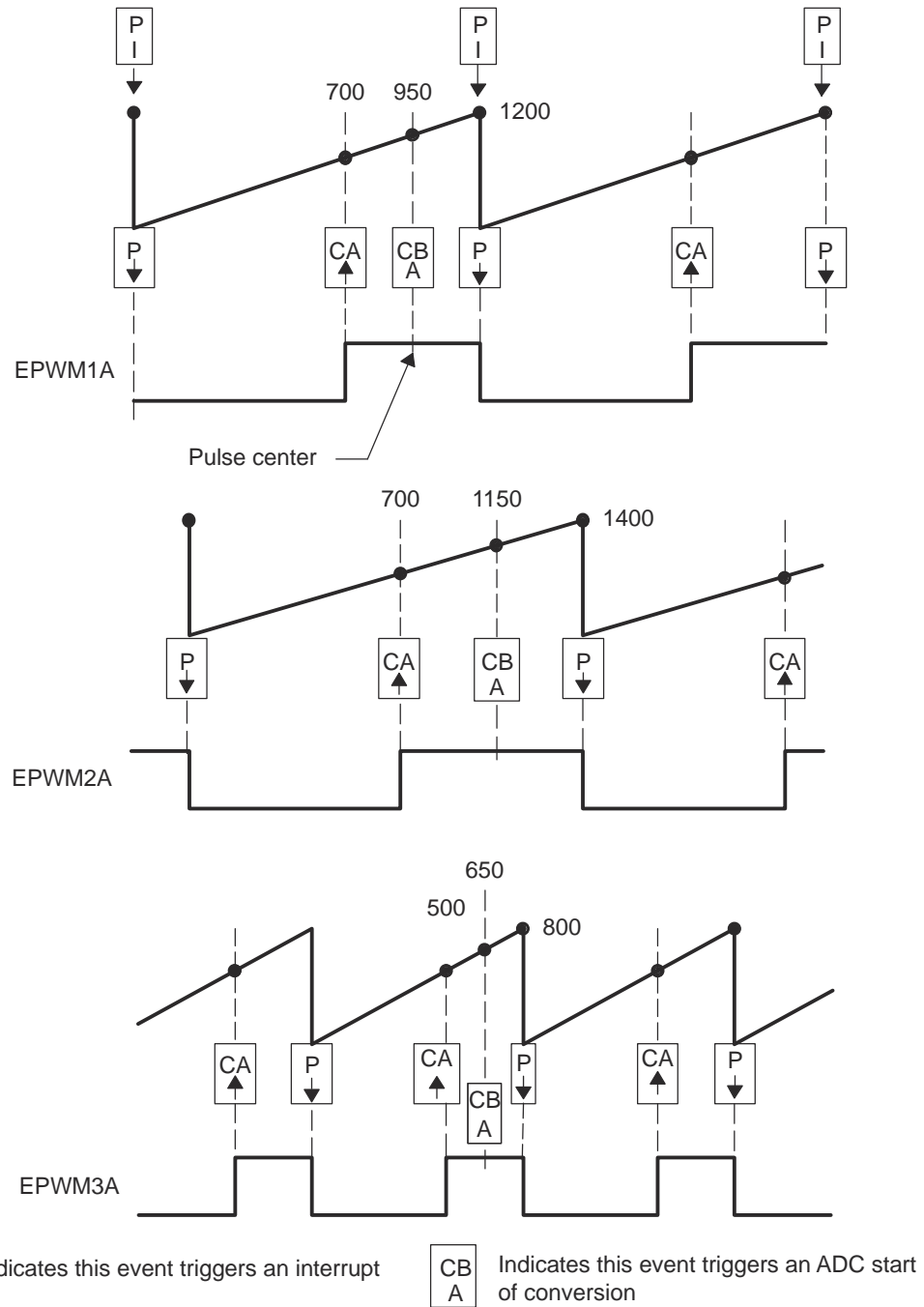


Figure 3-55. Buck Waveforms for Figure 3-54 (Note: Only three bucks shown here)

**Example 3-8. Code Snippet for Configuration in Figure 3-54**

```

//=====
// (Note: code for only 3 modules shown)
// Initialization Time
//=====
// EPWM Module 1 config
EPwm1Regs.TBPRD = 1200; // Period = 1201 TBCLK counts
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Asymmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm1Regs.TBCTL.bit.PRDLN = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.AQCTLA.bit.PRDLN = AQ_CLEAR;
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
// EPWM Module 2 config
EPwm2Regs.TBPRD = 1400; // Period = 1401 TBCLK counts
EPwm2Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Asymmetrical mode
EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm2Regs.TBCTL.bit.PRDLN = TB_SHADOW;
EPwm2Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.AQCTLA.bit.PRDLN = AQ_CLEAR;
EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;
// EPWM Module 3 config
EPwm3Regs.TBPRD = 800; // Period = 801 TBCLK counts
EPwm3Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm3Regs.TBCTL.bit.PRDLN = TB_SHADOW;
EPwm3Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm3Regs.AQCTLA.bit.PRDLN = AQ_CLEAR;
EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;
//
// Run Time (Note: Example execution of one run-time instant)
//=====
EPwm1Regs.CMPA.half.CMPA = 700; // adjust duty for output EPWM1A
EPwm2Regs.CMPA.half.CMPA = 700; // adjust duty for output EPWM2A
EPwm3Regs.CMPA.half.CMPA = 500; // adjust duty for output EPWM3A

```



### 3.3.4 Controlling Multiple Buck Converters With Same Frequencies

If synchronization is a requirement, ePWM module 2 can be configured as a slave and can operate at integer multiple (N) frequencies of module 1. The sync signal from master to slave ensures these modules remain locked. Figure 3-56 shows such a configuration; Figure 3-57 shows the waveforms generated by the configuration.

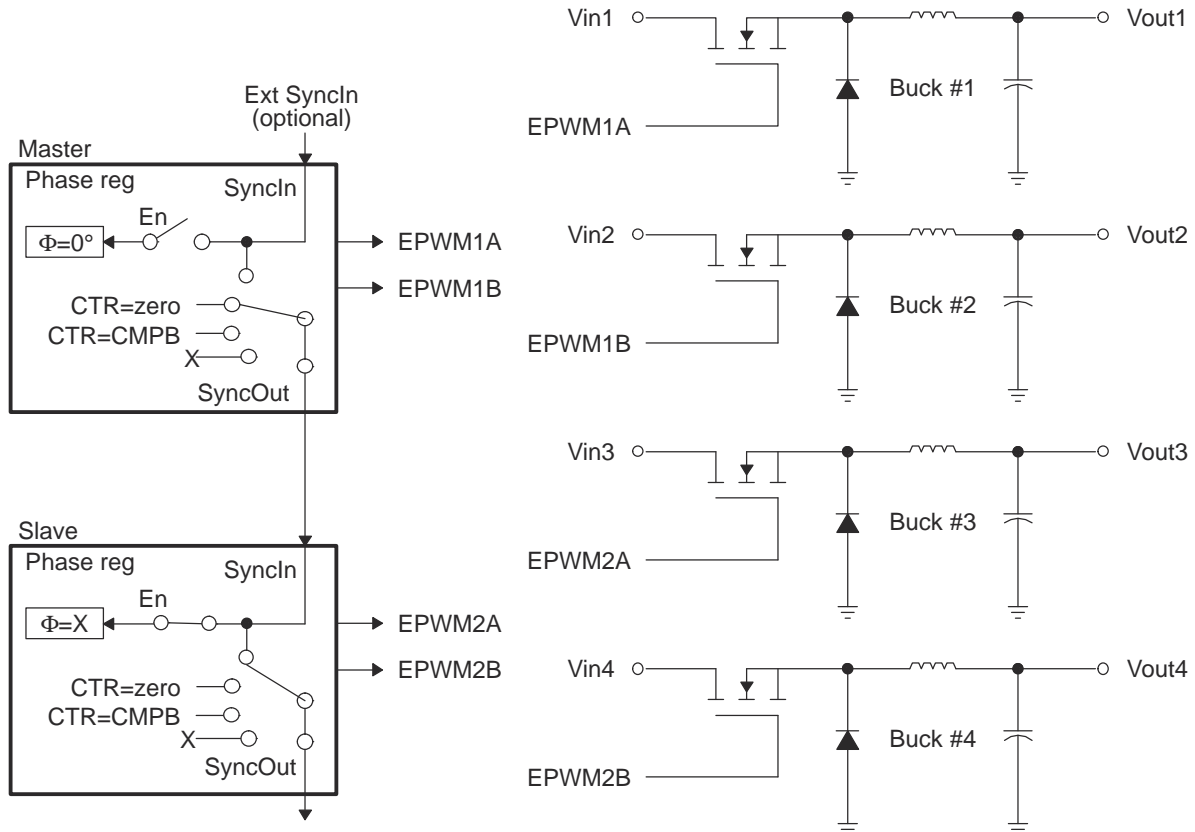


Figure 3-56. Control of Four Buck Stages. (Note:  $F_{PWM2} = N \times F_{PWM1}$ )

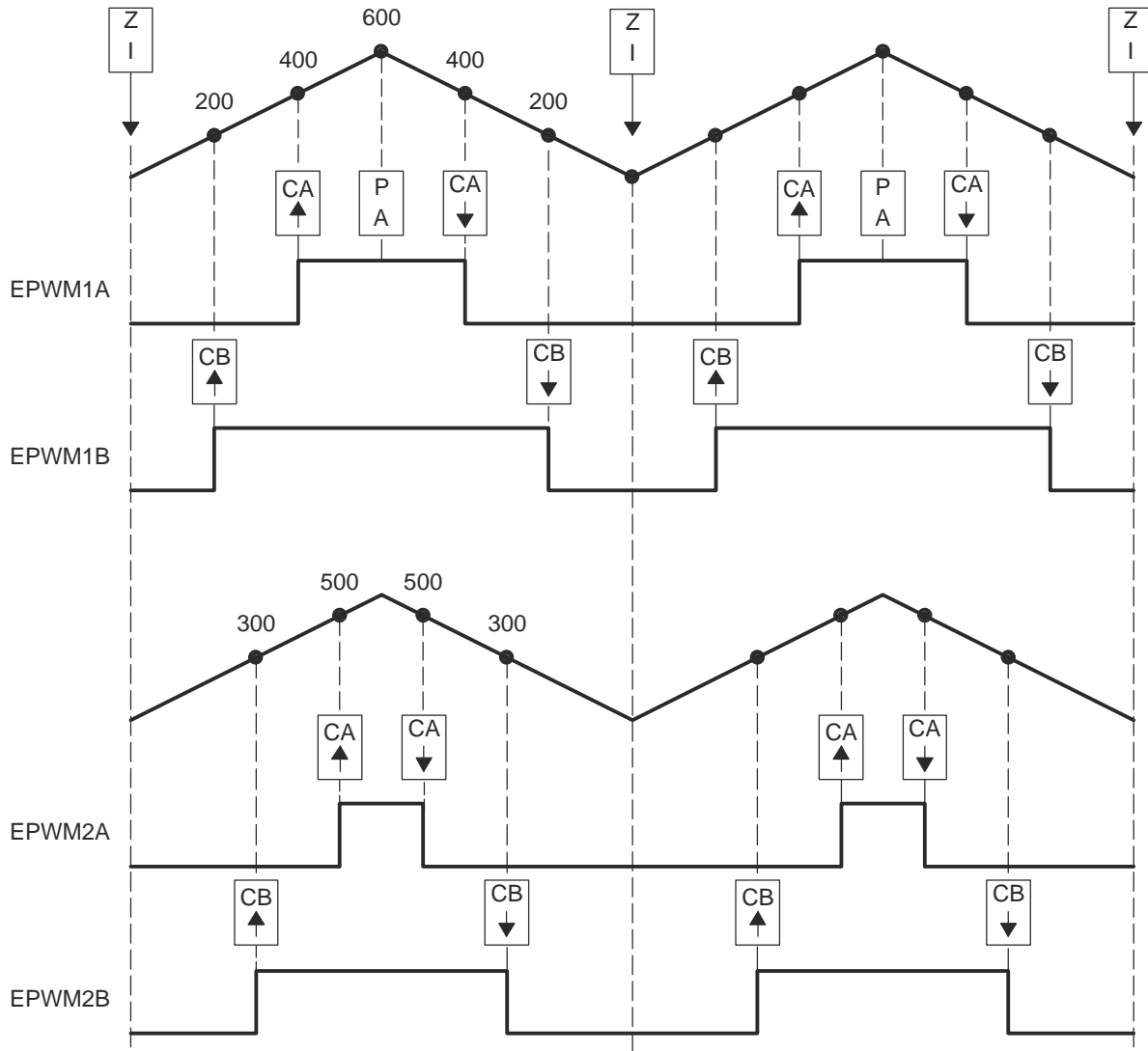


Figure 3-57. Buck Waveforms for Figure 3-56 (Note:  $F_{PWM2} = F_{PWM1}$ )

**Example 3-9. Code Snippet for Configuration in Figure 3-56**

```

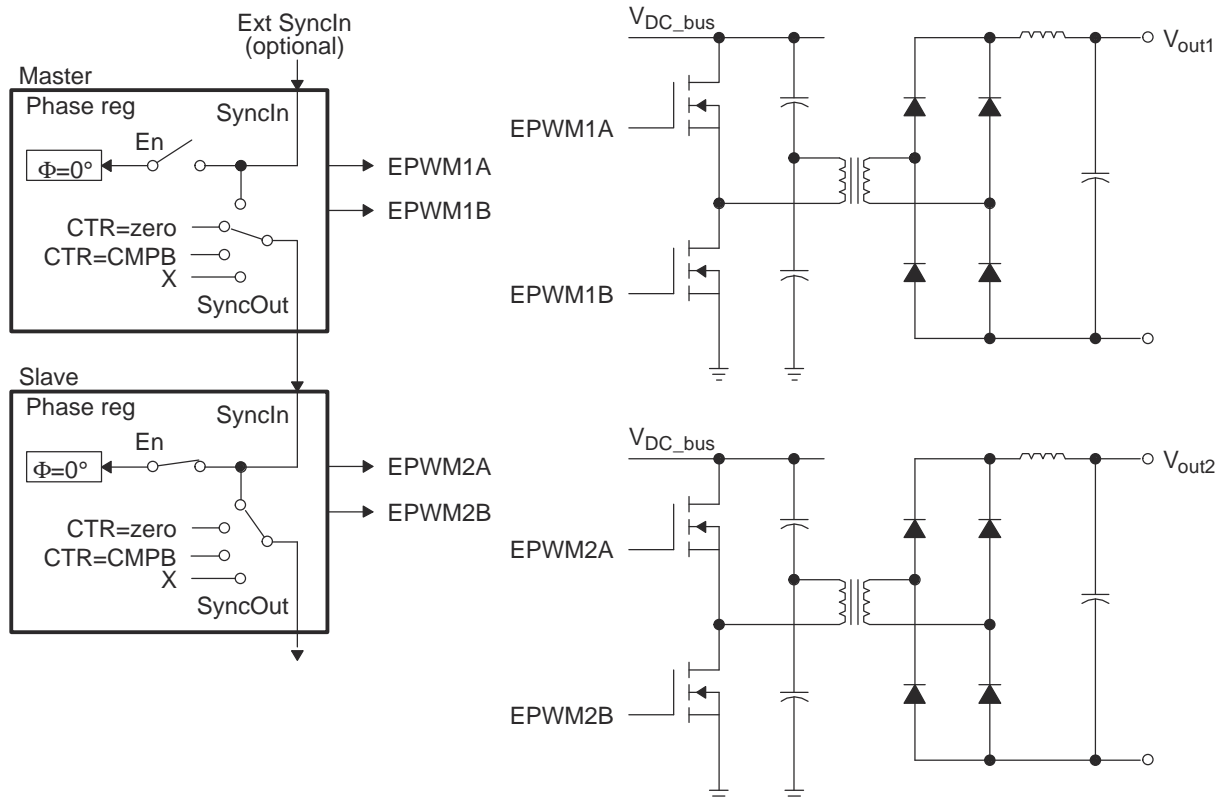
//=====
// EPWM Module 1 config
EPwm1Regs.TBPRD = 600; // Period = 1200 TBCLK counts
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Master module
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // Sync down-stream module
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM1A
EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.CBU = AQ_SET; // set actions for EPWM1B
EPwm1Regs.AQCTLB.bit.CBD = AQ_CLEAR;
// EPWM Module 2 config
EPwm2Regs.TBPRD = 600; // Period = 1200 TBCLK counts
EPwm2Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Slave module
EPwm2Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN; // sync flow-through
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM2A
EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm2Regs.AQCTLB.bit.CBU = AQ_SET; // set actions for EPWM2B
EPwm2Regs.AQCTLB.bit.CBD = AQ_CLEAR;
//
// Run Time (Note: Example execution of one run-time instance)
//=====
EPwm1Regs.CMPA.half.CMPA = 400; // adjust duty for output EPWM1A
EPwm1Regs.CMPB = 200; // adjust duty for output EPWM1B
EPwm2Regs.CMPA.half.CMPA = 500; // adjust duty for output EPWM2A
EPwm2Regs.CMPB = 300; // adjust duty for output EPWM2B

```

### 3.3.5 Controlling Multiple Half H-Bridge (HHB) Converters

Topologies that require control of multiple switching elements can also be addressed with these same ePWM modules. It is possible to control a Half-H bridge stage with a single ePWM module. This control can be extended to multiple stages. [Figure 3-58](#) shows control of two synchronized Half-H bridge stages where stage 2 can operate at integer multiple (N) frequencies of stage 1. [Figure 3-59](#) shows the waveforms generated by the configuration shown in [Figure 3-58](#).

Module 2 (slave) is configured for Sync flow-through; if required, this configuration allows for a third Half-H bridge to be controlled by PWM module 3 and also, most importantly, to remain in synchronization with master module 1.



**Figure 3-58. Control of Two Half-H Bridge Stages ( $F_{PWM2} = N \times F_{PWM1}$ )**

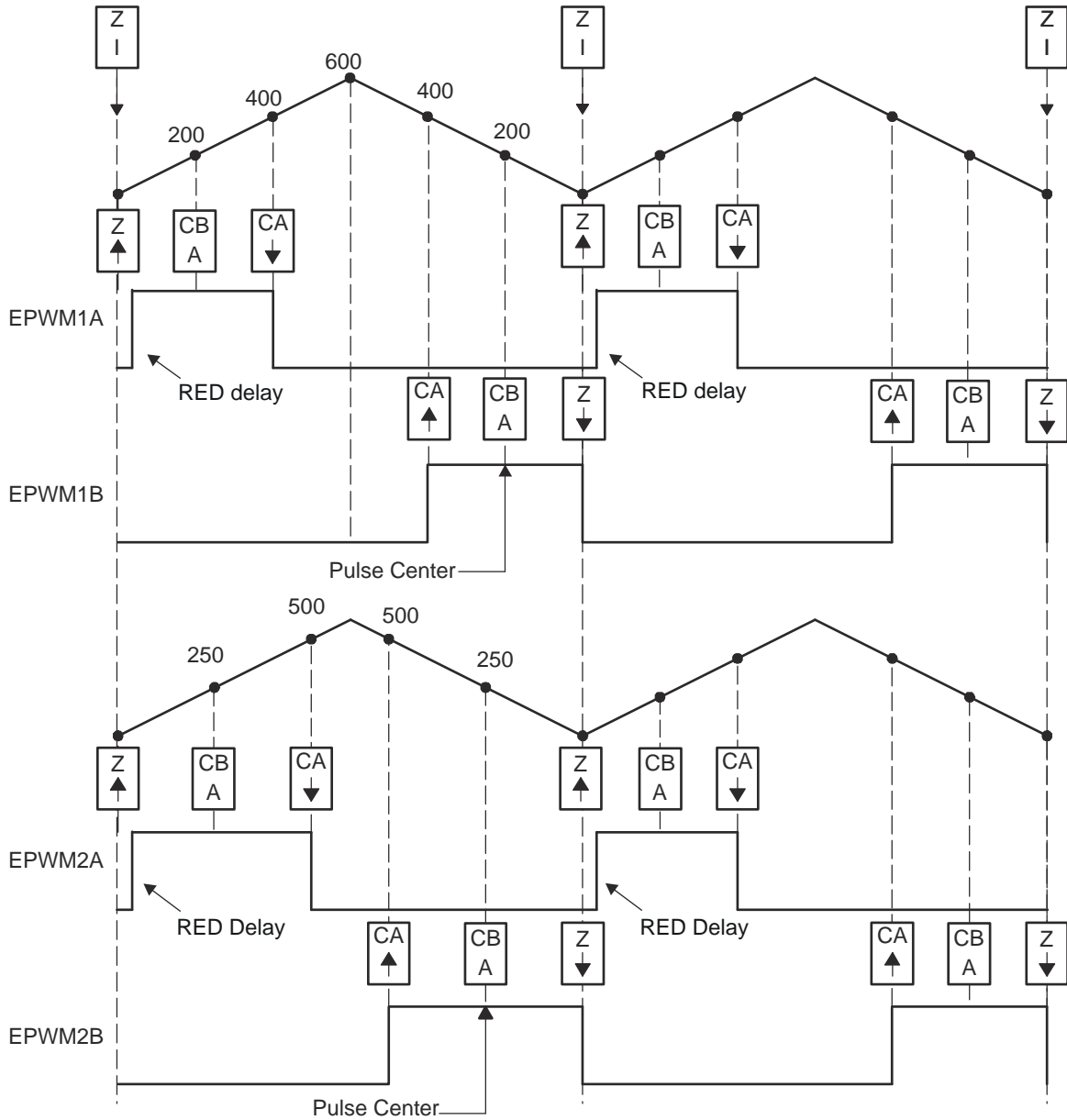


Figure 3-59. Half-H Bridge Waveforms for Figure 3-58 (Note: Here  $F_{PWM2} = F_{PWM1}$ )

**Example 3-10. Code Snippet for Configuration in Figure 3-58**

```

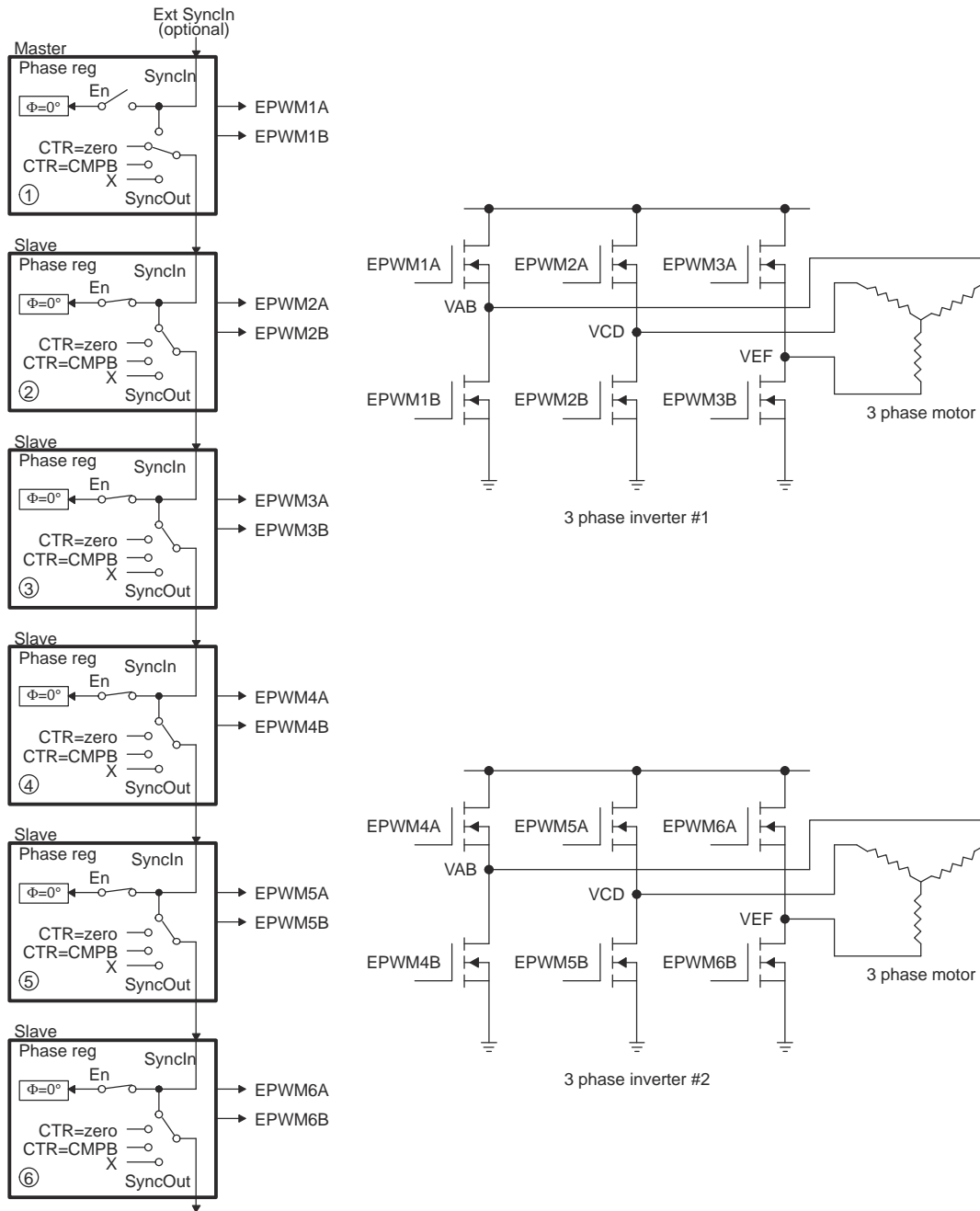
//=====
// Config
//=====
// Initialization Time
//=====
// EPWM Module 1 config
EPwm1Regs.TBPRD = 600; // Period = 1200 TBCLK counts
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Master module
EPwm1Regs.TBCTL.bit.PRDLID = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // Sync down-stream module
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET; // set actions for EPWM1A
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_CLEAR; // set actions for EPWM1B
EPwm1Regs.AQCTLB.bit.CAD = AQ_SET;
EPwm1Regs.DBCTL.bit.OUT_MODE = 2; // delay on EPWM1A
EPwm1Regs.DBRED = 10; // delay value
// EPWM Module 2 config
EPwm2Regs.TBPRD = 600; // Period = 1200 TBCLK counts
EPwm2Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Slave module
EPwm2Regs.TBCTL.bit.PRDLID = TB_SHADOW;
EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN; // sync flow-through
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET; // set actions for EPWM1A
EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR; // set actions for EPWM1B
EPwm2Regs.AQCTLB.bit.CAD = AQ_SET;
EPwm2Regs.DBCTL.bit.OUT_MODE = 2; // delay on EPWM2A
EPwm2Regs.DBRED = 10; // delay value
//=====
EPwm1Regs.CMPA.half.CMPA = 400; // adjust duty for output EPWM1A & EPWM1B
EPwm1Regs.CMPB = 200; // adjust point-in-time for ADCSOC trigger
EPwm2Regs.CMPA.half.CMPA = 500; // adjust duty for output EPWM2A & EPWM2B
EPwm2Regs.CMPB = 250; // adjust point-in-time for ADCSOC trigger

```

### 3.3.6 Controlling Dual 3-Phase Inverters for Motors (ACI and PMSM)

The idea of multiple modules controlling a single power stage can be extended to the 3-phase Inverter case. In such a case, six switching elements can be controlled using three PWM modules, one for each leg of the inverter. Each leg must switch at the same frequency and all legs must be synchronized. A master + two slaves configuration can easily address this requirement. [Figure 3-60](#) shows how six PWM modules can control two independent 3-phase Inverters; each running a motor.

As in the cases shown in the previous sections, we have a choice of running each inverter at a different frequency (module 1 and module 4 are masters as in [Figure 3-60](#)), or both inverters can be synchronized by using one master (module 1) and five slaves. In this case, the frequency of modules 4, 5, and 6 (all equal) can be integer multiples of the frequency for modules 1, 2, 3 (also all equal).



**Figure 3-60. Control of Dual 3-Phase Inverter Stages as Is Commonly Used in Motor Control**

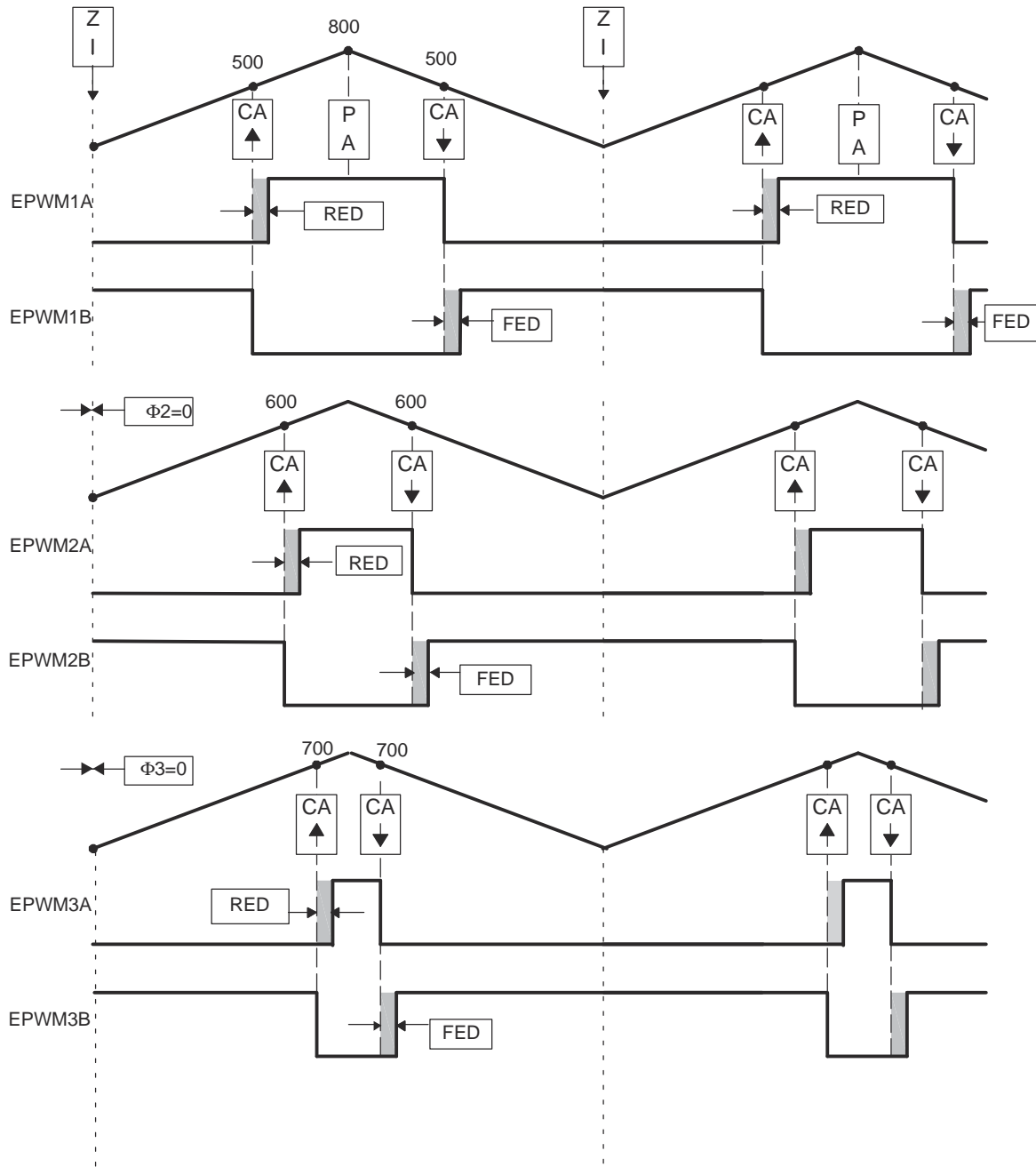


Figure 3-61. 3-Phase Inverter Waveforms for Figure 3-60 (Only One Inverter Shown)



**Example 3-11. Code Snippet for Configuration in Figure 3-60**

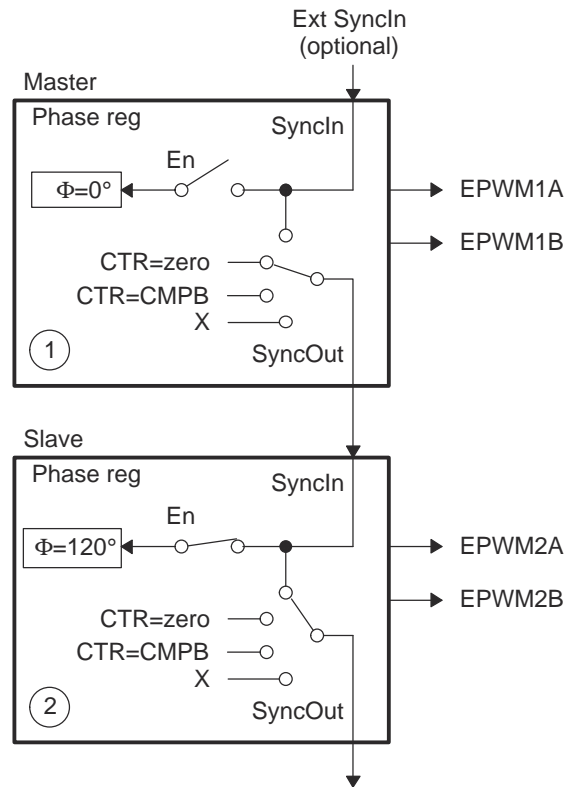
```

//=====
// Configuration
//=====
// Initialization Time
//=====
// EPWM Module 1 config
EPwm1Regs.TBPRD = 800; // Period = 1600 TBCLK counts
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Master module
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // Sync down-stream module
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM1A
EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
EPwm1Regs.DBFED = 50; // FED = 50 TBCLKs
EPwm1Regs.DBRED = 50; // RED = 50 TBCLKs
// EPWM Module 2 config
EPwm2Regs.TBPRD = 800; // Period = 1600 TBCLK counts
EPwm2Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Slave module
EPwm2Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN; // sync flow-through
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM2A
EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
EPwm2Regs.DBFED = 50; // FED = 50 TBCLKs
EPwm2Regs.DBRED = 50; // RED = 50 TBCLKs
// EPWM Module 3 config
EPwm3Regs.TBPRD = 800; // Period = 1600 TBCLK counts
EPwm3Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Slave module
EPwm3Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN; // sync flow-through
EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm3Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM3A
EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
EPwm3Regs.DBFED = 50; // FED = 50 TBCLKs
EPwm3Regs.DBRED = 50; // RED = 50 TBCLKs
// Run Time (Note: Example execution of one run-time instant)
//=====
EPwm1Regs.CMPA.half.CMPA = 500; // adjust duty for output EPWM1A
EPwm2Regs.CMPA.half.CMPA = 600; // adjust duty for output EPWM2A
EPwm3Regs.CMPA.half.CMPA = 700; // adjust duty for output EPWM3A

```

### 3.3.7 Practical Applications Using Phase Control Between PWM Modules

So far, none of the examples have made use of the phase register (TBPHS). It has either been set to zero or its value has been a don't care. However, by programming appropriate values into TBPHS, multiple PWM modules can address another class of power topologies that rely on phase relationship between legs (or stages) for correct operation. As described in the TB module section, a PWM module can be configured to allow a SyncIn pulse to cause the TBPHS register to be loaded into the TBCTR register. To illustrate this concept, [Figure 3-62](#) shows a master and slave module with a phase relationship of 120°, that is, the slave leads the master.



**Figure 3-62. Configuring Two PWM Modules for Phase Control**

Figure 3-63 shows the associated timing waveforms for this configuration. Here, TBPRD = 600 for both master and slave. For the slave, TBPHS = 200 ( $200/600 \times 360^\circ = 120^\circ$ ). Whenever the master generates a SyncIn pulse (CTR = PRD), the value of TBPHS = 200 is loaded into the slave TBCTR register so the slave time-base is always leading the master's time-base by  $120^\circ$ .

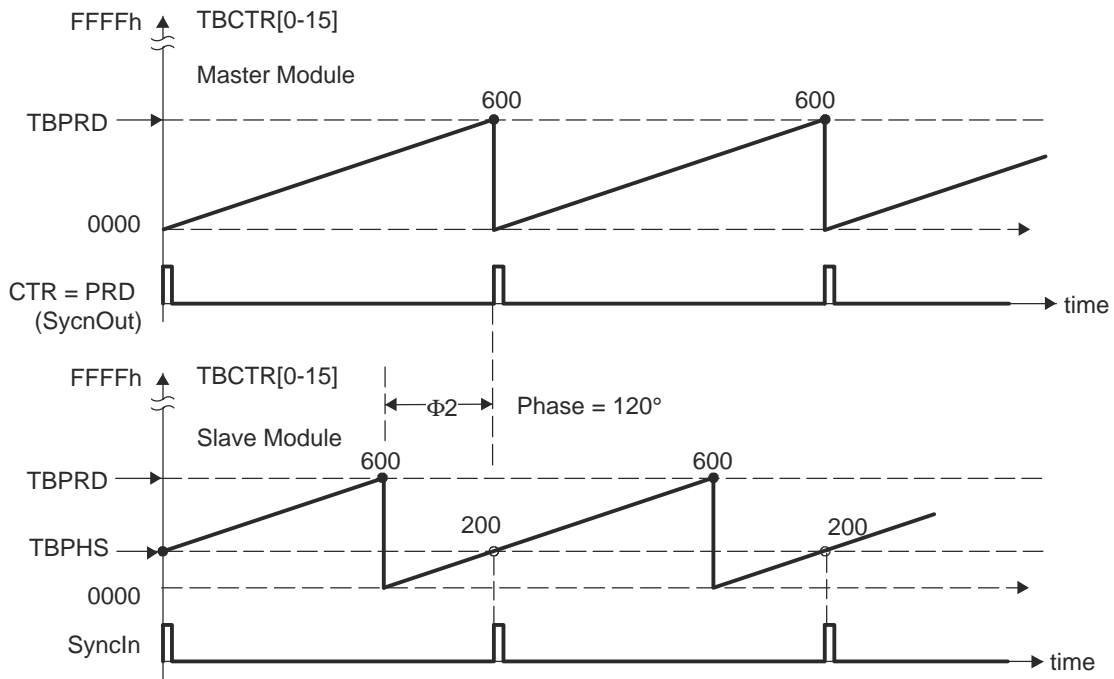


Figure 3-63. Timing Waveforms Associated With Phase Control Between 2 Modules

### 3.3.8 Controlling a 3-Phase Interleaved DC/DC Converter

A popular power topology that makes use of phase-offset between modules is shown in Figure 3-64. This system uses three PWM modules, with module 1 configured as the master. To work, the phase relationship between adjacent modules must be  $F = 120^\circ$ . This is achieved by setting the slave TBPHS registers 2 and 3 with values of 1/3 and 2/3 of the period value, respectively. For example, if the period register is loaded with a value of 600 counts, then TBPHS (slave 2) = 200 and TBPHS (slave 3) = 400. Both slave modules are synchronized to the master 1 module.

This concept can be extended to four or more phases, by setting the TBPHS values appropriately. The following formula gives the TBPHS values for N phases:

$$TBPHS(N,M) = (TBPRD/N) \times (M-1)$$

Where:

N = number of phases

M = PWM module number

For example, for the 3-phase case (N=3), TBPRD = 600,

$$TBPHS(3,2) = (600/3) \times (2-1) = 200 \text{ (that is, Phase value for Slave module 2)}$$

$$TBPHS(3,3) = 400 \text{ (Phase value for Slave module 3)}$$

Figure 3-65 shows the waveforms for the configuration in Figure 3-64.

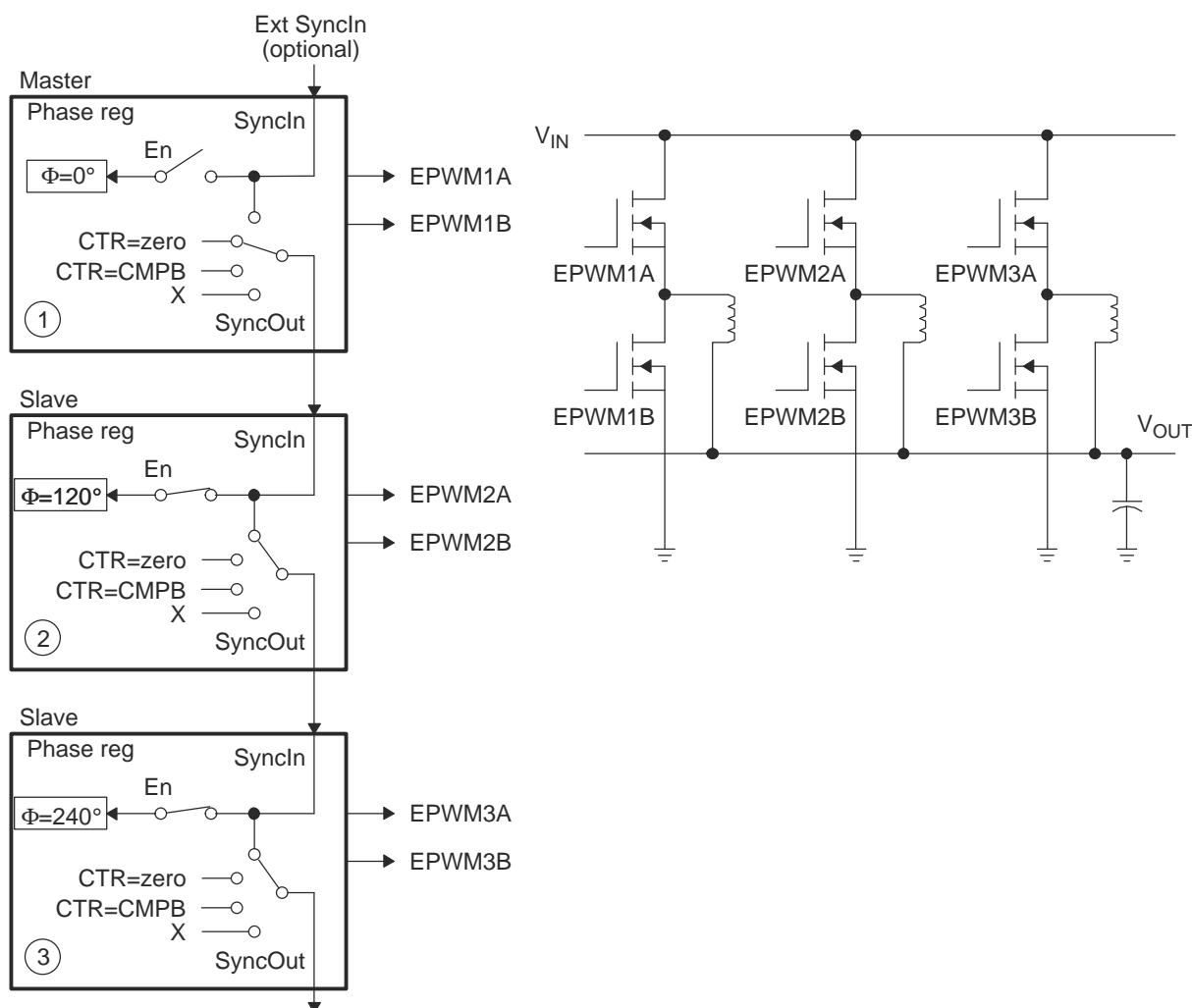


Figure 3-64. Control of a 3-Phase Interleaved DC/DC Converter

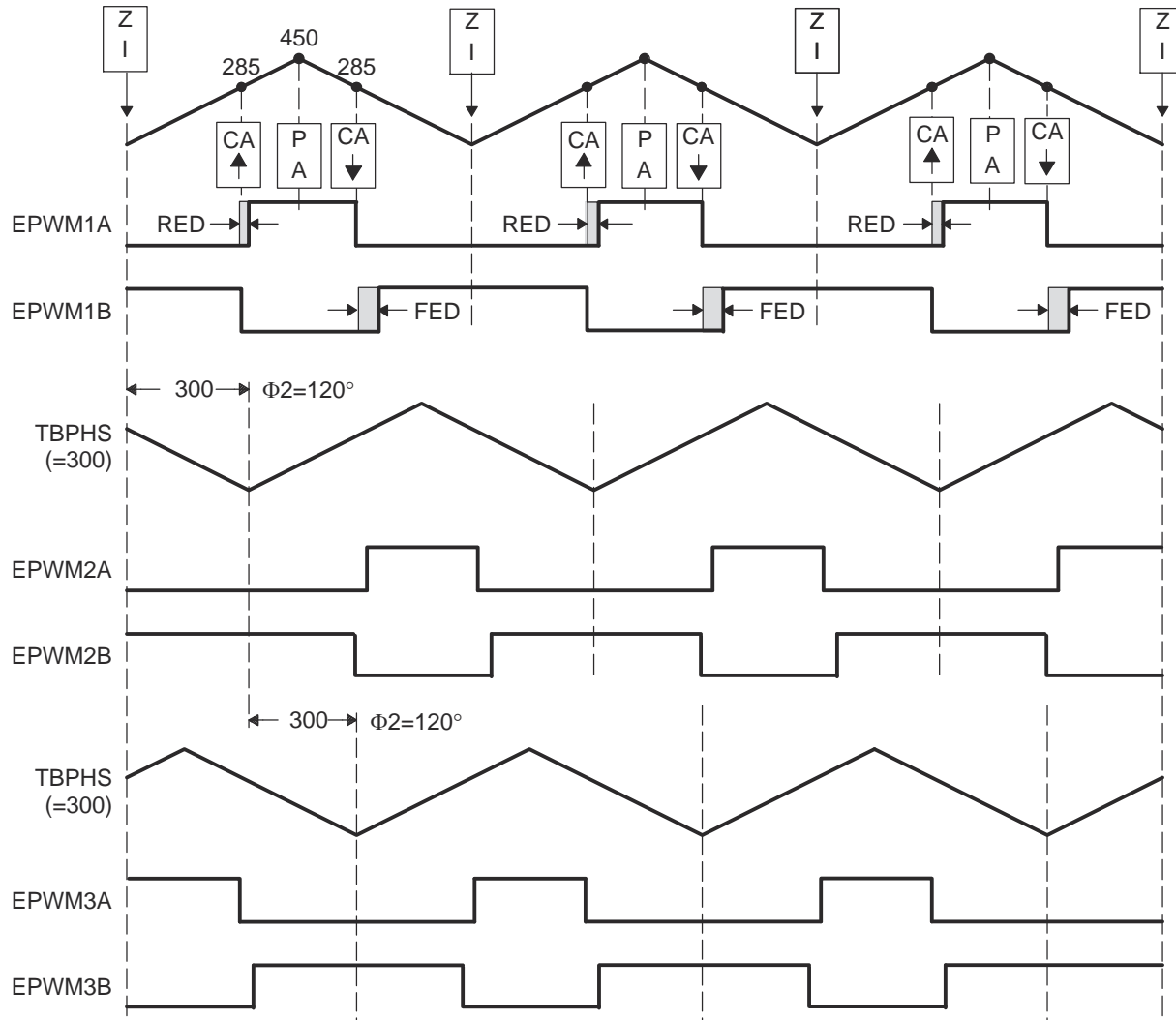


Figure 3-65. 3-Phase Interleaved DC/DC Converter Waveforms for [Figure 3-64](#)

**Example 3-12. Code Snippet for Configuration in Figure 3-64**

```

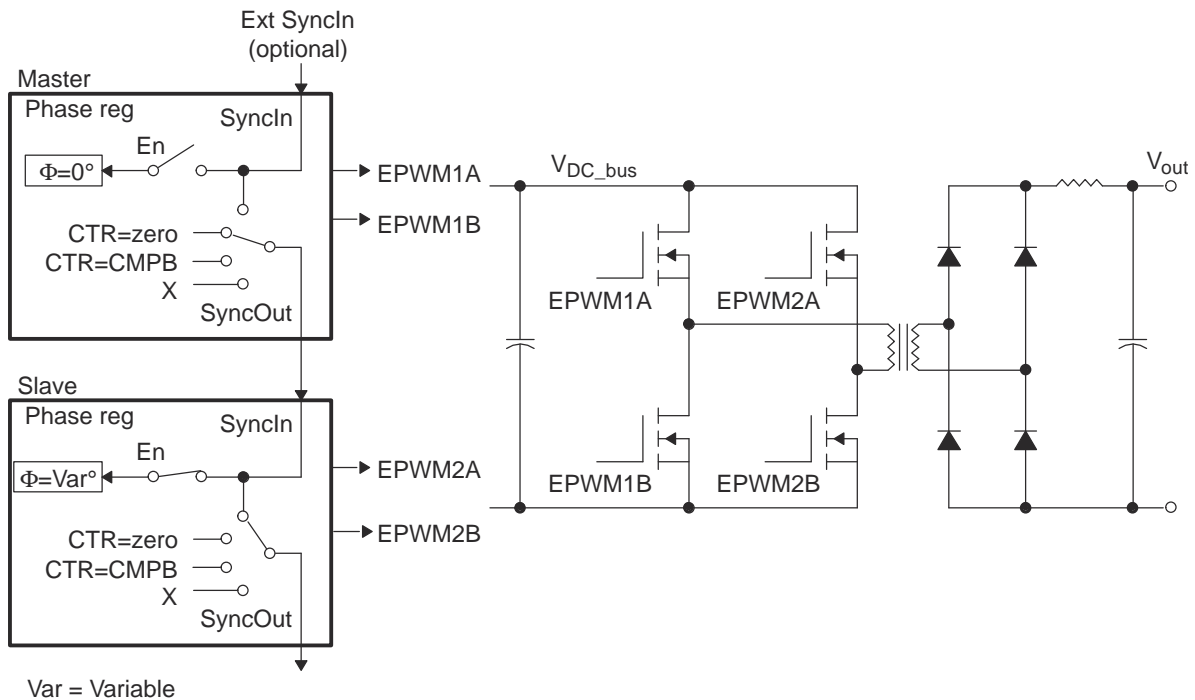
//=====
// Config
// Initialization Time
//=====
// EPWM Module 1 config
EPwm1Regs.TBPRD = 450; // Period = 900 TBCLK counts
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Master module
EPwm1Regs.TBCTL.bit.PRDL = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCSEL = TB_CTR_ZERO; // Sync down-stream module
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM1A
EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
EPwm1Regs.DBFED = 20; // FED = 20 TBCLKs
EPwm1Regs.DBRED = 20; // RED = 20 TBCLKs

EPwm2Regs.TBPRD = 450; // EPWM Module 2 config
EPwm2Regs.TBPHS.half.TBPHS = 300; // Period = 900 TBCLK counts
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Phase = 300/900 * 360 = 120 deg
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Symmetrical mode
EPwm2Regs.TBCTL.bit.PHSDIR = TB_DOWN; // Slave module
EPwm2Regs.TBCTL.bit.PRDL = TB_SHADOW; // Count DOWN on sync (=120 deg)
EPwm2Regs.TBCTL.bit.SYNCSEL = TB_SYNC_IN; // sync flow-through
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM2A
EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable dead-band module
EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi Complementary
EPwm2Regs.DBFED = 20; // FED = 20 TBCLKs
EPwm2Regs.DBRED = 20; // RED = 20 TBCLKs

EPwm3Regs.TBPRD = 450; // EPWM Module 3 config
EPwm3Regs.TBPHS.half.TBPHS = 300; // Period = 900 TBCLK counts
EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Phase = 300/900 * 360 = 120 deg
EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Symmetrical mode
EPwm3Regs.TBCTL.bit.PHSDIR = TB_UP; // Slave module
EPwm3Regs.TBCTL.bit.PRDL = TB_SHADOW; // Count UP on sync (=240 deg)
EPwm3Regs.TBCTL.bit.SYNCSEL = TB_SYNC_IN; // sync flow-through
EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm3Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM3Ai
EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;
EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
EPwm3Regs.DBFED = 20; // FED = 20 TBCLKs
EPwm3Regs.DBRED = 20; // RED = 20 TBCLKs
// Run Time (Note: Example execution of one run-time instant)
//=====
EPwm1Regs.CMPA.half.CMPA = 285; // adjust duty for output EPWM1A
EPwm2Regs.CMPA.half.CMPA = 285; // adjust duty for output EPWM2A
EPwm3Regs.CMPA.half.CMPA = 285; // adjust duty for output EPWM3A
    
```

### 3.3.9 Controlling Zero Voltage Switched Full Bridge (ZVSFB) Converter

The example given in [Figure 3-66](#) assumes a static or constant phase relationship between legs (modules). In such a case, control is achieved by modulating the duty cycle. It is also possible to dynamically change the phase value on a cycle-by-cycle basis. This feature lends itself to controlling a class of power topologies known as *phase-shifted full bridge*, or *zero voltage switched full bridge*. Here the controlled parameter is not duty cycle (this is kept constant at approximately 50 percent); instead it is the phase relationship between legs. Such a system can be implemented by allocating the resources of two PWM modules to control a single power stage, which in turn requires control of four switching elements. [Figure 3-67](#) shows a master/slave module combination synchronized together to control a full H-bridge. In this case, both master and slave modules are required to switch at the same PWM frequency. The phase is controlled by using the slave's phase register (TBPHS). The master's phase register is not used and therefore can be initialized to zero.



**Figure 3-66. Controlling a Full-H Bridge Stage ( $F_{PWM2} = F_{PWM1}$ )**

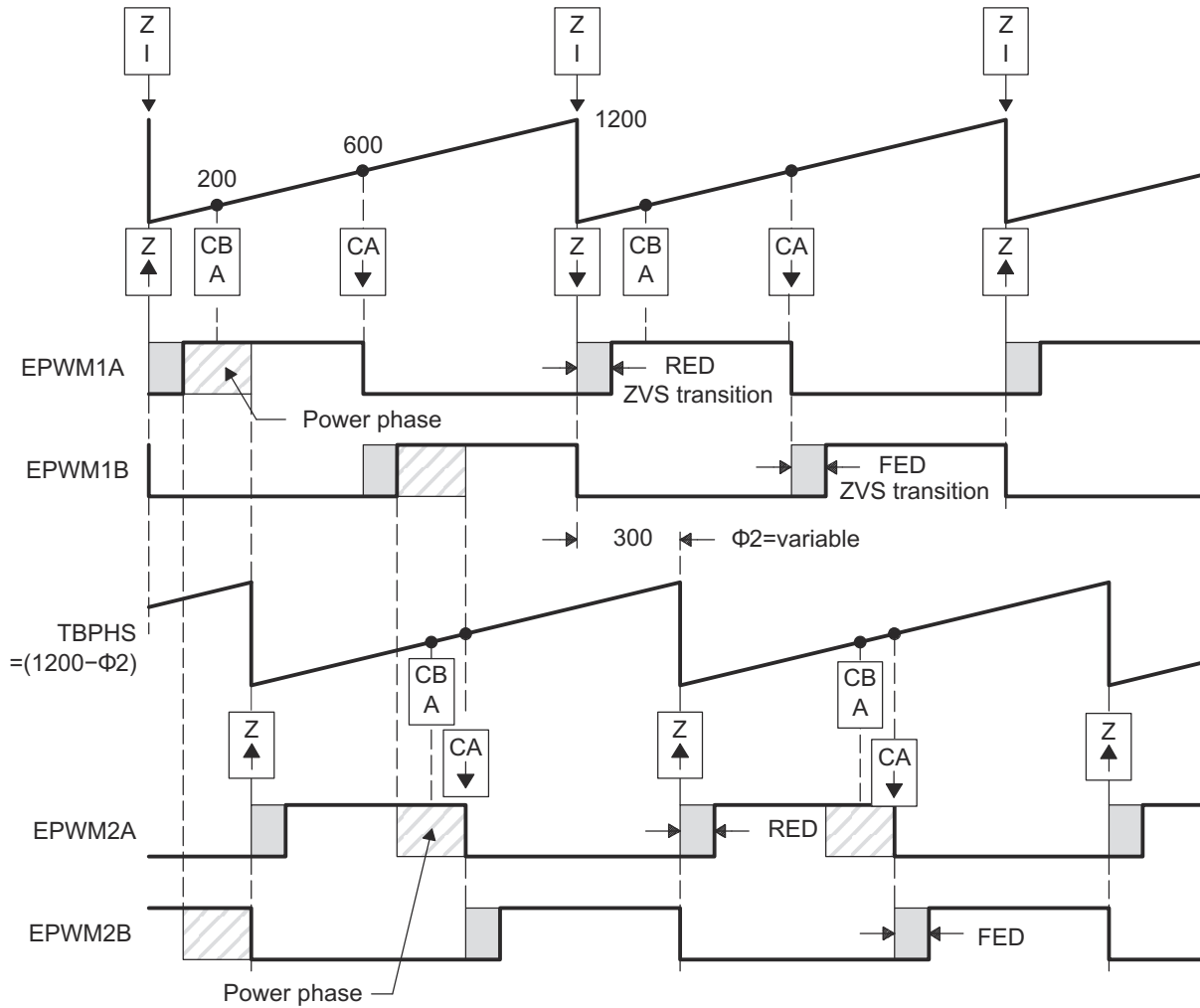


Figure 3-67. ZVS Full-H Bridge Waveforms



**Example 3-13. Code Snippet for Configuration in Figure 3-66**

```

//=====
// Config
//=====
// Initialization Time
//=====
// EPWM Module 1 config
EPwm1Regs.TBPRD = 1200; // Period = 1201 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 600; // Set 50% fixed duty for EPWM1A
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Asymmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Master module
EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // Sync down-stream module
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET; // set actions for EPWM1A
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
EPwm1Regs.DBFED = 50; // FED = 50 TBCLKs initially
EPwm1Regs.DBRED = 70; // RED = 70 TBCLKs initially
// EPWM Module 2 config
EPwm2Regs.TBPRD = 1200; // Period = 1201 TBCLK counts
EPwm2Regs.CMPA.half.CMPA = 600; // Set 50% fixed duty EPWM2A
EPwm2Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero initially
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Asymmetrical mode
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Slave module
EPwm2Regs.TBCTL.bit.PRDLD = TB_SHADOW;
EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN; // sync flow-through
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR=Zero
EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET; // set actions for EPWM2A
EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
EPwm2Regs.DBFED = 30; // FED = 30 TBCLKs initially
EPwm2Regs.DBRED = 40; // RED = 40 TBCLKs initially
// Run Time (Note: Example execution of one run-time instant)
//=====
EPwm2Regs.TBPHS = 1200-300; // Set Phase reg to
// 300/1200 * 360 = 90 deg
EPwm1Regs.DBFED = FED1_NewValue; // Update ZVS transition interval
EPwm1Regs.DBRED = RED1_NewValue; // Update ZVS transition interval
EPwm2Regs.DBFED = FED2_NewValue; // Update ZVS transition interval
EPwm2Regs.DBRED = RED2_NewValue; // Update ZVS transition interval
EPwm1Regs.CMPB = 200; // Adjust point-in-time for ADCSOC
trigger

```

### 3.3.10 Controlling a Peak Current Mode Controlled Buck Module

Peak current control techniques offer a number of benefits like automatic over current limiting, fast correction for input voltage variations, and reducing magnetic saturation. Figure 3-68 shows the use of ePWM1A along with the on-chip analog comparator for buck converter topology. The output current is sensed through a current sense resistor and fed to the positive terminal of the on-chip comparator. The internal programmable 10-bit DAC can be used to provide a reference peak current at the negative terminal of the comparator. Alternatively, an external reference could be connected at this input. The comparator output is an input to the digital compare submodule. The ePWM module is configured in such a way so as to trip the ePWM1A output as soon as the sensed current reaches the peak reference value. A cycle-by-cycle trip mechanism is used. Figure 3-69 shows the waveforms generated by the configuration.

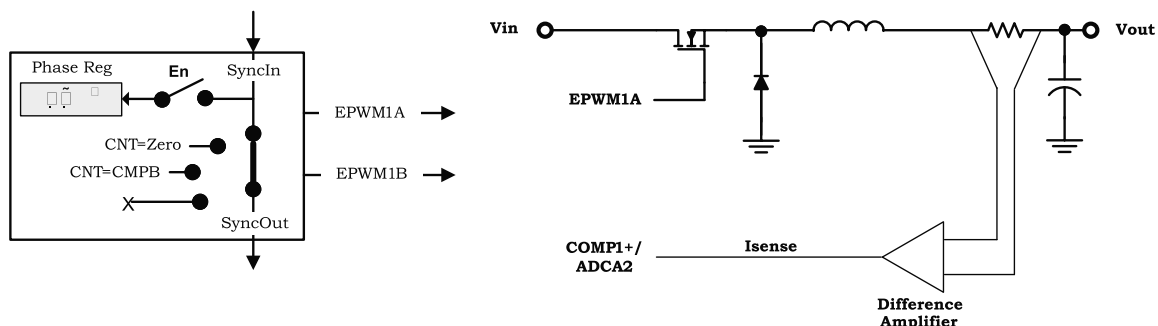


Figure 3-68. Peak Current Mode Control of a Buck Converter

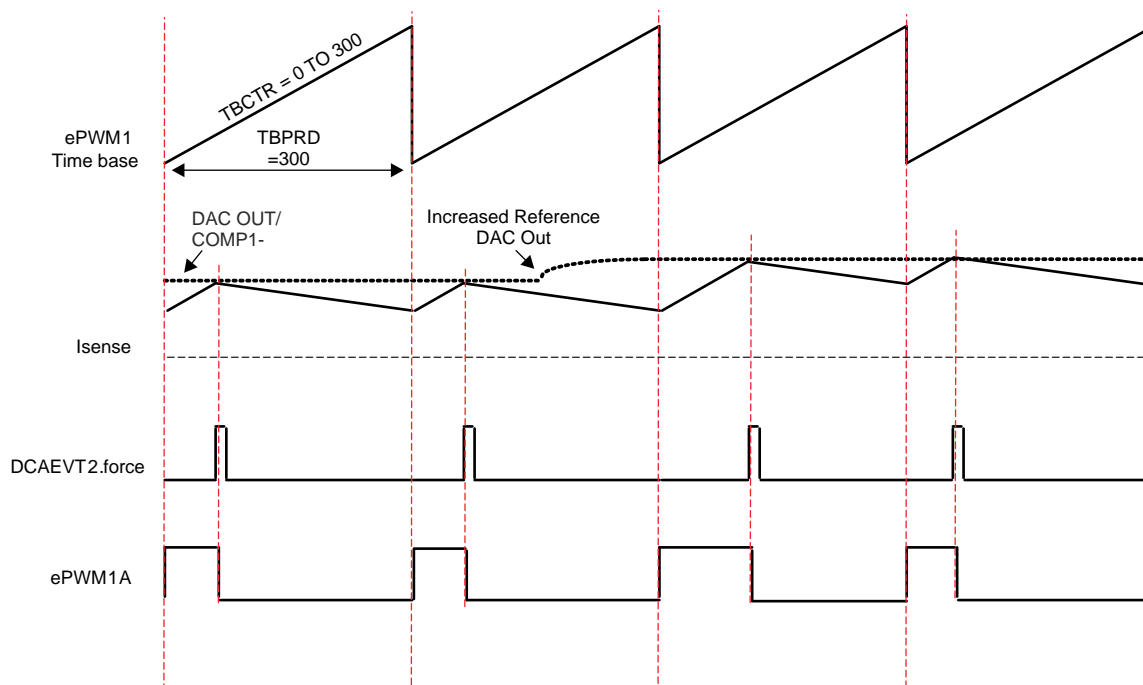


Figure 3-69. Peak Current Mode Control Waveforms for Figure 3-68

**Example 3-14. Code Snippet for Configuration in Figure 3-68**

```

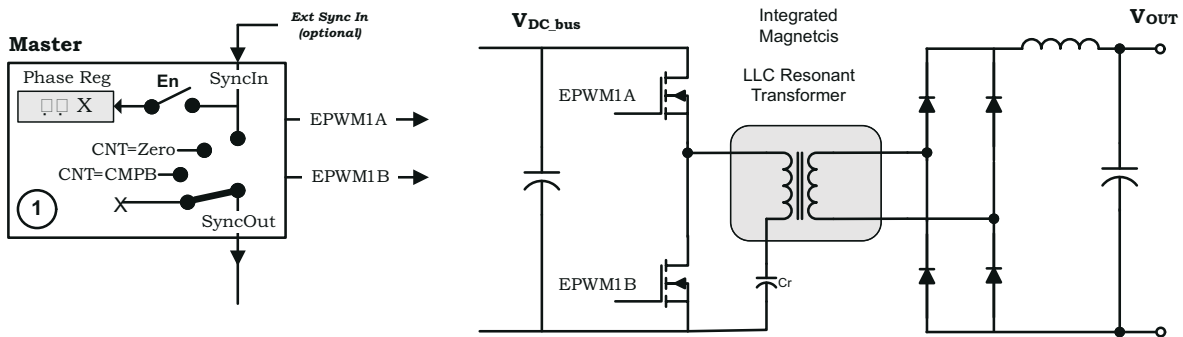
//=====
// Config
// Initialization Time
//=====
EPwm1Regs.TBPRD = 300;
// Period = 300 TBCLK counts // (200 KHz @ 60MHz clock)
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Asymmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Clock ratio to SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
Pwm1Regs.AQCTLA.bit.ZRO = AQ_SET; // Set PWM1A on Zero
// Define an event (DCAEVT2) based on
// Comparator 1 Output
EPwm1Regs.DCTRIPSEL.bit.DCAHCOMPSEL = DC_COMP1OUT; // DCAH = Comparator 1 output

EPwm1Regs.TZDCSEL.bit.DCAEVT2 = TZ_DCAH_HI; // DCAEVT2 = DCAH high(will become active
// as Comparator output goes high)
EPwm1Regs.DCACTL.bit.EVT2SRCSEL = DC_EVT2; // DCAEVT2 = DCAEVT2 (not filtered)
EPwm1Regs.DCACTL.bit.EVT2FRCSYNCSEL = DC_EVT_ASYNC; // Take async path // Enable DCAEVT2 as a
// one-shot trip source
// Note: DCxEVT1 events can be defined as
// one-shot.
// DCxEVT2 events can be defined as
// cycle-by-cycle.
EPwm1Regs.TZSEL.bit.DCAEVT2 = 1; // What do we want the DCAEVT1 and DCBEVT1
// events to do?
// DCAEVTx events can force EPWMxA
// DCBEVTx events can force EPWMxB
EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO; // EPWM1A will go low
//=====
// Run Time
//=====
// Adjust reference peak current to Comparator 1 negative input

```

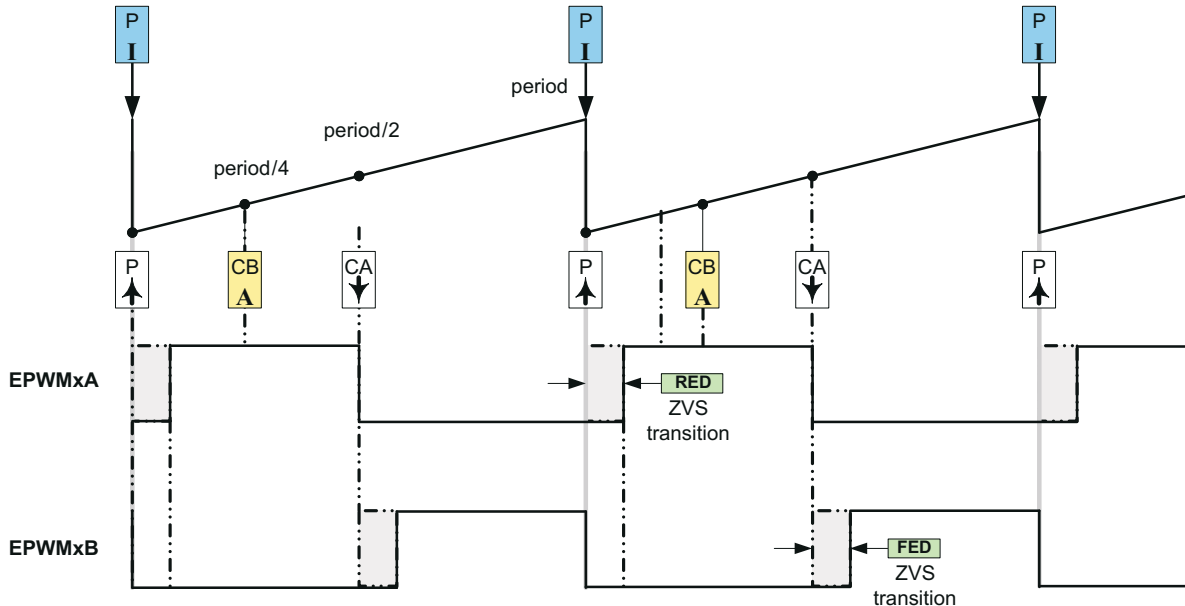
### 3.3.11 Controlling H-Bridge LLC Resonant Converter

For many years, various topologies of resonant converters have been well-known in the field of power electronics. In addition to these, H-bridge LLC resonant converter topology has recently gained popularity in many consumer electronics applications where high efficiency and power density are required. In this example, the single channel configuration of ePWM1 is detailed, yet the configuration can easily be extended to multichannel. Here, the controlled parameter is not duty cycle (this is kept constant at approximately 50 percent); instead it is frequency. Although the deadband is not controlled and kept constant as 300ns (that is, 18@60MHz TBCLK), it is up to the user to update it in real time to enhance the efficiency by adjusting enough time delay for soft switching.



NOTE:  $\Theta = X$  indicates value in phase register is 'don't care'

Figure 3-70. Control of Two Resonant Converter Stages



**P I** Indicates this event triggers an interrupt      **CB A** Indicates this event triggers an ADC start of conversion

Figure 3-71. H-Bridge LLC Resonant Converter PWM Waveforms

**Example 3-15. Code Snippet for Configuration in Figure 3-70**

```

//=====
// Config
//===== //
Initialization Time
//===== //
EPWMxA & EPWMxB config
EPwm1Regs.TBCTL.bit.PRDL = TB_IMMEDIATE; // Set immediate load
EPwm1Regs.TBPRD = period; // PWM frequency = 1 / period
EPwm1Regs.CMPA.half.CMPA = period/2; // Set duty as 50%
EPwm1Regs.CMPB = period/4; // Set duty as 25%
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set as master, phase =0
EPwm1Regs.TBCTR = 0; // Time base counter =0
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count-up mode: used for asymmetric PWM
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // Used to sync EPWM(n+1) "down-stream"
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Set the clock rate
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1; // Set the clock rate
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_PRD; // Load on CTR=PRD
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_PRD; // Load on CTR=PRD
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; // Shadow mode. Operates as a double buffer.
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW; // Shadow mode. Operates as a double buffer.
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET; // Set PWM1A on Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR; // Clear PWM1A on event A, up count
EPwm1Regs.AQCTLB.bit.CAU = AQ_SET; // Set PWM1B on event A, up count
EPwm1Regs.AQCTLB.bit.PR = AQ_CLEAR; // Clear PWM1B on PRD
EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL; // EPWMxA is the source for both delays
EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // Enable Dead-band module
EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active High Complementary (AHC)
EPwm1Regs.DBRED = 30; // RED = 30 TBCLKs initially
EPwm1Regs.DBFED = 30; // FED = 30 TBCLKs initially
// Configure TZ1 for short cct
// protection EALLOW;
// one-shot source EPwm1Regs.TZCTL.bit.TZA =

EPwm1Regs.TZSEL.bit.OSHT1 = 1;
TZ_FORCE_LO;

EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
EPwm1Regs.TZEINT.bit.OST = 1;
// set EPWM1A to low at fault
// set EPWM1B to low at fault instant
// Enable TZ interrupt EDIS;
// Enable HiRes option EALLOW;

EPwm1Regs.HRCNFG.all = 0x0;
EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP;
EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP;
EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_PRD;
EDIS; // Run Time (Note: Example execution of
// one run-time instant)

//=====
EPwm1Regs.TBPRD = period_new value; // Update new period
// EPwm1Regs.CMPA.half.CMPA= period_new
// value/2;
// Update new CMPA EPwm1Regs.CMPB= period_new
// value/4;
// Update new CMPB
// Update new CMPB
// Update new CMPB

```

### 3.4 Registers

This section includes the register layouts and bit description for the submodules.

#### 3.4.1 Time-Base Submodule Registers

This section describes the time-base submodule registers.

##### 3.4.1.1 Time-Base Control Register (TBCTL)

**Figure 3-72. Time-Base Control Register (TBCTL)**

15	14	13	12	10	9	8	
FREE, SOFT		PHSDIR	CLKDIV		HSPCLKDIV		
R/W-0		R/W-0	R/W-0		R/W-0,0,1		
7	6	5	4	3	2	1	0
HSPCLKDIV		SWFSYNC	SYNCOSEL		PRDL	PHSEN	CTRM
R/W-0,0,1		R/W-0	R/W-0		R/W-0	R/W-0	R/W-11

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

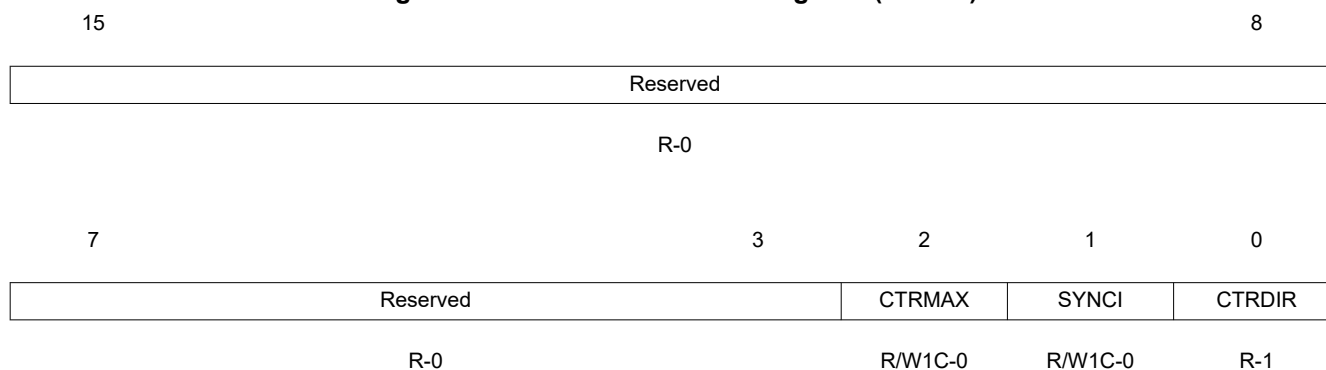
**Table 3-21. Time-Base Control Register (TBCTL) Field Descriptions**

Bit	Field	Value	Description
15-14	FREE, SOFT	00 01 1x	Emulation Mode Bits. These bits select the behavior of the ePWM time-base counter during emulation events: <ul style="list-style-type: none"> <li>00 Stop after the next time-base counter increment or decrement</li> <li>01 Stop when counter completes a whole cycle:                             <ul style="list-style-type: none"> <li>• Up-count mode: stop when the time-base counter = period (TBCTR = TBPRD)</li> <li>• Down-count mode: stop when the time-base counter = 0x0000 (TBCTR = 0x0000)</li> <li>• Up-down-count mode: stop when the time-base counter = 0x0000 (TBCTR = 0x0000)</li> </ul> </li> <li>1x Free run</li> </ul>
13	PHSDIR	0 1	Phase Direction Bit. This bit is only used when the time-base counter is configured in the up-down-count mode. The PHSDIR bit indicates the direction the time-base counter (TBCTR) will count after a synchronization event occurs and a new phase value is loaded from the phase (TBPHS) register. This is irrespective of the direction of the counter before the synchronization event.. In the up-count and down-count modes this bit is ignored. <ul style="list-style-type: none"> <li>0 Count down after the synchronization event.</li> <li>1 Count up after the synchronization event.</li> </ul>
12-10	CLKDIV	000 001 010 011 100 101 110 111	Time-base Clock Prescale Bits These bits determine part of the time-base clock prescale value. $TBCLK = SYSCLKOUT / (HSPCLKDIV \times CLKDIV)$ <ul style="list-style-type: none"> <li>000 /1 (default on reset)</li> <li>001 /2</li> <li>010 /4</li> <li>011 /8</li> <li>100 /16</li> <li>101 /32</li> <li>110 /64</li> <li>111 /128</li> </ul>

**Table 3-21. Time-Base Control Register (TBCTL) Field Descriptions (continued)**

Bit	Field	Value	Description
9-7	HSPCLKDIV	000 /1 001 /2 (default on reset) 010 /4 011 /6 100 /8 101 /10 110 /12 111 /14	High Speed Time-base Clock Prescale Bits  These bits determine part of the time-base clock prescale value. $TBCLK = SYSCLKOUT / (HSPCLKDIV \times CLKDIV)$  This divisor emulates the HSPCLK in the TMS320x281x system as used on the Event Manager (EV) peripheral.
6	SWFSYNC	0 1	Software Forced Synchronization Pulse  Writing a 0 has no effect and reads always return a 0. Writing a 1 forces a one-time synchronization pulse to be generated. This event is ORed with the EPWMxSYNCl input of the ePWM module. SWFSYNC is valid (operates) only when EPWMxSYNCl is selected by SYNCOSSEL = 00.
5-4	SYNCOSSEL	00 01 10 11	Synchronization Output Select. These bits select the source of the EPWMxSYNCO signal.  EPWMxSYNCO: 01 CTR = zero: Time-base counter equal to zero (TBCTR = 0x0000) 10 CTR = CMPB : Time-base counter equal to counter-compare B (TBCTR = CMPB) 11 Disable EPWMxSYNCO signal
3	PRDL	0 1	Active Period Register Load From Shadow Register Select  0 The period register (TBPRD) is loaded from its shadow register when the time-base counter, TBCTR, is equal to zero. A write or read to the TBPRD register accesses the shadow register. 1 Load the TBPRD register immediately without using a shadow register. A write or read to the TBPRD register directly accesses the active register.
2	PHSEN	0 1	Counter Register Load From Phase Register Enable  0 Do not load the time-base counter (TBCTR) from the time-base phase register (TBPHS) 1 Load the time-base counter with the phase register when an EPWMxSYNCl input signal occurs or when a software synchronization is forced by the SWFSYNC bit, or when a digital compare sync event occurs.
1-0	CTRM	00 01 10 11	Counter Mode  The time-base counter mode is normally configured once and not changed during normal operation. If you change the mode of the counter, the change will take effect at the next TBCLK edge and the current counter value shall increment or decrement from the value before the mode change.  These bits set the time-base counter mode of operation as follows:

### 3.4.1.2 Time-Base Status Register (TBSTS)

**Figure 3-73. Time-Base Status Register (TBSTS)**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

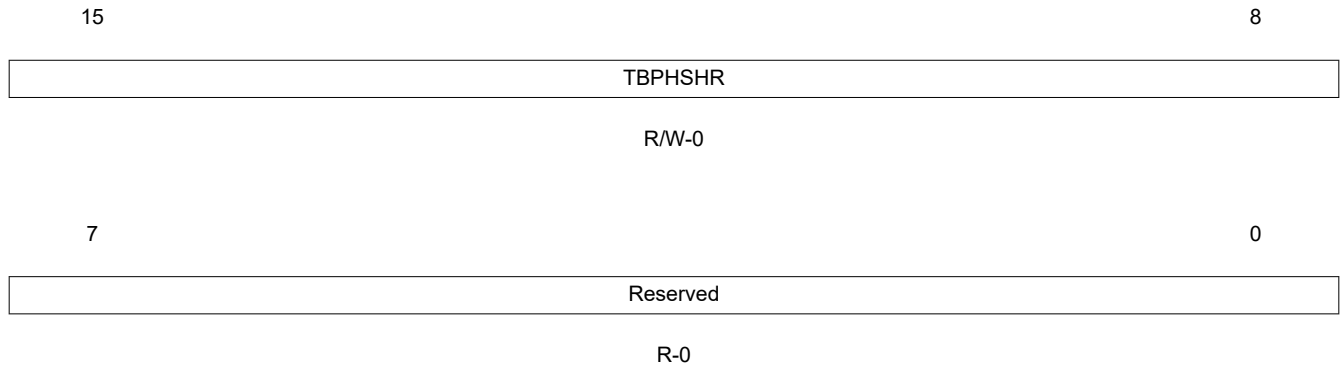
**Table 3-22. Time-Base Status Register (TBSTS) Field Descriptions**

Bit	Field	Value	Description
15-3	Reserved		Reserved
2	CTRMAX	0 1	Time-Base Counter Max Latched Status Bit Reading a 0 indicates the time-base counter never reached its maximum value. Writing a 0 will have no effect. Reading a 1 on this bit indicates that the time-base counter reached the max value 0xFFFF. Writing a 1 to this bit will clear the latched event.
1	SYNCI	0 1	Input Synchronization Latched Status Bit Writing a 0 will have no effect. Reading a 0 indicates no external synchronization event has occurred. Reading a 1 on this bit indicates that an external synchronization event has occurred (EPWMxSYNCI). Writing a 1 to this bit will clear the latched event.
0	CTRDIR	0 1	Time-Base Counter Direction Status Bit. At reset, the counter is frozen; therefore, this bit has no meaning. To make this bit meaningful, you must first set the appropriate mode via TBCTL[CTRMODE]. 0 Time-Base Counter is currently counting down. 1 Time-Base Counter is currently counting up.



### 3.4.1.3 Time-Base Phase High Resolution Register (TBPHSHR)

**Figure 3-74. Time-Base Phase High Resolution Register (TBPHSHR)**



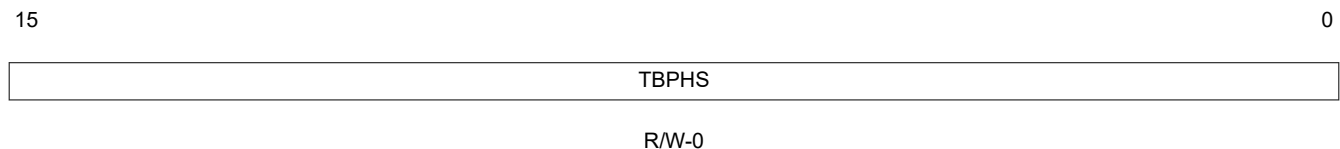
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-23. Time-Base Phase High Resolution Register (TBPHSHR) Field Descriptions**

Bit	Field	Value	Description
15-8	TBPHSHR	00-FFh	Time base phase high-resolution bits
7-0	Reserved		Reserved

### 3.4.1.4 Time-Base Phase Register (TBPHS)

**Figure 3-75. Time-Base Phase Register (TBPHS)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-24. Time-Base Phase Register (TBPHS) Field Descriptions**

Bits	Name	Value	Description
15-0	TBPHS	0000-FFFF	<p>These bits set time-base counter phase of the selected ePWM relative to the time-base that is supplying the synchronization input signal.</p> <ul style="list-style-type: none"> <li>If TBCTL[PHSEN] = 0, then the synchronization event is ignored and the time-base counter is not loaded with the phase.</li> <li>If TBCTL[PHSEN] = 1, then the time-base counter (TBCTR) will be loaded with the phase (TBPHS) when a synchronization event occurs. The synchronization event can be initiated by the input synchronization signal (EPWMxSYNCl) or by a software forced synchronization.</li> </ul>

### 3.4.1.5 Time-Base Counter Register (TBCTR)

**Figure 3-76. Time-Base Counter Register (TBCTR)**

15

0



R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-25. Time-Base Counter Register (TBCTR) Field Descriptions**

Bits	Name	Value	Description
15-0	TBCTR	0000-FFFF	Reading these bits gives the current time-base counter value.  Writing to these bits sets the current time-base counter value. The update happens as soon as the write occurs; the write is NOT synchronized to the time-base clock (TBCLK) and the register is not shadowed.

### 3.4.1.6 Time-Base Period Register (TBPRD)

**Figure 3-77. Time-Base Period Register (TBPRD)**

15

0



R/W-0

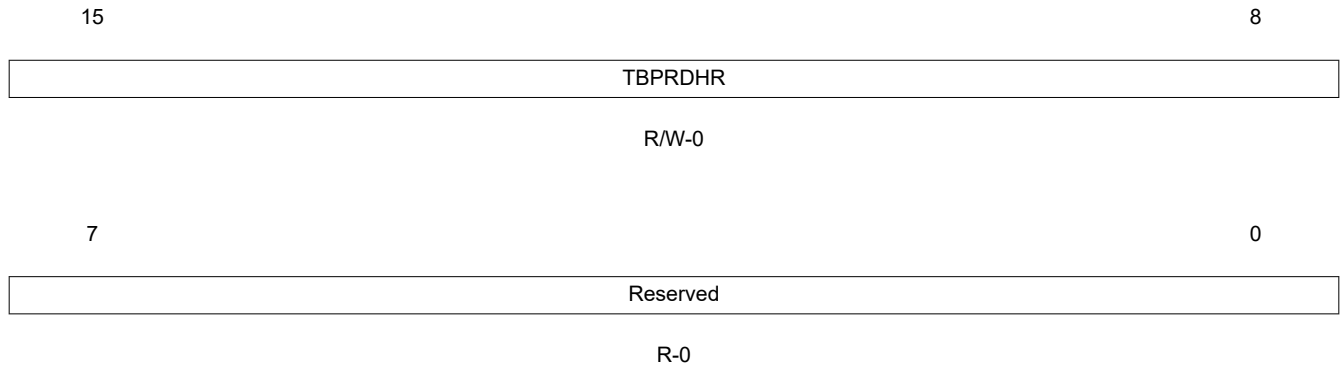
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-26. Time-Base Period Register (TBPRD) Field Descriptions**

Bit	Field	Value	Description
15-0	TBPRD	0000-FFFFh	These bits determine the period of the time-base counter. This sets the PWM frequency. Shadowing of this register is enabled and disabled by the TBCTL[PRDLD] bit. By default this register is shadowed. <ul style="list-style-type: none"> <li>• If TBCTL[PRDLD] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the active register will be loaded from the shadow register when the time-base counter equals zero.</li> <li>• If TBCTL[PRDLD] = 1, then the shadow is disabled and any write or read will go directly to the active register, that is the register actively controlling the hardware.</li> <li>• The active and shadow registers share the same memory map address.</li> </ul>

### 3.4.1.7 Time Base Period High Resolution Register (TBPRDHR)

**Figure 3-78. Time Base Period High Resolution Register (TBPRDHR)**



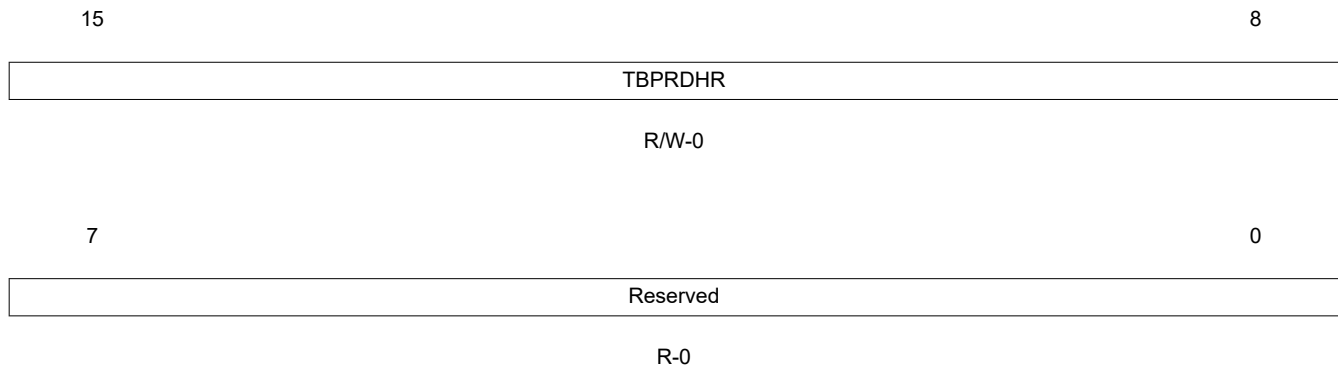
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-27. Time Base Period High Resolution Register (TBPRDHR) Field Descriptions**

Bit	Field	Value	Description
15-8	TBPRDHR	00-FFh	Period High Resolution Bits These 8-bits contain the high-resolution portion of the period value. The TBPRDHR register is not affected by the TBCTL[PRDL] bit. Reads from this register always reflect the shadow register. Likewise writes are also to the shadow register. The TBPRDHR register is only used when the high resolution period feature is enabled. This register is only available with ePWM modules which support high-resolution period control.
7-0	Reserved	0	Reserved

### 3.4.1.8 Time-Base Period High Resolution Mirror Register (TBPRDHRM)

**Figure 3-79. Time-Base Period High Resolution Mirror Register (TBPRDHRM)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-28. Time-Base Period High Resolution Mirror Register (TBPRDHRM) Field Descriptions**

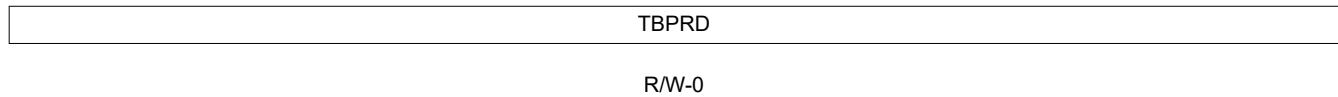
Bit	Field	Value	Description
15-8	TBPRDHR	00-FFh	Period High Resolution Bits These 8-bits contain the high-resolution portion of the period value TBPRD provides backwards compatibility with earlier ePWM modules. The mirror registers (TBPRDM and TBPRDHRM) allow for 32-bit writes to TBPRDHR in one access. Due to the odd-numbered memory address location of the TBPRD legacy register, a 32-bit write is not possible with TBPRD and TBPRDHR. The TBPRDHRM register is not affected by the TBCTL[PRDL] bit Writes to both the TBPRDHR and TBPRDM locations access the high-resolution (least significant 8-bit) portion of the Time Base Period value. The only difference is that unlike TBPRDHR, reads from the mirror register TBPRDHRM, are indeterminate (reserved for TI Test). The TBPRDHRM register is available with ePWM modules which support high-resolution period control and is used only when the high resolution period feature is enabled.
7-0	Reserved	00-FFh	Reserved for TI Test

### 3.4.1.9 Time-Base Period Mirror Register (TBPRDM)

**Figure 3-80. Time-Base Period Mirror Register (TBPRDM)**

15

0



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-29. Time-Base Period Mirror Register (TBPRDM) Field Descriptions**

Bit	Field	Value	Description
15-0	TBPRD	0000-FFFFh	<p>TBPRDM and TBPRD can both be used to access the time-base period.</p> <p>TBPRD provides backwards compatibility with earlier ePWM modules. The mirror registers (TBPRDM and TBPRDHRM) allow for 32-bit writes to TBPRDHR in one access. Due to the odd address memory location of the TBPRD legacy register, a 32-bit write is not possible.</p> <p>By default writes to this register are shadowed. Unlike the TBPRD register, reads of TBPRDM always return the active register value. Shadowing is enabled and disabled by the TBCTL[PRDL] bit.</p> <ul style="list-style-type: none"> <li>If TBCTL[PRDL] = 0, then the shadow is enabled and any write will automatically go to the shadow register. In this case the active register will be loaded from the shadow register when the time-base counter equals zero. Reads return the active value.</li> <li>If TBCTL[PRDL] = 1, then the shadow is disabled and any write to this register will go directly to the active register controlling the hardware. Likewise reads return the active value.</li> </ul>

### 3.4.2 Counter-Compare Submodule Registers

This section describes the counter-compare submodule control and status registers.

#### 3.4.2.1 Counter-Compare Control (CMPCTL) Register

**Figure 3-81. Counter-Compare Control (CMPCTL) Register**

15				10				9	8
Reserved							SHDWBFULL	SHDWAFULL	
R-0							R-0	R-0	
7	6	5	4	3	2	1	0		
Reserved	SHDWBMODE	Reserved	SHDWAMODE	LOADBMODE		LOADAMODE			
R-0	R/W-0	R-0	R/W-0	R/W-0		R/W-0			

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

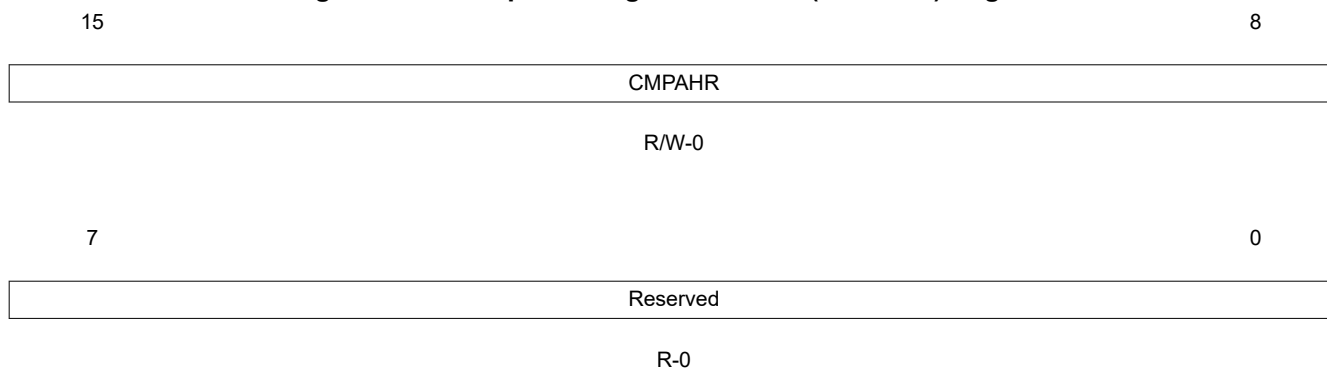
**Table 3-30. Counter-Compare Control (CMPCTL) Register Field Descriptions**

Bits	Name	Value	Description
15-10	Reserved		Reserved
9	SHDWBFULL	0 1	Counter-compare B (CMPB) Shadow Register Full Status Flag This bit self clears once a load-strobe occurs. 0 CMPB shadow FIFO not full yet 1 Indicates the CMPB shadow FIFO is full; a CPU write will overwrite current shadow value.
8	SHDWAFULL	0 1	Counter-compare A (CMPA) Shadow Register Full Status Flag The flag bit is set when a 32-bit write to CMPA:CMPAHR register or a 16-bit write to CMPA register is made. A 16-bit write to CMPAHR register will not affect the flag. This bit self clears once a load-strobe occurs. 0 CMPA shadow FIFO not full yet 1 Indicates the CMPA shadow FIFO is full, a CPU write will overwrite the current shadow value.
7	Reserved		Reserved
6	SHDWBMODE	0 1	Counter-compare B (CMPB) Register Operating Mode 0 Shadow mode. Operates as a double buffer. All writes via the CPU access the shadow register. 1 Immediate mode. Only the active compare B register is used. All writes and reads directly access the active register for immediate compare action.
5	Reserved		Reserved
4	SHDWAMODE	0 1	Counter-compare A (CMPA) Register Operating Mode 0 Shadow mode. Operates as a double buffer. All writes via the CPU access the shadow register. 1 Immediate mode. Only the active compare register is used. All writes and reads directly access the active register for immediate compare action
3-2	LOADBMODE	00 01 10 11	Active Counter-Compare B (CMPB) Load From Shadow Select Mode This bit has no effect in immediate mode (CMPCTL[SHDWBMODE] = 1). 00 Load on CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000) 01 Load on CTR = PRD: Time-base counter equal to period (TBCTR = TBPRD) 10 Load on either CTR = Zero or CTR = PRD 11 Freeze (no loads possible)

**Table 3-30. Counter-Compare Control (CMPCTL) Register Field Descriptions (continued)**

Bits	Name	Value	Description
1-0	LOADAMODE		Active Counter-Compare A (CMPA) Load From Shadow Select Mode. This bit has no effect in immediate mode (CMPCTL[SHDWAMODE] = 1).
		00	Load on CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000)
		01	Load on CTR = PRD: Time-base counter equal to period (TBCTR = TBPRD)
		10	Load on either CTR = Zero or CTR = PRD
		11	Freeze (no loads possible)

### 3.4.2.2 Compare A High Resolution (CMPAHR) Register

**Figure 3-82. Compare A High Resolution (CMPAHR) Register**

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-31. Compare A High Resolution (CMPAHR) Register Field Descriptions**

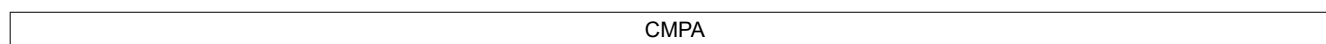
Bit	Field	Value	Description
15-8	CMPAHR	00-FFh	These 8-bits contain the high-resolution portion (least significant 8-bits) of the counter-compare A value. CMPA:CMPAHR can be accessed in a single 32-bit read/write. Shadowing is enabled and disabled by the CMPCTL[SHDWAMODE] bit as described for the CMPA register.
7-0	Reserved		Reserved for TI Test

### 3.4.2.3 Counter-Compare A (CMPA) Register

**Figure 3-83. Counter-Compare A (CMPA) Register**

15

0



R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-32. Counter-Compare A (CMPA) Register Field Descriptions**

Bits	Name	Description
15-0	CMPA	<p>The value in the active CMPA register is continuously compared to the time-base counter (TBCTR). When the values are equal, the counter-compare module generates a "time-base counter equal to counter compare A" event. This event is sent to the action-qualifier where it is qualified and converted it into one or more actions. These actions can be applied to either the EPWMxA or the EPWMxB output depending on the configuration of the AQCTLA and AQCTLB registers. The actions that can be defined in the AQCTLA and AQCTLB registers include:</p> <ul style="list-style-type: none"> <li>• Do nothing; the event is ignored.</li> <li>• Clear: Pull the EPWMxA and/or EPWMxB signal low</li> <li>• Set: Pull the EPWMxA and/or EPWMxB signal high</li> <li>• Toggle the EPWMxA and/or EPWMxB signal</li> </ul> <p>Shadowing of this register is enabled and disabled by the CMPCTL[SHDWAMODE] bit. By default this register is shadowed.</p> <ul style="list-style-type: none"> <li>• If CMPCTL[SHDWAMODE] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the CMPCTL[LOADAMODE] bit field determines which event will load the active register from the shadow register.</li> <li>• Before a write, the CMPCTL[SHDWAFULL] bit can be read to determine if the shadow register is currently full.</li> <li>• If CMPCTL[SHDWAMODE] = 1, then the shadow register is disabled and any write or read will go directly to the active register, that is the register actively controlling the hardware.</li> <li>• In either mode, the active and shadow registers share the same memory map address.</li> </ul>



### 3.4.2.4 Counter-Compare B (CMPB) Register

**Figure 3-84. Counter-Compare B (CMPB) Register**

15

0

CMPB
------

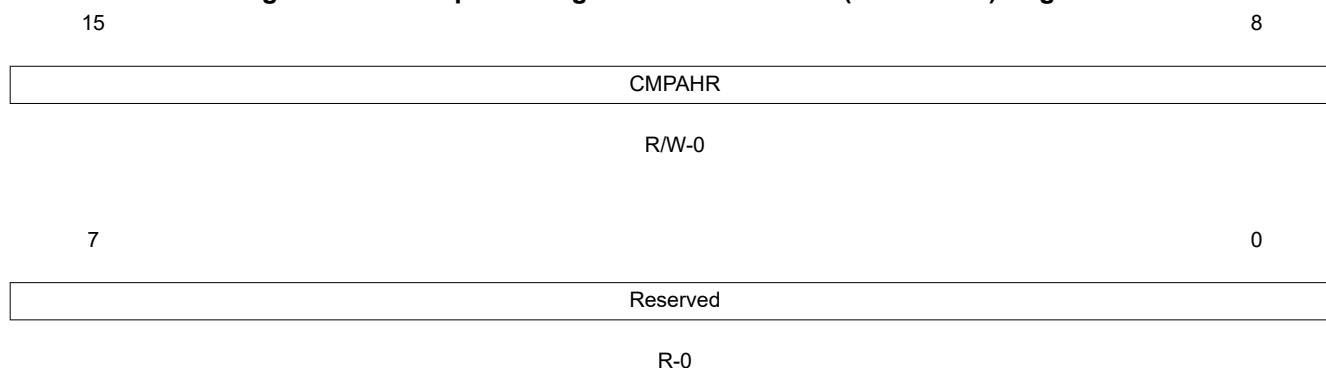
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-33. Counter-Compare B (CMPB) Register Field Descriptions**

Bits	Name	Description
15-0	CMPB	<p>The value in the active CMPB register is continuously compared to the time-base counter (TBCTR). When the values are equal, the counter-compare module generates a "time-base counter equal to counter compare B" event. This event is sent to the action-qualifier where it is qualified and converted it into one or more actions. These actions can be applied to either the EPWMxA or the EPWMxB output depending on the configuration of the AQCTLA and AQCTLB registers. The actions that can be defined in the AQCTLA and AQCTLB registers include:</p> <ul style="list-style-type: none"> <li>• Do nothing. event is ignored.</li> <li>• Clear: Pull the EPWMxA and/or EPWMxB signal low</li> <li>• Set: Pull the EPWMxA and/or EPWMxB signal high</li> <li>• Toggle the EPWMxA and/or EPWMxB signal</li> </ul> <p>Shadowing of this register is enabled and disabled by the CMPCTL[SHDWBMODE] bit. By default this register is shadowed.</p> <ul style="list-style-type: none"> <li>• If CMPCTL[SHDWBMODE] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the CMPCTL[LOADBMODE] bit field determines which event will load the active register from the shadow register:</li> <li>• Before a write, the CMPCTL[SHDWBFULL] bit can be read to determine if the shadow register is currently full.</li> <li>• If CMPCTL[SHDWBMODE] = 1, then the shadow register is disabled and any write or read will go directly to the active register, that is the register actively controlling the hardware.</li> <li>• In either mode, the active and shadow registers share the same memory map address.</li> </ul>

### 3.4.2.5 Compare A High-Resolution Mirror (CMPAHRM) Register

**Figure 3-85. Compare A High-Resolution Mirror (CMPAHRM) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-34. Compare A High-Resolution Mirror (CMPAHRM) Register Field Descriptions**

Bit	Field	Value	Description
15-8	CMPAHR	00-FFh	Compare A High Resolution Bits Writes to both the CMPAHR and CMPAHRM locations access the high-resolution (least significant 8-bit) portion of the Counter Compare A value. The only difference is that unlike CMPAHR, reads from the mirror register, CMPAHRM, are indeterminate (reserved for TI Test). By default writes to this register are shadowed. Shadowing is enabled and disabled by the CMPCTL[SHDWAMODE] bit as described for the CMPAM register.
7-0	Reserved		Reserved for TI Test

### 3.4.2.6 Counter-Compare A Mirror (CMPAM) Register

**Figure 3-86. Counter-Compare A Mirror (CMPAM) Register**

15

0

CMPA
------

R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-35. Counter-Compare A Mirror (CMPAM) Register Field Descriptions**

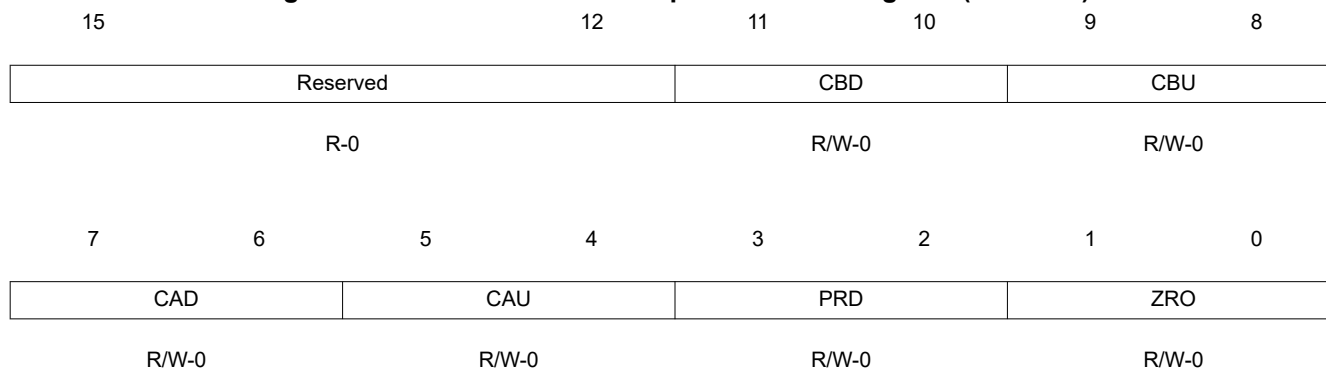
Bit	Field	Value	Description
15-0	CMPA	0000-FFFFh	<p>CMPA and CMPAM can both be used to access the counter-compare A value. The only difference is that the mirror register always reads back the active value.</p> <p>By default writes to this register are shadowed. Unlike the CMPA register, reads of CMPAM always return the active register value. Shadowing is enabled and disabled by the CMPCTL[SHDWAMODE] bit.</p> <ul style="list-style-type: none"> <li>• If CMPCTL[SHDWAMODE] = 0, then the shadow is enabled and any write will automatically go to the shadow register. All reads will reflect the active register value. In this case, the CMPCTL[LOADAMODE] bit field determines which event will load the active register from the shadow register.</li> <li>• Before a write, the CMPCTL[SHDWAFULL] bit can be read to determine if the shadow register is currently full.</li> <li>• If CMPCTL[SHDWAMODE] = 1, then the shadow register is disabled and any write will go directly to the active register, that is the register actively controlling the hardware.</li> </ul>

### 3.4.3 Action-Qualifier Submodule Registers

This section describes the action-qualifier submodule registers.

#### 3.4.3.1 Action-Qualifier Output A Control Register (AQCTLA)

**Figure 3-87. Action-Qualifier Output A Control Register (AQCTLA)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

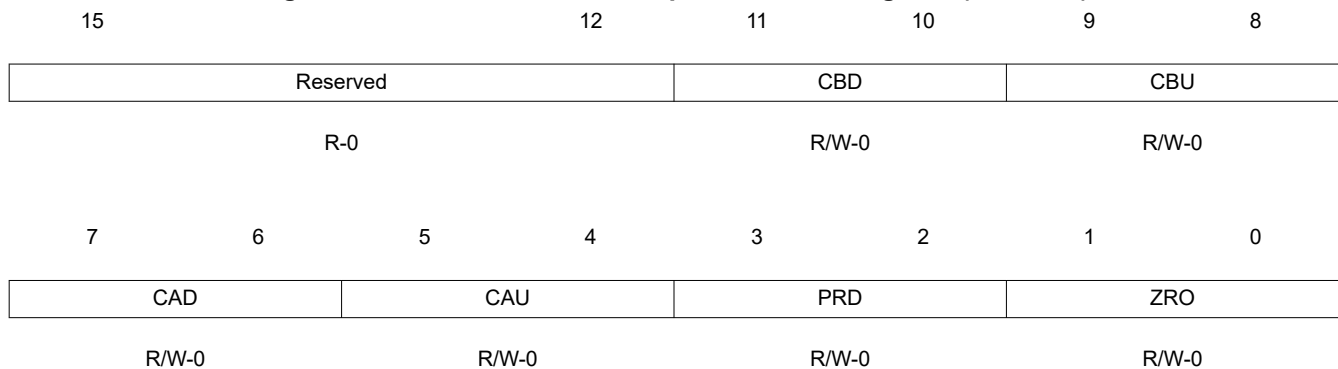
**Table 3-36. Action-Qualifier Output A Control Register (AQCTLA) Field Descriptions**

Bits	Name	Value	Description
15-12	Reserved		Reserved
11-10	CBD	00 Do nothing (action disabled) 01 Clear: force EPWMxA output low. 10 Set: force EPWMxA output high. 11 Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.	Action when the time-base counter equals the active CMPB register and the counter is decrementing.
9-8	CBU	00 Do nothing (action disabled) 01 Clear: force EPWMxA output low. 10 Set: force EPWMxA output high. 11 Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.	Action when the counter equals the active CMPB register and the counter is incrementing.
7-6	CAD	00 Do nothing (action disabled) 01 Clear: force EPWMxA output low. 10 Set: force EPWMxA output high. 11 Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.	Action when the counter equals the active CMPA register and the counter is decrementing.
5-4	CAU	00 Do nothing (action disabled) 01 Clear: force EPWMxA output low. 10 Set: force EPWMxA output high. 11 Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.	Action when the counter equals the active CMPA register and the counter is incrementing.
3-2	PRD	00 Do nothing (action disabled) 01 Clear: force EPWMxA output low. 10 Set: force EPWMxA output high. 11 Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.	Action when the counter equals the period. Note: By definition, in count up-down mode when the counter equals period the direction is defined as 0 or counting down.

**Table 3-36. Action-Qualifier Output A Control Register (AQCTLA) Field Descriptions (continued)**

Bits	Name	Value	Description
1-0	ZRO		Action when counter equals zero.  Note: By definition, in count up-down mode when the counter equals 0 the direction is defined as 1 or counting up.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxA output low.
		10	Set: force EPWMxA output high.
		11	Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.

### 3.4.3.2 Action-Qualifier Output B Control Register (AQCTLB)

**Figure 3-88. Action-Qualifier Output B Control Register (AQCTLB)**

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-37. Action-Qualifier Output B Control Register (AQCTLB) Field Descriptions**

Bits	Name	Value	Description
15-12	Reserved		
11-10	CBD		Action when the counter equals the active CMPB register and the counter is decrementing.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxB output low.
		10	Set: force EPWMxB output high.
		11	Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.
9-8	CBU		Action when the counter equals the active CMPB register and the counter is incrementing.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxB output low.
		10	Set: force EPWMxB output high.
		11	Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.
7-6	CAD		Action when the counter equals the active CMPA register and the counter is decrementing.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxB output low.
		10	Set: force EPWMxB output high.
		11	Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.

**Table 3-37. Action-Qualifier Output B Control Register (AQCTLB) Field Descriptions (continued)**

Bits	Name	Value	Description
5-4	CAU	00 Do nothing (action disabled) 01 Clear: force EPWMxB output low. 10 Set: force EPWMxB output high. 11 Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.	Action when the counter equals the active CMPA register and the counter is incrementing.
3-2	PRD	00 Do nothing (action disabled) 01 Clear: force EPWMxB output low. 10 Set: force EPWMxB output high. 11 Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.	Action when the counter equals the period. Note: By definition, in count up-down mode when the counter equals period the direction is defined as 0 or counting down.
1-0	ZRO	00 Do nothing (action disabled) 01 Clear: force EPWMxB output low. 10 Set: force EPWMxB output high. 11 Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.	Action when counter equals zero. Note: By definition, in count up-down mode when the counter equals 0 the direction is defined as 1 or counting up.

### 3.4.3.3 Action-Qualifier Software Force Register (AQSFRC)

**Figure 3-89. Action-Qualifier Software Force Register (AQSFRC)**

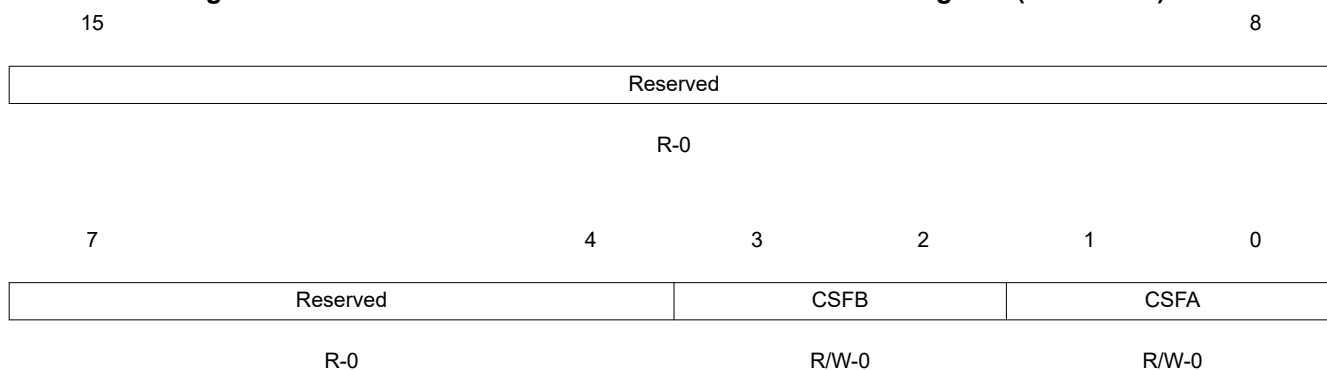


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-38. Action-Qualifier Software Force Register (AQSFRC) Field Descriptions**

Bit	Field	Value	Description
15-8	Reserved		
7-6	RLDCSF	00 01 10 11	AQCSFRC Active Register Reload From Shadow Options Load on event counter equals zero Load on event counter equals period Load on event counter equals zero or counter equals period Load immediately (the active register is directly accessed by the CPU and is not loaded from the shadow register).
5	OTSF	0 1	One-Time Software Forced Event on Output B Writing a 0 (zero) has no effect. Always reads back a 0 This bit is auto cleared once a write to this register is complete (that is, a forced event is initiated). This is a one-shot forced event. It can be overridden by another subsequent event on output B. Initiates a single s/w forced event
4-3	ACTSFB	00 01 10 11	Action when One-Time Software Force B Is invoked Does nothing (action disabled) Clear (low) Set (high) Toggle (Low -> High, High -> Low) <b>Note:</b> This action is not qualified by counter direction (CNT_dir)
2	OTSFA	0 1	One-Time Software Forced Event on Output A Writing a 0 (zero) has no effect. Always reads back a 0. This bit is auto cleared once a write to this register is complete (that is, a forced event is initiated). Initiates a single software forced event
1-0	ACTSFA	00 01 10 11	Action When One-Time Software Force A Is Invoked Does nothing (action disabled) Clear (low) Set (high) Toggle (Low → High, High → Low) <b>Note:</b> This action is not qualified by counter direction (CNT_dir)

### 3.4.3.4 Action-Qualifier Continuous Software Force Register (AQCSFRC)

**Figure 3-90. Action-Qualifier Continuous Software Force Register (AQCSFRC)**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-39. Action-Qualifier Continuous Software Force Register (AQCSFRC) Field Descriptions**

Bits	Name	Value	Description
15-4	Reserved		Reserved
3-2	CSFB	00 01 10 11	Continuous Software Force on Output B In immediate mode, a continuous force takes effect on the next TBCLK edge. In shadow mode, a continuous force takes effect on the next TBCLK edge after a shadow load into the active register. To configure shadow mode, use AQSFRC[RLDCSF]. Software forcing is disabled and has no effect Forces a continuous low on output B Forces a continuous high on output B Software forcing is disabled and has no effect
1-0	CSFA	00 01 10 11	Continuous Software Force on Output A In immediate mode, a continuous force takes effect on the next TBCLK edge. In shadow mode, a continuous force takes effect on the next TBCLK edge after a shadow load into the active register. Software forcing is disabled and has no effect Forces a continuous low on output A Forces a continuous high on output A Software forcing is disabled and has no effect



### 3.4.4 Dead-Band Submodule Registers

This section describes the dead-band submodule registers.

#### 3.4.4.1 Dead-Band Generator Control (DBCTL) Register

**Figure 3-91. Dead-Band Generator Control (DBCTL) Register**

15	14					8	
HALFCYCLE		Reserved					
R/W-0		R-0					
7	6	5	4	3	2	1	0
Reserved		IN_MODE		POLSEL		OUT_MODE	
R-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-40. Dead-Band Generator Control (DBCTL) Register Field Descriptions**

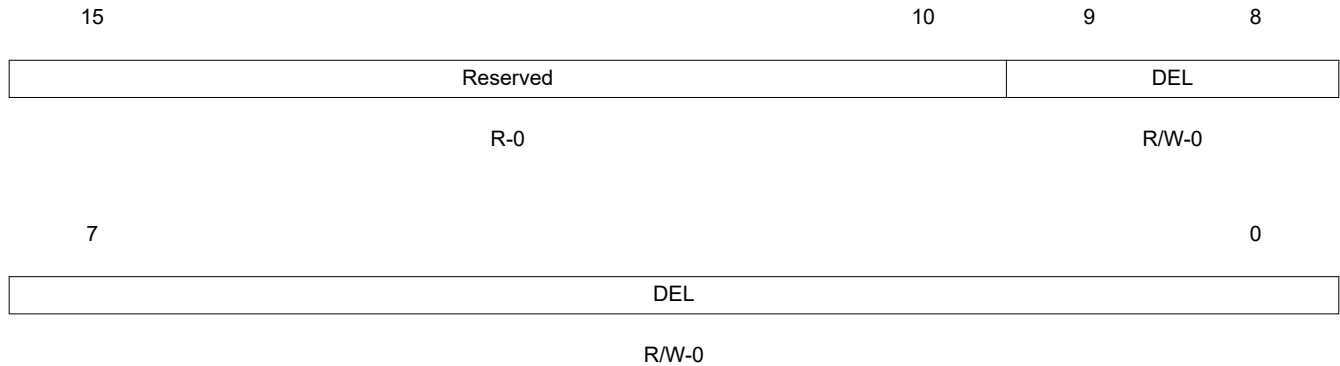
Bits	Name	Value	Description
15	HALFCYCLE	0 1	Half Cycle Clocking Enable Bit: Full cycle clocking enabled. The dead-band counters are clocked at the TBCLK rate. Half cycle clocking enabled. The dead-band counters are clocked at TBCLK*2.
14-6	Reserved		Reserved
5-4	IN_MODE	00 01 10 11	Dead Band Input Mode Control Bit 5 controls the S5 switch and bit 4 controls the S4 switch shown in <a href="#">Figure 3-29</a> . This allows you to select the input source to the falling-edge and rising-edge delay. To produce classical dead-band waveforms the default is EPWMxA In is the source for both falling and rising-edge delays. EPWMxA In (from the action-qualifier) is the source for both falling-edge and rising-edge delay. EPWMxB In (from the action-qualifier) is the source for rising-edge delayed signal. EPWMxA In (from the action-qualifier) is the source for falling-edge delayed signal. EPWMxA In (from the action-qualifier) is the source for rising-edge delayed signal. EPWMxB In (from the action-qualifier) is the source for falling-edge delayed signal. EPWMxB In (from the action-qualifier) is the source for both rising-edge delay and falling-edge delayed signal.
3-2	POLSEL	00 01 10 11	Polarity Select Control Bit 3 controls the S3 switch and bit 2 controls the S2 switch shown in <a href="#">Figure 3-29</a> . This allows you to selectively invert one of the delayed signals before it is sent out of the dead-band submodule. The following descriptions correspond to classical upper/lower switch control as found in one leg of a digital motor control inverter. These assume that DBCTL[OUT_MODE] = 1,1 and DBCTL[IN_MODE] = 0,0. Other enhanced modes are also possible, but not regarded as typical usage modes. Active high (AH) mode. Neither EPWMxA nor EPWMxB is inverted (default). Active low complementary (ALC) mode. EPWMxA is inverted. Active high complementary (AHC). EPWMxB is inverted. Active low (AL) mode. Both EPWMxA and EPWMxB are inverted.

**Table 3-40. Dead-Band Generator Control (DBCTL) Register Field Descriptions (continued)**

Bits	Name	Value	Description
1-0	OUT_MODE		<p>Dead-band Output Mode Control</p> <p>Bit 1 controls the S1 switch and bit 0 controls the S0 switch shown in <a href="#">Figure 3-29</a>. This allows you to selectively enable or bypass the dead-band generation for the falling-edge and rising-edge delay.</p>
		00	<p>Dead-band generation is bypassed for both output signals. In this mode, both the EPWMxA and EPWMxB output signals from the action-qualifier are passed directly to the PWM-chopper submodule.</p> <p>In this mode, the POLSEL and IN_MODE bits have no effect.</p>
		01	<p>Disable rising-edge delay. The EPWMxA signal from the action-qualifier is passed straight through to the EPWMxA input of the PWM-chopper submodule.</p> <p>The falling-edge delayed signal is seen on output EPWMxB. The input signal for the delay is determined by DBCTL[IN_MODE].</p>
		10	<p>The rising-edge delayed signal is seen on output EPWMxA. The input signal for the delay is determined by DBCTL[IN_MODE].</p> <p>Disable falling-edge delay. The EPWMxB signal from the action-qualifier is passed straight through to the EPWMxB input of the PWM-chopper submodule.</p>
		11	<p>Dead-band is fully enabled for both rising-edge delay on output EPWMxA and falling-edge delay on output EPWMxB. The input signal for the delay is determined by DBCTL[IN_MODE].</p>

### 3.4.4.2 Dead-Band Generator Rising Edge Delay (DBRED) Register

**Figure 3-92. Dead-Band Generator Rising Edge Delay (DBRED) Register**



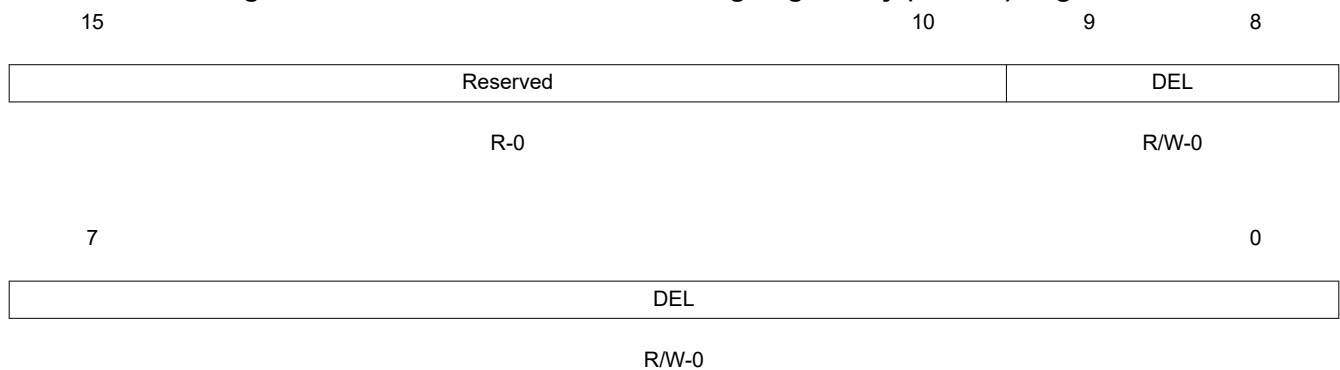
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-41. Dead-Band Generator Rising Edge Delay (DBRED) Register Field Descriptions**

Bits	Name	Value	Description
15-10	Reserved		Reserved
9-0	DEL		Rising Edge Delay Count. 10-bit counter.

### 3.4.4.3 Dead-Band Generator Falling Edge Delay (DBFED) Register

**Figure 3-93. Dead-Band Generator Falling Edge Delay (DBFED) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-42. Dead-Band Generator Falling Edge Delay (DBFED) Register Field Descriptions**

Bits	Name	Value	Description
15-10	Reserved		Reserved
9-0	DEL		Falling Edge Delay Count. 10-bit counter

### 3.4.5 Trip-Zone Submodule Control and Status Registers

This section describes the trip-zone submodule control and status registers.

#### 3.4.5.1 Trip-Zone Select Register (TZSEL)

**Figure 3-94. Trip-Zone Select Register (TZSEL)**

15	14	13	12	11	10	9	8
DCBEVT1	DCAEVT1	OSHT6	OSHT5	OSHT4	OSHT3	OSHT2	OSHT1
R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
DCBEVT2	DCAEVT2	CBC6	CBC5	CBC4	CBC3	CBC2	CBC1
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-43. Trip-Zone Submodule Select Register (TZSEL) Field Descriptions**

Bits	Name	Value	Description
<b>One-Shot (OSHT) Trip-zone enable/disable. When any of the enabled pins go low, a one-shot trip event occurs for this ePWM module. When the event occurs, the action defined in the TZCTL register (Section 3.4.5.3) is taken on the EPWMxA and EPWMxB outputs. The one-shot trip condition remains latched until the user clears the condition via the TZCLR register (Section 3.4.5.6).</b>			
15	DCBEVT1	0 1	Digital Compare Output B Event 1 Select Disable DCBEVT1 as one-shot-trip source for this ePWM module. Enable DCBEVT1 as one-shot-trip source for this ePWM module.
14	DCAEVT1	0 1	Digital Compare Output A Event 1 Select Disable DCAEVT1 as one-shot-trip source for this ePWM module. Enable DCAEVT1 as one-shot-trip source for this ePWM module.
13	OSHT6	0 1	Trip-zone 6 ( $\overline{TZ6}$ ) Select Disable $\overline{TZ6}$ as a one-shot trip source for this ePWM module. Enable $\overline{TZ6}$ as a one-shot trip source for this ePWM module.
12	OSHT5	0 1	Trip-zone 5 ( $\overline{TZ5}$ ) Select Disable $\overline{TZ5}$ as a one-shot trip source for this ePWM module Enable $\overline{TZ5}$ as a one-shot trip source for this ePWM module
11	OSHT4	0 1	Trip-zone 4 ( $\overline{TZ4}$ ) Select Disable $\overline{TZ4}$ as a one-shot trip source for this ePWM module Enable $\overline{TZ4}$ as a one-shot trip source for this ePWM module
10	OSHT3	0 1	Trip-zone 3 ( $\overline{TZ3}$ ) Select Disable $\overline{TZ3}$ as a one-shot trip source for this ePWM module Enable $\overline{TZ3}$ as a one-shot trip source for this ePWM module
9	OSHT2	0 1	Trip-zone 2 ( $\overline{TZ2}$ ) Select Disable $\overline{TZ2}$ as a one-shot trip source for this ePWM module Enable $\overline{TZ2}$ as a one-shot trip source for this ePWM module
8	OSHT1	0 1	Trip-zone 1 ( $\overline{TZ1}$ ) Select Disable $\overline{TZ1}$ as a one-shot trip source for this ePWM module Enable $\overline{TZ1}$ as a one-shot trip source for this ePWM module

**Table 3-43. Trip-Zone Submodule Select Register (TZSEL) Field Descriptions (continued)**

Bits	Name	Value	Description
<b>Cycle-by-Cycle (CBC) Trip-zone enable/disable. When any of the enabled pins go low, a cycle-by-cycle trip event occurs for this ePWM module. When the event occurs, the action defined in the TZCTL register (Section 3.4.5.3) is taken on the EPWMxA and EPWMxB outputs. A cycle-by-cycle trip condition is automatically cleared when the time-base counter reaches zero.</b>			
7	DCBEVT2	0	Digital Compare Output B Event 2 Select Disable DCBEVT2 as a CBC trip source for this ePWM module
		1	Enable DCBEVT2 as a CBC trip source for this ePWM module
6	DCAEVT2	0	Digital Compare Output A Event 2 Select Disable DCAEVT2 as a CBC trip source for this ePWM module
		1	Enable DCAEVT2 as a CBC trip source for this ePWM module
5	CBC6	0	Trip-zone 6 ( $\overline{TZ6}$ ) Select Disable $\overline{TZ6}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ6}$ as a CBC trip source for this ePWM module
4	CBC5	0	Trip-zone 5 ( $\overline{TZ5}$ ) Select Disable $\overline{TZ5}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ5}$ as a CBC trip source for this ePWM module
3	CBC4	0	Trip-zone 4 ( $\overline{TZ4}$ ) Select Disable $\overline{TZ4}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ4}$ as a CBC trip source for this ePWM module
2	CBC3	0	Trip-zone 3 ( $\overline{TZ3}$ ) Select Disable $\overline{TZ3}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ3}$ as a CBC trip source for this ePWM module
1	CBC2	0	Trip-zone 2 ( $\overline{TZ2}$ ) Select Disable $\overline{TZ2}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ2}$ as a CBC trip source for this ePWM module
0	CBC1	0	Trip-zone 1 ( $\overline{TZ1}$ ) Select Disable $\overline{TZ1}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ1}$ as a CBC trip source for this ePWM module

### 3.4.5.2 Trip Zone Digital Compare Event Select Register (TZDCSEL)

**Figure 3-95. Trip Zone Digital Compare Event Select Register (TZDCSEL)**

15	12	11	9	8	6	5	3	2	0
Reserved		DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1				
R-0		R/W-0	R/W-0	R/W-0	R/W-0		R/W-0		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-44. Trip Zone Digital Compare Event Select Register (TZDCSEL) Field Descriptions**

Bit	Field	Value	Description
15-12	Reserved		Reserved

**Table 3-44. Trip Zone Digital Compare Event Select Register (TZDCSEL) Field Descriptions (continued)**

Bit	Field	Value	Description
11-9	DCBEVT2		Digital Compare Output B Event 2 Selection
		000	Event disabled
		001	DCBH = low, DCBL = don't care
		010	DCBH = high, DCBL = don't care
		011	DCBL = low, DCBH = don't care
		100	DCBL = high, DCBH = don't care
		101	DCBL = high, DCBH = low
		110	reserved
8-6	DCBEVT1		Digital Compare Output B Event 1 Selection
		000	Event disabled
		001	DCBH = low, DCBL = don't care
		010	DCBH = high, DCBL = don't care
		011	DCBL = low, DCBH = don't care
		100	DCBL = high, DCBH = don't care
		101	DCBL = high, DCBH = low
		110	reserved
5-3	DCAEVT2		Digital Compare Output A Event 2 Selection
		000	Event disabled
		001	DCAH = low, DCAL = don't care
		010	DCAH = high, DCAL = don't care
		011	DCAL = low, DCAH = don't care
		100	DCAL = high, DCAH = don't care
		101	DCAL = high, DCAH = low
		110	reserved
2-0	DCAEVT1		Digital Compare Output A Event 1 Selection
		000	Event disabled
		001	DCAH = low, DCAL = don't care
		010	DCAH = high, DCAL = don't care
		011	DCAL = low, DCAH = don't care
		100	DCAL = high, DCAH = don't care
		101	DCAL = high, DCAH = low
		110	reserved
111	reserved		

### 3.4.5.3 Trip-Zone Control Register (TZCTL)

**Figure 3-96. Trip-Zone Control Register (TZCTL)**

15	12	11	10	9	8
Reserved			DCBEVT2	DCBEVT1	
R-0			R/W-0	R/W-0	
7	6	5	4	3	2
DCAEVT2		DCAEVT1		TZB	TZA
R/W-0		R/W-0		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

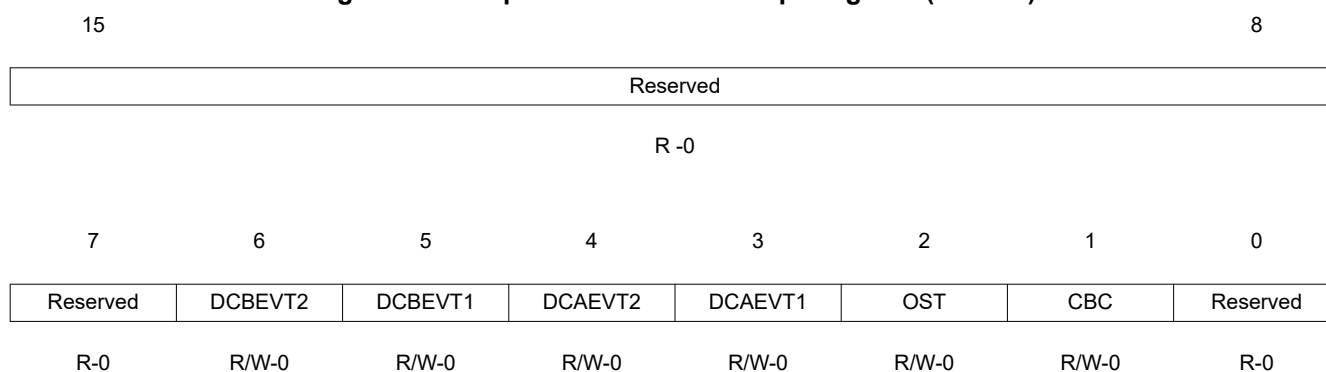
**Table 3-45. Trip-Zone Control Register Field Descriptions**

Bit	Field	Value	Description
15-12	Reserved		Reserved
11-10	DCBEVT2	00 01 10 11	Digital Compare Output B Event 2 Action On EPWMxB: High-impedance (EPWMxB = High-impedance state) Force EPWMxB to a high state. Force EPWMxB to a low state. Do Nothing, trip action is disabled
9-8	DCBEVT1	00 01 10 11	Digital Compare Output B Event 1 Action On EPWMxB: High-impedance (EPWMxB = High-impedance state) Force EPWMxB to a high state. Force EPWMxB to a low state. Do Nothing, trip action is disabled
7-6	DCAEVT2	00 01 10 11	Digital Compare Output A Event 2 Action On EPWMxA: High-impedance (EPWMxA = High-impedance state) Force EPWMxA to a high state. Force EPWMxA to a low state. Do Nothing, trip action is disabled
5-4	DCAEVT1	00 01 10 11	Digital Compare Output A Event 1 Action On EPWMxA: High-impedance (EPWMxA = High-impedance state) Force EPWMxA to a high state. Force EPWMxA to a low state. Do Nothing, trip action is disabled
3-2	TZB	00 01 10 11	When a trip event occurs the following action is taken on output EPWMxB. Which trip-zone pins can cause an event is defined in the TZSEL register. High-impedance (EPWMxB = High-impedance state) Force EPWMxB to a high state Force EPWMxB to a low state Do nothing, no action is taken on EPWMxB.

**Table 3-45. Trip-Zone Control Register Field Descriptions (continued)**

Bit	Field	Value	Description
1-0	TZA		When a trip event occurs the following action is taken on output EPWMxA. Which trip-zone pins can cause an event is defined in the TZSEL register.
		00	High-impedance (EPWMxA = High-impedance state)
		01	Force EPWMxA to a high state
		10	Force EPWMxA to a low state
		11	Do nothing, no action is taken on EPWMxA.

### 3.4.5.4 Trip-Zone Enable Interrupt Register (TZEINT)

**Figure 3-97. Trip-Zone Enable Interrupt Register (TZEINT)**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

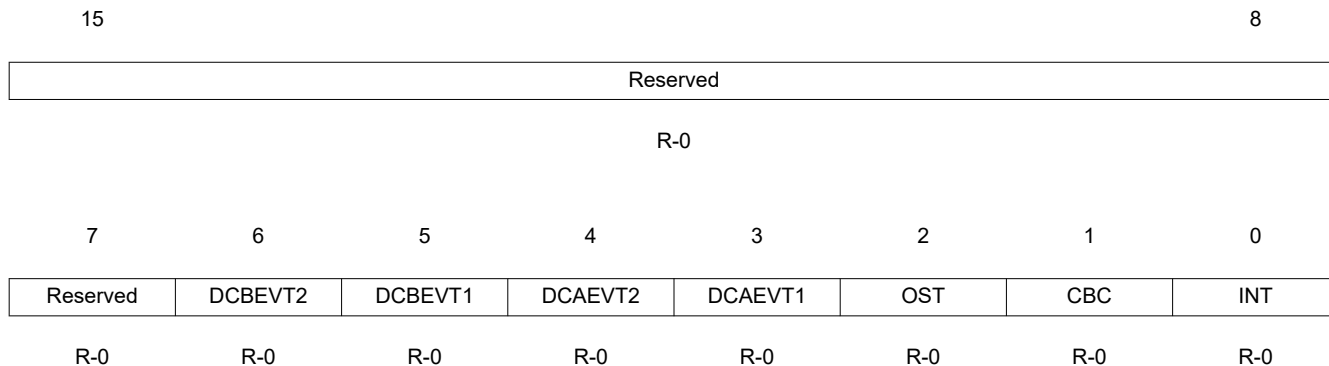
**Table 3-46. Trip-Zone Enable Interrupt Register (TZEINT) Field Descriptions**

Bits	Name	Value	Description
15-3	Reserved		Reserved
6	DCBEVT2	0 1	Digital Comparator Output B Event 2 Interrupt Enable Disabled Enabled
5	DCBEVT1	0 1	Digital Comparator Output B Event 1 Interrupt Enable Disabled Enabled
4	DCAEVT2	0 1	Digital Comparator Output A Event 2 Interrupt Enable Disabled Enabled
3	DCAEVT1	0 1	Digital Comparator Output A Event 1 Interrupt Enable Disabled Enabled
2	OST	0 1	Trip-zone One-Shot Interrupt Enable Disable one-shot interrupt generation Enable Interrupt generation; a one-shot trip event will cause a EPWMx_TZINT PIE interrupt.
1	CBC	0 1	Trip-zone Cycle-by-Cycle Interrupt Enable Disable cycle-by-cycle interrupt generation. Enable interrupt generation; a cycle-by-cycle trip event will cause an EPWMx_TZINT PIE interrupt.
0	Reserved		Reserved



### 3.4.5.5 Trip-Zone Flag Register (TZFLG)

**Figure 3-98. Trip-Zone Flag Register (TZFLG)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-47. Trip-Zone Flag Register Field Descriptions**

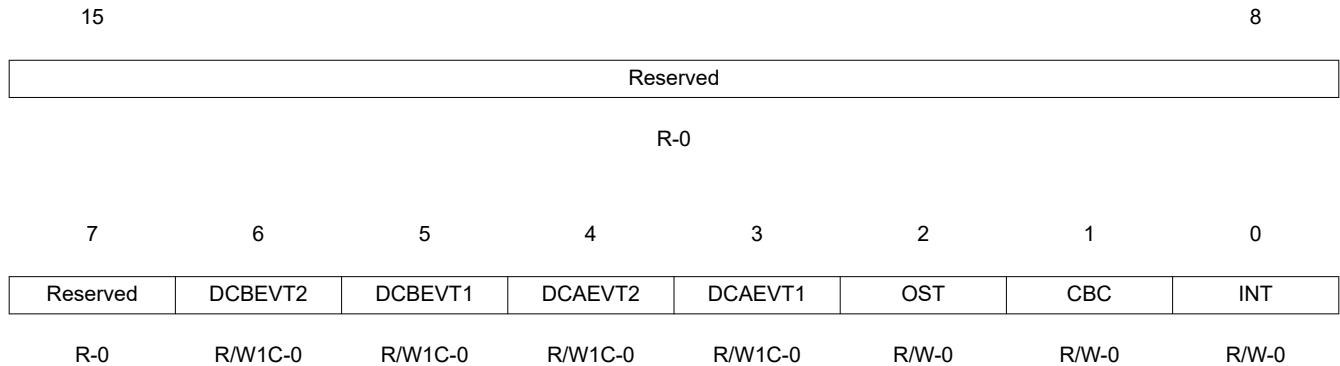
Bit	Field	Value	Description
15-7	Reserved		Reserved
6	DCBEVT2	0	Latched Status Flag for Digital Compare Output B Event 2 Indicates no trip event has occurred on DCBEVT2
		1	Indicates a trip event has occurred for the event defined for DCBEVT2
5	DCBEVT1	0	Latched Status Flag for Digital Compare Output B Event 1 Indicates no trip event has occurred on DCBEVT1
		1	Indicates a trip event has occurred for the event defined for DCBEVT1
4	DCAEVT2	0	Latched Status Flag for Digital Compare Output A Event 2 Indicates no trip event has occurred on DCAEVT2
		1	Indicates a trip event has occurred for the event defined for DCAEVT2
3	DCAEVT1	0	Latched Status Flag for Digital Compare Output A Event 1 Indicates no trip event has occurred on DCAEVT1
		1	Indicates a trip event has occurred for the event defined for DCAEVT1
2	OST	0	Latched Status Flag for A One-Shot Trip Event No one-shot trip event has occurred.
		1	Indicates a trip event has occurred on a pin selected as a one-shot trip source. This bit is cleared by writing the appropriate value to the TZCLR register .
1	CBC	0	Latched Status Flag for Cycle-By-Cycle Trip Event No cycle-by-cycle trip event has occurred.
		1	Indicates a trip event has occurred on a signal selected as a cycle-by-cycle trip source. The TZFLG[CBC] bit will remain set until it is manually cleared by the user. If the cycle-by-cycle trip event is still present when the CBC bit is cleared, then CBC will be immediately set again. The specified condition on the signal is automatically cleared when the ePWM time-base counter reaches zero (TBCTR = 0x0000) if the trip condition is no longer present. The condition on the signal is only cleared when the TBCTR = 0x0000 no matter where in the cycle the CBC flag is cleared. This bit is cleared by writing the appropriate value to the TZCLR register .

**Table 3-47. Trip-Zone Flag Register Field Descriptions (continued)**

Bit	Field	Value	Description
0	INT	0	Latched Trip Interrupt Status Flag Indicates no interrupt has been generated.
		1	Indicates an EPWMx_TZINT PIE interrupt was generated because of a trip condition. No further EPWMx_TZINT PIE interrupts will be generated until this flag is cleared. If the interrupt flag is cleared when either CBC or OST is set, then another interrupt pulse will be generated. Clearing all flag bits will prevent further interrupts. This bit is cleared by writing the appropriate value to the TZCLR register .

### 3.4.5.6 Trip-Zone Clear Register (TZCLR)

**Figure 3-99. Trip-Zone Clear Register (TZCLR)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-48. Trip-Zone Clear Register (TZCLR) Field Descriptions**

Bit	Field	Value	Description
15-7	Reserved		Reserved
6	DCBEVT2	0 1	Clear Flag for Digital Compare Output B Event 2 Writing 0 has no effect. This bit always reads back 0. Writing 1 clears the DCBEVT2 event trip condition.
5	DCBEVT1	0 1	Clear Flag for Digital Compare Output B Event 1 Writing 0 has no effect. This bit always reads back 0. Writing 1 clears the DCBEVT1 event trip condition.
4	DCAEVT2	0 1	Clear Flag for Digital Compare Output A Event 2 Writing 0 has no effect. This bit always reads back 0. Writing 1 clears the DCAEVT2 event trip condition.
3	DCAEVT1	0 1	Clear Flag for Digital Compare Output A Event 1 Writing 0 has no effect. This bit always reads back 0. Writing 1 clears the DCAEVT1 event trip condition.
2	OST	0 1	Clear Flag for One-Shot Trip (OST) Latch Has no effect. Always reads back a 0. Clears this Trip (set) condition.
1	CBC	0 1	Clear Flag for Cycle-By-Cycle (CBC) Trip Latch Has no effect. Always reads back a 0. Clears this Trip (set) condition.
0	INT	0 1	Global Interrupt Clear Flag Has no effect. Always reads back a 0. Clears the trip-interrupt flag for this ePWM module (TZFLG[INT]). <b>NOTE:</b> No further EPWMx_TZINT PIE interrupts will be generated until the flag is cleared. If the TZFLG[INT] bit is cleared and any of the other flag bits are set, then another interrupt pulse will be generated. Clearing all flag bits will prevent further interrupts.



### 3.4.6 Event-Trigger Submodule Registers

This section describes the event-trigger submodule registers.

#### 3.4.6.1 Event-Trigger Selection Register (ETSEL)

**Figure 3-101. Event-Trigger Selection Register (ETSEL)**

15	14	12	11	10	8
SOCBEN	SOCBSEL	SOCAEN	SOCASEL		
R/W-0	R/W-0	R/W-0	R/W-0		
7	4	3	2	0	
Reserved		INTEN	INTSEL		
R-0		R/W-0	R/W-0		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-50. Event-Trigger Selection Register (ETSEL) Field Descriptions**

Bits	Name	Value	Description
15	SOCBEN	0 1	Enable the ADC Start of Conversion B (EPWMxSOCB) Pulse Disable EPWMxSOCB. Enable EPWMxSOCB pulse.
14-12	SOCBSEL	000 001 010 011 100 101 110 111	EPWMxSOCB Selection Options These bits determine when a EPWMxSOCB pulse will be generated. Enable DCBEVT1.soc event Enable event time-base counter equal to zero. (TBCTR = 0x0000) Enable event time-base counter equal to period (TBCTR = TBPRD) Enable event time-base counter equal to zero or period (TBCTR = 0x0000 or TBCTR = TBPRD). This mode is useful in up-down count mode. Enable event time-base counter equal to CMPA when the timer is incrementing. Enable event time-base counter equal to CMPA when the timer is decrementing. Enable event: time-base counter equal to CMPB when the timer is incrementing. Enable event: time-base counter equal to CMPB when the timer is decrementing.
11	SOCAEN	0 1	Enable the ADC Start of Conversion A (EPWMxSOCA) Pulse Disable EPWMxSOCA. Enable EPWMxSOCA pulse.
10-8	SOCASEL	000 001 010 011 100 101 110 111	EPWMxSOCA Selection Options These bits determine when a EPWMxSOCA pulse will be generated. Enable DCAEVT1.soc event Enable event time-base counter equal to zero. (TBCTR = 0x0000) Enable event time-base counter equal to period (TBCTR = TBPRD) Enable event time-base counter equal to zero or period (TBCTR = 0x0000 or TBCTR = TBPRD). This mode is useful in up-down count mode. Enable event time-base counter equal to CMPA when the timer is incrementing. Enable event time-base counter equal to CMPA when the timer is decrementing. Enable event: time-base counter equal to CMPB when the timer is incrementing. Enable event: time-base counter equal to CMPB when the timer is decrementing.
7-4	Reserved		Reserved

**Table 3-50. Event-Trigger Selection Register (ETSEL) Field Descriptions (continued)**

Bits	Name	Value	Description
3	INTEN	0	Enable ePWM Interrupt (EPWMx_INT) Generation Disable EPWMx_INT generation
		1	Enable EPWMx_INT generation
2-0	INTSEL	000	ePWM Interrupt (EPWMx_INT) Selection Options Reserved
		001	Enable event time-base counter equal to zero. (TBCTR = 0x0000)
		010	Enable event time-base counter equal to period (TBCTR = TBPRD)
		011	Enable event time-base counter equal to zero or period (TBCTR = 0x0000 or TBCTR = TBPRD). This mode is useful in up-down count mode.
		100	Enable event time-base counter equal to CMPA when the timer is incrementing.
		101	Enable event time-base counter equal to CMPA when the timer is decrementing.
		110	Enable event: time-base counter equal to CMPB when the timer is incrementing.
		111	Enable event: time-base counter equal to CMPB when the timer is decrementing.

### 3.4.6.2 Event-Trigger Prescale Register (ETPS)

**Figure 3-102. Event-Trigger Prescale Register (ETPS)**

15	14	13	12	11	10	9	8
SOCBCNT		SOCBPRD		SOCACNT		SOCAPRD	
R-0		R/W-0		R-0		R/W-0	
7		4	3	2	1	0	
Reserved				INTCNT		INTPRD	
R-0				R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

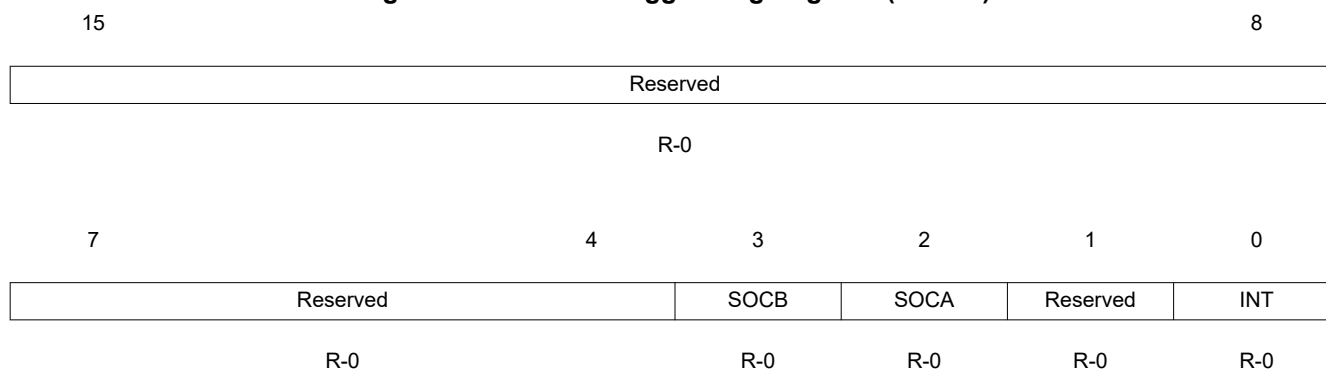
**Table 3-51. Event-Trigger Prescale Register (ETPS) Field Descriptions**

Bits	Name	Description
15-14	SOCBCNT	ePWM ADC Start-of-Conversion B Event (EPWMxSOCB) Counter Register These bits indicate how many selected ETSEL[SOCBSEL] events have occurred:
		00 No events have occurred.
		01 1 event has occurred.
		10 2 events have occurred.
		11 3 events have occurred.
13-12	SOCBPRD	ePWM ADC Start-of-Conversion B Event (EPWMxSOCB) Period Select These bits determine how many selected ETSEL[SOCBSEL] events need to occur before an EPWMxSOCB pulse is generated. To be generated, the pulse must be enabled (ETSEL[SOCBEN] = 1). The SOCB pulse will be generated even if the status flag is set from a previous start of conversion (ETFLG[SOCB] = 1). Once the SOCB pulse is generated, the ETPS[SOCBCNT] bits will automatically be cleared.
		00 Disable the SOCB event counter. No EPWMxSOCB pulse will be generated
		01 Generate the EPWMxSOCB pulse on the first event: ETPS[SOCBCNT] = 0,1
		10 Generate the EPWMxSOCB pulse on the second event: ETPS[SOCBCNT] = 1,0
		11 Generate the EPWMxSOCB pulse on the third event: ETPS[SOCBCNT] = 1,1

**Table 3-51. Event-Trigger Prescale Register (ETPS) Field Descriptions (continued)**

Bits	Name		Description
11-10	SOCACNT		ePWM ADC Start-of-Conversion A Event (EPWMxSOCA) Counter Register These bits indicate how many selected ETSEL[SOCASEL] events have occurred: 00 No events have occurred. 01 1 event has occurred. 10 2 events have occurred. 11 3 events have occurred.
9-8	SOCAPRD		ePWM ADC Start-of-Conversion A Event (EPWMxSOCA) Period Select These bits determine how many selected ETSEL[SOCASEL] events need to occur before an EPWMxSOCA pulse is generated. To be generated, the pulse must be enabled (ETSEL[SOCASEN] = 1). The SOCA pulse will be generated even if the status flag is set from a previous start of conversion (ETFLG[SOCA] = 1). Once the SOCA pulse is generated, the ETPS[SOCACNT] bits will automatically be cleared. 00 Disable the SOCA event counter. No EPWMxSOCA pulse will be generated 01 Generate the EPWMxSOCA pulse on the first event: ETPS[SOCACNT] = 0,1 10 Generate the EPWMxSOCA pulse on the second event: ETPS[SOCACNT] = 1,0 11 Generate the EPWMxSOCA pulse on the third event: ETPS[SOCACNT] = 1,1
7-4	Reserved		Reserved
3-2	INTCNT		ePWM Interrupt Event (EPWMx_INT) Counter Register These bits indicate how many selected ETSEL[INTSEL] events have occurred. These bits are automatically cleared when an interrupt pulse is generated. If interrupts are disabled, ETSEL[INT] = 0 or the interrupt flag is set, ETFLG[INT] = 1, the counter will stop counting events when it reaches the period value ETPS[INTCNT] = ETPS[INTPRD]. 00 No events have occurred. 01 1 event has occurred. 10 2 events have occurred. 11 3 events have occurred.
1-0	INTPRD		ePWM Interrupt (EPWMx_INT) Period Select These bits determine how many selected ETSEL[INTSEL] events need to occur before an interrupt is generated. To be generated, the interrupt must be enabled (ETSEL[INT] = 1). If the interrupt status flag is set from a previous interrupt (ETFLG[INT] = 1) then no interrupt will be generated until the flag is cleared via the ETCLR[INT] bit. This allows for one interrupt to be pending while another is still being serviced. Once the interrupt is generated, the ETPS[INTCNT] bits will automatically be cleared. Writing a INTPRD value that is the same as the current counter value will trigger an interrupt if it is enabled and the status flag is clear. Writing a INTPRD value that is less than the current counter value will result in an undefined state. If a counter event occurs at the same instant as a new zero or non-zero INTPRD value is written, the counter is incremented. 00 Disable the interrupt event counter. No interrupt will be generated and ETFRC[INT] is ignored. 01 Generate an interrupt on the first event INTCNT = 01 (first event) 10 Generate interrupt on ETPS[INTCNT] = 1,0 (second event) 11 Generate interrupt on ETPS[INTCNT] = 1,1 (third event)

### 3.4.6.3 Event-Trigger Flag Register (ETFLG)

**Figure 3-103. Event-Trigger Flag Register (ETFLG)**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-52. Event-Trigger Flag Register (ETFLG) Field Descriptions**

Bits	Name	Value	Description
15-4	Reserved		Reserved
3	SOCB	0 1	Latched ePWM ADC Start-of-Conversion B (EPWMxSOCB) Status Flag Indicates no EPWMxSOCB event occurred Indicates that a start of conversion pulse was generated on EPWMxSOCB. The EPWMxSOCB output will continue to be generated even if the flag bit is set.
2	SOCA	0 1	Latched ePWM ADC Start-of-Conversion A (EPWMxSOCA) Status Flag Unlike the ETFLG[INT] flag, the EPWMxSOCA output will continue to pulse even if the flag bit is set. Indicates no event occurred Indicates that a start of conversion pulse was generated on EPWMxSOCA. The EPWMxSOCA output will continue to be generated even if the flag bit is set.
1	Reserved		Reserved
0	INT	0 1	Latched ePWM Interrupt (EPWMx_INT) Status Flag Indicates no event occurred Indicates that an ePWMx interrupt (EWPMx_INT) was generated. No further interrupts will be generated until the flag bit is cleared. Up to one interrupt can be pending while the ETFLG[INT] bit is still set. If an interrupt is pending, it will not be generated until after the ETFLG[INT] bit is cleared. See <a href="#">Figure 3-42</a> .



### 3.4.6.4 Event-Trigger Clear Register (ETCLR)

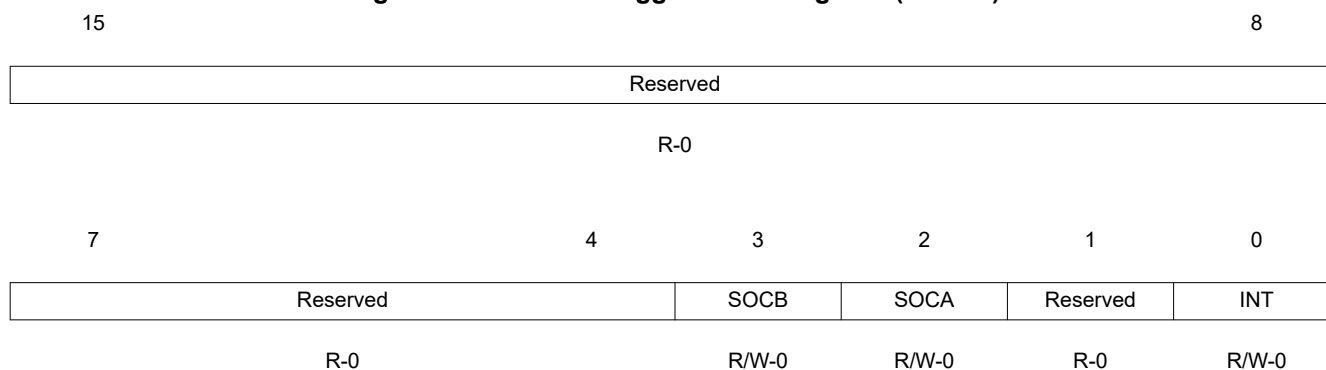
**Figure 3-104. Event-Trigger Clear Register (ETCLR)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-53. Event-Trigger Clear Register (ETCLR) Field Descriptions**

Bits	Name	Value	Description
15-4	Reserved		Reserved
3	SOCB	0 1	ePWM ADC Start-of-Conversion B (EPWMxSOCB) Flag Clear Bit Writing a 0 has no effect. Always reads back a 0 Clears the ETFLG[SOCB] flag bit
2	SOCA	0 1	ePWM ADC Start-of-Conversion A (EPWMxSOCA) Flag Clear Bit Writing a 0 has no effect. Always reads back a 0 Clears the ETFLG[SOCA] flag bit
1	Reserved		Reserved
0	INT	0 1	ePWM Interrupt (EPWMx_INT) Flag Clear Bit Writing a 0 has no effect. Always reads back a 0 Clears the ETFLG[INT] flag bit and enable further interrupts pulses to be generated

**3.4.6.5 Event-Trigger Force Register (ETFRC)**
**Figure 3-105. Event-Trigger Force Register (ETFRC)**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-54. Event-Trigger Force Register (ETFRC) Field Descriptions**

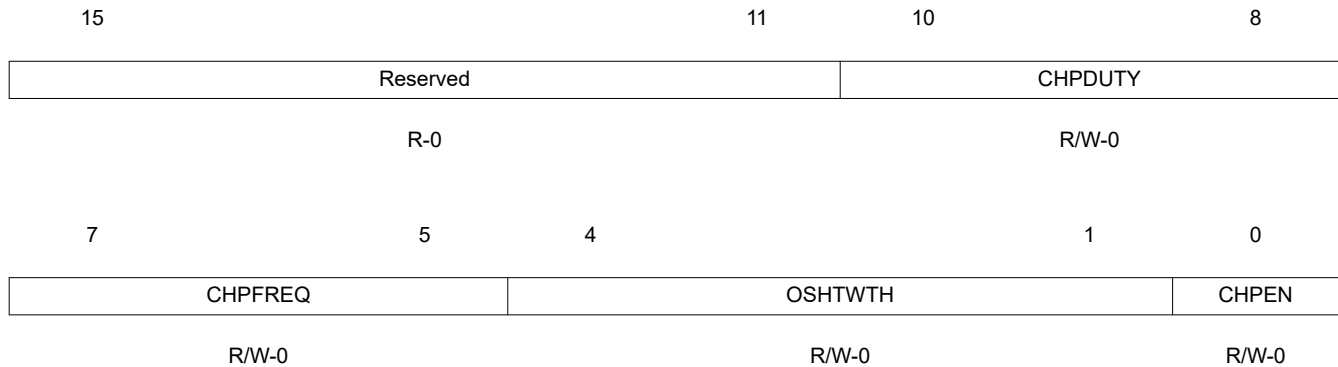
Bits	Name	Value	Description
15-4	Reserved		Reserved
3	SOCB	0 1	SOCB Force Bit. The SOCB pulse will only be generated if the event is enabled in the ETSEL register. The ETFLG[SOCB] flag bit will be set regardless.  0 Has no effect. Always reads back a 0. 1 Generates a pulse on EPWMxSOCB and sets the SOCBFLG bit. This bit is used for test purposes.
2	SOCA	0 1	SOCA Force Bit. The SOCA pulse will only be generated if the event is enabled in the ETSEL register. The ETFLG[SOCA] flag bit will be set regardless.  0 Writing 0 to this bit will be ignored. Always reads back a 0. 1 Generates a pulse on EPWMxSOCA and set the SOCAFLG bit. This bit is used for test purposes.
1	Reserved	0	Reserved
0	INT	0 1	INT Force Bit. The interrupt will only be generated if the event is enabled in the ETSEL register. The INT flag bit will be set regardless.  0 Writing 0 to this bit will be ignored. Always reads back a 0. 1 Generates an interrupt on $\overline{\text{EPWMxINT}}$ and set the INT flag bit. This bit is used for test purposes.

### 3.4.7 PWM-Chopper Submodule Control Register

This section describes the PWM-chopper submodule control register.

#### 3.4.7.1 PWM-Chopper Control (PCCTL) Register

**Figure 3-106. PWM-Chopper Control (PCCTL) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-55. PWM-Chopper Control (PCCTL) Register Field Descriptions**

Bits	Name	Value	Description
15-11	Reserved		Reserved
10-8	CHPDUTY	000 001 010 011 100 101 110 111	Chopping Clock Duty Cycle Duty = 1/8 (12.5%) Duty = 2/8 (25.0%) Duty = 3/8 (37.5%) Duty = 4/8 (50.0%) Duty = 5/8 (62.5%) Duty = 6/8 (75.0%) Duty = 7/8 (87.5%) Reserved
7-5	CHPFREQ	000 001 010 011 100 101 110 111	Chopping Clock Frequency Divide by 1 (no prescale, = 11.25 MHz at 90 MHz SYSCLKOUT) Divide by 2 (5.63 MHz at 90 MHz SYSCLKOUT) Divide by 3 (3.75 MHz at 90 MHz SYSCLKOUT) Divide by 4 (2.81 MHz at 90 MHz SYSCLKOUT) Divide by 5 (2.25 MHz at 90 MHz SYSCLKOUT) Divide by 6 (1.88 MHz at 90 MHz SYSCLKOUT) Divide by 7 (1.61 MHz at 90 MHz SYSCLKOUT) Divide by 8 (1.41 MHz at 90 MHz SYSCLKOUT)

**Table 3-55. PWM-Chopper Control (PCCTL) Register Field Descriptions (continued)**

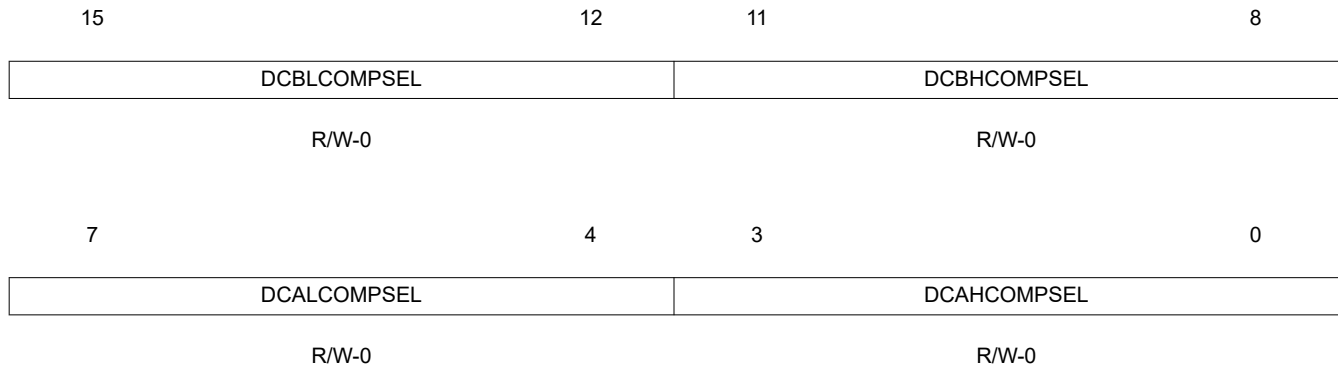
Bits	Name	Value	Description
4-1	OSHTWTH		One-Shot Pulse Width
		0000	1 x SYSCLKOUT / 8 wide ( = 133.3 nS at 60 MHz SYSCLKOUT)
		0001	2 x SYSCLKOUT / 8 wide ( = 266.7 nS at 60 MHz SYSCLKOUT)
		0010	3 x SYSCLKOUT / 8 wide ( = 400 nS at 60 MHz SYSCLKOUT)
		0011	4 x SYSCLKOUT / 8 wide ( = 533.3 nS at 60 MHz SYSCLKOUT)
		0100	5 x SYSCLKOUT / 8 wide ( = 666.7 nS at 60 MHz SYSCLKOUT)
		0101	6 x SYSCLKOUT / 8 wide ( = 800 nS at 60 MHz SYSCLKOUT)
		0110	7 x SYSCLKOUT / 8 wide ( = 933.3 nS at 60 MHz SYSCLKOUT)
		0111	8 x SYSCLKOUT / 8 wide ( = 1067 nS at 60 MHz SYSCLKOUT)
		1000	9 x SYSCLKOUT / 8 wide ( = 1200 nS at 60 MHz SYSCLKOUT)
		1001	10 x SYSCLKOUT / 8 wide ( = 1333 nS at 60 MHz SYSCLKOUT)
		1010	11 x SYSCLKOUT / 8 wide ( = 1467 nS at 60 MHz SYSCLKOUT)
		1011	12 x SYSCLKOUT / 8 wide ( = 1600 nS at 60 MHz SYSCLKOUT)
		1100	13 x SYSCLKOUT / 8 wide ( = 1733 nS at 60 MHz SYSCLKOUT)
		1101	14 x SYSCLKOUT / 8 wide ( = 1867 nS at 60 MHz SYSCLKOUT)
		1110	15 x SYSCLKOUT / 8 wide ( = 2000 nS at 60 MHz SYSCLKOUT)
1111	16 x SYSCLKOUT / 8 wide ( = 2133 nS at 60 MHz SYSCLKOUT)		
0	CHPEN		PWM-chopping Enable
		0	Disable (bypass) PWM chopping function
		1	Enable chopping function

### 3.4.8 Digital Compare Submodule Registers

This section describes the digital compare submodule registers.

#### 3.4.8.1 Digital Compare Trip Select (DCTRIPSEL) Register

**Figure 3-107. Digital Compare Trip Select (DCTRIPSEL) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-56. Digital Compare Trip Select (DCTRIPSEL) Register Field Descriptions**

Bit	Field	Value	Description
15-12	DCBLCOMPSEL	0000 0001 0010 1000 1001	Digital Compare B Low Input Select Defines the source for the DCBL input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.  $\overline{TZ1}$ input $\overline{TZ2}$ input $\overline{TZ3}$ input COMP1OUT input COMP2OUT input  Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.
11-8	DCBHCOMPSEL	0000 0001 0010 1000 1001	Digital Compare B High Input Select Defines the source for the DCBH input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.  $\overline{TZ1}$ input $\overline{TZ2}$ input $\overline{TZ3}$ input COMP1OUT input COMP2OUT input  Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.

**Table 3-56. Digital Compare Trip Select (DCTRIPSEL) Register Field Descriptions (continued)**

Bit	Field	Value	Description
7-4	DCALCOMPSEL		Digital Compare A Low Input Select Defines the source for the DCAL input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.
3-0	DCAHCOMPSEL		Digital Compare A High Input Select Defines the source for the DCAH input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.

### 3.4.8.2 Digital Compare A Control (DCACTL) Register

**Figure 3-108. Digital Compare A Control (DCACTL) Register**

15	10	9	8
Reserved		EVT2FRC SYNCSEL	EVT2SRCSEL
R-0		R/W-0	R/W-0
7	4	3	2
7	4	3	2
Reserved		EVT1SYNCE	EVT1SOCE
R-0		R/W-0	R/W-0
Reserved		EVT1FRC SYNCSEL	EVT1SRCSEL
R-0		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-57. Digital Compare A Control (DCACTL) Register Field Descriptions**

Bit	Field	Value	Description
15-10	Reserved		Reserved
9	EVT2FRC SYNCSEL	0 1	DCAEVT2 Force Synchronization Signal Select Source Is Synchronous Signal Source Is Asynchronous Signal
8	EVT2SRCSEL	0 1	DCAEVT2 Source Signal Select Source Is DCAEVT2 Signal Source Is DCEVTFILT Signal
7-4	Reserved		Reserved
3	EVT1SYNCE	0 1	DCAEVT1 SYNC, Enable/Disable SYNC Generation Disabled SYNC Generation Enabled
2	EVT1SOCE	0 1	DCAEVT1 SOC, Enable/Disable SOC Generation Disabled SOC Generation Enabled
1	EVT1FRC SYNCSEL	0 1	DCAEVT1 Force Synchronization Signal Select Source Is Synchronous Signal Source Is Asynchronous Signal
0	EVT1SRCSEL	0 1	DCAEVT1 Source Signal Select Source Is DCAEVT1 Signal Source Is DCEVTFILT Signal

### 3.4.8.3 Digital Compare B Control (DCBCTL) Register

**Figure 3-109. Digital Compare B Control (DCBCTL) Register**

15	10	9	8		
Reserved			EVT2FRC SYNCSEL	EVT2SRCSEL	
R-0			R/W-0	R/W-0	
7	4	3	2	1	0
Reserved			EVT1SYNCE	EVT1SOCE	EVT1FRC SYNCSEL
R-0			R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-58. Digital Compare B Control (DCBCTL) Register Field Descriptions**

Bit	Field	Value	Description
15-10	Reserved		Reserved
9	EVT2FRC SYNCSEL	0 1	DCBEVT2 Force Synchronization Signal Select Source Is Synchronous Signal Source Is Asynchronous Signal
8	EVT2SRCSEL	0 1	DCBEVT2 Source Signal Select Source Is DCBEVT2 Signal Source Is DCEVTFILT Signal
7-4	Reserved		Reserved
3	EVT1SYNCE	0 1	DCBEVT1 SYNC, Enable/Disable SYNC Generation Disabled SYNC Generation Enabled
2	EVT1SOCE	0 1	DCBEVT1 SOC, Enable/Disable SOC Generation Disabled SOC Generation Enabled
1	EVT1FRC SYNCSEL	0 1	DCBEVT1 Force Synchronization Signal Select Source Is Synchronous Signal Source Is Asynchronous Signal
0	EVT1SRCSEL	0 1	DCBEVT1 Source Signal Select Source Is DCBEVT1 Signal Source Is DCEVTFILT Signal



### 3.4.8.4 Digital Compare Filter Control (DCFCTL) Register

**Figure 3-110. Digital Compare Filter Control (DCFCTL) Register**

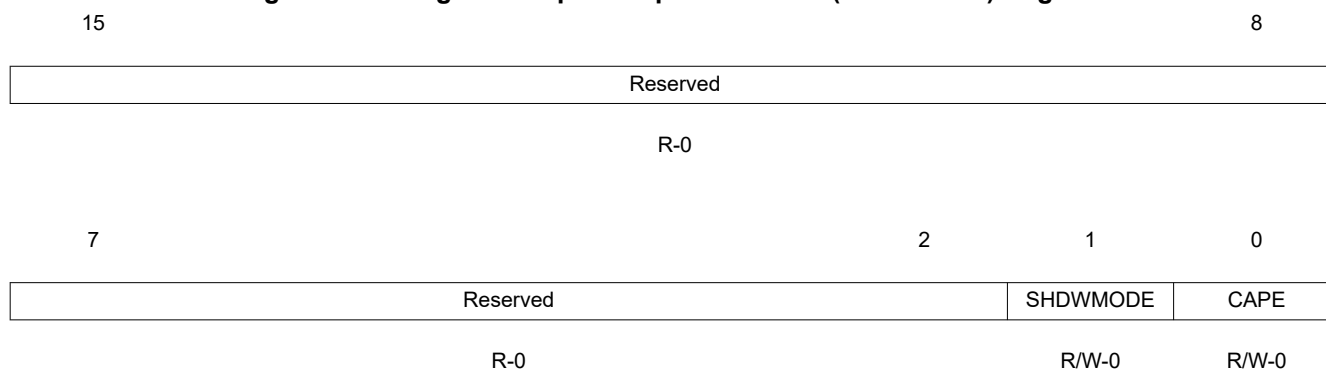
15	13	12	8				
Reserved		Reserved					
R-0		R-0					
7	6	5	4	3	2	1	0
Reserved	Reserved	PULSESEL	BLANKINV	BLANKE	SRCSEL		
R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-59. Digital Compare Filter Control (DCFCTL) Register Field Descriptions**

Bit	Field	Value	Description
15-13	Reserved		Reserved
12-8	Reserved		Reserved for TI Test
7	Reserved		Reserved
6	Reserved		Reserved for TI Test
5-4	PULSESEL	00 01 10 11	Pulse Select For Blanking & Capture Alignment Time-base counter equal to period (TBCTR = TBPRD) Time-base counter equal to zero (TBCTR = 0x0000) Reserved Reserved
3	BLANKINV	0 1	Blanking Window Inversion Blanking window not inverted Blanking window inverted
2	BLANKE	0 1	Blanking Window Enable/Disable Blanking window is disabled Blanking window is enabled
1-0	SRCSEL	00 01 10 11	Filter Block Signal Source Select Source Is DCAEVT1 Signal Source Is DCAEVT2 Signal Source Is DCBEVT1 Signal Source Is DCBEVT2 Signal

### 3.4.8.5 Digital Compare Capture Control (DCCAPCTL) Register

**Figure 3-111. Digital Compare Capture Control (DCCAPCTL) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-60. Digital Compare Capture Control (DCCAPCTL) Register Field Descriptions**

Bit	Field	Value	Description
15-2	Reserved		Reserved
1	SHDWMODE	0	TBCTR Counter Capture Shadow Select Mode Enable shadow mode. The DCCAP active register is copied to shadow register on a TBCTR = TBPRD or TBCTR = zero event as defined by the DCFCTL[PULSESEL] bit. CPU reads of the DCCAP register will return the shadow register contents.
		1	Active Mode. In this mode the shadow register is disabled. CPU reads from the DCCAP register will always return the active register contents.
0	CAPE	0	TBCTR Counter Capture Enable/Disable Disable the time-base counter capture.
		1	Enable the time-base counter capture.

### 3.4.8.6 Digital Compare Filter Offset (DCOFFSET) Register

**Figure 3-112. Digital Compare Filter Offset (DCOFFSET) Register**

15

0

DCOFFSET
----------

R-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-61. Digital Compare Filter Offset (DCOFFSET) Register Field Descriptions**

Bit	Field	Value	Description
15-0	OFFSET	0000- FFFFh	<p>Blanking Window Offset</p> <p>These 16-bits specify the number of TBCLK cycles from the blanking window reference to the point when the blanking window is applied. The blanking window reference is either period or zero as defined by the DCFCTL[PULSESEL] bit.</p> <p>This offset register is shadowed and the active register is loaded at the reference point defined by DCFCTL[PULSESEL]. The offset counter is also initialized and begins to count down when the active register is loaded. When the counter expires, the blanking window is applied. If the blanking window is currently active, then the blanking window counter is restarted.</p>

### 3.4.8.7 Digital Compare Filter Offset Counter (DCOFFSETCNT) Register

**Figure 3-113. Digital Compare Filter Offset Counter (DCOFFSETCNT) Register**

15

0

OFFSETCNT
-----------

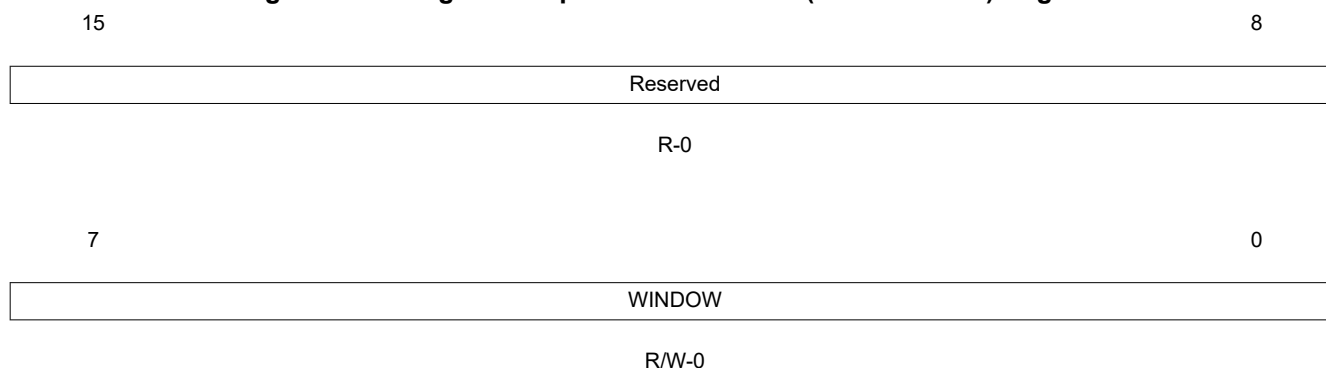
R-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-62. Digital Compare Filter Offset Counter (DCOFFSETCNT) Register Field Descriptions**

Bit	Field	Value	Description
15-0	OFFSETCNT	0000- FFFFh	<p>Blanking Offset Counter</p> <p>These 16-bits are read only and indicate the current value of the offset counter. The counter counts down to zero and then stops until it is re-loaded on the next period or zero event as defined by the DCFCTL[PULSESEL] bit.</p> <p>The offset counter is not affected by the free/soft emulation bits. That is, it will always continue to count down if the device is halted by an emulation stop.</p>

### 3.4.8.8 Digital Compare Filter Window (DCFWINDOW) Register

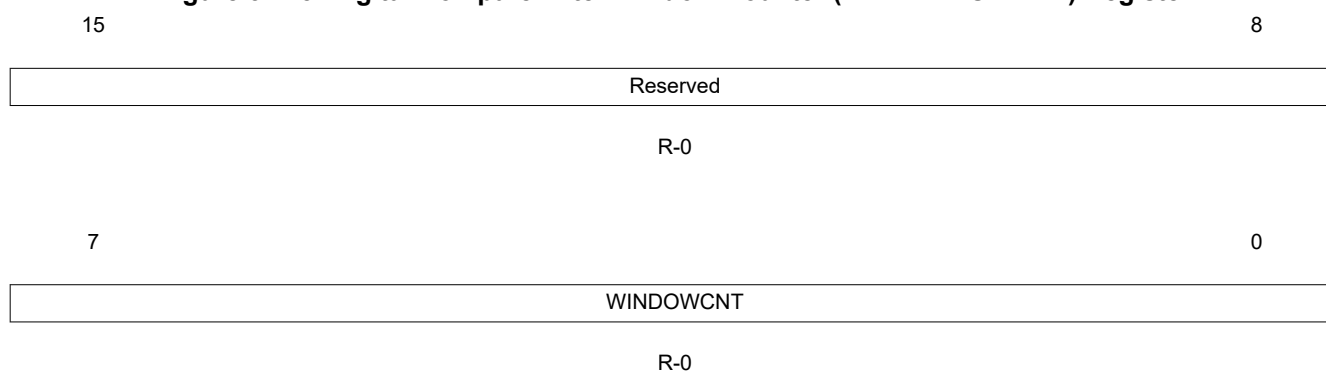
**Figure 3-114. Digital Compare Filter Window (DCFWINDOW) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-63. Digital Compare Filter Window (DCFWINDOW) Register Field Descriptions**

Bit	Field	Value	Description
15-8	Reserved		Reserved
7-0	WINDOW	00h 01-FFh	Blanking Window Width No blanking window is generated. Specifies the width of the blanking window in TBCLK cycles. The blanking window begins when the offset counter expires. When this occurs, the window counter is loaded and begins to count down. If the blanking window is currently active and the offset counter expires, the blanking window counter is restarted. The blanking window can cross a PWM period boundary.

### 3.4.8.9 Digital Compare Filter Window Counter (DCFWINDOWCNT) Register

**Figure 3-115. Digital Compare Filter Window Counter (DCFWINDOWCNT) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-64. Digital Compare Filter Window Counter (DCFWINDOWCNT) Register Field Descriptions**

Bit	Field	Value	Description
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7-0	WINDOWCNT	00-FF	Blanking Window Counter These 8 bits are read only and indicate the current value of the window counter. The counter counts down to zero and then stops until it is re-loaded when the offset counter reaches zero again.

### 3.4.8.10 Digital Compare Counter Capture (DCCAP) Register

**Figure 3-116. Digital Compare Counter Capture (DCCAP) Register**

15

0

DCCAP
R-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 3-65. Digital Compare Counter Capture (DCCAP) Register Field Descriptions**

Bit	Field	Value	Description
15-0	DCCAP	0000-FFFFh	<p>Digital Compare Time-Base Counter Capture</p> <p>To enable time-base counter capture, set the DCCAPCLT[CAPE] bit to 1.</p> <p>If enabled, reflects the value of the time-base counter (TBCTR) on the low to high edge transition of a filtered (DCEVTFLT) event. Further capture events are ignored until the next period or zero as selected by the DCFCTL[PULSESEL] bit.</p> <p>Shadowing of DCCAP is enabled and disabled by the DCCAPCTL[SHDWMODE] bit. By default this register is shadowed.</p> <ul style="list-style-type: none"> <li>If DCCAPCTL[SHDWMODE] = 0, then the shadow is enabled. In this mode, the active register is copied to the shadow register on the TBCTR = TBPRD or TBCTR = zero as defined by the DCFCTL[PULSESEL] bit. CPU reads of this register will return the shadow register value.</li> <li>If DCCAPCTL[SHDWMODE] = 1, then the shadow register is disabled. In this mode, CPU reads will return the active register value.</li> </ul> <p>The active and shadow registers share the same memory map address.</p>

### 3.4.9 Proper Interrupt Initialization Procedure

When the ePWM peripheral clock is enabled it may be possible that interrupt flags may be set due to spurious events due to the ePWM registers not being properly initialized. The proper procedure for initializing the ePWM peripheral is as follows:

1. Disable global interrupts (CPU INTM flag)
2. Disable ePWM interrupts
3. Set TBCLKSYNC=0
4. Initialize peripheral registers
5. Set TBCLKSYNC=1
6. Clear any spurious ePWM flags (including PIEIFR)
7. Enable ePWM interrupts
8. Enable global interrupts

This page intentionally left blank.

This chapter is used in conjunction with [Chapter 3](#). The HRPWM module is a type 1 HRPWM. See the [C2000 Real-Time Control Peripheral Reference Guide](#) for a list of all devices with an HRPWM module of the same type, to determine the differences between types, and for a list of device-specific differences within a type.

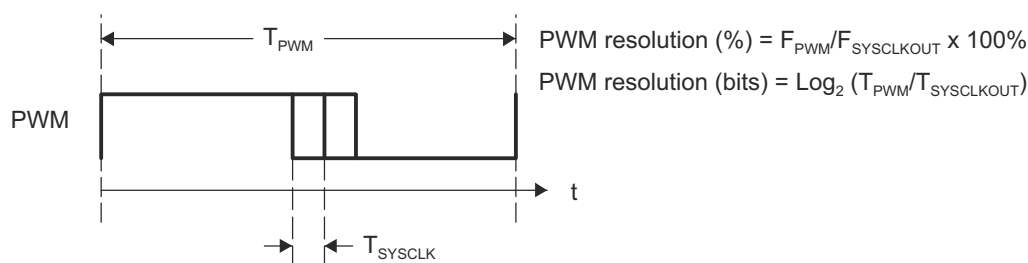
The HRPWM module extends the time resolution capabilities of the conventionally derived digital pulse width modulator (PWM). HRPWM is typically used when PWM resolution falls below ~9-10 bits. The key features of HRPWM are:

- Extended time resolution capability
- Used in both duty cycle and phase-shift control methods
- Finer time granularity control or edge positioning using extensions to the Compare A and Phase registers
- Implemented using the A signal path of PWM, that is, on the EPWMxA output
- Self-check diagnostics software mode to check if the micro edge positioner (MEP) logic is running optimally
- Enables high-resolution output on B signal path of PWM via PWM A and B channel path swapping
- Enables high-resolution output on B signal output via inversion of A signal output
- Enables high-resolution period control on the ePWMxA output on devices with a type 1 ePWM module. See the device-specific data manual to determine if your device has a type 1 ePWM module for high-resolution period support. The ePWMxB output will have a  $\pm 1$ -2 cycle jitter in this mode.

<b>4.1 Introduction</b> .....	<b>344</b>
<b>4.2 Operational Description of HRPWM</b> .....	<b>345</b>
<b>4.3 SFO Library Software - SFO_TI_Build_V6.lib</b> .....	<b>363</b>
<b>4.4 HRPWM Registers</b> .....	<b>367</b>

## 4.1 Introduction

The ePWM peripheral is used to perform a function that is mathematically equivalent to a digital-to-analog converter (DAC). As shown in [Figure 4-1](#), the effective resolution for conventionally generated PWM is a function of PWM frequency (or period) and system clock frequency.



**Figure 4-1. Resolution Calculations for Conventionally Generated PWM**

If the required PWM operating frequency does not offer sufficient resolution in PWM mode, you may want to consider HRPWM. As an example of improved performance offered by HRPWM, [Table 4-1](#) shows resolution in bits for various PWM frequencies. These values assume a MEP step size of 180 ps. See your device-specific data sheet for typical and maximum performance specifications for the MEP.

**Table 4-1. Resolution for PWM and HRPWM**

PWM Frequency (kHz)	Regular Resolution (PWM)				High Resolution (HRPWM)	
	60 MHz SYSCLKOUT		50 MHz SYSCLKOUT		Bits	%
	Bits	%	Bits	%		
20	11.6	0.0	11.3	0	18.1	0.000
50	10.2	0.1	10	0.1	16.8	0.001
100	9.2	0.2	9	0.2	15.8	0.002
150	8.6	0.3	8.4	0.3	15.2	0.003
200	8.2	0.3	8	0.4	14.8	0.004
250	7.9	0.4	7.6	0.5	14.4	0.005
500	6.9	0.8	6.6	1	13.4	0.009
1000	5.9	1.7	5.6	2	12.4	0.018
1500	5.3	2.5	5.1	3	11.9	0.027
2000	4.9	3.3	4.6	4	11.4	0.036

Although each application may differ, typical low-frequency PWM operation (below 250 kHz) may not require HRPWM. HRPWM capability is most useful for high-frequency PWM requirements of power conversion topologies such as:

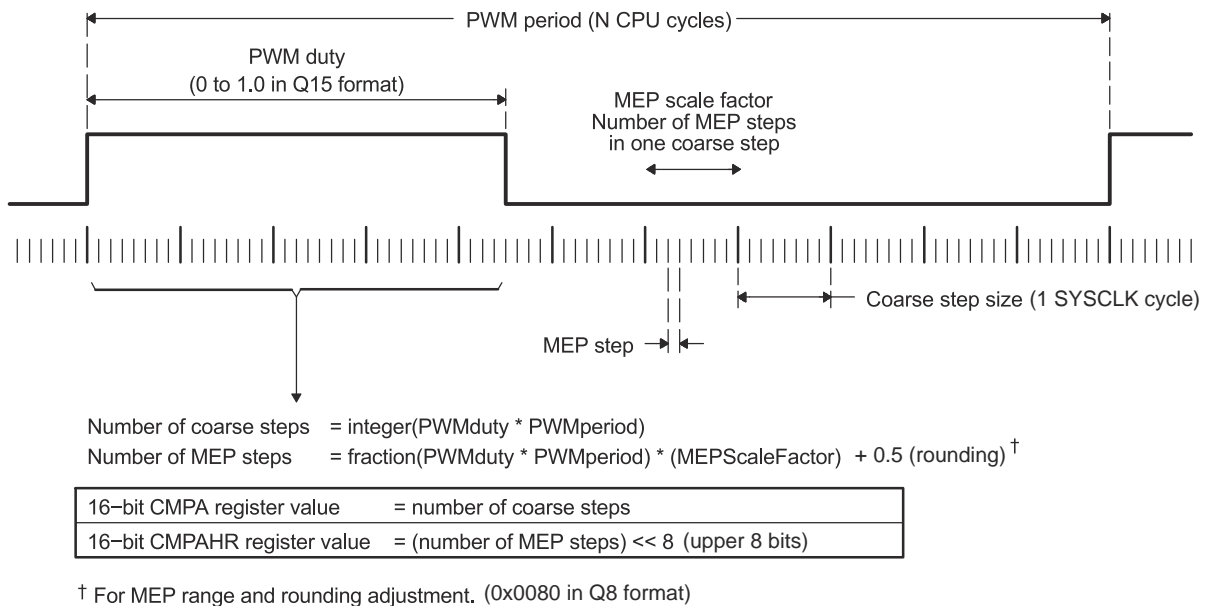
- Single-phase buck, boost, and flyback
- Multi-phase buck, boost, and flyback
- Phase-shifted full bridge
- Direct modulation of D-Class power amplifiers



## 4.2 Operational Description of HRPWM

The HRPWM is based on micro edge positioner (MEP) technology. MEP logic is capable of positioning an edge very finely by sub-dividing one coarse system clock of a conventional PWM generator. The time step accuracy is on the order of 150 ps. See the device-specific data sheet for the typical MEP step size on a particular device. The HRPWM also has a self-check software diagnostics mode to check if the MEP logic is running optimally, under all operating conditions. Details on software diagnostics and functions are in [Section 4.2.4](#).

Figure 4-2 shows the relationship between one coarse system clock and edge position in terms of MEP steps, which are controlled with an 8-bit field in the Compare A extension register (CMPAHR).



**Figure 4-2. Operating Logic Using MEP**

To generate an HRPWM waveform, configure the TBM, CCM, and AQM registers as you would to generate a conventional PWM of a given frequency and polarity. The HRPWM works together with the TBM, CCM, and AQM registers to extend edge resolution, and should be configured accordingly. Although many programming combinations are possible, only a few are needed and practical. These methods are described in [Section 4.2.5](#).

Registers discussed but not found in this chapter are in [Chapter 3](#).

The HRPWM operation is controlled and monitored using the registers listed in [Table 4-2](#).

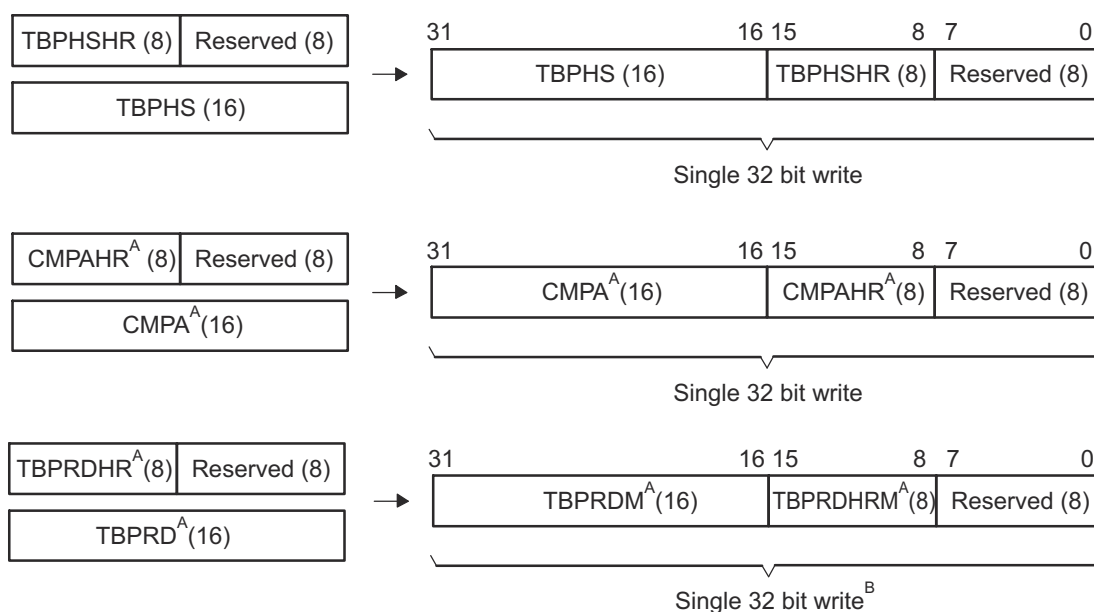
**Table 4-2. HRPWM Registers**

Register	Address Offset	Shadowed	Description
TBPHSHR	0x0002	No	Extension Register for HRPWM Phase (8 bits)
TBPRDHR	0x0006	Yes	Extension Register for HRPWM Period (8 bits)
CMPAHR	0x0008	Yes	Extension Register for HRPWM Duty (8 bits)
HRCNFG	0x0020	No	HRPWM Configuration Register
HRPWR	0x0021	No	HRPWM Power Register
HRMSTEP	0x0026	No	HRPWM MEP Step Register
HRPCTL	0x0028	No	High Resolution Period Control Register
TBPRDHRM	0x002A	Yes	Extension Mirror Register for HRPWM Period (8 bits)
CMPAHRM	0x002C	Yes	Extension Mirror Register for HRPWM Duty (8 bits)

### 4.2.1 Controlling the HRPWM Capabilities

The MEP of the HRPWM is controlled by three extension registers, each 8-bits wide. These HRPWM registers are concatenated with the 16-bit TBPHS, TBPRD, and CMPA registers used to control PWM operation, see [Figure 4-3](#).

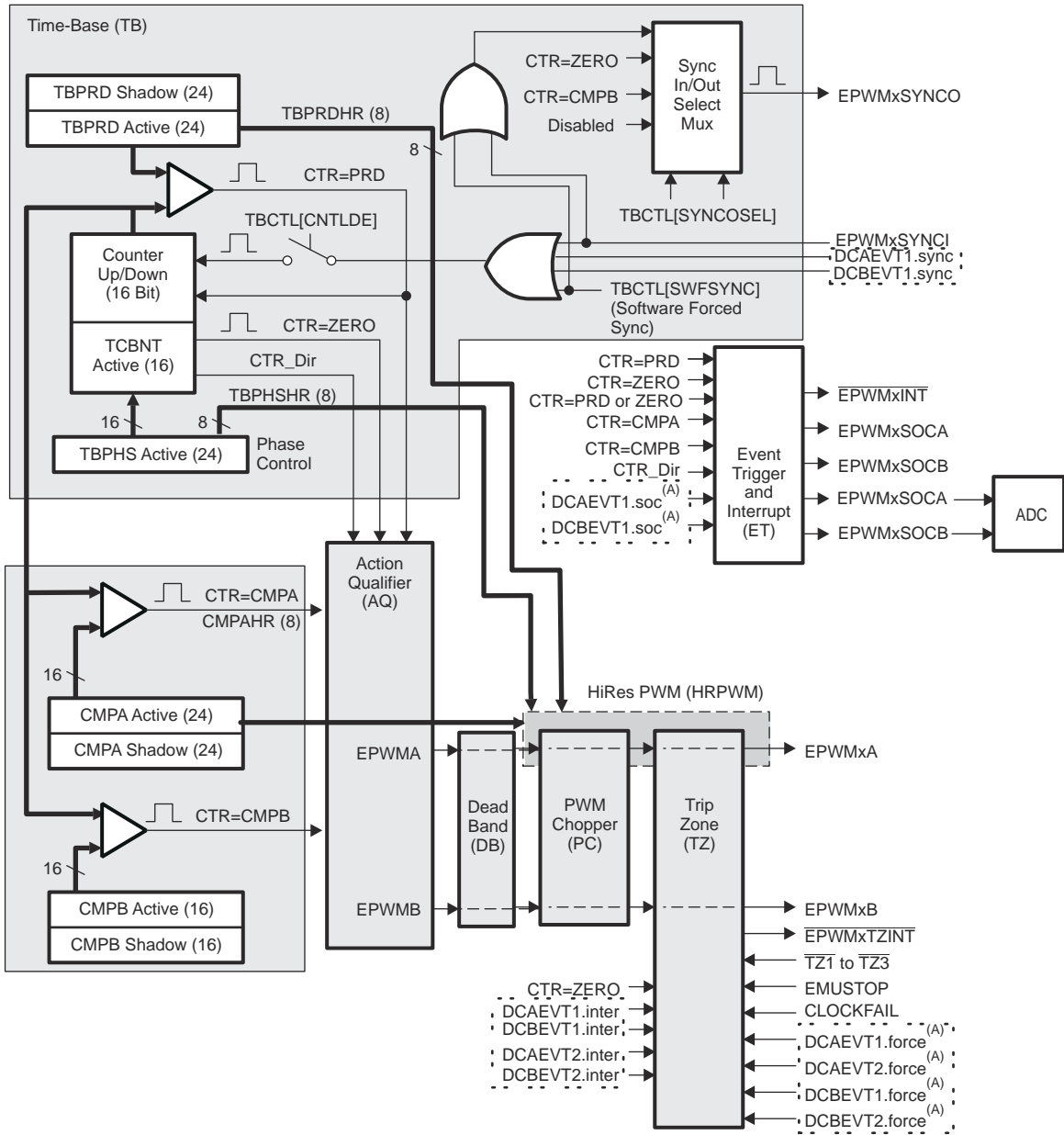
- TBPHSHR - Time-Base Phase High Resolution Register
- CMPAHR - Counter-Compare A High Resolution Register
- TBPRDHR - Time-Base Period High Resolution Register (available on some devices)



- A. These registers are mirrored and can be written to at two different memory locations (mirrored registers have an "M" suffix (that is, CMPA mirror = CMPAM). Reads of the high-resolution mirror registers will result in indeterminate values.
- B. TBPRDHR and TBPRD may be written to as a 32-bit value only at the mirrored address
- Not all devices may have TBPRD and TBPRDHR registers. See device-specific data sheet for more information

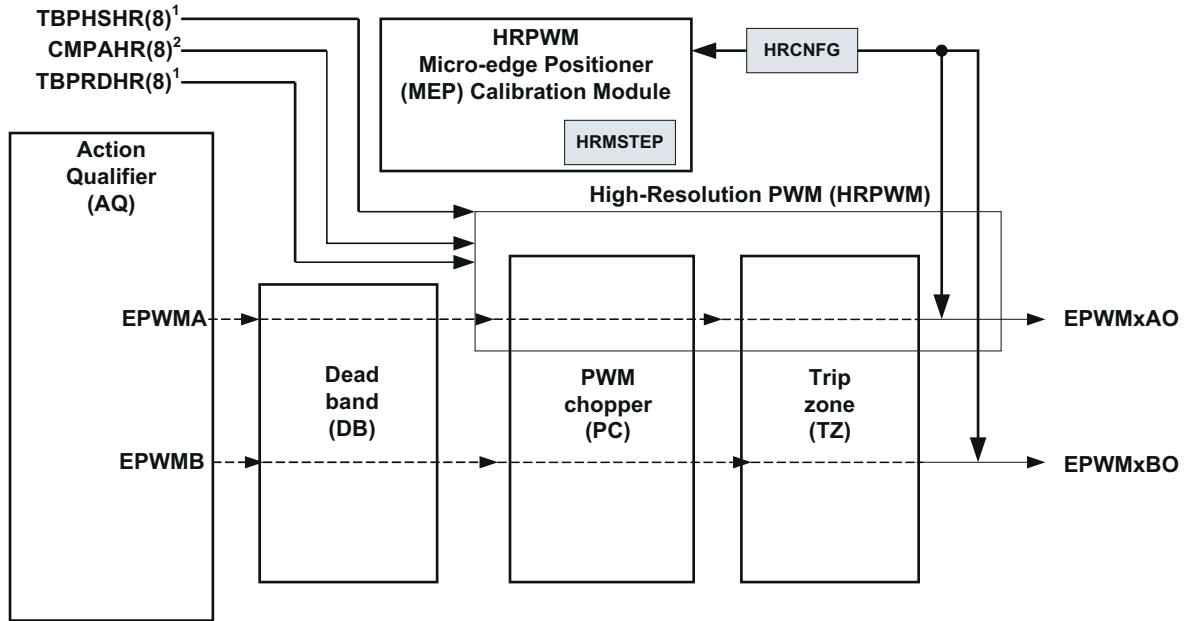
**Figure 4-3. HRPWM Extension Registers and Memory Configuration**

HRPWM capabilities are controlled using the Channel A PWM signal path. HRPWM support on the channel B signal path is available by properly configuring the HRCNFG register. [Figure 4-4](#) shows how the HRPWM interfaces with the 8-bit extension registers.



A. These events are generated by the type 1 ePWM digital compare (DC) submodule based on the levels of the COMPxOUT and  $\bar{TZ}$  signals.

Figure 4-4. HRPWM System Interface



1. From ePWM time-base (TB) submodule.
2. From ePWM counter-compare (CC) submodule.

Figure 4-5. HRPWM Block Diagram

### 4.2.2 Configuring the HRPWM

Once the ePWM has been configured to provide conventional PWM of a given frequency and polarity, the HRPWM is configured by programming the HRCNFG register located at offset address 20h. This register provides the following configuration options:

- Edge Mode** The MEP can be programmed to provide precise position control on the rising edge (RE), falling edge (FE), or both edges (BE) at the same time. FE and RE are used for power topologies requiring duty cycle control (CMPA high-resolution control), while BE is used for topologies requiring phase shifting, for example, phase shifted full bridge (TBPHS or TBPRD high-resolution control).
- Control Mode** The MEP is programmed to be controlled either from the CMPAHR register (duty cycle control) or the TBPHSHR register (phase control). RE or FE control mode should be used with CMPAHR register. BE control mode should be used with TBPHSHR register. When the MEP is controlled from the TBPRDHR register (period control) the duty cycle and phase can also be controlled via their respective high-resolution registers.
- Shadow Mode** This mode provides the same shadowing (double buffering) option as in regular PWM mode. This option is valid only when operating from the CMPAHR and TBPRDHR registers and should be chosen to be the same as the regular load option for the CMPA register. If TBPHSHR is used, then this option has no effect.
- High-Resolution B Signal Control** The B signal path of an ePWM channel can generate a high-resolution output by either swapping the A and B outputs (the high-resolution signal will appear on ePWMxB instead of ePWMxA) or by outputting an inverted version of the high-resolution ePWMxA signal on the ePWMxB pin.
- Auto-conversion Mode** This mode is used in conjunction with the scale factor optimization software only. For a type 1 HRPWM module, if auto-conversion is enabled,  $CMPAHR = \text{fraction}(PWMduty * PWMperiod) \ll 8$ . The scale factor optimization software will calculate the MEP scale factor in background code and automatically update the HRMSTEP register with the calculated number of MEP steps per coarse step. The MEP Calibration Module will then use the values in the HRMSTEP and CMPAHR register to automatically calculate the appropriate number of MEP steps represented by the fractional duty cycle and move the high-resolution ePWM signal edge accordingly. If auto-conversion is disabled, the CMPAHR register behaves like a type 0 HRPWM module and  $CMPAHR = (\text{fraction}(PWMduty * PWMperiod) * MEP \text{ Scale Factor} + 0.5) \ll 8$ . All of these calculations will need to be performed by user code in this mode, and the HRMSTEP register is ignored. Auto-conversion for high-resolution period has the same behavior as auto-conversion for high-resolution duty cycle. Auto-conversion must always be enabled for high-resolution period mode.

### 4.2.3 Principle of Operation

The MEP logic is capable of placing an edge in one of 255 (8 bits) discrete time steps (see device-specific data sheet for typical MEP step size). The MEP works with the TBM and CCM registers to be certain that time steps are optimally applied and that edge placement accuracy is maintained over a wide range of PWM frequencies, system clock frequencies, and other operating conditions. Table 4-3 shows the typical range of operating frequencies supported by the HRPWM.

**Table 4-3. Relationship Between MEP Steps, PWM Frequency, and Resolution**

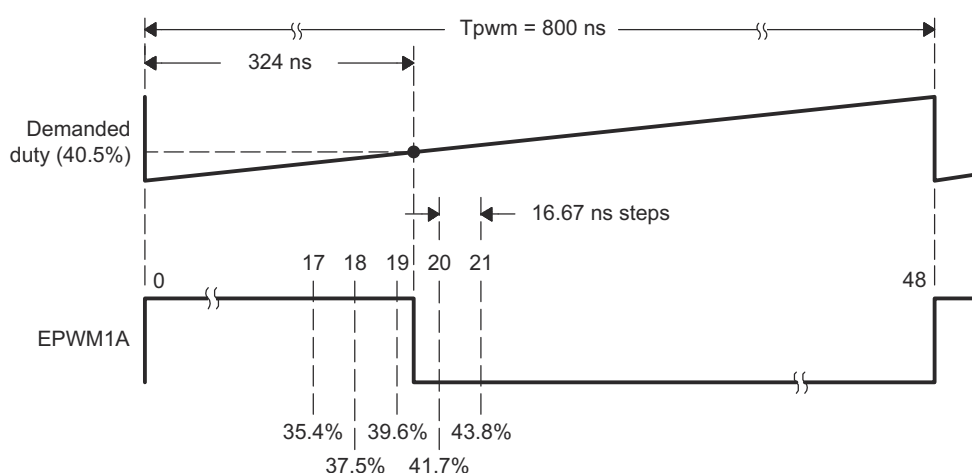
System (MHz)	MEP Steps Per SYSCLKOUT <sup>(1) (2) (3)</sup>	PWM Minimum (Hz) <sup>(4)</sup>	PWM Maximum (MHz)	Resolution @ Maximum (Bits) <sup>(5)</sup>
50.0	111	763	2.50	11.1
60.0	93	916	3.00	10.9

- (1) System frequency = SYSCLKOUT, that is, CPU clock. TBCLK = SYSCLKOUT.
- (2) Table data based on a MEP time resolution of 180 ps (this is an example value. See the device-specific data sheet for MEP limits)
- (3) MEP steps applied =  $T_{SYSCLKOUT}/180$  ps in this example.
- (4) PWM minimum frequency is based on a maximum period value, that is, TBPRD = 65535. PWM mode is asymmetrical up-count.
- (5) Resolution in bits is given for the maximum PWM frequency stated.

#### 4.2.3.1 Edge Positioning

In a typical power control loop (for example, switch modes, digital motor control [DMC], uninterruptible power supply [UPS]), a digital controller (PID, 2pole/2zero, lag/lead, and so on) issues a duty command, usually expressed in a per unit or percentage terms. Assume that for a particular operating point, the demanded duty cycle is 0.405 or 40.5% on time and the required converter PWM frequency is 1.25 MHz. In conventional PWM generation with a system clock of 60 MHz, the duty cycle choices are in the vicinity of 40.5%. In Figure 4-6, a compare value of 19 counts (that is, duty = 39.6%) is the closest to 40.5% that you can attain. This is equivalent to an edge position of 316.7 ns instead of the desired 324 ns. This data is shown in Table 4-4.

By utilizing the MEP, you can achieve an edge position much closer to the desired point of 324 ns. Table 4-4 shows that in addition to the CMPA 44 steps of the MEP (CMPAHR register) will position the edge at 323.92 ns, resulting in almost zero error. In this example, it is assumed that the MEP has a step resolution of 180 ps.



**Figure 4-6. Required PWM Waveform for a Requested Duty = 30.0%**

**Table 4-4. CMPA versus Duty (left), and [CMPA:CMPAHR] versus Duty (right)**

CMPA (count) <sup>(1) (2) (3)</sup>	Duty	High Time (ns)	CMPA (count)	CMPAHR (count)	Duty	High Time (ns)
15	31.25%	250	19	40	40.40%	323.2
16	33.33%	267	19	41	40.42%	323.38
17	35.42%	283	19	42	40.45%	323.56
18	37.50%	300	19	43	40.47%	323.74
19	39.58%	316	19	44	40.49%	323.92
20	41.67%	333	19	45	40.51%	324.1
21	43.75%	350	19	46	40.54%	324.28
			19	47	40.56%	324.46
Required			19	48	40.58%	324.64
19.4	40.50%	324	19	49	40.60%	324.82

(1) System clock, SYSCLKOUT and TBCLK = 60 MHz, 16.67 ns

(2) For a PWM Period register value of 48 counts, PWM Period = 48 x 16.67 ns = 800 ns, PWM frequency = 1/800 ns = 1.25 MHz

(3) Assumed MEP step size for the above example = 180 ps

See the device-specific data manual for typical and maximum MEP values.

#### 4.2.3.2 Scaling Considerations

The mechanics of how to position an edge precisely in time has been demonstrated using the resources of the standard CMPA and MEP (CMPAHR) registers. In a practical application, however, it is necessary to seamlessly provide the CPU a mapping function from a per-unit (fractional) duty cycle to a final integer (non-fractional) representation that is written to the [CMPA:CMPAHR] register combination. This section describes the mapping from a per-unit duty cycle only. The method for mapping from a per-unit period is described in [Section 4.2.3.4](#).

To do this, first examine the scaling or mapping steps involved. It is common in control software to express duty cycle in a per-unit or percentage basis. This has the advantage of performing all needed math calculations without concern for the final absolute duty cycle, expressed in clock counts or high time in ns. Furthermore, it makes the code more transportable across multiple converter types running different PWM frequencies.

To implement the mapping scheme, a two-step scaling procedure is required.

#### Assumptions for this example:

System clock , SYSCLKOUT	=	16.67 ns (60MHz)
PWM frequency	=	1.25 MHz (1/800 ns)
Required PWM duty cycle, <b>PWMDuty</b>	=	0.300 (30.0%)
PWM period in terms of coarse steps, <b>PWMperiod</b> (800 ns/ 11.1 ns)	=	48
Number of MEP steps per coarse step at 180 ps ( 11.1 ns/ 180 ps), <b>MEP_ScaleFactor</b>	=	93
Value to keep CMPAHR within the range of 1-255 and fractional rounding constant (default value). In the event that $\text{frac}(\text{PWMDuty} * \text{PWMperiod}) * \text{MEP\_ScaleFactor}$ results in a value with a decimal portion $\geq 0.5$ , this rounding constant will round the CMPAHR value up 1 MEP step.	=	0.5 (0 080h in Q8 format)

**Step 1: Percentage Integer Duty value conversion for CMPA register**

CMPA register value =  $\text{int}(\text{PWMDuty} \times \text{PWMperiod})$ ; int means integer part  
=  $\text{int}(0.405 \times 48)$   
CMPA register value = 19 (13h)

**Step 2: Fractional value conversion for CMPAHR register**

CMPAHR register value =  $(\text{frac}(\text{PWMDuty} \times \text{PWMperiod}) \times \text{MEP\_ScaleFactor} + 0.5) \ll 8$ ; frac means fractional part  
=  $(\text{frac}(19.4) \times 93 + 0.5) \ll 8$ ; Shift is to move the value as CMPAHR high byte  
=  $((0.4 \times 93 + 0.5) \ll 8)$   
=  $((37.2 + 0.5) \ll 8)$   
=  $37.7 \times 256$ ; Shifting left by 8 is the same as multiplying by 256.  
= 9,651  
CMPAHR value = 25B3h; lower 8 bits will be ignored by hardware.

---

**Note**

If the AUTOCONV bit (HRCNFG.6) is set and the MEP\_ScaleFactor is in the HRMSTEP register, then CMPAHR register value =  $\text{frac}(\text{PWMDuty} \times \text{PWMperiod} \ll 8)$ . The rest of the conversion calculations are performed automatically in hardware, and the correct MEP-scaled signal edge appears on the ePWM channel output. If AUTOCONV is not set, the above calculations must be performed by software.

---

**Note**

The MEP scale factor (MEP\_ScaleFactor) varies with the system clock and DSP operating conditions. TI provides an MEP scale factor optimizing (SFO) software C function, which uses the built in diagnostics in each HRPWM and returns the best scale factor for a given operating point.

The scale factor varies slowly over a limited range so the optimizing C function can be run very slowly in a background loop.

The CMPA and CMPAHR registers are configured in memory so that the 32-bit data capability of the 28x CPU can write this as a single concatenated value, [CMPA:CMPAHR]. The TBPRDM and TBPRDHRM (mirror) registers are similarly configured in memory.

The mapping scheme has been implemented in both C and assembly, as shown in [Section 4.2.5](#). The actual implementation takes advantage of the 32-bit CPU architecture of the 28xx, and is somewhat different from the steps shown in [Section 4.2.3.2](#).

For time critical control loops where every cycle counts, the assembly version is recommended. This is a cycle optimized function (11 SYSCLKOUT cycles ) that takes a Q15 duty value as input and writes a single [CMPA:CMPAHR] value.

---



### 4.2.3.3 Duty Cycle Range Limitation

In high resolution mode, the MEP is not active for 100% of the PWM period. It becomes operational:

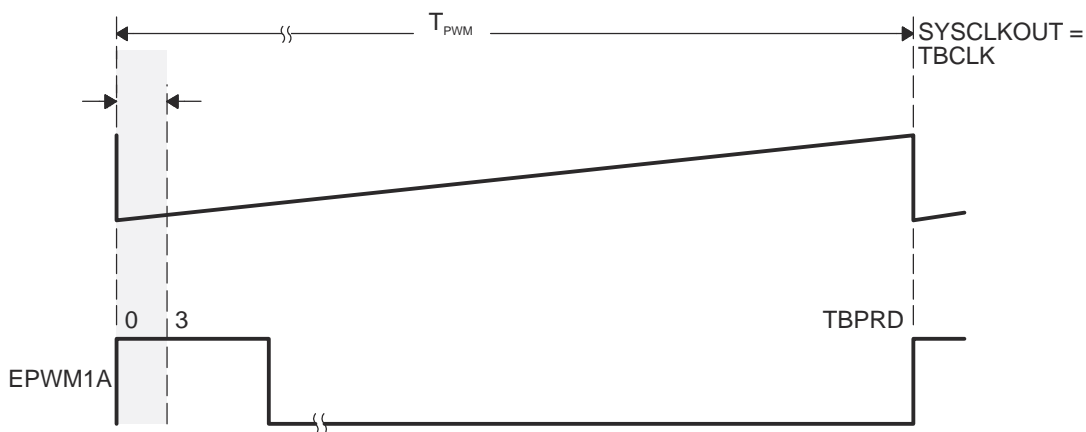
- 3 SYSCLK cycles after the period starts when high-resolution period (TBPRDHR) control is not enabled.
- When high resolution period (TBPRDHR) control is enabled via the HRPCTL register:
  - In up-count mode: 3 SYSCLK cycles after the period starts until 3 SYSCLK cycles before the period ends.
  - In up-down count mode: when counting up, 3 cycles after CTR = 0 until 3 cycles before CTR = PRD, and when counting down, 3 cycles after CTR = PRD until 3 cycles before CTR = 0.

To better understand the useable duty cycle range, see [Table 4-5](#). Duty cycle range limitations are illustrated in [Figure 4-7](#) to [Figure 4-10](#). This limitation imposes a duty cycle limit on the MEP. For example, precision edge control is not available all the way down to 0% duty cycle. When high-resolution period control is disabled, although for the first three cycles, the HRPWM capabilities are not available, regular PWM duty control is still fully operational down to 0% duty. In most applications, this should not be an issue as the controller regulation point is usually not designed to be close to 0% duty cycle. When high-resolution period control is enabled (HRPCTL[HRPE]=1), the duty cycle must not fall within the restricted range. Otherwise, there may be undefined behavior on the ePWMxA output.

**Table 4-5. Duty Cycle Range Limitation for 3 SYSCLK/TBCLK Cycles**

PWM Frequency <sup>(1)</sup> (kHz)	3 Cycles Minimum Duty	3 Cycles Maximum Duty <sup>(2)</sup>
200	0.67%	99.00%
400	1.33%	98.00%
600	2.00%	97.00%
800	2.67%	96.00%
1000	3.33%	95.00%
1200	4.00%	94.00%
1400	4.67%	93.00%
1600	5.33%	92.00%
1800	6.00%	91.00%
2000	6.67%	90.00%

- (1) System clock -  $T_{SYSCLKOUT} = 16.67$  ns System clock = TBCLK = 60 MHz  
 (2) This limitation applies only if high-resolution period (TBPRDHR) control is enabled.



**Figure 4-7. Low % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0)**

If the application demands HRPWM operation in the low percent duty cycle region, then the HRPWM can be configured to operate in count-down mode with the rising edge position (REP) controlled by the MEP when high-resolution period is disabled (HRPCTL[HRPE] = 0). This is illustrated in Figure 4-8. In this case, low percent duty limitation is no longer an issue. However, there will be a maximum duty limitation with same percent numbers as given in Table 4-5.

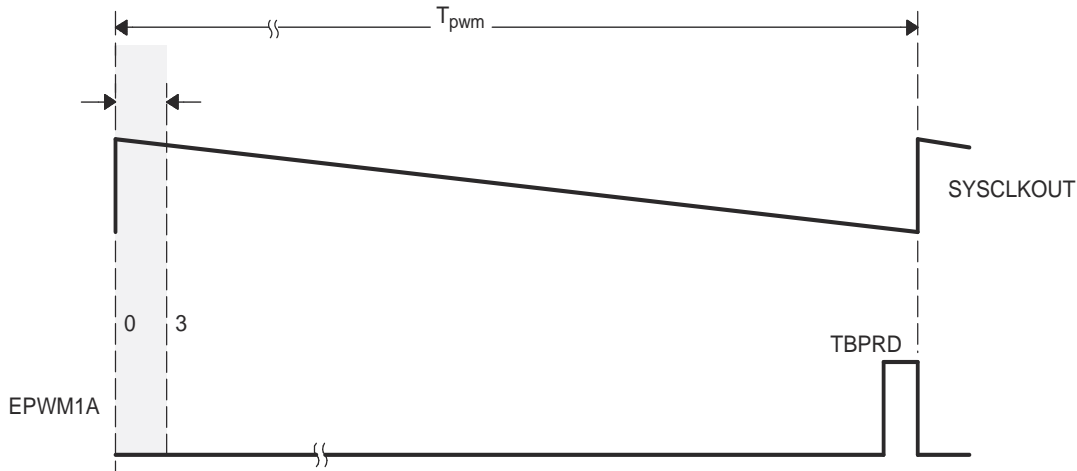


Figure 4-8. High % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0)

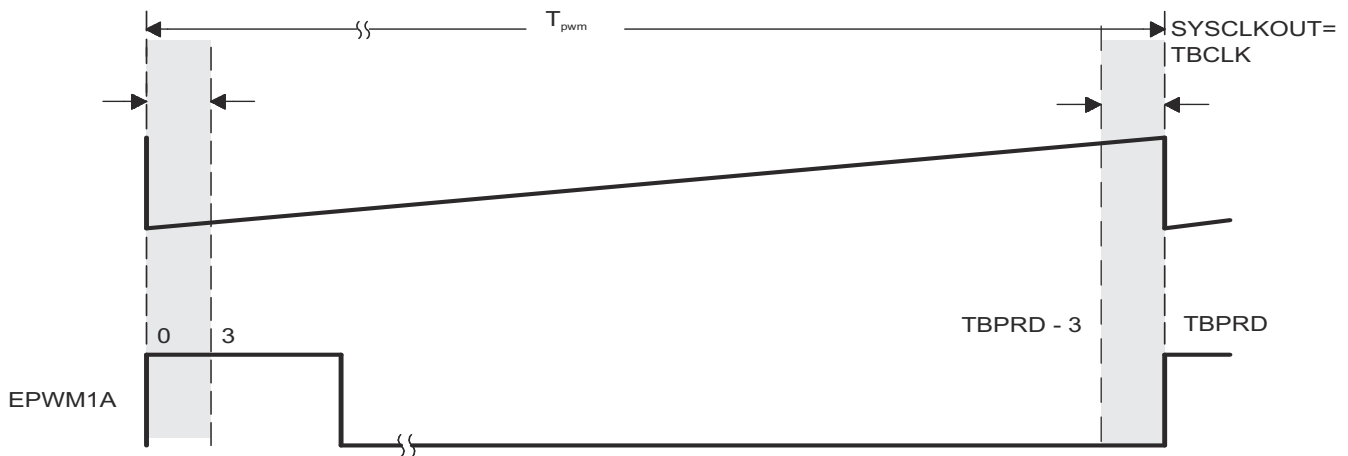


Figure 4-9. Up-Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)

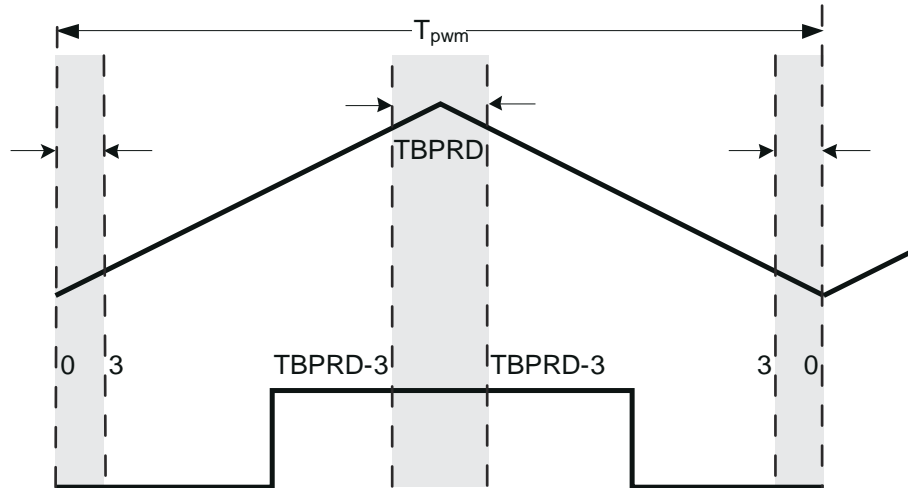


Figure 4-10. Up-Down Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)

**Note**

If the application has enabled high-resolution period control (HRPCTL[HRPE]=1), the duty cycle must not fall within the restricted range. Otherwise, there will be undefined behavior on the ePWM output.

**4.2.3.4 High Resolution Period**

High resolution period control using the MEP logic is supported on devices with a Type 1 ePWM module via the TBPRDHR(M) register.

**Note**

When high-resolution period control is enabled, the ePWMxB output will have +/- 1 TBCLK cycle jitter in up-count mode and +/- 2 TBCLK cycle jitter in up-down count mode.

The scaling procedure described for duty cycle in [Section 4.2.3.2](#) applies for high-resolution period as well:

**Assumptions for this example:**

System clock , SYSCLKOUT	= 16.67 ns (60 MHz)
Required PWM frequency	= 175 kHz (TBPRD value of 342.857)
Number of MEP steps per coarse step at 180 ps (MEP_ScaleFactor)	= 93 (16.67 ns/180 ps)
Value to keep TBPRDHR within range of 1-255 and fractional rounding constant (default value)	= 0.5 (0080h in Q8 format)

**Problem:**

In up-count mode:

If TBPRD = 342, then PWM frequency = 174.93 kHz (period =  $(342+1) \cdot T_{TBCLK}$ ).

TBPRD = 341, then PWM frequency = 175.44 kHz (period =  $(341+1) \cdot T_{TBCLK}$ ).

In up-down count mode:

If TBPRD = 172, then PWM frequency = 174.42 kHz (period =  $(172 \cdot 2) \cdot T_{TBCLK}$ ).

If TBPRD = 171, then PWM frequency = 175.44 kHz (period =  $(171 \cdot 2) \cdot T_{TBCLK}$ ).

**Solution:**

With 93 MEP steps per coarse step at 180 ps each:

**Step 1: Percentage Integer Period value conversion for TBPRD register**

Integer period value =  $342 \cdot T_{TBCLK}$   
=  $\text{int}(342.857) \cdot T_{TBCLK}$   
=  $\text{int}(\text{PWMperiod}) \cdot T_{TBCLK}$

In up-count mode:

TBPRD register value = 341 (TBPRD = period value - 1)  
= 0155h

In up-down count mode:

TBPRD register value = 171 (TBPRD = period value / 2)  
= 00ABh

**Step 2: Fractional value conversion for TBPRDHR register**

TBPRDHR register value =  $(\text{frac}(\text{PWMperiod}) \cdot \text{MEP\_ScaleFactor} + 0.5)$  (shift is to move the value as TBPRDHR high byte)

If auto-conversion enabled and HRMSTEP =

MEP\_ScaleFactor value (93): =  $\text{frac}(\text{PWMperiod}) \ll 8$

TBPRDHR register value =  $\text{frac}(342.857) \ll 8$

=  $0.857 \times 256$

= DB00h

The autoconversion will then automatically perform the calculation such that TBPRDHR MEP delay is scaled by hardware to:

=  $((\text{TBPRDHR}(15:0) \gg 8) \times \text{HRMSTEP} + 80h) \gg 8$

=  $(00DBh \times 93 + 80h) \gg 8$

= 500Fh  $\gg 8$

Period MEP delay

= 0050h MEP Steps

#### 4.2.3.4.1 High-Resolution Period Configuration

To use High Resolution Period, the ePWMx module must be initialized, following the steps in this exact order:

1. Enable ePWMx clock
2. Disable TBCLKSYNC
3. Configure ePWMx registers - AQ, TBPRD, CC, etc.
  - ePWMx may only be configured for up-count or up-down count modes. High-resolution period is not compatible with down-count mode.
  - TBCLK must equal SYSCLKOUT
  - TBPRD and CC registers must be configured for shadow loads.
  - CMPCTL[LOADAMODE]
    - In up-count mode: CMPCTL[LOADAMODE] = 1 (load on CTR = PRD)
    - In up-down count mode: CMPCTL[LOADAMODE] = 2 (load on CTR=0 or CTR=PRD)
4. Configure HRPWM register such that:
  - HRCNFG[HRLOAD] = 2 (load on either CTR = 0 or CTR = PRD)
  - HRCNFG[AUTOCONV] = 1 (Enable auto-conversion)
  - HRCNFG[EDGMODE] = 3 (MEP control on both edges)
5. For TBPHS: TBPHSHR synchronization with high-resolution period, set both HRPCTL[TBPSHRLOADE] = 1 and TBCTL[PHSEN] = 1. In up-down count mode these bits must be set to 1 regardless of the contents of TBPHSHR.
6. Enable high-resolution period control (HRPCTL[HRPE] = 1)
7. Enable TBCLKSYNC
8. TBCTL[SWFSYNC] = 1
9. HRMSTEP must contain an accurate MEP scale factor (# of MEP steps per SYSCLKOUT coarse step) because auto-conversion is enabled. The MEP scale factor can be acquired via the SFO() function described in [Section 4.3](#).
10. To control high-resolution period, write to the TBPRDHR(M) registers.

---

#### Note

When high-resolution period mode is enabled, an EPWMxSYNC pulse will introduce +/- 1 - 2 cycle jitter to the PWM (+/- 1 cycle in up-count mode and +/- 2 cycle in up-down count mode). For this reason, TBCTL[SYNCOSEL] should not be set to 1 (CTR = 0 is EPWMxSYNCO source) or 2 (CTR = CMPB is EPWMxSYNCO source). Otherwise, the jitter will occur on every PWM cycle with the synchronization pulse.

When TBCTL[SYNCOSEL] = 0 (EPWMxSYNCO is EPWMxSYNCO source), a software synchronization pulse should be issued only once during high-resolution period initialization. If a software sync pulse is applied while the PWM is running, the jitter will appear on the PWM output at the time of the sync pulse.

---

#### 4.2.4 Scale Factor Optimizing Software (SFO)

The micro edge positioner (MEP) logic is capable of placing an edge in one of 255 discrete time steps. As previously mentioned, the size of these steps is on the order of 150 ps (see device-specific data sheet for typical MEP step size on your device). The MEP step size varies based on worst-case process parameters, operating temperature, and voltage. MEP step size increases with decreasing voltage and increasing temperature and decreases with increasing voltage and decreasing temperature. Applications that use the HRPWM feature should use the TI-supplied MEP scale factor optimizer (SFO) software function. The SFO function helps to dynamically determine the number of MEP steps per SYSCLKOUT period while the HRPWM is in operation.

To utilize the MEP capabilities effectively during the Q15 duty (or period) to [CMPA:CMPAHR] or [TBPRD(M):TBPRDHR(M)] mapping function (see [Section 4.2.3.2](#)), the correct value for the MEP scaling factor (MEP\_ScaleFactor) needs to be known by the software. To accomplish this, the HRPWM module has built in self-check and diagnostics capabilities that can be used to determine the optimum MEP\_ScaleFactor value for any operating condition. TI provides a C-callable library containing one SFO function that utilizes this hardware and determines the optimum MEP\_ScaleFactor. As such, MEP Control and Diagnostics registers are reserved for TI use.

A detailed description of the SFO library - SFO\_TI\_Build\_V6.lib software can be found in [Section 4.3](#).

#### 4.2.5 HRPWM Examples Using Optimized Assembly Code

The best way to understand how to use the HRPWM capabilities is through two real examples:

1. Simple buck converter using asymmetrical PWM (that is, count-up) with active high polarity.
2. DAC function using simple R+C reconstruction filter.

The following examples all have Initialization/configuration code written in C. To make these easier to understand, the #defines shown below are used. Note that the #defines introduced in [Chapter 3](#) are also used.

[Example 4-1](#) assumes an MEP step size of 150 ps and does not use the SFO library.

##### **Example 4-1. #Defines for HRPWM Header Files**

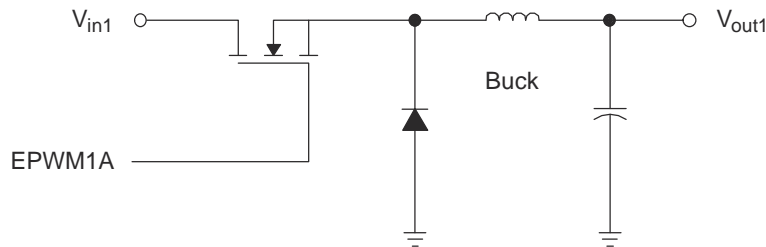
```
// HRPWM (High Resolution PWM) //
=====
// HRCNFG
#define HR_Disable 0x0
#define HR_REP 0x1           // Rising Edge position
#define HR_FEP 0x2           // Falling Edge position
#define HR_BEP 0x3           // Both Edge position #define HR_CMP 0x0 // CMPAHR controlled
#define HR_PHS 0x1           // TBPHSHR controlled #define HR_CTR_ZERO 0x0 // CTR = Zero event
#define HR_CTR_PRD 0x1        // CTR = Period event
#define HR_CTR_ZERO_PRD 0x2  // CTR = ZERO or Period event
#define HR_NORM_B 0x0         // Normal ePWMxB output
#define HR_INVERT_B 0x1       // ePWMxB is inverted ePWMxA output
```

### 4.2.5.1 Implementing a Simple Buck Converter

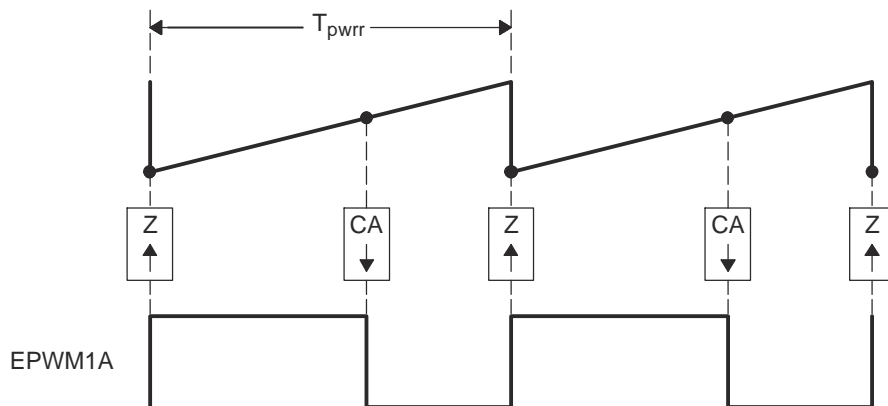
In this example, the PWM requirements for SYSCLKOUT = 60 MHz are:

- PWM frequency = 600 kHz (TBPRD = 100 )
- PWM mode = asymmetrical, up-count
- Resolution = 12.7 bits (with a MEP step size of 150 ps)

Figure 4-11 and Figure 4-12 show the required PWM waveform. As explained previously, configuration for the ePWM1 module is almost identical to the normal case except that the appropriate MEP options need to be enabled/selected.



**Figure 4-11. Simple Buck Controlled Converter Using a Single PWM**



**Figure 4-12. PWM Waveform Generated for Simple Buck Controlled Converter**

The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

**Example 4-2** shows the Initialization code. The first part is configured for conventional PWM. The second part sets up the HRPWM resources.

This example assumes an MEP step size of 150 ps and does not use the SFO library.

#### **Example 4-2. HRPWM Buck Converter Initialization Code**

```
void HrBuckDrvCnf(void)
{
    // Config for conventional PWM first
    EPwm1Regs.TBCTL.bit.PRDL = TB_IMMEDIATE;           // set Immediate load
    EPwm1Regs.TBPRD = 100;                             // Period set for 600 kHz PWM
    hrbuck_period = 200;                               // Used for Q15 to Q0 scaling
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;           // EPWM1 is the Master
    EPwm1Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    // Note: ChB is initialized here only for comparison purposes, it is not required
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;     // optional
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;     // optional
    EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
    EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;               // optional
    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;             // optional
    // Now configure the HRPWM resources
    EALLOW;                                           // Note these registers are protected
                                                    // and act only on ChA
    EPwm1Regs.HRCNFG.all = 0x0;                       // clear all bits first
    EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP;           // Control Falling Edge Position
    EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP;           // CMPAHR controls the MEP
    EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO;       // Shadow load on CTR=Zero
    EDIS;
    MEP_ScaleFactor = 111*256;                        // Start with typical Scale Factor
                                                    // value for 60 MHz
                                                    // Note: Use SFO functions to update
                                                    // MEP_ScaleFactor dynamically
}

```

**Example 4-3** shows an assembly example of run-time code for the HRPWM buck converter.

#### **Example 4-3. HRPWM Buck Converter Run-Time Code**

```
EPWM1_BASE .set 0x6800
CMPAHR1 .set EPWM1_BASE+0x8
;=====
HRBUCK_DRV; (can execute within an ISR or loop)
;=====
    MOVW DP, # HRBUCK_In
    MOVL XAR2,@_HRBUCK_In      ; Pointer to Input Q15 Duty (XAR2)
    MOVL XAR3,#CMPAHR1        ; Pointer to HRPWM CMPA reg (XAR3)
; Output for EPWM1A (HRPWM)
    MOV T,*XAR2 ; T <= Duty
    MPYU ACC,T,@_hrbuck_period ; Q15 to Q0 scaling based on Period
    MOV T,@_MEP_ScaleFactor    ; MEP scale factor (from optimizer s/w)
    MPYU P,T,@AL               ; P <= T * AL, Optimizer scaling
    MOVH @AL,P                 ; AL <= P, move result back to ACC
    ADD ACC, #0x080            ; MEP range and rounding adjustment
    MOVL *XAR3,ACC             ; CMPA: CMPAHR(31:8) <= ACC
; Output for EPWM1B (Regular Res) Optional - for comparison purpose only
    MOV **XAR3[2],AH           ; Store ACCH to regular CMPB

```



#### 4.2.5.2 Implementing a DAC Function Using an R+C Reconstruction Filter

In this example, the PWM requirements are:

- PWM frequency = 400 kHz (that is, TBPRD = 150)
- PWM mode = Asymmetrical, Up-count
- Resolution = 14 bits (MEP step size = 150 ps)

Figure 4-13 and Figure 4-14 show the DAC function and the required PWM waveform. As explained previously, configuration for the ePWM1 module is almost identical to the normal case except that the appropriate MEP options need to be enabled/selected.

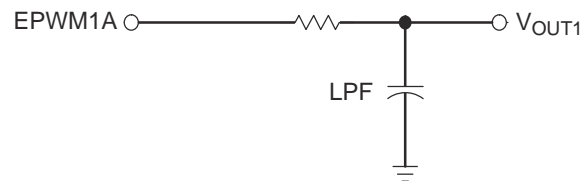


Figure 4-13. Simple Reconstruction Filter for a PWM Based DAC

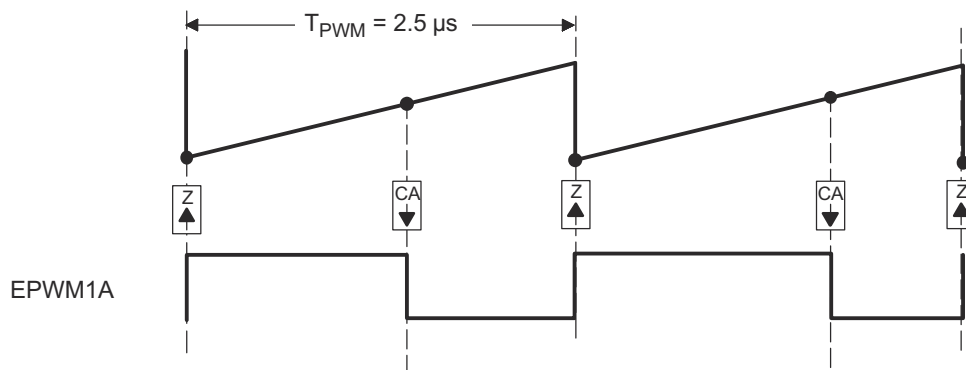


Figure 4-14. PWM Waveform Generated for the PWM DAC Function

The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

This example assumes a typical MEP\_ScaleFactor and does not use the SFO library.

**Example 4-4** shows the Initialization code. The first part is configured for conventional PWM. The second part sets up the HRPWM resources.

#### Example 4-4. PWM DAC Function Initialization Code

```
void HrPwmDacDrvCnf(void)
{
// Config for conventional PWM first
EPwm1Regs.TBCTL.bit.PRDL = TB_IMMEDIATE;           // Set Immediate load
EPwm1Regs.TBPRD = 150;                             // Period set for 400 kHz PWM
hrDAC_period = 150;                                 // Used for Q15 to Q0 scaling
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;            // EPWM1 is the Master
EPwm1Regs.TBCTL.bit.SYNCSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
// Note: ChB is initialized here only for comparison purposes, it is not required
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;     // optional
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;       // optional
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;                // optional
EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;              // optional
// Now configure the HRPWM resources
EALLOW;                                           // Note these registers are protected
                                                    // and act only on ChA.
EPwm1Regs.HRCNFG.all = 0x0; // Clear all bits first
EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP;           // Control falling edge position
EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP;          // CMPAHR controls the MEP.
EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO;      // Shadow load on CTR=Zero.
EDIS;
MEP_ScaleFactor = 111*256;                        // Start with typical Scale Factor
                                                    // value for 60 MHz.
                                                    // Use SFO functions to update MEP_ScaleFactor
                                                    // dynamically.
}

```

**Example 4-5** shows an assembly example of run-time code that can execute in a high-speed ISR loop.

#### Example 4-5. PWM DAC Function Run-Time Code

```
EPWM1_BASE .set 0x6800
CMPAHR1 .set EPWM1_BASE+0x8
;=====
HRPWM_DAC_DRV; (can execute within an ISR or loop)
;=====
    MOVW DP, # HRDAC_In
    MOVL XAR2,@_HRDAC_In           ; Pointer to input Q15 duty (XAR2)
    MOVL XAR3,#CMPAHR1            ; Pointer to HRPWM CMPA reg (XAR3)
; Output for EPWM1A (HRPWM
    MOV T,*XAR2                    ; T <= duty
    MPY ACC,T,@_hrDAC_period       ; Q15 to Q0 scaling based on period
    ADD ACC,@_HrDAC_period<<15    ; Offset for bipolar operation
    MOV T,@_MEP_ScaleFactor        ; MEP scale factor (from optimizer s/w)
    MPYU P,T,@AL                  ; P <= T * AL, optimizer scaling
    MOVH @AL,P                    ; AL <= P, move result back to ACC
    ADD ACC,#0x080                ; MEP range and rounding adjustment
    MOVL *XAR3,ACC                ; CMPA: CMPAHR(31:8) <= ACC
; Output for EPWM1B (Regular Res) Optional - for comparison purpose only
    MOV *+XAR3[2],AH              ; Store ACCH to regular CMPB

```

### 4.3 SFO Library Software - SFO\_TI\_Build\_V6.lib

Table 4-6 lists several features of the SFO\_TI\_Build\_V6.lib library.

**Table 4-6. SFO Library Features**

	SYSCLK Freq	SFO_TI_Build_V6.lib	Unit
Maximum HRPWM channels supported	-	8	channels
Total static variable memory size	-	11	words
Completion-checking?	-	Yes	-
Typical time required for SFO() to update MEP_ScaleFactor, if called repetitively without interrupts	80 MHz	1.3	milliseconds
	60 MHz	2.23	milliseconds

Following is a functional description of the SFO library routine, SFO().

#### 4.3.1 Scale Factor Optimizer Function - int SFO()

This routine drives the micro-edge positioner (MEP) calibration module to run SFO diagnostics and determine the appropriate MEP scale factor (number of MEP steps per coarse SYSCLKOUT step) for a device at any given time.

If SYSCLKOUT = TBCLK = 60 MHz and assuming the MEP step size is 150 ps, the typical scale factor value at 60 MHz = 111 MEP steps per TBCLK unit (16.67 ns)

The function returns a MEP scale factor value:

MEP\_ScaleFactor = Number of MEP steps/SYSCLKOUT.

#### Constraints when using this function:

- SFO() can be used with a minimum SYSCLKOUT = TBCLK = 50 MHz. MEP diagnostics logic uses SYSCLKOUT and not TBCLK, so the SYSCLKOUT restriction is an important constraint. Below 50 MHz, with device process variation, the MEP step size may decrease under cold temperature and high core voltage conditions to such a point, that 255 MEP steps will not span an entire SYSCLKOUT cycle.
- At any time, SFO() can be called to run SFO diagnostics on the MEP calibration module

#### Usage:

- SFO() can be called at any time in the background while the ePWM channels are running in HRPWM mode. The scale factor result obtained in MEP\_ScaleFactor can be applied to all ePWM channels running in HRPWM mode because the function makes use of the diagnostics logic in the MEP calibration module (which runs independently of ePWM channels).
- This routine returns a 1 when calibration is finished, and a new scale factor has been calculated or a 0 if calibration is still running. The routine returns a 2 if there is an error, and the MEP\_ScaleFactor is greater than the maximum 255 fine steps per coarse SYSCLKOUT cycle. In this case, the HRMSTEP register will maintain the last MEP scale factor value less than 256 for auto conversion.
- All ePWM modules operating in HRPWM incur only a 3-SYSCLKOUT cycle minimum duty cycle limitation when high-resolution period control is not used. If high-resolution period control is enabled, there is an additional duty cycle limitation 3-SYSCLKOUT cycles before the end of the PWM period (see [Section 4.2.3.3](#)).
- In SFO\_TI\_Build\_V6b.lib, the SFO() function also updates the HRMSTEP register with the scale factor result. If the HRCNFG[AUTOCONV] bit is set, the application software is responsible only for setting CMPAHR = fraction(PWMduty\*PWMperiod)<<8 or TBPRDHR = fraction (PWMperiod) while running SFO() in the background. The MEP Calibration Module will then use the values in the HRMSTEP and CMPAHR/TBPRDHR register to automatically calculate the appropriate number of MEP steps represented by the fractional duty cycle or period and move the high-resolution ePWM signal edge accordingly. In SFO\_TI\_Build\_V6.lib, the SFO() function does not automatically update the HRMSTEP register. Therefore, after the SFO function completes, the application software must write MEP\_ScaleFactor to the HRMSTEP register (EALLOW-protected).

- If the HRCNFG[AUTOCONV] bit is clear, the HRMSTEP register is ignored. The application software will need to perform the necessary calculations manually so that:
  - $CMPAHR = (\text{fraction}(\text{PWMduty} * \text{PWMperiod}) * \text{MEP Scale Factor}) \ll 8 + 0x080$ .
  - Similar behavior applies for TBPHSHR. Auto-conversion must be enabled when using TBPRDHR.

The routine can be run as a background tasks in a slow loop requiring negligible CPU cycles. The repetition rate at which an SFO function needs to be executed depends on the application's operating environment. As with all digital CMOS devices temperature and supply voltage variations have an effect on MEP operation. However, in most applications these parameters vary slowly and therefore it is often sufficient to execute the SFO function once every 5 to 10 seconds or so. If more rapid variations are expected, then execution may have to be performed more frequently to match the application. Note, there is no high limit restriction on the SFO function repetition rate, hence it can execute as quickly as the background loop is capable.

While using the HRPWM feature, HRPWM logic will not be active for the first 3 SYSCLKOUT cycles of the PWM period (and the last 3 SYSCLKOUT cycles of the PWM period if TBPRDHR is used). While running the application in this configuration, if high-resolution period control is disabled (HRPCTL[HRPE=0]) and the CMPAHR register value is less than 3 cycles, then its CMPAHR register must be cleared to zero. If high-resolution period control is enabled (HRPCTL[HRPE=1]), the CMPAHR register value must not fall below 3 or above TBPRD-3. This would avoid any unexpected transitions on the PWM signal.

### 4.3.2 Software Usage

The software library function SFO(), calculates the MEP scale factor for the HRPWM-supported ePWM modules. The scale factor is an integer value in the range 1-255, and represents the number of micro step edge positions available for a system clock period. The scale factor value is returned in an integer variable called MEP\_ScaleFactor. For example, see [Table 4-7](#).

**Table 4-7. Factor Values**

Software Function call	Functional Description	Updated Variables
SFO()	Returns MEP scale factor in MEP_ScaleFactor Returns MEP scale factor in the HRMSTEP register in SFO_TI_Build_V6b.lib	MEP_ScaleFactor and HRMSTEP register.

To use the HRPWM feature of the ePWMs it is recommended that the SFO function be used as described here.

#### Step 1. Add "Include" Files

The SFO\_V6.h file needs to be included as follows. This include file is mandatory while using the SFO library function. For the TMS320F2802x devices, the F2802x C/C++ Header Files and Peripheral Examples in controlSUITE DSP2802x\_Device.h and DSP2802x\_Epwm\_defines.h are necessary. For other device families, the device-specific equivalent files in the header files and peripheral examples software packages for those devices should be used. These include files are optional if customized header files are used in the end applications.

#### Example 4-6. A Sample of How to Add "Include" Files

```
#include "DSP2802x_Device.h"           // DSP2802x Headerfile
#include "DSP2802x_EPwm_defines.h"    // init defines
#include "SFO_V6.h"                   // SFO lib functions (needed for HRPWM)
```

## Step 2. Element Declaration

Declare an integer variable for the scale factor value as shown in [Example 4-7](#).

### Example 4-7. Declaring an Element

```
int MEP_ScaleFactor = 0; //scale factor value
volatile struct EPWM_REGS *ePWM[] = {0, &EPwm1Regs, &EPwm2Regs, &EPwm3Regs,
&EPwm4Regs};
```

## Step 3. MEP\_ScaleFactor Initialization

The SFO() function does not require a starting scale factor value in MEP\_ScaleFactor. Prior to using the MEP\_ScaleFactor variable in application code, SFO() should be called to drive the MEP calibration module to calculate an MEP\_ScaleFactor value.

As part of the one-time initialization code prior to using MEP\_ScaleFactor, include the following shown in [Example 4-8](#).

### Example 4-8. Initializing With a Scale Factor Value

```
MEP_ScaleFactor initialized using function SFO ()
while (SFO() == 0) {} // MEP_ScaleFactor calculated by MEP Cal Module
```

## Step 4. Application Code

While the application is running, fluctuations in both device temperature and supply voltage may be expected. To be sure that optimal Scale Factors are used for each ePWM module, the SFO function should be re-run periodically as part of a slower back-ground loop. Some examples of this are shown in [Example 4-9](#).

---

### Note

See the HRPWM\_SFO example in the device-specific C/C++ header files and peripheral examples available from the TI website.

---

### Example 4-9. SFO Function Calls

```
main ()
{
    int status;
    // User code
    // ePWM1, 2, 3, 4 are running in HRPWM mode
    // The status variable returns 1 once a new MEP_ScaleFactor has been
    // calculated by the MEP Calibration Module running SFO
    // diagnostics.
    status = SFO();
    if(status==2) {ESTOP0;} // The function returns a 2 if MEP_ScaleFactor is greater
    // than the maximum 255 allowed (error condition)
}
```

### 4.3.3 SFO Library Version Software Differences

There are two different versions of the SFO library - SFO\_TI\_Build\_V6.lib, and SFO\_TI\_Build\_V6b.lib. SFO\_TI\_Build\_V6.lib does not update the HRMSTEP register with the value in MEP\_ScaleFactor, while SFO\_TI\_Build\_V6b.lib updates the register. Therefore, if using SFO\_TI\_Build\_V6.lib and auto-conversion is enabled, the application should write MEP\_Scalefactor in the HRMSTEP register as shown in [Example 4-10](#).

**Example 4-10. Manually Updating the HRMSTEP Register if Using SFO\_TI\_Build\_V6b.lib**

```
main ()
{
    int status;

    status = SFO_INCOMPLETE;
    while (status==SFO_INCOMPLETE) {
        status = SFO();
    }
    if(status!=SFO_ERROR) { // IF SFO() is complete with no errors
        EALLOW;
        EPwm1Regs.HRMSTEP=MEP_ScaleFactor;
        EDIS;
    }
}
```

## 4.4 HRPWM Registers

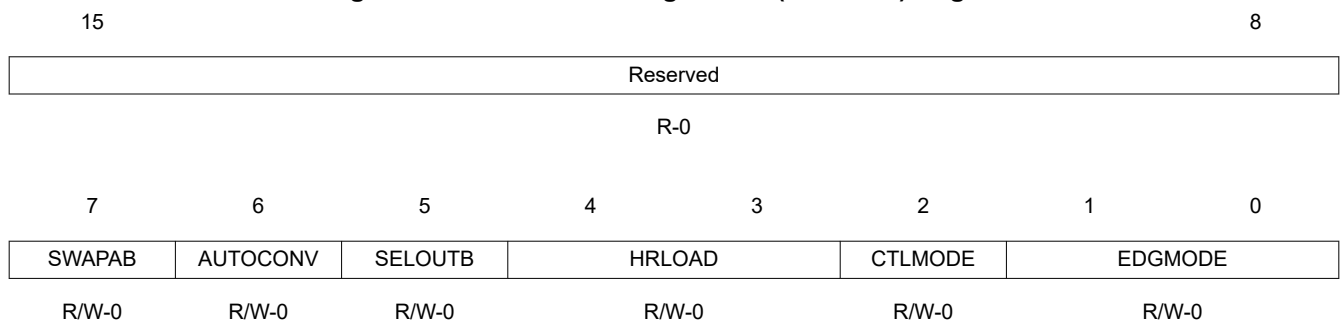
A summary of the registers required for the HRPWM is listed in [Table 4-8](#).

**Table 4-8. HRPWM Registers**

Name	Offset	Size(x16)	Shadow	Description	Bit Description
<b>Time-Base Registers</b>					
TBCTL	0x0000	1		Time-Base Control Register	<a href="#">Section 3.4.1.1</a>
TBSTS	0x0001	1		Time-Base Status Register	<a href="#">Section 3.4.1.2</a>
TBPHSHR	0x0002	1		Time-Base Phase High Resolution Register	<a href="#">Section 3.4.1.3</a>
TBPHS	0x0003	1		Time-Base Phase Register	<a href="#">Section 3.4.1.4</a>
TBCTR	0x0004	1		Time-Base Counter Register	<a href="#">Section 3.4.1.5</a>
TBPRD	0x0005	1	Yes	Time-Base Period Register Set	<a href="#">Section 3.4.1.6</a>
TBPRDHR	0x0006	1	Yes	Time-Base Period High Resolution Register Set	<a href="#">Section 3.4.1.7</a>
<b>Counter-Compare Registers</b>					
CMPCTL	0x0007	1		Counter-Compare Control Register	<a href="#">Section 3.4.2.1</a>
CMPAHR	0x0008	1	Yes	Counter-Compare A High Resolution Register	<a href="#">Section 3.4.2.2</a>
CMPA	0x0009	1	Yes	Counter-Compare A Register	<a href="#">Section 3.4.2.3</a>
CMPB	0x000A	1	Yes	Counter-Compare B Register	<a href="#">Section 3.4.2.4</a>
<b>HRPWM Registers</b>					
HRCNFG	0x0020	1		HRPWM Configuration Register	<a href="#">Section 4.4.1</a>
HRPWR	0x0021	1		HRPWM Power Register	<a href="#">Section 4.4.2</a>
HRMSTEP	0x0026	1		HRPWM MEP Step Register	<a href="#">Section 4.4.3</a>
<b>High Resolution Period and Mirror Registers</b>					
HRPCTL	0x0028	1		High Resolution Period Control Register	<a href="#">Section 4.4.4</a>
TBPRDHRM	0x002A	1	Writes	Time-Base Period High Resolution Mirror Register	<a href="#">Section 3.4.1.8</a>
TBPRDM	0x002B	1	Writes	Time-Base Period Mirror Register	<a href="#">Section 3.4.1.9</a>
CMPAHRM	0x002C	1	Writes	Counter-Compare A High Resolution Mirror Register	<a href="#">Section 3.4.2.5</a>
CMPAM	0x002D	1	Writes	Counter-Compare A Mirror Register	<a href="#">Section 3.4.2.6</a>

### 4.4.1 HRPWM Configuration (HRCNFG) Register

**Figure 4-15. HRPWM Configuration (HRCNFG) Register**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 4-9. HRPWM Configuration (HRCNFG) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-8	Reserved		Reserved
7	SWAPAB	0 1	Swap ePWM A and B Output Signals This bit enables the swapping of the A and B signal outputs. The selection is as follows: 0 ePWMxA and ePWMxB outputs are unchanged. 1 ePWMxA signal appears on ePWMxB output and ePWMxB signal appears on ePWMxA output.
6	AUTOCONV	0 1	Auto Convert Delay Line Value Selects whether the fractional duty cycle/period/phase in the CMPAHR/TBPRDHR/TBPHSHR register is automatically scaled by the MEP scale factor in the HRMSTEP register or manually scaled by calculations in application software. The SFO library function automatically updates the HRMSTEP register with the appropriate MEP scale factor. 0 Automatic HRMSTEP scaling is disabled. 1 Automatic HRMSTEP scaling is enabled. If application software is manually scaling the fractional duty cycle, or phase (that is, software sets $CMPAHR = (\text{fraction}(PWMduty * PWMperiod) * MEP \text{ Scale Factor}) \ll 8 + 0x080$ for duty cycle), then this mode must be disabled.
5	SELOUTB	0 1	EPWMxB Output Select Bit This bit selects which signal is output on the ePWMxB channel output. 0 ePWMxB output is normal. 1 ePWMxB output is inverted version of ePWMxA signal.
4-3	HRLOAD	00 01 10 11	Shadow Mode Bit Selects the time event that loads the CMPAHR shadow value into the active register. 00 Load on CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000) 01 Load on CTR = PRD: Time-base counter equal to period (TBCTR = TBPRD) 10 Load on either CTR = Zero or CTR = PRD 11 Reserved
2	CTLMODE	0 1	Control Mode Bits Selects the register (CMP/TBPRD or TBPHS) that controls the MEP: 0 CMPAHR(8) or TBPRDHR(8) Register controls the edge position (that is, this is duty or period control mode). (Default on Reset) 1 TBPHSHR(8) Register controls the edge position (that is, this is phase control mode).
1-0	EDGMODE	00 01 10 11	Edge Mode Bits Selects the edge of the PWM that is controlled by the micro-edge position (MEP) logic: 00 HRPWM capability is disabled (default on reset) 01 MEP control of rising edge (CMPAHR) 10 MEP control of falling edge (CMPAHR) 11 MEP control of both edges (TBPHSHR or TBPRDHR)

(1) This register is EALLOW protected.



#### 4.4.2 High Resolution Power (HRPWR) Register

**Figure 4-16. High Resolution Power (HRPWR) Register**

15	10      9	6      5	0
Reserved	MEPOFF	Reserved	
R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 4-10. High Resolution Power (HRPWR) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-10	Reserved		Reserved for TI Test
9-6	MEPOFF	0-Fh	MEP Calibration OFF bits When not using MEP calibration, setting these bits to all 1's disables MEP calibration logic in the HRPWM and reduces power consumption.
5-0	Reserved		Reserved for TI Test

(1) This register is EALLOW protected.

#### 4.4.3 High Resolution Micro Step (HRMSTEP) Register

**Figure 4-17. High Resolution Micro Step (HRMSTEP) Register**

15	8      7	0
Reserved	HRMSTEP	
R-0	R/W-0	

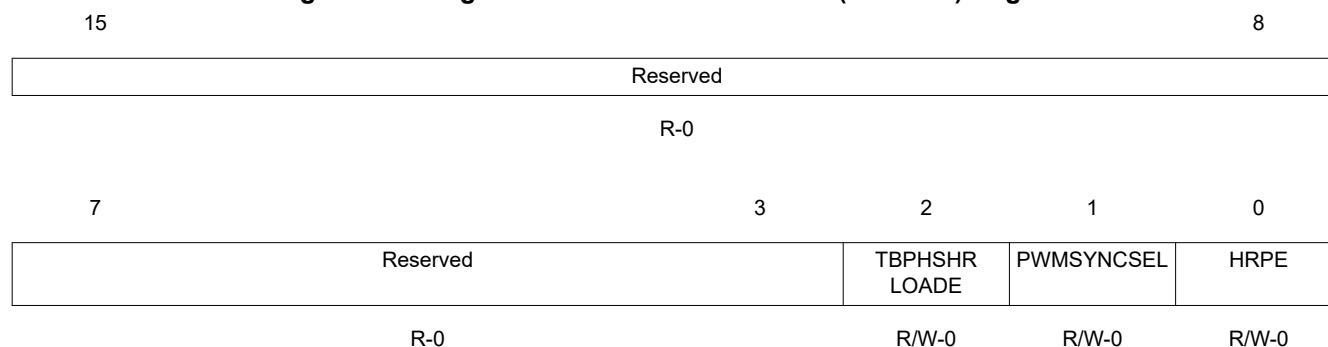
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 4-11. High Resolution Micro Step (HRMSTEP) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-8	Reserved		Reserved
7-0	HRMSTEP	00-FFh	High Resolution MEP Step When auto-conversion is enabled (HRCNFG[AUTOCONV] = 1), This 8-bit field contains the MEP_ScaleFactor (number of MEP steps per coarse steps) used by the hardware to automatically convert the value in the CMPAHR, TBPHSHR, or TBPRDHR register to a scaled micro-edge delay on the high-resolution ePWM output. The value in this register is written by the SFO calibration software at the end of each calibration run.

(1) This register is EALLOW protected.

#### 4.4.4 High Resolution Period Control (HRPCTL) Register

**Figure 4-18. High Resolution Period Control (HRPCTL) Register**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 4-12. High Resolution Period Control (HRPCTL) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup> <sup>(2)</sup>
15-3	Reserved		Reserved
2	TBPHSHRLOADE	0 Disables synchronization of high-resolution phase on a SYNCIN, TBCTL[SWFSYNC] or digital compare event: 1 Synchronize the high-resolution phase on a SYNCIN, TBCTL[SWFSYNC] or digital comparator synchronization event. The phase is synchronized using the contents of the high-resolution phase TBPHSHR register.  The TBCTL[PHSEN] bit which enables the loading of the TBCTR register with TBPHS register value on a SYNCIN or TBCTL[SWFSYNC] event works independently. However, users need to enable this bit also if they want to control phase in conjunction with the high-resolution period feature.  This bit and the TBCTL[PHSEN] bit must be set to 1 when high-resolution period is enabled for up-down count mode even if TBPHSHR = 0x0000. This bit does not need to be set when only high-resolution duty is enabled.	
1	PWMSYNCSEL	0 PWMSYNC is generated by TBCTR = PRD pulse. 1 PWMSYNC is generated by TBCTR = 0 pulse.	
0	HRPE	0 High resolution period feature disabled. In this mode the ePWM behaves as a Type 0 ePWM. 1 High resolution period enabled. In this mode the HRPWM module can control high-resolution of both the duty and frequency.  When high-resolution period is enabled, TBCTL[CTRMODE] = 0,1 (down-count mode) is not supported.	

(1) This register is EALLOW protected.

(2) This register is used with Type 1 ePWM modules (support high-resolution period) only.

This chapter describes the enhanced capture (eCAP) module, which is used in systems where accurate timing of external events is important.

The enhanced capture (eCAP) module is a Type 0 eCAP. See the [C2000 Real-Time Control Peripheral Reference Guide](#) for a list of all devices with an eCAP module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

<b>5.1 Introduction</b> .....	<b>372</b>
<b>5.2 Description</b> .....	<b>372</b>
<b>5.3 Capture and APWM Operating Mode</b> .....	<b>373</b>
<b>5.4 Capture Mode Description</b> .....	<b>375</b>
<b>5.5 Application of the eCAP Module</b> .....	<b>383</b>
<b>5.6 Application of the APWM Mode</b> .....	<b>387</b>
<b>5.7 eCAP Registers</b> .....	<b>387</b>

## 5.1 Introduction

### 5.1.1 Features

The features of the eCAP module include:

- Speed measurements of rotating machinery (for example, toothed sprockets sensed by way of Hall sensors)
- Elapsed time measurements between position sensor pulses
- Period and duty cycle measurements of pulse train signals
- Decoding current or voltage amplitude derived from duty cycle encoded current/voltage sensors

The eCAP module features described in this chapter include:

- 4-event time-stamp registers (each 32 bits)
- Edge polarity selection for up to four sequenced time-stamp capture events
- Interrupt on either of the four events
- Single-shot capture of up to four event time-stamps
- Continuous mode capture of time stamps in a four-deep circular buffer
- Absolute time-stamp capture
- Difference (Delta) mode time-stamp capture
- All above resources are dedicated to a single input pin
- When not used in capture mode, the eCAP module can be configured as a single-channel PWM output

### 5.1.2 ECAP Related Collateral

#### Getting Started Materials

- [Leveraging High Resolution Capture \(HRCAP\) for Single Wire Data Transfer Application Report](#)

## 5.2 Description

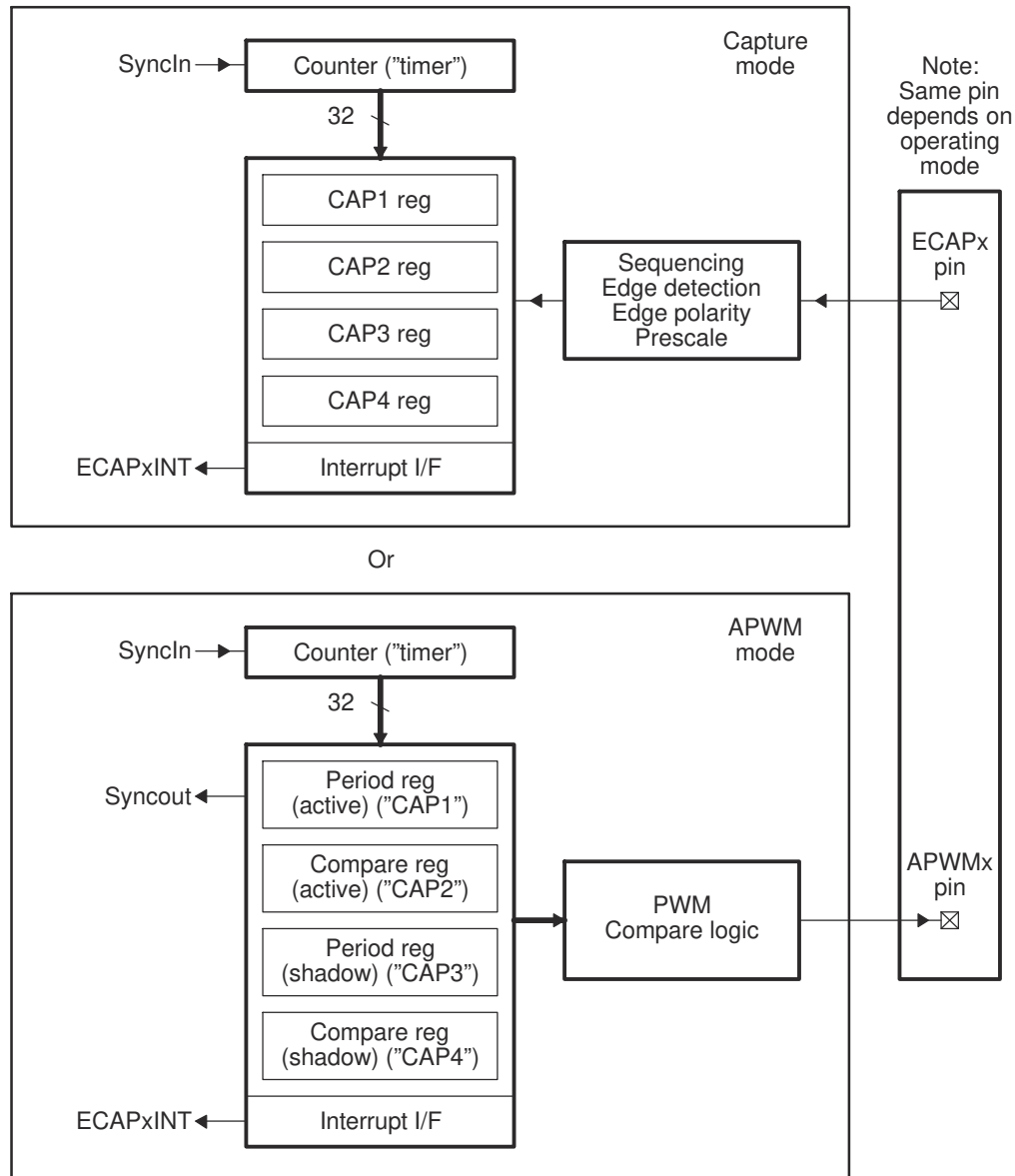
The eCAP module represents one complete capture channel that can be instantiated multiple times, depending on the target device. In the context of this guide, one eCAP channel has the following independent key resources:

- Dedicated input capture pin
- 32-bit time base (counter)
- 4 x 32-bit time-stamp capture registers (CAP1-CAP4)
- Four-stage sequencer (modulo4 counter) that is synchronized to external events, eCAP pin rising/falling edges.
- Independent edge polarity (rising/falling edge) selection for all four events
- Input capture signal prescaling (from 2-62 or bypass)
- One-shot compare register (two bits) to freeze captures after 1-4 time-stamp events
- Control for continuous time-stamp captures using a four-deep circular buffer (CAP1-CAP4) scheme
- Interrupt capabilities on any of the four capture events

### 5.3 Capture and APWM Operating Mode

You can use the eCAP module resources to implement a single-channel PWM generator (with 32-bit capabilities) when it is not being used for input captures. The counter operates in count-up mode, providing a time-base for asymmetrical pulse width modulation (PWM) waveforms. The CAP1 and CAP2 registers become the active period and compare registers, respectively, while CAP3 and CAP4 registers become the period and compare shadow registers, respectively. Figure 5-1 is a high-level view of both the capture and auxiliary pulse-width modulator (APWM) modes of operation.

Figure 5-2 further describes the output of the eCAP in APWM mode based on the CMP and PRD values.



- A. A single pin is shared between CAP and APWM functions. In capture mode, it is an input; in APWM mode, it is an output.
- B. In APWM mode, writing any value to CAP1/CAP2 active registers also writes the same value to the corresponding shadow registers CAP3/CAP4. This emulates immediate mode. Writing to the shadow registers CAP3/CAP4 invokes the shadow mode.

**Figure 5-1. Capture and APWM Modes of Operation**

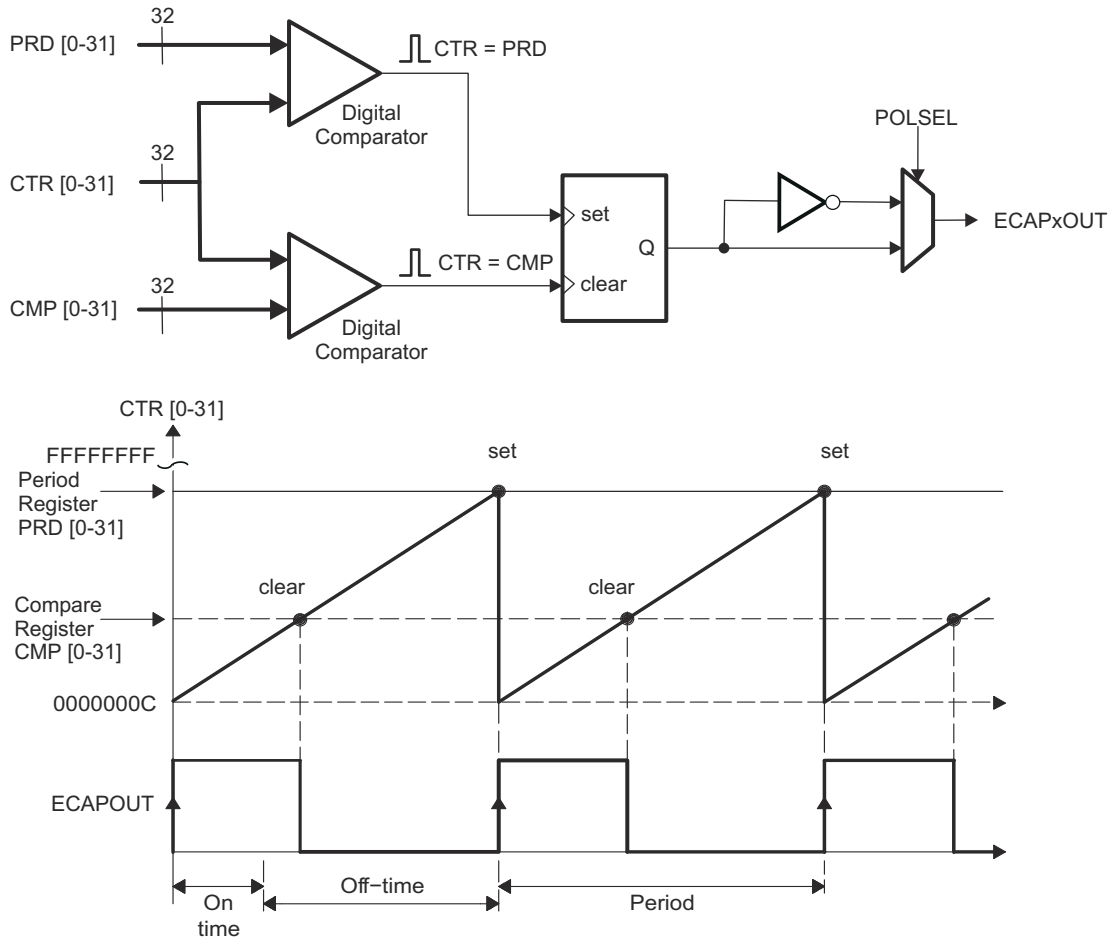


Figure 5-2. Counter Compare and PRD Effects on the eCAP Output in APWM Mode

### 5.4 Capture Mode Description

Figure 5-3 shows the various components that implement the capture function.

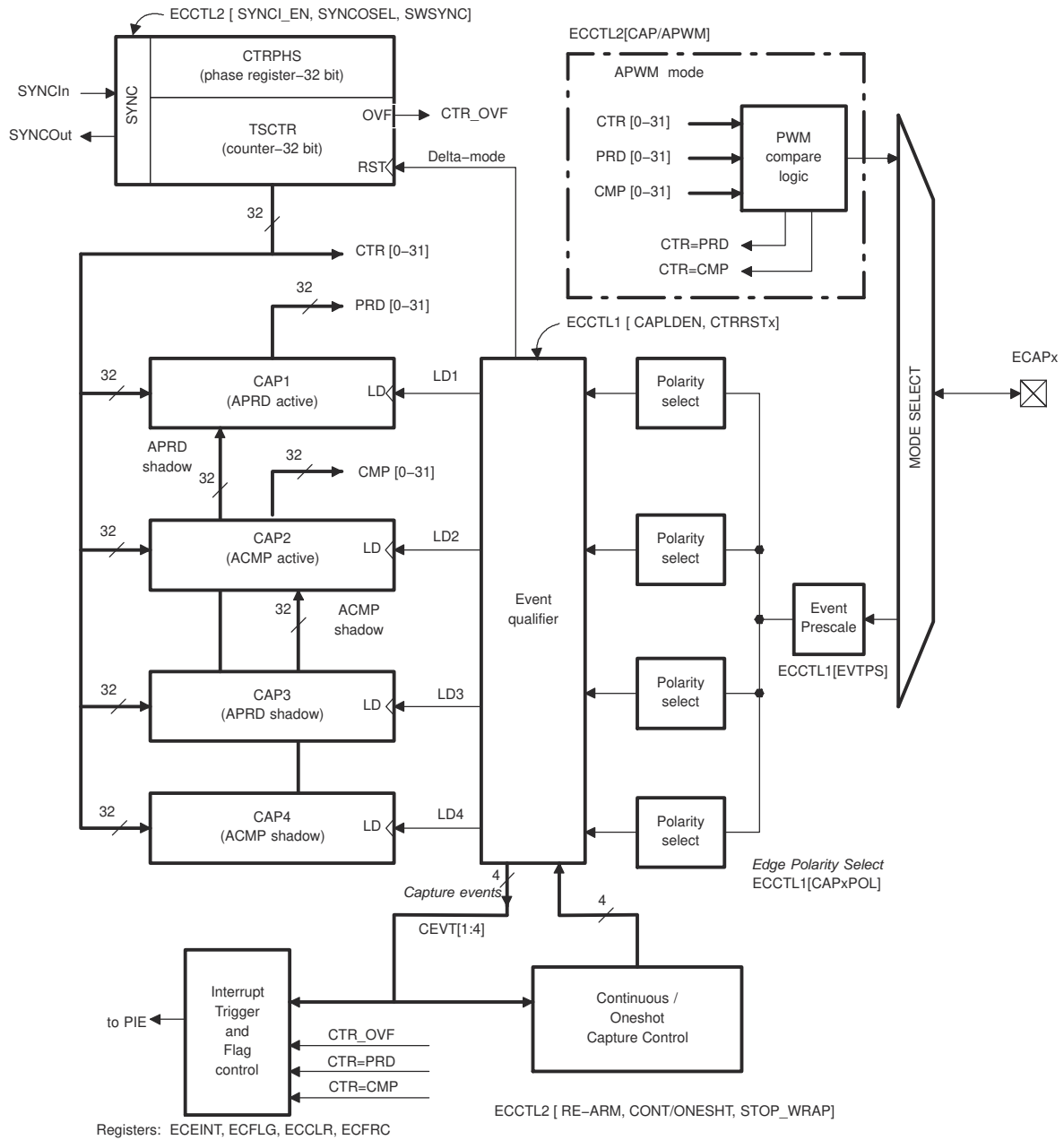
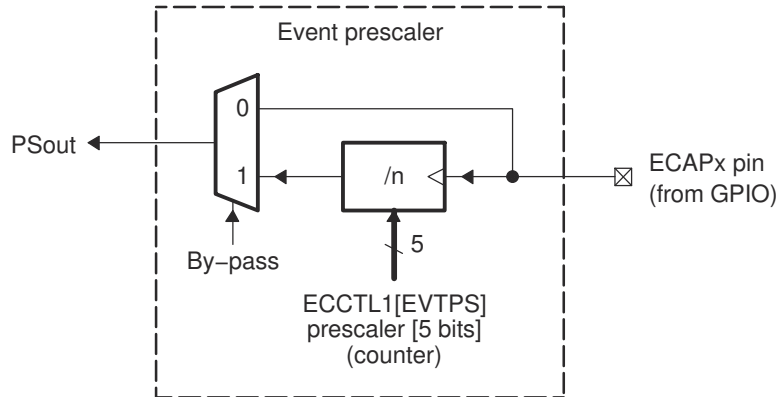


Figure 5-3. eCAP Block Diagram

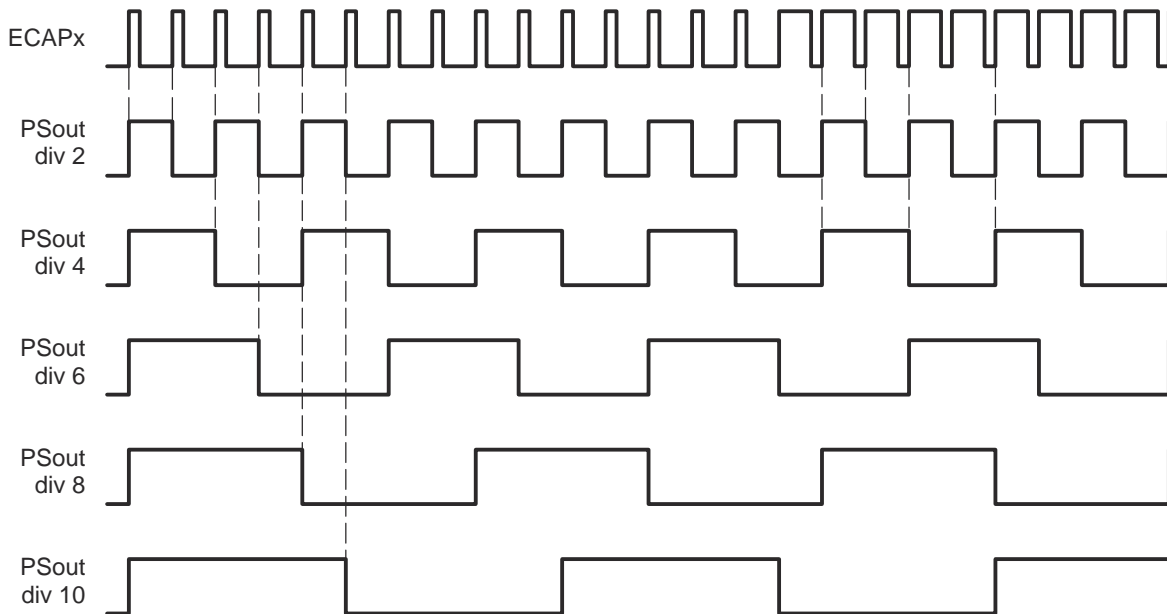
### 5.4.1 Event Prescaler

An input capture signal (pulse train) can be prescaled by  $N = 2-62$  (in multiples of 2) or can bypass the prescaler. This is useful when very high frequency signals are used as inputs. Figure 5-4 shows a functional diagram and Figure 5-5 shows the operation of the prescale function.



- A. When a prescale value of 1 is chosen (ECCTL1[13:9] = 0,0,0,0,0), the input capture signal bypasses the prescale logic completely.
- B. The first Rise edge after Prescale configuration change is not passed to Capture logic, prescaler value takes into effect on the second rising edge after the configuration.

**Figure 5-4. Event Prescale Control**



**Figure 5-5. Prescale Function Waveforms**



### 5.4.2 Edge Polarity Select and Qualifier

Functionality and features include:

- Four independent edge polarity (rising edge/falling edge) selection muxes are used, one for each capture event.
- Each edge (up to 4) is event qualified by the Modulo4 sequencer.
- The edge event is gated to its respective CAPx register by the Mod4 counter. The CAPx register is loaded on the falling edge.

### 5.4.3 Continuous/One-Shot Control

Operation of eCAP in Continuous/One-Shot mode:

- The Mod4 (2-bit) counter is incremented via edge qualified events (CEVT1-CEVT4).
- The Mod4 counter continues counting (0->1->2->3->0) and wraps around unless stopped.
- During one-shot operation, a 2-bit stop register (STOP\_WRAP) is used to compare the Mod4 counter output, and when equal, stops the Mod4 counter and inhibits further loads of the CAP1-CAP4 registers. In this mode if TSCCTR counter is configured to reset on capture event (CEVTx) by configuring ECCTL1.CTRRSTx bit, it will still keep resetting the TSCCTR counter on capture event (CEVTx) after the STOP\_WRAP value is reached and re-arm (REARM) has not occurred.

The continuous/one-shot block controls the start, stop and reset (zero) functions of the Mod4 counter, via a mono-shot type of action that can be triggered by the stop-value comparator and re-armed via software control.

Once armed, the eCAP module waits for 1-4 (defined by stop-value) capture events before freezing both the Mod4 counter and contents of CAP1-4 registers (time stamps).

Re-arming prepares the eCAP module for another capture sequence. Also, re-arming clears (to zero) the Mod4 counter and permits loading of CAP1-4 registers again, providing the CAPLDEN bit is set.

In continuous mode, the Mod4 counter continues to run (0->1->2->3->0, the one-shot action is ignored, and capture values continue to be written to CAP1-4 in a circular buffer sequence.

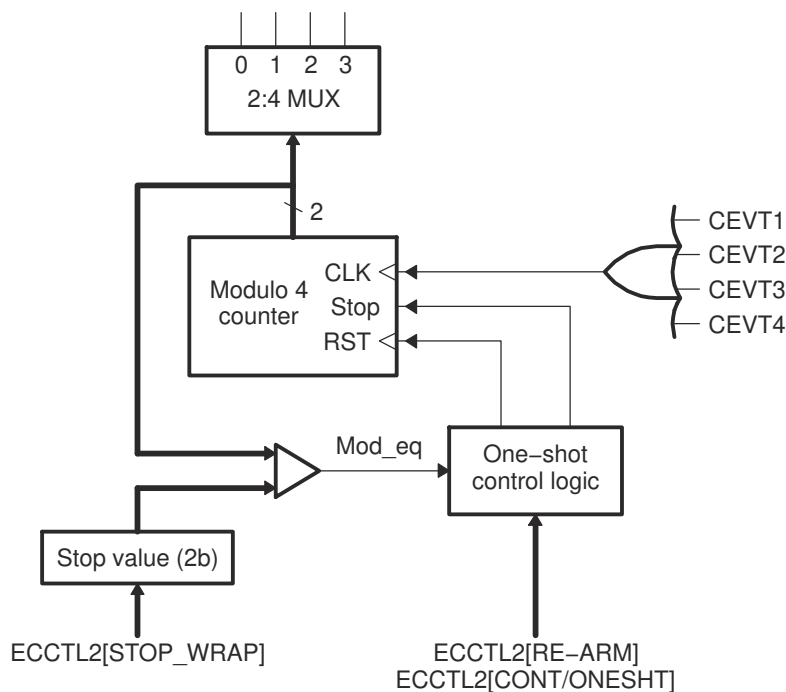


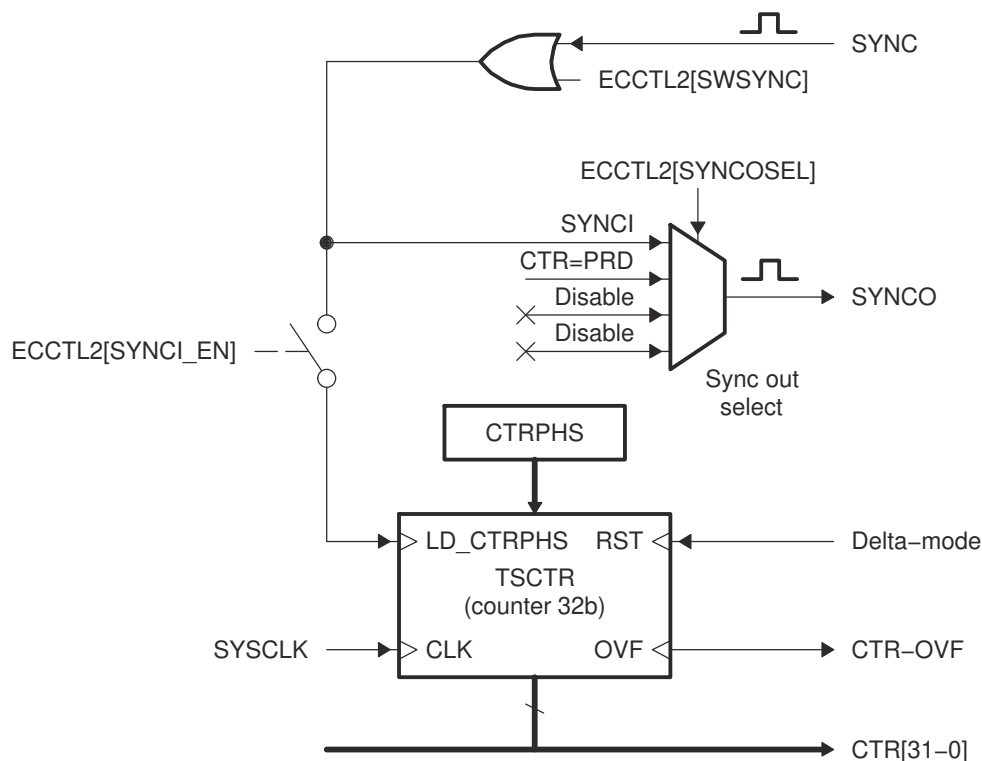
Figure 5-6. Details of the Continuous/One-shot Block

### 5.4.4 32-Bit Counter and Phase Control

This counter provides the time-base for event captures, and is clocked via the system clock.

A phase register is provided to achieve synchronization with other counters, via a hardware and software forced sync. This is useful in APWM mode when a phase offset between modules is needed.

On any of the four event loads, an option to reset the 32-bit counter is given. This is useful for time difference capture. The 32-bit counter value is captured first, then it is reset to 0 by any of the LD1-LD4 signals.



**Figure 5-7. Details of the Counter and Synchronization Block**

### 5.4.5 CAP1-CAP4 Registers

These 32-bit registers are fed by the 32-bit counter timer bus, CTR[0-31] and are loaded (capture a time-stamp) when their respective LD inputs are strobed.

Control bit CAPLDEN can inhibit loading of the capture registers. During one-shot operation, this bit is cleared (loading is inhibited) automatically when a stop condition occurs, StopValue = Mod4.

CAP1 and CAP2 registers become the active period and compare registers, respectively, in APWM mode.

CAP3 and CAP4 registers become the respective shadow registers (APRD and ACMP) for CAP1 and CAP2 during APWM operation.



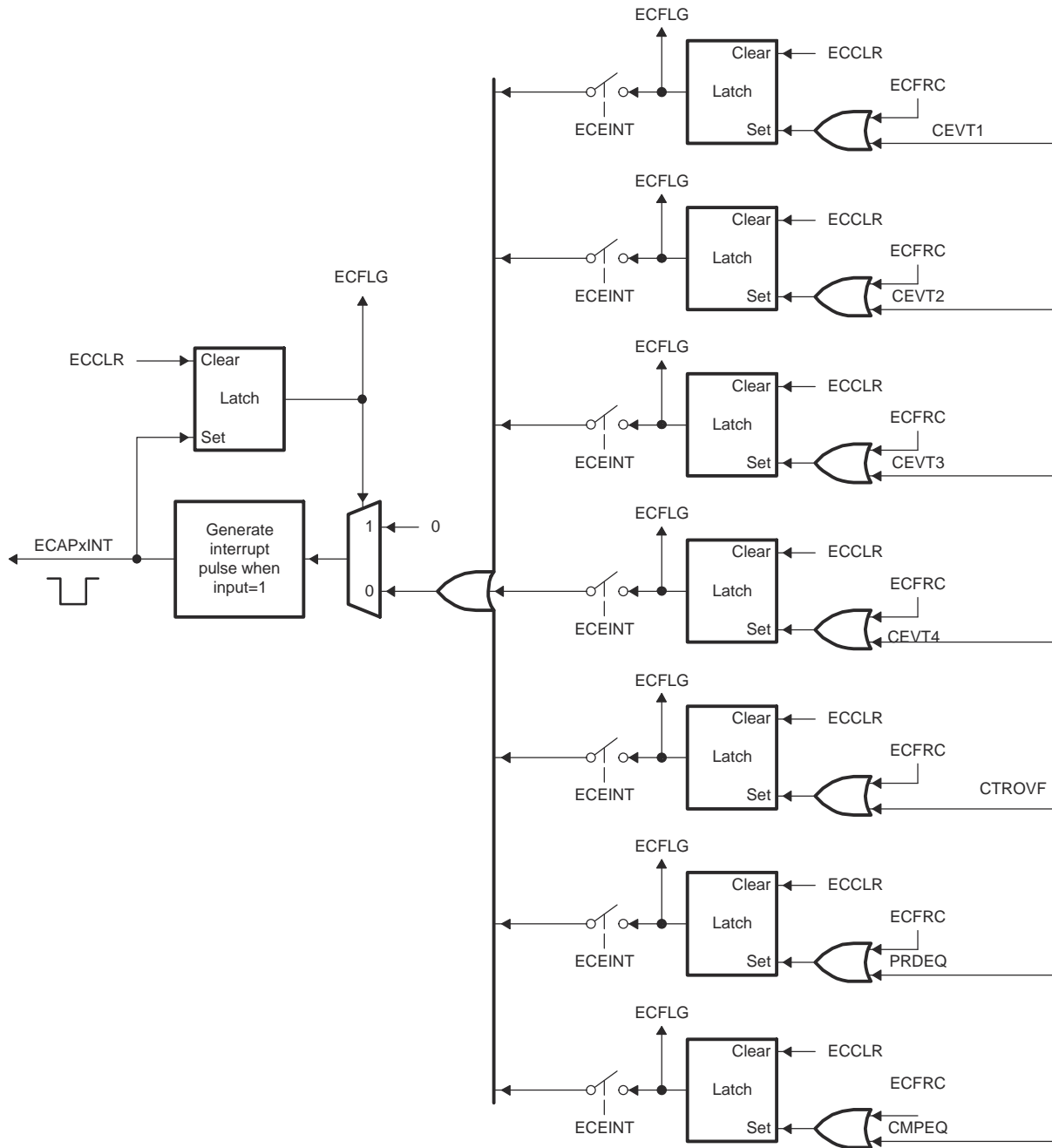


Figure 5-8. Interrupts in eCAP Module

### 5.4.7 Shadow Load and Lockout Control

In capture mode, this logic inhibits (locks out) any shadow loading of CAP1 or CAP2 from APRD and ACMP registers, respectively.

In APWM mode, shadow loading is active and two choices are permitted:

- Immediate - APRD or ACMP are transferred to CAP1 or CAP2 immediately upon writing a new value.
- On period equal,  $CTR[31:0] = PRD[31:0]$ .

### 5.4.8 APWM Mode Operation

Main operating highlights of the APWM section:

- The time-stamp counter bus is made available for comparison by way of 2 digital (32-bit) comparators.
- When CAP1/2 registers are not used in capture mode, their contents can be used as Period and Compare values in APWM mode.
- Double buffering is achieved via shadow registers APRD and ACMP (CAP3/4). The shadow register contents are transferred over to CAP1/2 registers, either immediately upon a write, or on a  $CTR = PRD$  trigger.
- In APWM mode, writing to CAP1/CAP2 active registers will also write the same value to the corresponding shadow registers CAP3/CAP4. This emulates immediate mode. Writing to the shadow registers CAP3/CAP4 will invoke the shadow mode.
- During initialization, you must write to the active registers for both period and compare. This automatically copies the initial values into the shadow values. For subsequent compare updates, during run-time, you only need to use the shadow registers.

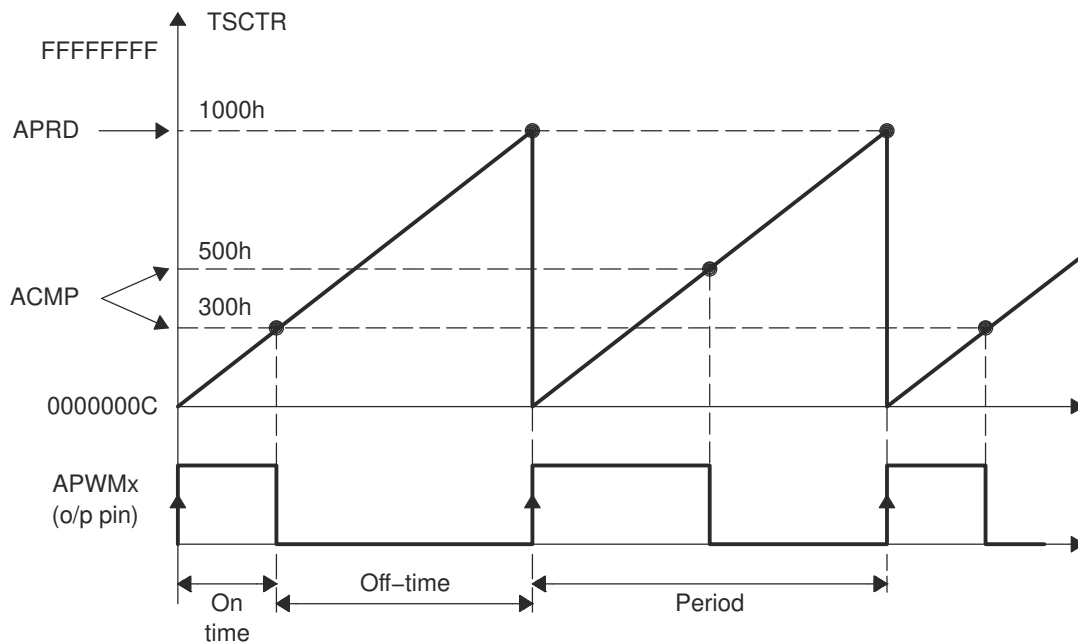


Figure 5-9. PWM Waveform Details Of APWM Mode Operation

The behavior of APWM active high mode (APWMPOL == 0) is as follows:

CMP = 0x00000000, output low for duration of period (0% duty)

CMP = 0x00000001, output high 1 cycle

CMP = 0x00000002, output high 2 cycles

CMP = PERIOD, output high except for 1 cycle (<100% duty)

CMP = PERIOD+1, output high for complete period (100% duty)

CMP > PERIOD+1, output high for complete period

The behavior of APWM active low mode (APWMPOL == 1) is as follows:

CMP = 0x00000000, output high for duration of period (0% duty)

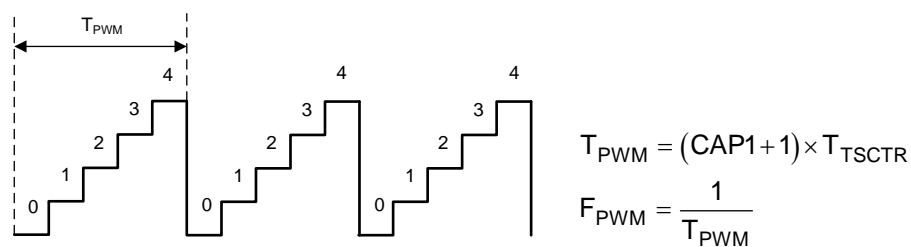
CMP = 0x00000001, output low 1 cycle

CMP = 0x00000002, output low 2 cycles

CMP = PERIOD, output low except for 1 cycle (<100% duty)

CMP = PERIOD+1, output low for complete period (100% duty)

CMP > PERIOD+1, output low for complete period



**Figure 5-10. Time-Base Frequency and Period Calculation**

## 5.5 Application of the eCAP Module

The following sections will provide applications examples to show how to operate the eCAP module.

### 5.5.1 Example 1 - Absolute Time-Stamp Operation Rising Edge Trigger

Figure 5-11 shows an example of continuous capture operation (Mod4 counter wraps around). In this figure, TSCTR counts-up without resetting and capture events are qualified on the rising edge only, this gives period (and frequency) information.

On an event, the TSCTR contents (time-stamp) is first captured, then Mod4 counter is incremented to the next state. When the TSCTR reaches FFFFFFFF (maximum value), it wraps around to 00000000 (not shown in Figure 5-11), if this occurs, the CTROVF (counter overflow) flag is set, and an interrupt (if enabled) occurs. CTROVF (counter overflow) Flag is set, and an Interrupt (if enabled) occurs. Captured Time-stamps are valid at the point indicated by the diagram (after the 4th event), hence event CEVT4 can conveniently be used to trigger an interrupt and the CPU can read data from the CAPx registers.

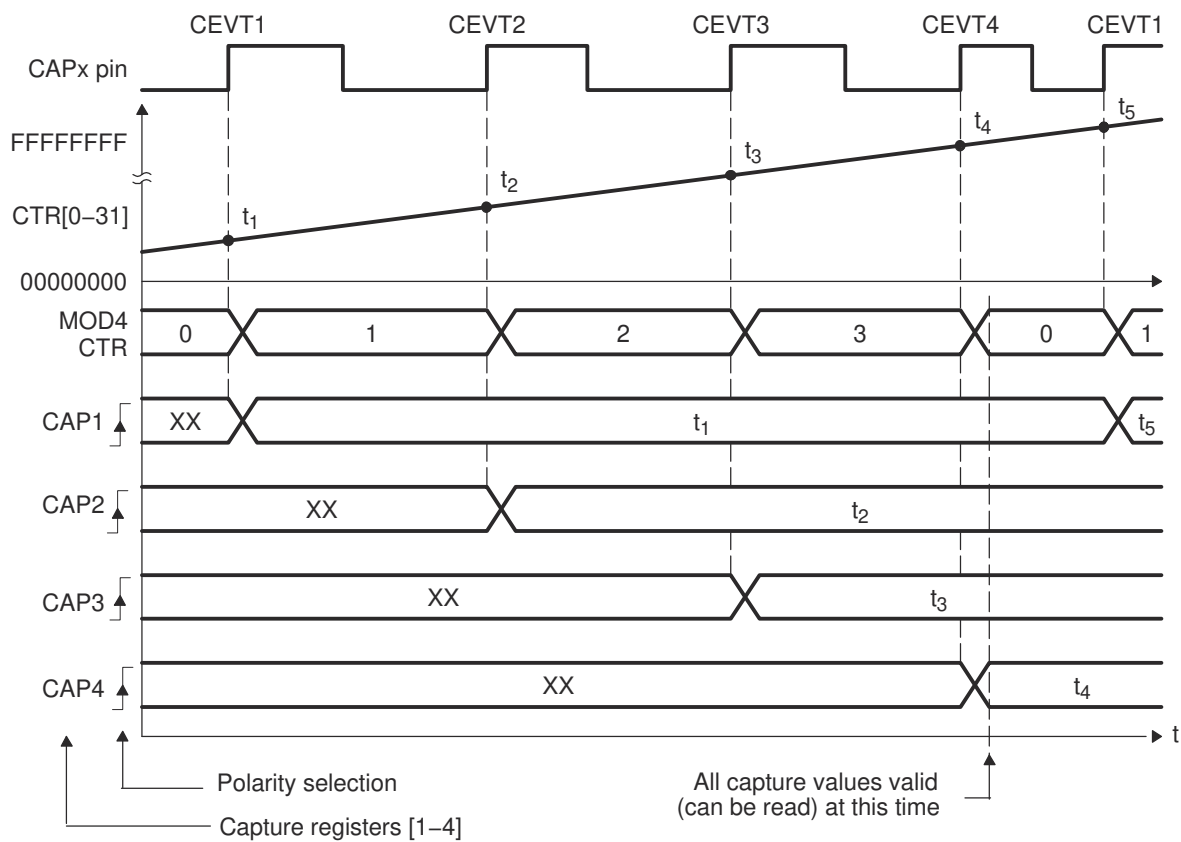
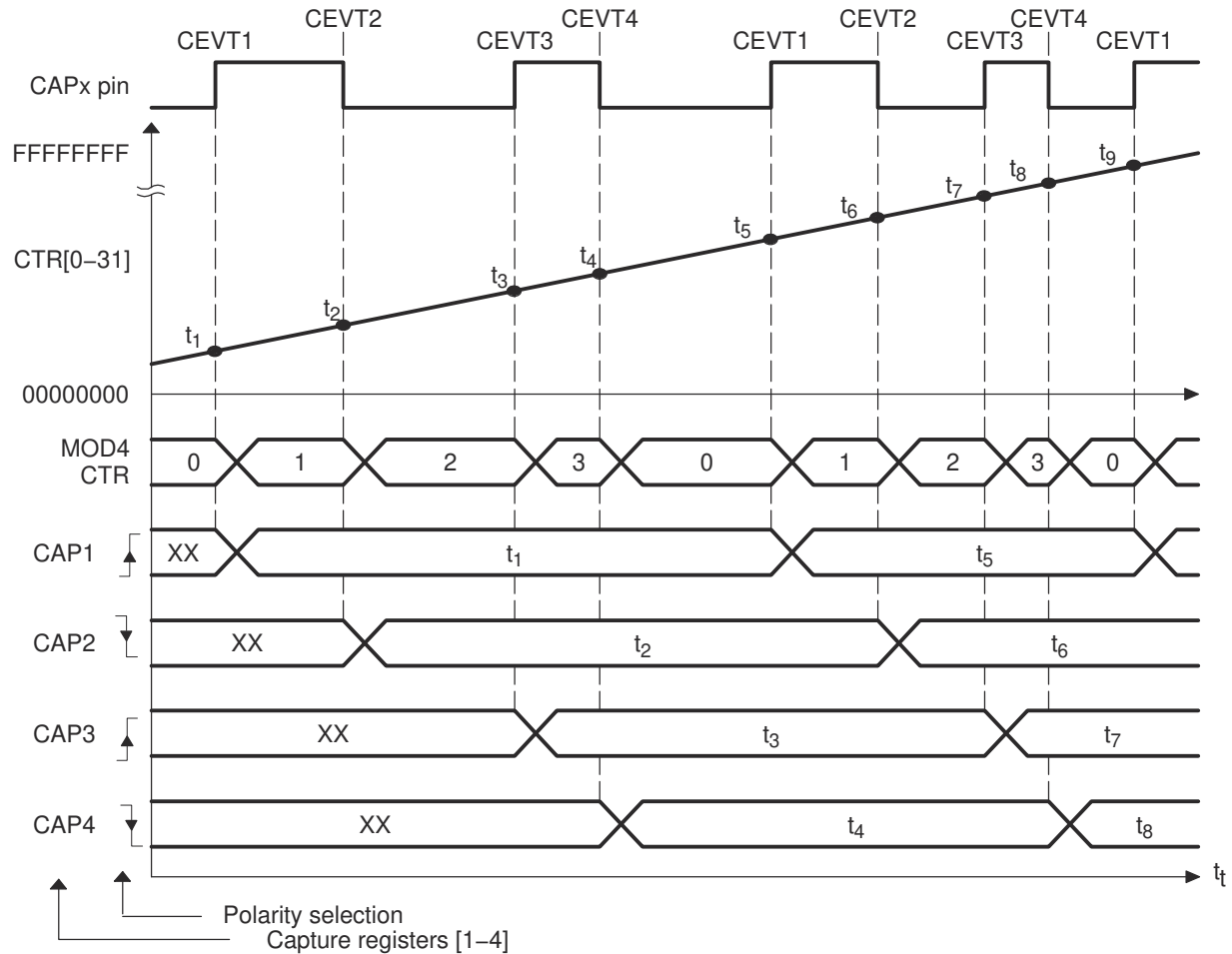


Figure 5-11. Capture Sequence for Absolute Time-stamp and Rising Edge Detect

### 5.5.2 Example 2 - Absolute Time-Stamp Operation Rising and Falling Edge Trigger

In Figure 5-12, the eCAP operating mode is almost the same as in the previous section except capture events are qualified as either rising or falling edge, this now gives both period and duty cycle information, that is: Period1 =  $t_3 - t_1$ , Period2 =  $t_5 - t_3$ , ...and so on. Duty Cycle1 (on-time %) =  $(t_2 - t_1) / \text{Period1} \times 100\%$ , and so on. Duty Cycle1 (off-time %) =  $(t_3 - t_2) / \text{Period1} \times 100\%$ , and so on.



**Figure 5-12. Capture Sequence for Absolute Time-stamp With Rising and Falling Edge Detect**



### 5.5.3 Example 3 - Time Difference (Delta) Operation Rising Edge Trigger

Figure 5-13 shows how the eCAP module can be used to collect Delta timing data from pulse train waveforms. Here Continuous Capture mode (TSCTR counts-up without resetting, and Mod4 counter wraps around) is used. In Delta-time mode, TSCTR is Reset back to Zero on every valid event. Here Capture events are qualified as Rising edge only. On an event, TSCTR contents (Time-Stamp) is captured first, and then TSCTR is reset to Zero. The Mod4 counter then increments to the next state. If TSCTR reaches FFFFFFFF (Max value), before the next event, it wraps around to 00000000 and continues, a CANTOVF (counter overflow) Flag is set, and an Interrupt (if enabled) occurs. The advantage of Delta-time Mode is that the CAPx contents directly give timing data without the need for CPU calculations, that is,  $Period1 = T_1$ ,  $Period2 = T_2$ , and so on. As shown in Figure 5-13, the CEVT1 event is a good trigger point to read the timing data,  $T_1, T_2, T_3, T_4$  are all valid here.

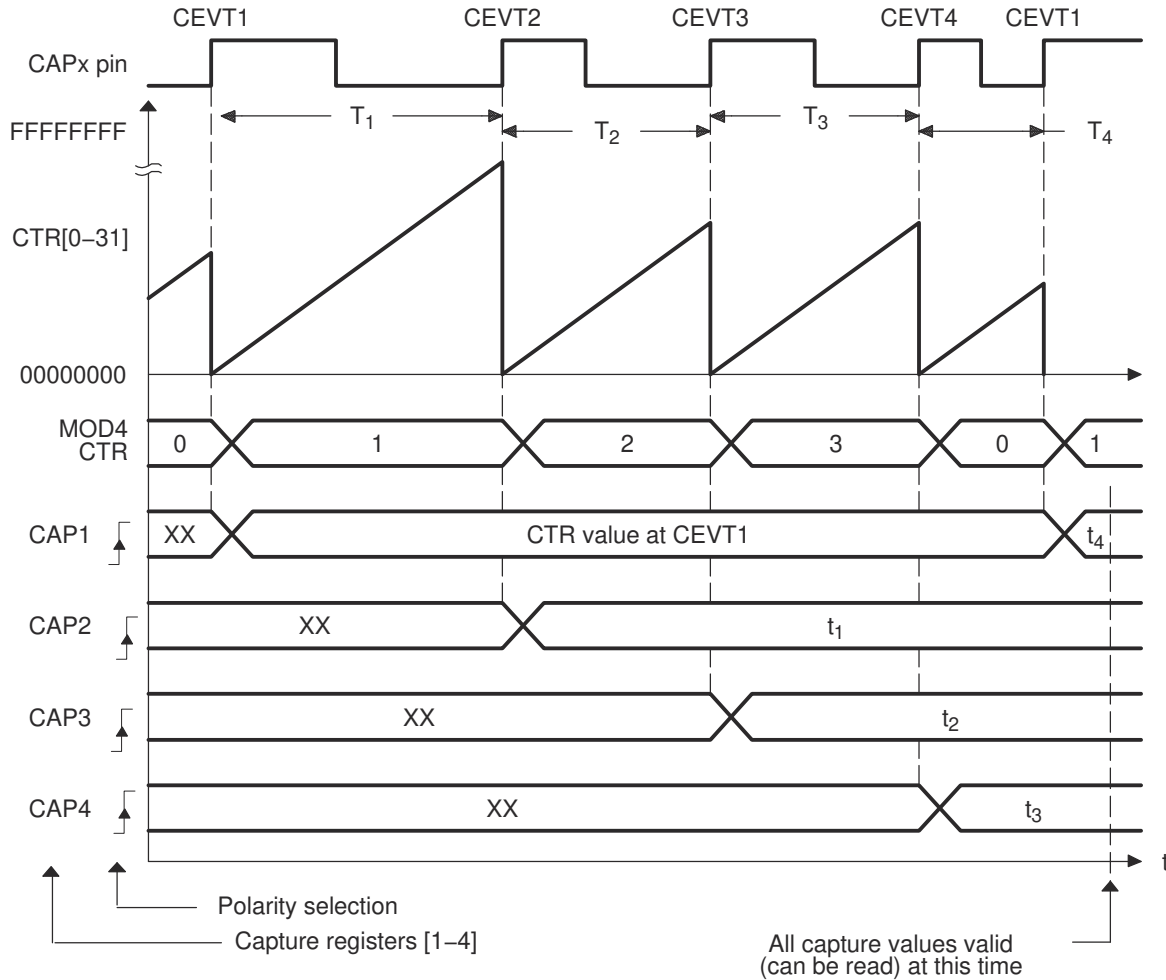
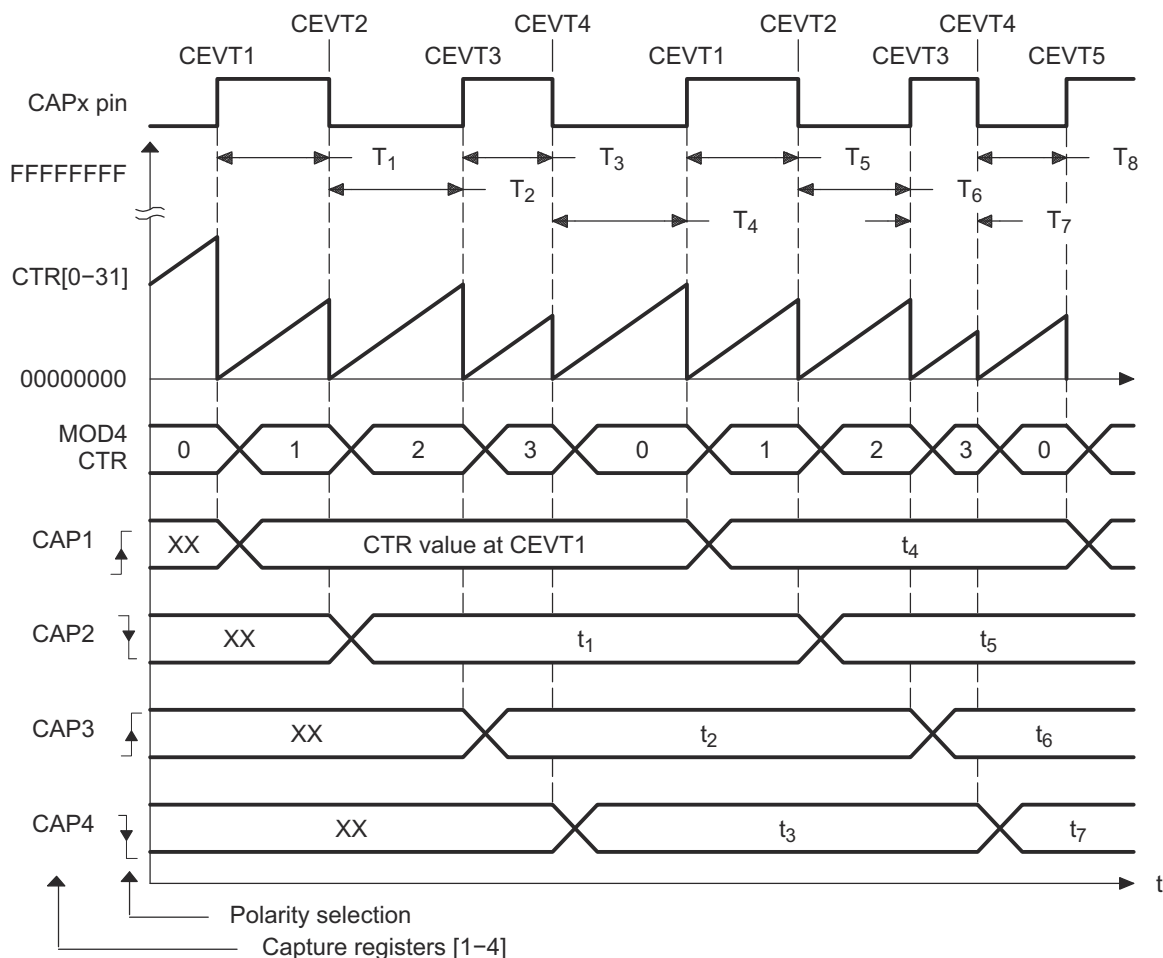


Figure 5-13. Capture Sequence for Delta Mode Time-stamp and Rising Edge Detect

### 5.5.4 Example 4 - Time Difference (Delta) Operation Rising and Falling Edge Trigger

In Figure 5-14, the eCAP operating mode is almost the same as in previous section except Capture events are qualified as either Rising or Falling edge, this now gives both Period and Duty cycle information, that is:  $\text{Period1} = T_1 + T_2$ ,  $\text{Period2} = T_3 + T_4$ , and so on.  $\text{Duty Cycle1 (on-time \%)} = T_1 / \text{Period1} \times 100\%$ ,  $\text{Duty Cycle1 (off-time \%)} = T_2 / \text{Period1} \times 100\%$ , and so on.

During initialization, you must write to the active registers for both period and compare. This action will automatically copy the init values into the shadow values. For subsequent compare updates during run-time, the shadow registers must be used.



**Figure 5-14. Capture Sequence for Delta Mode Time-stamp With Rising and Falling Edge Detect**

## 5.6 Application of the APWM Mode

In this example, the eCAP module is configured to operate as a PWM generator. Here, a very simple single-channel PWM waveform is generated from the APWMx output pin. The PWM polarity is active high, which means that the compare value (CAP2 reg is now a compare register) represents the on-time (high level) of the period. Alternatively, if the APWMPOL bit is configured for active low, then the compare value represents the off-time.

### 5.6.1 Example 1 - Simple PWM Generation (Independent Channel/s)

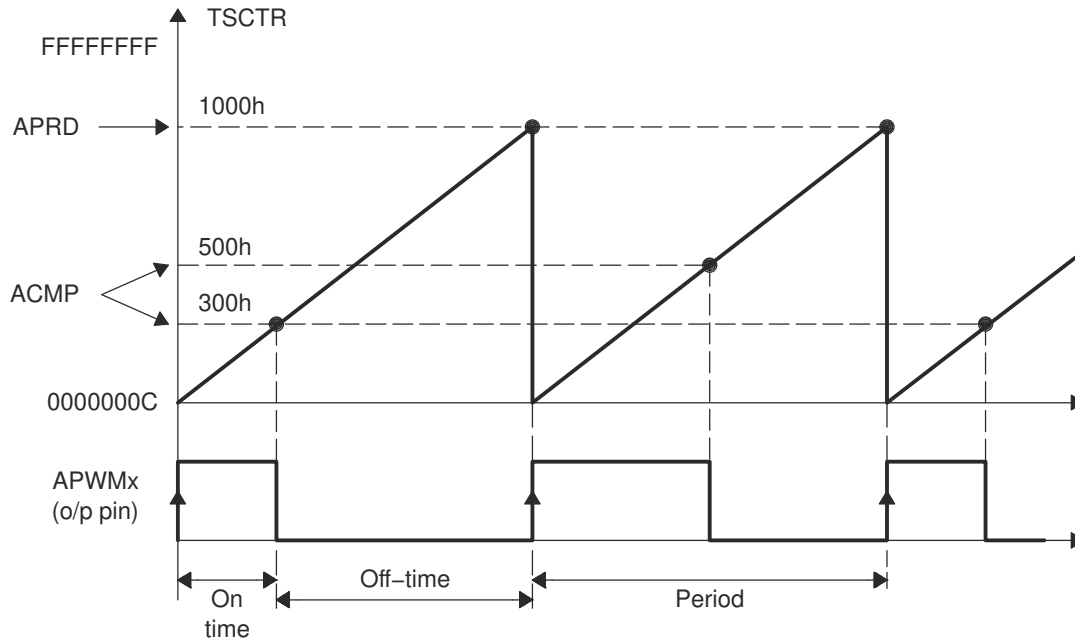


Figure 5-15. PWM Waveform Details of APWM Mode Operation

## 5.7 eCAP Registers

This section describes the Enhanced Capture Registers.

### 5.7.1 eCAP Base Addresses

Table 5-1. eCAP Base Address Table (C28)

Bit Field Name		Base Address
Instance	Structure	
ECap1Regs	ECAP_REGS	0x0000_6A00

## 5.7.2 ECAP\_REGS Registers

Table 5-2 lists the ECAP\_REGS registers. All register offset addresses not listed in Table 5-2 should be considered as reserved locations and the register contents should not be modified.

**Table 5-2. ECAP\_REGS Registers**

Offset	Acronym	Register Name	Write Protection	Section
0h	TSCTR	Time-Stamp Counter		<a href="#">Section 5.7.2.1</a>
2h	CTRPHS	Counter Phase Offset Value Register		<a href="#">Section 5.7.2.2</a>
4h	CAP1	Capture 1 Register		<a href="#">Section 5.7.2.3</a>
6h	CAP2	Capture 2 Register		<a href="#">Section 5.7.2.4</a>
8h	CAP3	Capture 3 Register		<a href="#">Section 5.7.2.5</a>
Ah	CAP4	Capture 4 Register		<a href="#">Section 5.7.2.6</a>
14h	ECCTL1	Capture Control Register 1		<a href="#">Section 5.7.2.7</a>
15h	ECCTL2	Capture Control Register 2		<a href="#">Section 5.7.2.8</a>
16h	ECEINT	Capture Interrupt Enable Register		<a href="#">Section 5.7.2.9</a>
17h	ECFLG	Capture Interrupt Flag Register		<a href="#">Section 5.7.2.10</a>
18h	ECCLR	Capture Interrupt Clear Register		<a href="#">Section 5.7.2.11</a>
19h	ECFRC	Capture Interrupt Force Register		<a href="#">Section 5.7.2.12</a>

Complex bit access types are encoded to fit into small table cells. Table 5-3 shows the codes that are used for access types in this section.

**Table 5-3. ECAP\_REGS Access Type Codes**

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R-0	Read Returns 0s
Write Type		
W	W	Write
W1C	W1C	Write 1 to clear
W1S	W1S	Write 1 to set
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

### 5.7.2.1 TSCTR Register (Offset = 0h) [reset = 0h]

Time-Stamp Counter

**Figure 5-16. TSCTR Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSCTR																															
R/W-0h																															

**Table 5-4. TSCTR Register Field Descriptions**

Bit	Field	Type	Reset	Description
31-0	TSCTR	R/W	0h	Active 32-bit counter register that is used as the capture time-base Reset type: SYSRSn

### 5.7.2.2 CTRPHS Register (Offset = 2h) [reset = 0h]

Counter Phase Offset Value Register

**Figure 5-17. CTRPHS Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTRPHS																															
R/W-0h																															

**Table 5-5. CTRPHS Register Field Descriptions**

Bit	Field	Type	Reset	Description
31-0	CTRPHS	R/W	0h	Counter phase value register that can be programmed for phase lag/lead. This register CTRPHS is loaded into TSCTR upon either a SYNCI event or S/W force via a control bit. Used to achieve phase control synchronization with respect to other eCAP and EPWM timebases. Reset type: SYSRSn

### 5.7.2.3 CAP1 Register (Offset = 4h) [reset = 0h]

Capture 1 Register

**Figure 5-18. CAP1 Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP1																															
R/W-0h																															

**Table 5-6. CAP1 Register Field Descriptions**

Bit	Field	Type	Reset	Description
31-0	CAP1	R/W	0h	This register can be loaded (written) by: <ul style="list-style-type: none"> <li>- Time-Stamp counter value (TSCTR) during a capture event</li> <li>- Software - may be useful for test purposes or initialization</li> <li>- ARPD shadow register (CAP3) when used in APWM mode</li> </ul> Reset type: SYSRSn

### 5.7.2.4 CAP2 Register (Offset = 6h) [reset = 0h]

Capture 2 Register

**Figure 5-19. CAP2 Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP2																															
R/W-0h																															

**Table 5-7. CAP2 Register Field Descriptions**

Bit	Field	Type	Reset	Description
31-0	CAP2	R/W	0h	This register can be loaded (written) by: <ul style="list-style-type: none"> <li>- Time-Stamp ( counter value) during a capture event</li> <li>- Software - may be useful for test purposes</li> <li>- ACMP shadow register (CAP4) when used in APWM mode</li> </ul> Reset type: SYSRSn

### 5.7.2.5 CAP3 Register (Offset = 8h) [reset = 0h]

Capture 3 Register

**Figure 5-20. CAP3 Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP3																															
R/W-0h																															

**Table 5-8. CAP3 Register Field Descriptions**

Bit	Field	Type	Reset	Description
31-0	CAP3	R/W	0h	In CMP mode, this is a time-stamp capture register. In APWM mode, this is the period shadow (APRD) register. You can update the PWM period value through this register. CAP3 (APRD) shadows CAP1 in this mode. Reset type: SYSRSn

### 5.7.2.6 CAP4 Register (Offset = Ah) [reset = 0h]

Capture 4 Register

**Figure 5-21. CAP4 Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP4																															
R/W-0h																															

**Table 5-9. CAP4 Register Field Descriptions**

Bit	Field	Type	Reset	Description
31-0	CAP4	R/W	0h	In CMP mode, this is a time-stamp capture register. In APWM mode, this is the compare shadow (ACMP) register. You can update the PWM compare value via this register. CAP4 (ACMP) shadows CAP2 in this mode. Reset type: SYSRSn

### 5.7.2.7 ECCTL1 Register (Offset = 14h) [reset = 0h]

Capture Control Register 1

**Figure 5-22. ECCTL1 Register**

15		14		13		12		11		10		9		8	
FREE_SOFT				PRESCALE								CAPLDEN			
R/W-0h				R/W-0h								R/W-0h			
7		6		5		4		3		2		1		0	
CTRRST4		CAP4POL		CTRRST3		CAP3POL		CTRRST2		CAP2POL		CTRRST1		CAP1POL	
R/W-0h		R/W-0h		R/W-0h		R/W-0h		R/W-0h		R/W-0h		R/W-0h		R/W-0h	

**Table 5-10. ECCTL1 Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-14	FREE_SOFT	R/W	0h	Emulation Control Reset type: SYSRSn 0h (R/W) = TSCTR counter stops immediately on emulation suspend 1h (R/W) = TSCTR counter runs until = 0 2h (R/W) = TSCTR counter is unaffected by emulation suspend (Run Free) 3h (R/W) = TSCTR counter is unaffected by emulation suspend (Run Free)
13-9	PRESCALE	R/W	0h	Event Filter prescale select Reset type: SYSRSn 0h (R/W) = Divide by 1 (that is, no prescale, by-pass the prescaler) 1h (R/W) = Divide by 2 2h (R/W) = Divide by 4 3h (R/W) = Divide by 6 4h (R/W) = Divide by 8 5h (R/W) = Divide by 10 1Eh (R/W) = Divide by 60 1Fh (R/W) = Divide by 62
8	CAPLDEN	R/W	0h	Enable Loading of CAP1-4 registers on a capture event. Note that this bit does not disable CEVTn events from being generated. Reset type: SYSRSn 0h (R/W) = Disable CAP1-4 register loads at capture event time. 1h (R/W) = Enable CAP1-4 register loads at capture event time.
7	CTRRST4	R/W	0h	Counter Reset on Capture Event 4 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 4 (absolute time stamp operation) 1h (R/W) = Reset counter after Capture Event 4 time-stamp has been captured (used in difference mode operation)
6	CAP4POL	R/W	0h	Capture Event 4 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 4 triggered on a rising edge (RE) 1h (R/W) = Capture Event 4 triggered on a falling edge (FE)
5	CTRRST3	R/W	0h	Counter Reset on Capture Event 3 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 3 (absolute time stamp) 1h (R/W) = Reset counter after Event 3 time-stamp has been captured (used in difference mode operation)

**Table 5-10. ECCTL1 Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
4	CAP3POL	R/W	0h	Capture Event 3 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 3 triggered on a rising edge (RE) 1h (R/W) = Capture Event 3 triggered on a falling edge (FE)
3	CTRRST2	R/W	0h	Counter Reset on Capture Event 2 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 2 (absolute time stamp) 1h (R/W) = Reset counter after Event 2 time-stamp has been captured (used in difference mode operation)
2	CAP2POL	R/W	0h	Capture Event 2 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 2 triggered on a rising edge (RE) 1h (R/W) = Capture Event 2 triggered on a falling edge (FE)
1	CTRRST1	R/W	0h	Counter Reset on Capture Event 1 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 1 (absolute time stamp) 1h (R/W) = Reset counter after Event 1 time-stamp has been captured (used in difference mode operation)
0	CAP1POL	R/W	0h	Capture Event 1 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 1 triggered on a rising edge (RE) 1h (R/W) = Capture Event 1 triggered on a falling edge (FE)



### 5.7.2.8 ECCTL2 Register (Offset = 15h) [reset = 6h]

Capture Control Register 2

**Figure 5-23. ECCTL2 Register**

15	14	13	12	11	10	9	8
RESERVED					APWMPOL	CAP_APWM	SWSYNC
R-0h					R/W-0h	R/W-0h	R-0/W1S-0h
7	6	5	4	3	2	1	0
SYNCO_SEL	SYNCL_EN	TSCTRSTOP	REARM	STOP_WRAP		CONT_ONESHT	
R/W-0h		R/W-0h	R/W-0h	R-0/W1S-0h	R/W-3h		R/W-0h

**Table 5-11. ECCTL2 Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-11	RESERVED	R	0h	Reserved
10	APWMPOL	R/W	0h	APWM output polarity select. This is applicable only in APWM operating mode. Reset type: SYSRSn 0h (R/W) = Output is active high (Compare value defines high time) 1h (R/W) = Output is active low (Compare value defines low time)
9	CAP_APWM	R/W	0h	CAP/APWM operating mode select Reset type: SYSRSn 0h (R/W) = ECAP module operates in capture mode. This mode forces the following configuration: - Inhibits TSCTR resets via CTR = PRD event - Inhibits shadow loads on CAP1 and 2 registers - Permits user to enable CAP1-4 register load - CAPx/APWMx pin operates as a capture input 1h (R/W) = ECAP module operates in APWM mode. This mode forces the following configuration: - Resets TSCTR on CTR = PRD event (period boundary) - Permits shadow loading on CAP1 and 2 registers - Disables loading of time-stamps into CAP1-4 registers - CAPx/APWMx pin operates as a APWM output
8	SWSYNC	R-0/W1S	0h	Software-forced Counter (TSCTR) Synchronizer. This provides the user a method to generate a synchronization pulse through software. In APWM mode, the synchronization pulse can also be sourced from the CTR = PRD event. Reset type: SYSRSn 0h (R/W) = Writing a zero has no effect. Reading always returns a zero 1h (R/W) = Writing a one forces a TSCTR shadow load of current ECAP module and any ECAP modules down-stream providing the SYNCO_SEL bits are 0,0. After writing a 1, this bit returns to a zero. Note: Selection CTR = PRD is meaningful only in APWM mode however, you can choose it in CAP mode if you find doing so useful.
7-6	SYNCO_SEL	R/W	0h	Sync-Out Select Reset type: SYSRSn 0h (R/W) = Select sync-in event to be the sync-out signal (pass through) 1h (R/W) = Select CTR = PRD event to be the sync-out signal 2h (R/W) = Disable sync out signal 3h (R/W) = Disable sync out signal

**Table 5-11. ECCTL2 Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
5	SYNCl_EN	R/W	0h	Counter (TSCTR) Sync-In select mode Reset type: SYSRSn 0h (R/W) = Disable sync-in option 1h (R/W) = Enable counter (TSCTR) to be loaded from CTRPHS register upon either a SYNCl signal or a S/W force event.
4	TSCTRSTOP	R/W	0h	Time Stamp (TSCTR) Counter Stop (freeze) Control Reset type: SYSRSn 0h (R/W) = TSCTR stopped 1h (R/W) = TSCTR free-running
3	REARM	R-0/W1S	0h	Re-Arming Control. Note: The re-arm function is valid in one shot or continuous mode. Reset type: SYSRSn 0h (R/W) = Has no effect (reading always returns a 0) 1h (R/W) = Arms the one-shot sequence as follows: 1) Resets the Mod4 counter to zero 2) Unfreezes the Mod4 counter 3) Enables capture register loads
2-1	STOP_WRAP	R/W	3h	Stop value for one-shot mode. This is the number (between 1-4) of captures allowed to occur before the CAP(1-4) registers are frozen, that is, capture sequence is stopped. Wrap value for continuous mode. This is the number (between 1-4) of the capture register in which the circular buffer wraps around and starts again. Notes: STOP_WRAP is compared to Mod4 counter and, when equal, 2 actions occur: - Mod4 counter is stopped (frozen) - Capture register loads are inhibited In one-shot mode, further interrupt events are blocked until re-armed. Reset type: SYSRSn 0h (R/W) = Stop after Capture Event 1 in one-shot mode Wrap after Capture Event 1 in continuous mode. 1h (R/W) = Stop after Capture Event 2 in one-shot mode Wrap after Capture Event 2 in continuous mode. 2h (R/W) = Stop after Capture Event 3 in one-shot mode Wrap after Capture Event 3 in continuous mode. 3h (R/W) = Stop after Capture Event 4 in one-shot mode Wrap after Capture Event 4 in continuous mode.
0	CONT_ONESHT	R/W	0h	Continuous or one-shot mode control (applicable only in capture mode) Reset type: SYSRSn 0h (R/W) = Operate in continuous mode 1h (R/W) = Operate in one-Shot mode

### 5.7.2.9 ECEINT Register (Offset = 16h) [reset = 0h]

The interrupt enable bits (CEVT1, ...) block any of the selected events from generating an interrupt. Events will still be latched into the flag bit (ECFLG register) and can be forced/cleared via the ECFRC/ECCLR registers. The proper procedure for configuring peripheral modes and interrupts is as follows:

1. Disable global interrupts
2. Stop eCAP counter
3. Disable eCAP interrupts
4. Configure peripheral registers
5. Clear spurious eCAP interrupt flags
6. Enable eCAP interrupts
7. Start eCAP counter
8. Enable global interrupts

**Figure 5-24. ECEINT Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_EQ_CMP	CTR_EQ_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	RESERVED
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h

**Table 5-12. ECEINT Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_EQ_CMP	R/W	0h	Counter Equal Compare Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Compare Equal as an Interrupt source 1h (R/W) = Enable Compare Equal as an Interrupt source
6	CTR_EQ_PRD	R/W	0h	Counter Equal Period Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Period Equal as an Interrupt source 1h (R/W) = Enable Period Equal as an Interrupt source
5	CTROVF	R/W	0h	Counter Overflow Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disabled counter Overflow as an Interrupt source 1h (R/W) = Enable counter Overflow as an Interrupt source
4	CEVT4	R/W	0h	Capture Event 4 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 4 as an Interrupt source 1h (R/W) = Capture Event 4 Interrupt Enable
3	CEVT3	R/W	0h	Capture Event 3 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 3 as an Interrupt source 1h (R/W) = Enable Capture Event 3 as an Interrupt source
2	CEVT2	R/W	0h	Capture Event 2 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 2 as an Interrupt source 1h (R/W) = Enable Capture Event 2 as an Interrupt source
1	CEVT1	R/W	0h	Capture Event 1 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 1 as an Interrupt source 1h (R/W) = Enable Capture Event 1 as an Interrupt source
0	RESERVED	R	0h	Reserved

### 5.7.2.10 ECFLG Register (Offset = 17h) [reset = 0h]

Capture Interrupt Flag Register

**Figure 5-25. ECFLG Register**

15								14		13		12		11		10		9		8			
RESERVED																							
R-0h																							
7			6			5			4			3			2			1			0		
CTR_CMP			CTR_PRD			CTROVF			CEVT4			CEVT3			CEVT2			CEVT1			INT		
R-0h			R-0h			R-0h			R-0h			R-0h			R-0h			R-0h			R-0h		

**Table 5-13. ECFLG Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_CMP	R	0h	Compare Equal Compare Status Flag. This flag is active only in APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) reached the compare register value (ACMP)
6	CTR_PRD	R	0h	Counter Equal Period Status Flag. This flag is only active in APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) reached the period register value (APRD) and was reset.
5	CTROVF	R	0h	Counter Overflow Status Flag. This flag is active in CAP and APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) has made the transition from FFFFFFFF " 00000000
4	CEVT4	R	0h	Capture Event 4 Status Flag This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the fourth event occurred at ECAPx pin
3	CEVT3	R	0h	Capture Event 3 Status Flag. This flag is active only in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the third event occurred at ECAPx pin.
2	CEVT2	R	0h	Capture Event 2 Status Flag. This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the second event occurred at ECAPx pin.
1	CEVT1	R	0h	Capture Event 1 Status Flag. This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the first event occurred at ECAPx pin.
0	INT	R	0h	Global Interrupt Status Flag Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates that an interrupt was generated.

### 5.7.2.11 ECCLR Register (Offset = 18h) [reset = 0h]

Capture Interrupt Clear Register

**Figure 5-26. ECCLR Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	INT
R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h

**Table 5-14. ECCLR Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_CMP	R-0/W1C	0h	Counter Equal Compare Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CTR=COMP flag.
6	CTR_PRD	R-0/W1C	0h	Counter Equal Period Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CTR=PRD flag.
5	CTROVF	R-0/W1C	0h	Counter Overflow Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CTROVF flag.
4	CEVT4	R-0/W1C	0h	Capture Event 4 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT4 flag.
3	CEVT3	R-0/W1C	0h	Capture Event 3 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT3 flag.
2	CEVT2	R-0/W1C	0h	Capture Event 2 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT2 flag.
1	CEVT1	R-0/W1C	0h	Capture Event 1 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT1 flag.
0	INT	R-0/W1C	0h	ECAP Global Interrupt Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the INT flag and enable further interrupts to be generated if any of the event flags are set to 1

### 5.7.2.12 ECFRC Register (Offset = 19h) [reset = 0h]

Capture Interrupt Force Register

**Figure 5-27. ECFRC Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	RESERVED
R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0h

**Table 5-15. ECFRC Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_CMP	R-0/W1S	0h	Force Counter Equal Compare Interrupt. This event is only active in APWM mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CTR=CMP flag.
6	CTR_PRD	R-0/W1S	0h	Force Counter Equal Period Interrupt. This event is only active in APWM mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CTR=PRD flag.
5	CTROVF	R-0/W1S	0h	Force Counter Overflow. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 to this bit sets the CTROVF flag.
4	CEVT4	R-0/W1S	0h	Force Capture Event 4. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CEVT4 flag.
3	CEVT3	R-0/W1S	0h	Force Capture Event 3. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CEVT3 flag.
2	CEVT2	R-0/W1S	0h	Force Capture Event 2. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CEVT2 flag.
1	CEVT1	R-0/W1S	0h	Force Capture Event 1. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Sets the CEVT1 flag.
0	RESERVED	R	0h	Reserved

The Analog-to-Digital Converter (ADC) module is a Type 3 ADC. See the [C2000 Real-Time Control Peripherals Reference Guide](#) for a list of all devices with modules of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

<b>6.1 Introduction</b> .....	<b>400</b>
<b>6.2 SOC Principle of Operation</b> .....	<b>403</b>
<b>6.3 ONESHOT Single Conversion Support</b> .....	<b>409</b>
<b>6.4 ADC Conversion Priority</b> .....	<b>410</b>
<b>6.5 Sequential Sampling Mode</b> .....	<b>413</b>
<b>6.6 Simultaneous Sampling Mode</b> .....	<b>413</b>
<b>6.7 EOC and Interrupt Operation</b> .....	<b>414</b>
<b>6.8 Power-Up Sequence</b> .....	<b>415</b>
<b>6.9 ADC Calibration</b> .....	<b>415</b>
<b>6.10 Internal/External Reference Voltage Selection</b> .....	<b>416</b>
<b>6.11 ADC Timings</b> .....	<b>417</b>
<b>6.12 Internal Temperature Sensor</b> .....	<b>422</b>
<b>6.13 ADC Registers</b> .....	<b>424</b>

## 6.1 Introduction

The ADC module described in this chapter is a 12-bit recyclic ADC; part SAR, part pipelined. The analog circuits of this converter, referred to as the "core" in this document, include the front-end analog multiplexers (MUXs), sample-and-hold (S+H) circuits, the conversion core, voltage regulators, and other analog supporting circuits. Digital circuits, referred to as the "wrapper" in this document, include programmable conversions, result registers, interface to analog circuits, interface to device peripheral bus, and interface to other on-chip modules.

### 6.1.1 Features

The core of the ADC contains a single 12-bit converter fed by two sample and hold circuits. The sample and hold circuits can be sampled simultaneously or sequentially. These, in turn, are fed by a total of up to 16 analog input channels. See your device-specific data sheet for the specific number of channels available. The converter can be configured to run with an internal bandgap reference to create true-voltage based conversions or with a pair of external voltage references (VREFHI/LO) to create ratiometric based conversions.

Contrary to previous ADC types, this ADC is not sequencer based. It is easy for the user to create a series of conversions from a single trigger. However, the basic principle of operation is centered around the configurations of individual conversions, called SOC's, or Start-Of-Conversions.

Functions of the ADC module include:

- 12-bit ADC core with built-in dual sample-and-hold (S+H)
- Simultaneous sampling or sequential sampling modes
- Full range analog input: 0 V to 3.3 V fixed, or VREFHI/VREFLO ratiometric
- Up to 16-channel, multiplexed inputs
- 16 SOC's, configurable for trigger, sample window, and channel
- 16 result registers (individually addressable) to store conversion values
- Multiple trigger sources
  - S/W - software immediate start
  - ePWM 1-4
  - GPIO XINT2
  - CPU Timers 0/1/2
  - ADCINT1/2
- 9 flexible PIE interrupts, can configure interrupt request after any conversion

### 6.1.2 ADC Related Collateral

#### Foundational Materials

- [ADC Input Circuit Evaluation for C2000 MCUs \(TINA-TI\) Application Report](#)
- [PSpice for TI design and simulation tool](#)
- [Real-Time Control Reference Guide](#)
  - Refer to the ADC section
- [TI Precision Labs - ADCs](#) (Video)
  - Start with the "Introduction to analog-to-digital converters (ADCs)" section.
- [TI Precision Labs - ADCs](#)
- [TI Precision Labs: Driving the reference input on a SAR ADC](#) (Video)
- [TI Precision Labs: Introduction to analog-to-digital converters \(ADCs\)](#) (Video)
- [TI Precision Labs: SAR ADC input driver design](#) (Video)
- [TI e2e: Connecting VDDA to VREFHI](#)
- [TI e2e: Topologies for ADC Input Protection](#)
- [TI e2e: Why does the ADC Input Voltage drop with sampling?](#)
  - Sampling a high impedance voltage divider with ADC
- [Understanding Data Converters Application Report](#)



### Getting Started Materials

- [ADC-PWM Synchronization Using ADC Interrupt](#)
  - NOTE: This is a non-TI (third party) site.
- [C2000 ADC \(Type-3\) Performance Versus ACQPS Application Report](#)
- [Hardware Design Guide for F2800x C2000 Real-Time MCU Series](#)

### Expert Materials

- [Analog Engineer's Calculator](#)
- [Analog Engineer's Pocket Reference](#)
- [Debugging an integrated ADC in a microcontroller using an oscilloscope](#)
- [TI Precision Labs: ADC AC specifications \(Video\)](#)
- [TI Precision Labs: ADC Error sources \(Video\)](#)
- [TI Precision Labs: ADC Noise \(Video\)](#)
- [TI Precision Labs: Analog-to-digital converter \(ADC\) drive topologies \(Video\)](#)
- [TI Precision Labs: Electrical overstress on data converters \(Video\)](#)
- [TI Precision Labs: High-speed ADC fundamentals \(Video\)](#)
- [TI Precision Labs: SAR & Delta-Sigma: Understanding the Difference \(Video\)](#)
- [TI e2e: ADC Bandwidth Clarification](#)
- [TI e2e: ADC Calibration and Total Unadjusted Error](#)
- [TI e2e: ADC Reference Driver Options](#)
- [TI e2e: ADC Resolution with Oversampling](#)
- [TI e2e: ADC configuration for interleaved mode](#)
- [TI e2e: Simultaneous Sampling with Single ADC](#)

### 6.1.3 Block Diagram

Figure 6-1 shows the block diagram of the ADC module.

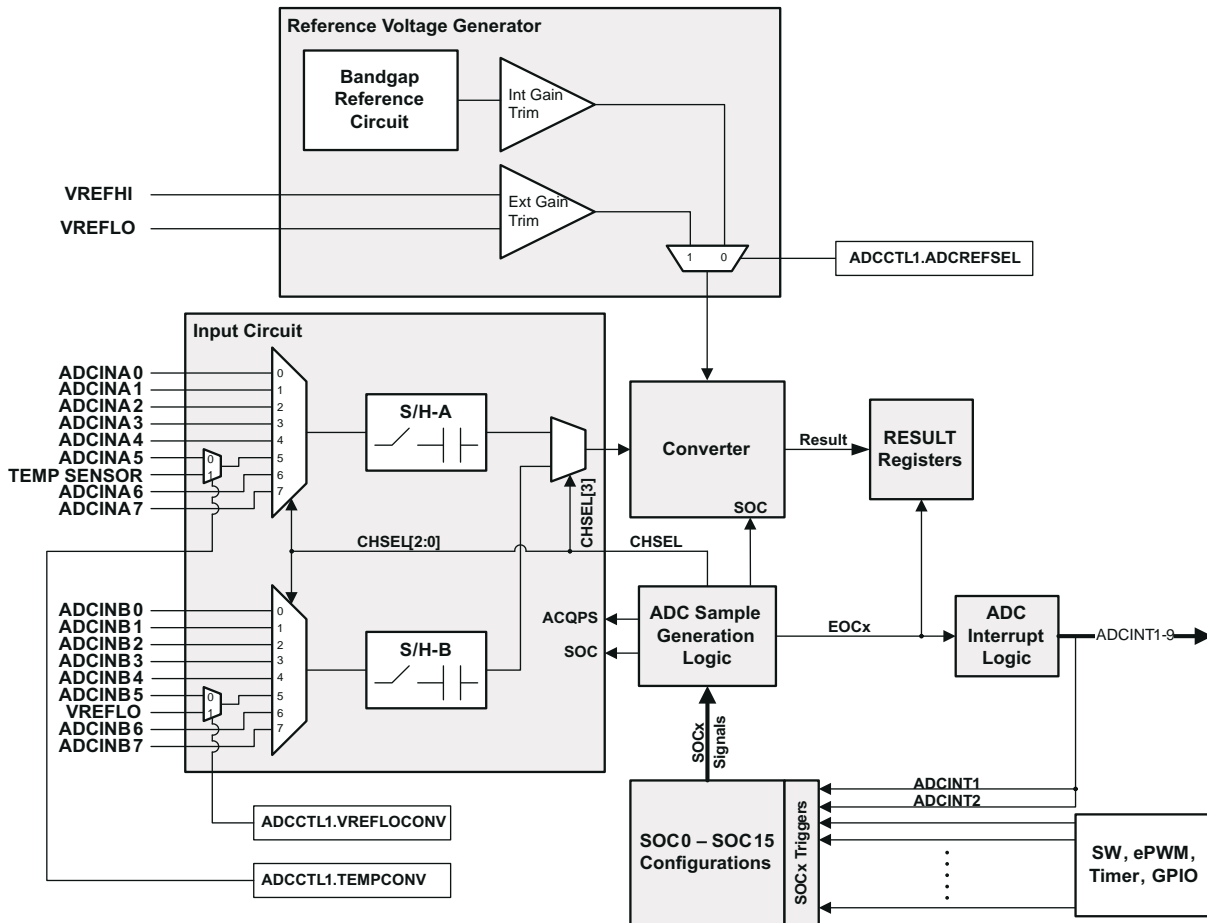


Figure 6-1. ADC Block Diagram

## 6.2 SOC Principle of Operation

Contrary to previous ADC types, this ADC is not sequencer based. Instead, it is SOC based. The term SOC is configuration set defining the single conversion of a single channel. In that set there are three configurations: the trigger source that starts the conversion, the channel to convert, and the acquisition (sample) window size. Each SOC is independently configured and can have any combination of the trigger, channel, and sample window size available. Multiple SOCs can be configured for the same trigger, channel, and/or acquisition window as desired. This provides a very flexible means of configuring conversions ranging from individual samples of different channels with different triggers, to oversampling the same channel using a single trigger, to creating your own series of conversions of different channels all from a single trigger.

The trigger source for SOCx is configured by a combination of the TRIGSEL field in the ADCSOCxCTL register and the appropriate bits in the ADCINTSOCSEL1 or ADCINTSOCSEL2 register. Software can also force an SOC event with the ADCSOCFRC1 register. The channel and sample window size for SOCx are configured with the CHSEL and ACQPS fields of the ADCSOCxCTL register.

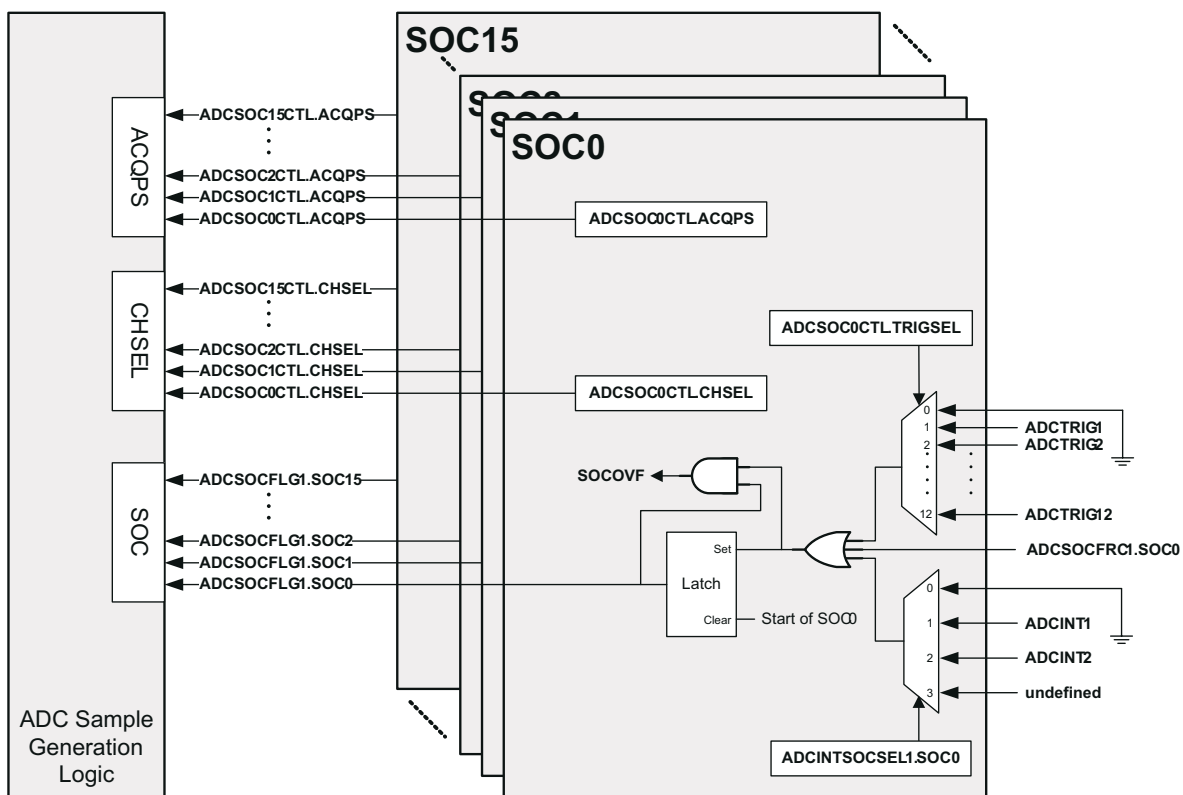


Figure 6-2. SOC Block Diagram

For example, to configure a single conversion on channel ADCINA1 to occur when the ePWM3 timer reaches its period match, you must first setup ePWM3 to output an SOCA or SOCB signal on a period match (see [Chapter 3](#)). In this case, we will use SOCA. Then, set up one of the SOCs using its ADCSOCxCTL register. It makes no difference which SOC we choose, so we will use SOC0. The fastest allowable sample window for the ADC is 7 cycles. Choosing the fastest time for the sample window, channel ADCINA1 for the channel to convert, and ePWM3 for the SOC0 trigger, we'll set the ACQPS field to 6, the CHSEL field to 1, and the TRIGSEL field to 9, respectively. The resulting value written into the register is:

```
ADCSOC0CTL = 4846h; // (ACQPS=6, CHSEL=1, TRIGSEL=9)
```

When configured as such, a single conversion of ADCINA1 will be started on an ePWM3 SOCA event with the resulting value stored in the ADCRESULT0 register.

If instead ADCINA1 needed to be oversampled by 3X, then SOC1, SOC2, and SOC3 could all be given the same configuration as SOC0.

```
ADCSOC1CTL = 4846h;           // (ACQPS=6, CHSEL=1, TRIGSEL=9)
ADCSOC2CTL = 4846h;           // (ACQPS=6, CHSEL=1, TRIGSEL=9)
ADCSOC3CTL = 4846h;           // (ACQPS=6, CHSEL=1, TRIGSEL=9)
```

When configured as such, four conversions of ADCINA1 will be started in series on an ePWM3 SOCA event with the resulting values stored in the ADCRESULT0 – ADCRESULT3 registers.

Another application may require 3 different signals to be sampled from the same trigger. This can be done by simply changing the CHSEL field for SOC0-SOC2 while leaving the TRIGSEL field unchanged.

```
ADCSOC0CTL = 4846h;           // (ACQPS=6, CHSEL=1, TRIGSEL=9)
ADCSOC1CTL = 4886h;           // (ACQPS=6, CHSEL=2, TRIGSEL=9)
ADCSOC2CTL = 48C6h;           // (ACQPS=6, CHSEL=3, TRIGSEL=9)
```

When configured this way, three conversions will be started in series on an ePWM3 SOCA event. The result of the conversion on channel ADCINA1 will show up in ADCRESULT0. The result of the conversion on channel ADCINA2 will show up in ADCRESULT1. The result of the conversion on channel ADCINA3 will show up in ADCRESULT2. The channel converted and the trigger have no bearing on where the result of the conversion shows up. The RESULT register is associated with the SOC.

#### Note

These examples are incomplete. Clocks must be enabled via the PCLKCR0 register and the ADC must be powered to work correctly. For a description of the PCLKCR0 register, see the *System Control and Interrupts* chapter. For the power-up sequence of the ADC, see [Section 6.8](#). The CLKDIV2EN bit in the ADCCTL2 register must also be set to a proper value to obtain correct frequency of operation. For more information on the ADCCTL2 register, refer to [Section 6.13](#).

### 6.2.1 ADC Acquisition (Sample and Hold) Window

External drivers vary in their ability to drive an analog signal quickly and effectively. Some circuits require longer times to properly transfer the charge into the sampling capacitor of an ADC. To address this, the ADC supports control over the sample window length for each individual SOC configuration. Each ADCSOCxCTL register has a 6-bit field, ACQPS, that determines the sample and hold (S+H) window size. The value written to this field is one less than the number of cycles desired for the sampling window for that SOC. Thus, a value of 15 in this field will give 16 clock cycles of sample time. The minimum number of sample cycles allowed is 7 (ACQPS=6). The total sampling time is found by adding the sample window size to the conversion time of the ADC, 13 ADC clocks. Examples of various sample times are shown in [Table 6-1](#).

**Table 6-1. Sample Timings with Different Values of ACQPS**

ADC Clock	ACQPS	Sample Window	Conversion Time (13 cycles)	Total Time to Process Analog Voltage <sup>(1)</sup>
40 MHz	6	175.00 ns	325.00 ns	500.00 ns
40 MHz	25	625.00 ns	325.00 ns	950.00 ns
60 MHz	6	116.67 ns	216.67 ns	333.33 ns
60 MHz	25	433.67 ns	216.67 ns	650 ns

(1) The total times are for a single conversion and do not include pipelining effects that increase the average speed over time.

As shown in [Figure 6-3](#), the ADCIN pins can be modeled as an RC circuit. With VREFLO connected to ground, a voltage swing from 0 to 3.3 V on ADCIN yields a typical RC time constant of 2 ns.

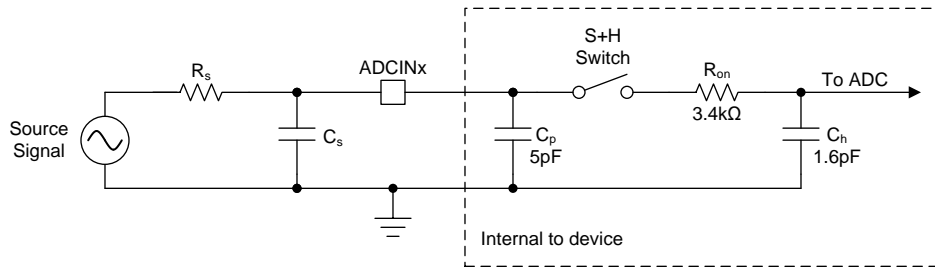


Figure 6-3. ADCINx Input Model

**Note**

The ADC does not precondition the Ch capacitor voltage before conversions, therefore the following behaviors apply:

1. There is no predetermined ADC conversion value when the ADCIN pin is not connected to a Source Signal.
2. Residual charge will remain on Ch between ADC conversions.
3. Sequential conversions may suffer from cross-talk if the ACQPS window is too short for Ch to settle.

For correct operation, the input signal to the ADC must be allowed adequate time to charge the sample and hold capacitor, C<sub>h</sub>. Typically, the S+H duration is chosen such that C<sub>h</sub> will be charged to within ½ LSB or ¼ LSB of the final value, depending on the tolerable settling error.

The S+H time required to satisfy the settling error is largely influenced by the bandwidth of the source signal. Therefore, the following recommendations for approximating the S+H duration will be simplified into two practical scenarios of either high bandwidth or low bandwidth signals. A high bandwidth source signal will be characterized as being able to meet the settling error and real-time requirements of the system using a supported ACQPS setting. A low bandwidth source signal is one that requires a longer S+H duration than is acceptable.

**6.2.1.1 ACQPS Approximation for High Bandwidth Signals**

Signals that must be sampled frequently with minimal phase delay (such as feedback sensors used in control-loop calculations) are high bandwidth signals. These signal paths require a small R<sub>s</sub>C<sub>s</sub> time constant as seen by the ADCINx pin. An external signal buffer (such as an op-amp) may be used to boost the sampling bandwidth; such buffers should ideally have a bandwidth that is high enough to fully charge C<sub>h</sub> within the selected ACPQS S+H window.

**6.2.1.1.1 ACQPS Approximation Equations for High Bandwidth Signals**

An approximation of the required settling time can be determined using an RC settling model. The time constant (τ) for the model is given by the equation:

$$\tau = (R_s + R_{on})(C_h) + (R_s)(C_s + C_p)$$

And the number of time constants needed is given by the equation:

$$k = \ln\left(\frac{2^n}{\text{settling error}}\right) - \ln\left(\frac{C_s + C_p}{C_h}\right)$$

So the total S+H time (t<sub>S+H</sub>) should be set to at least:

$$t_{s+h} = k \cdot \tau$$

Finally,  $t_{S+H}$  is used to determine the minimum value to program into the ACQPS field of the ADCSOCxCTL registers:

$$ACQPS = (t_{s+h} \cdot f_{ADCCLK}) - 1$$

Where the following parameters are provided by the ADC input model:

- $n$  = ADC resolution (in bits)
- $R_{on}$  = ADC sampling switch resistance
- $C_h$  = ADC sampling capacitor
- $C_p$  = ADC parasitic pin capacitance for the channel

And the following parameters are dependent on the application design:

- settling error = tolerable settling error (in LSBs)
- $R_s$  = ADC driving circuit source impedance
- $C_s$  = ADC driving circuit source capacitance on ADC input pin
- $f_{ADCCLK}$  = ADC clock frequency

#### 6.2.1.1.2 ACQPS Approximation Example for High Bandwidth Signals

For example, assuming the following parameters:

- $n$  = 12-bits
- settling error =  $\frac{1}{4}$  LSB
- $R_{on} = 3400\Omega$
- $C_h = 1.6pF$
- $C_p = 5pF$
- $R_s = 56\Omega$
- $C_s = 2.2nF$
- $f_{ADCCLK} = 30MHz$

The time constant would be calculated as:

$$\tau = (56\Omega + 3400\Omega)(1.6pF) + (56\Omega)(2200pF + 5pF) = 5.5ns + 123.5ns = 129ns$$

And the number of required time constants would be:

$$k = \ln\left(\frac{2^{12} \text{ LSB}}{0.25 \text{ LSB}}\right) - \ln\left(\frac{2200pF + 5pF}{1.6pF}\right) = 9.7 - 7.2 = 2.5$$

So the S+H time should be set to at least:

$$t_{s+h} = 2.5 \cdot 129ns = 322.5ns$$

Finally, the minimum ACQPS value is calculated and rounded up to the nearest supported value:

$$ACQPS = (322.5ns \cdot 30MHz) - 1 = 8.7 \rightarrow 9$$

While this gives a rough estimate of the required acquisition window, a better method would be to setup a circuit with the ADC input model, a model of the source impedance/capacitance, and any board parasitics in SPICE (or similar software) and simulate to verify that the sampling capacitor settles to the desired accuracy.

### 6.2.1.2 ACQPS Approximation for Low Bandwidth Signals

Signals that are sampled infrequently and are tolerant of low-pass filtering (such as ambient temperature sensors) can be treated as low bandwidth signals. A large  $C_s$  that is sized to be much larger than  $C_h$  will allow the ADC to sample the  $R_s C_s$  filtered signal quickly at the expense of increased phase delay.  $C_h$  will receive the bulk of its charge from  $C_s$  during the S+H window, and  $C_s$  will recover its charge through  $R_s$  between ADC samples.

#### 6.2.1.2.1 ACQPS Approximation Equations for Low Bandwidth Signals

The desired settling accuracy will determine the value of  $C_s$ :

$$C_s = C_h \left( \frac{2^n}{\text{settling error}} \right)$$

The desired recovery time and acceptable charge error will determine the value of  $R_s$ :

$$R_s = \frac{t_{\text{recovery}}}{n_\tau \cdot C_s}$$

With this configuration,  $C_s$  acts as the effective source signal, which simplifies the equations for calculating ( $k$ ) and ( $\tau$ ) as follows:

$$\tau = R_{on} \cdot C_h$$

$$k = \ln \left( \frac{2^n}{\text{settling error}} \right)$$

So the total S+H time ( $t_{S+H}$ ) should be set to at least:

$$t_{S+h} = k \cdot \tau$$

Finally,  $t_{S+H}$  is used to determine the minimum value to program into the ACQPS field of the ADCSOCxCTL registers:

$$ACQPS = (t_{S+h} \cdot f_{ADCCLK}) - 1$$

Where the following parameters are provided by the ADC input model:

- $n$  = ADC resolution (in bits)
- $R_{on}$  = ADC sampling switch resistance
- $C_h$  = ADC sampling capacitor
- $C_p$  = ADC parasitic pin capacitance for the channel

And the following parameters are dependent on the application design:

- settling error = tolerable settling error (in LSBs)
- $t_{\text{recovery}}$  = amount of time between ADC samples
- $n_\tau$  = number of RC time constants that comprise  $t_{\text{recovery}}$
- $R_s$  = ADC driving circuit source impedance
- $C_s$  = ADC driving circuit source capacitance on ADC input pin
- $f_{ADCCLK}$  = ADC clock frequency

The selection of  $n_\tau$  will determine how much the voltage on  $C_s$  is able to recover between samples. An insufficient amount of recovery time will introduce a droop error by way of an undercharged  $C_s$ . [Table 6-2](#) shows the relationship between  $n_\tau$  and the estimated droop error.

**Table 6-2. Estimated Droop Error from  $n_T$  Value**

$n_T$	Droop Error (% of Settling Error)	Droop Error for $C_s$ sized to $\frac{1}{4}$ LSB Settling Error (LSB)	Total Error (Droop Error + $\frac{1}{4}$ LSB Settling Error)
0.25	352%	0.88 LSB	1.13 LSB
0.50	154%	0.39 LSB	0.64 LSB
0.75	90%	0.23 LSB	0.48 LSB
1.00	58%	0.15 LSB	0.40 LSB
2.00	16%	0.04 LSB	0.29 LSB
3.00	5%	0.01 LSB	0.26 LSB
4.00	2%	0.01 LSB	0.26 LSB
5.00	1%	0.00 LSB	0.25 LSB

### 6.2.1.2.2 ACQPS Approximation Example for Low Bandwidth Signals

For example, assuming the following parameters:

- $n = 12$ -bits
- settling error =  $\frac{1}{4}$  LSB
- $t_{\text{recovery}} = 1$  ms
- $n_T = 2$
- $R_{\text{on}} = 3400 \Omega$
- $C_h = 1.6$  pF
- $C_p = 5$  pF
- $f_{\text{ADCCLK}} = 30$  MHz

The minimum source capacitance would be calculated and rounded up to a common value:

$$C_s = 1.6pF \left( \frac{2^{12} \text{ LSB}}{0.25 \text{ LSB}} \right) = 26.2nF \rightarrow 33nF$$

Then the maximum source resistance would be calculated and rounded down to a common value:

$$R_s = \frac{1ms}{2 \cdot 33nF} = 15.2k\Omega \rightarrow 12k\Omega$$

Now the time constant ( $\tau$ ) and multiple ( $k$ ) can be calculated as:

$$\tau = 3400\Omega \cdot 1.6pF = 5.4ns$$

$$k = \ln \left( \frac{2^{12} \text{ LSB}}{0.25 \text{ LSB}} \right) = 9.7$$

So the S+H time should be set to at least:

$$t_{s+h} = 9.7 \cdot 5.4ns = 52.4ns$$

Finally, the minimum ACQPS value is calculated and rounded up to the nearest supported value:

$$ACQPS = (52.4ns \cdot 30MHz) - 1 = 0.6 \rightarrow 6$$

While this gives a rough estimate of the required acquisition window, a better method would be to setup a circuit with the ADC input model, a model of the source impedance/capacitance, and any board parasitics in SPICE (or similar software) and simulate to verify that the sampling capacitor settles to the desired accuracy.



### 6.2.2 Trigger Operation

Each SOC can be configured to start on one of many input triggers. Multiple SOC's can be configured for the same channel if desired. Following is a list of the available input triggers:

- Software
- CPU Timers 0/1/2 interrupts
- XINT2 SOC
- ePWM1-4 SOCA and SOCB

See the ADCSOCxCTL register bit definitions for the configuration details of these triggers.

Additionally ADCINT1 and ADCINT2 can be fed back to trigger another conversion. This configuration is controlled in the ADCINTSOCSEL1/2 registers. This mode is useful if a continuous stream of conversions is desired. See Section 6.7 for information on the ADC interrupt signals.

### 6.2.3 Channel Selection

Each SOC can be configured to convert any of the available ADCIN input channels. When an SOC is configured for sequential sampling mode, the four bit CHSEL field of the ADCSOCxCTL register defines which channel to convert. When an SOC is configured for simultaneous sampling mode, the most significant bit of the CHSEL field is dropped and the lower three bits determine which pair of channels are converted.

ADCINA0 is shared with VREFHI, and therefore cannot be used as a variable input source when using external reference voltage mode. See Section 6.10 for details on this mode.

### 6.3 ONESHOT Single Conversion Support

This mode will allow you to perform a single conversion on the next triggered SOC in the round robin scheme. The ONESHOT mode is only valid for channels present in the round robin wheel. Channels which are not configured for triggered SOC in the round robin scheme will get priority based on contents of the SOC PRIORITY field in the ADCSOC PRIORITY CTL register.

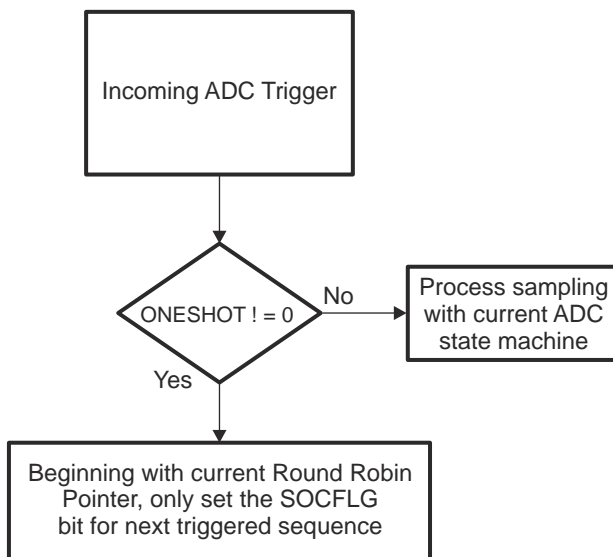


Figure 6-4. ONESHOT Single Conversion

The effect of ONESHOT mode on Sequential Mode and Simultaneous Mode is explained below.

**Sequential mode:** Only the next active SOC in RR mode (one up from current RR pointer) will be allowed to generate SOC; all other triggers for other SOC slots will be ignored.

**Simultaneous mode:** If current RR pointer has SOC with simultaneous enabled; active SOC will be incremented by 2 from the current RR pointer. This is because simultaneous mode will create result for SOCx and SOCx+1, and SOCx+1 will never be triggered by the user.

---

#### Note

ONESHOT = 1 and SOC PRIORITY = 10h is not a valid combination for above implementation reasons. This should not be a desired mode of operation by the user in any case. The limitation of the above is that the next SOC's must eventually be triggered, or else the ADC will not generate new SOC's for other out-of-order triggers. Any non-orthogonal channels should be placed in the priority mode which is unaffected by ONESHOT mode

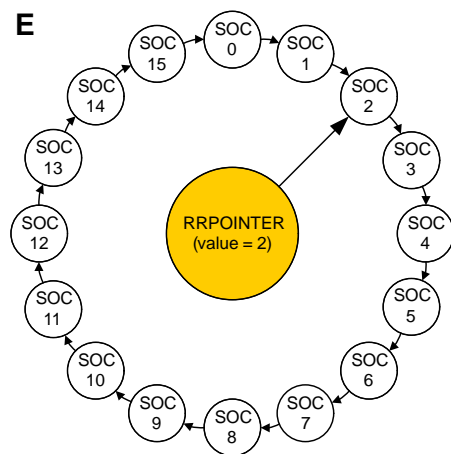
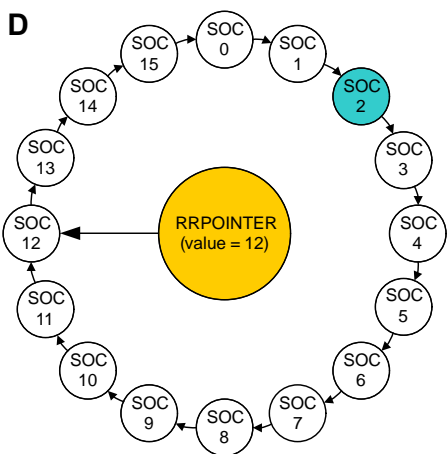
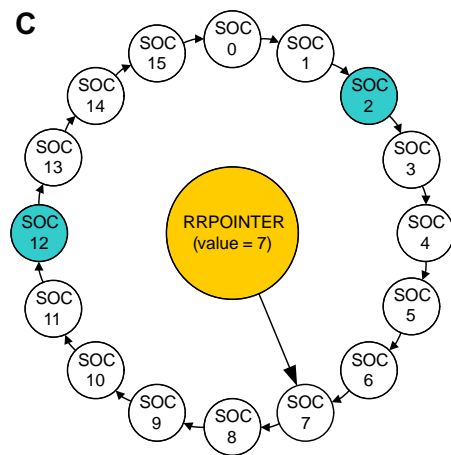
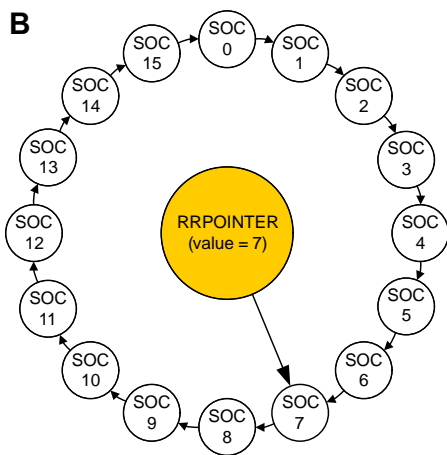
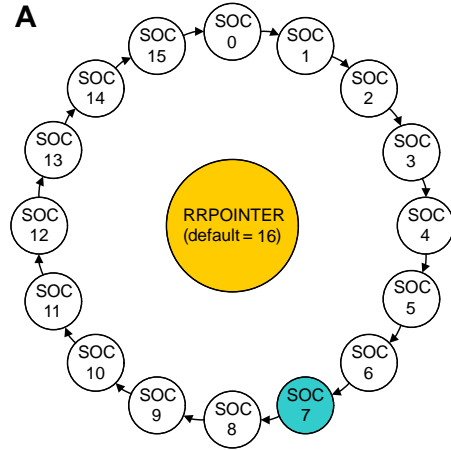
---

## 6.4 ADC Conversion Priority

When multiple SOC flags are set at the same time, one of two forms of priority determines the order in which they are converted. The default priority method is round robin. In this scheme, no SOC has an inherent higher priority than another. Priority depends on the round robin pointer (RR POINTER). The RR POINTER reflected in the ADCSOC PRIORITY CTL register points to the last SOC converted. The highest priority SOC is given to the next value greater than the RR POINTER value, wrapping around back to SOC0 after SOC15. At reset the value is 16 since 0 indicates a conversion has already occurred. When RR POINTER equals 16, the highest priority is given to SOC0. The RR POINTER is reset by a device reset, when the ADCCTL1.RESET bit is set, or when the SOC PRIORITY CTL register is written.

An example of the round robin priority method is given in [Figure 6-5](#) .

- A** After reset, SOC0 is highest priority SOC ; SOC7 receives trigger ; SOC7 configured channel is converted immediately .
- B** RRPOINTER changes to point to SOC 7 ; SOC8 is now highest priority SOC .
- C** SOC2 & SOC12 triggers rcvd. simultaneously ; SOC12 is first on round robin wheel ; SOC12 configured channel is converted while SOC2 stays pending .
- D** RRPOINTER changes to point to SOC 12 ; SOC2 configured channel is now converted .
- E** RRPOINTER changes to point to SOC 2 ; SOC3 is now highest priority SOC .



**Figure 6-5. Round Robin Priority Example**

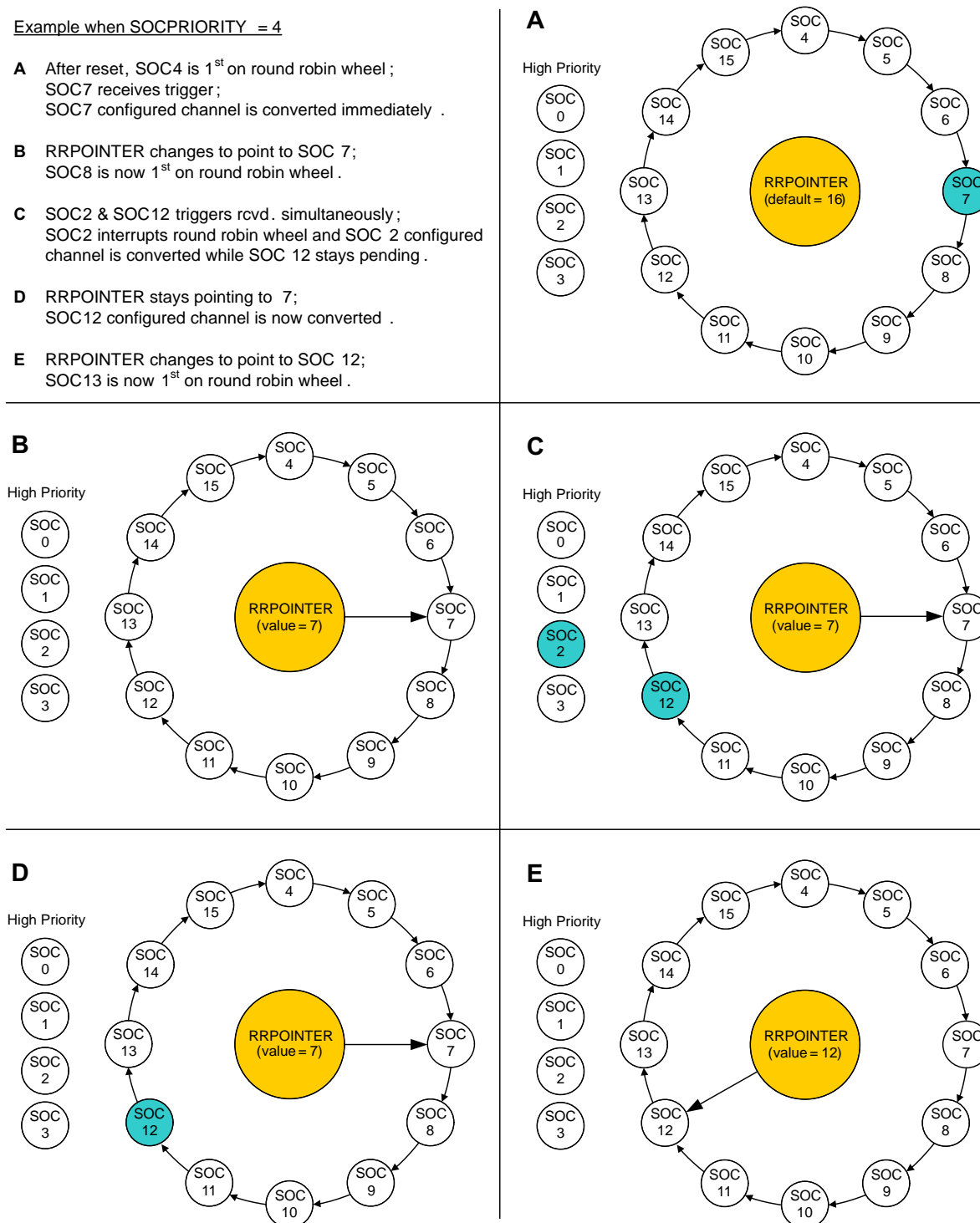
The SOC PRIORITY field in the ADCSOC PRIORITY CTL register can be used to assign high priority from a single to all of the SOC's. When configured as high priority, an SOC will interrupt the round robin wheel after any current conversion completes and insert itself in as the next conversion. After its conversion completes, the round robin wheel will continue where it was interrupted. If two high priority SOC's are triggered at the same time, the SOC with the lower number will take precedence.

High priority mode is assigned first to SOC0, then in increasing numerical order. The value written in the SOC PRIORITY field defines the first SOC that is not high priority. In other words, if a value of 4 is written into SOC PRIORITY, then SOC0, SOC1, SOC2, and SOC3 are defined as high priority, with SOC0 the highest.

An example using high priority SOC's is given in Figure 6-6 .

Example when SOC PRIORITY = 4

- A** After reset, SOC4 is 1<sup>st</sup> on round robin wheel ;  
SOC7 receives trigger ;  
SOC7 configured channel is converted immediately .
- B** RRPOINTER changes to point to SOC 7 ;  
SOC8 is now 1<sup>st</sup> on round robin wheel .
- C** SOC2 & SOC12 triggers rcvd. simultaneously ;  
SOC2 interrupts round robin wheel and SOC 2 configured channel is converted while SOC 12 stays pending .
- D** RRPOINTER stays pointing to 7 ;  
SOC12 configured channel is now converted .
- E** RRPOINTER changes to point to SOC 12 ;  
SOC13 is now 1<sup>st</sup> on round robin wheel .



**Figure 6-6. High Priority Example**

## 6.5 Sequential Sampling Mode

The default behavior of the ADC is to treat triggered SOCx as single conversions to be processed sequentially. Sequential sampling can convert both A-channels and B-channels without restriction on ordering.

However, a by-product of supporting the Simultaneous Sampling Mode is that the sampling capacitor from the paired simultaneous channel will also be connected to its respective input at the same time as the desired sequential sampling ACQPS window; the ADC will not convert the sample from the paired channel. (The Simultaneous Sampling Mode is described in [Section 6.6](#), and the sampling capacitor is denoted as Ch in [Figure 6-3](#).)

For example, assume that SOC0 is configured to convert ADCINB3, and SOC1 is configured to convert ADCINA5. If SOC0 and SOC1 are triggered together, the following sequence of simplified events would take place:

1. SOC0 Sample: A-channel and B-channel sampling capacitors are connected to ADCINA3 and ADCINB3 for SOC0 ACQPS window
2. SOC0 Convert: ADC converts B-channel sampling capacitor and stores result to ADCRESULT0
3. SOC1 Sample: A-channel and B-channel sampling capacitors are connected to ADCINA5 and ADCINB5 for SOC1 ACQPS window
4. SOC1 Convert: ADC converts A-channel sampling capacitor and stores result to ADCRESULT1

The extraneous sampling capacitor exposure should be taken into consideration for input signals that have a slow recovery or settling time. Typical examples of slow inputs are sensors with high impedance outputs and signals that are conditioned with low-pass filters.

## 6.6 Simultaneous Sampling Mode

In some applications it is important to keep the delay between the sampling of two signals minimal. The ADC contains dual sample and hold circuits to allow two different channels to be sampled simultaneously. Simultaneous sampling mode is configured for a pair of SOCx's with the ADCSAMPLEMODE register. The even-numbered SOCx and the following odd-numbered SOCx (SOC0 and SOC1) are coupled together with one enable bit (SIMULEN0, in this case). The coupling behavior is as follows:

- Either SOCx's trigger will start a pair of conversions.
- The pair of channels converted will consist of the A-channel and the B-channel corresponding to the value of the CHSEL field of the triggered SOCx. The valid values in this mode are 0-7.
- Both channels will be sampled simultaneously.
- The A channel will always convert first.
- The even EOCx pulse will be generated based off of the A-channel conversion, the odd EOCx pulse will be generated off of the B-channel conversion. See [Section 1.6](#) for an explanation of the EOCx signals.
- The result of the A-channel conversion is placed in the even ADCRESULTx register and the result of the B-channel conversion is written to the odd ADCRESULTx register.

For example, if the ADCSAMPLEMODE.SIMULEN0 bit is set, and SOC0 is configured as follows:

CHSEL = 2 (ADCINA2/ADCINB2 pair)

TRIGSEL = 5 (ADCTRIG5 = ePWM1.ADCSOCA)

When the ePWM1 sends out an ADCSOCA trigger, both ADCINA2 and ADCINB2 will be sampled simultaneously (assuming priority). Immediately after, the ADCINA2 channel will be converted and its value will be stored in the ADCRESULT0 register. Depending on the ADCCTL1.INTPULSEPOS setting, the EOC0 pulse will either occur when the conversion of ADCINA2 begins or completes. Then the ADCINB2 channel will be converted and its value will be stored in the ADCRESULT1 register. Depending on the ADCCTL1.INTPULSEPOS setting, the EOC1 pulse will either occur when the conversion of ADCINB2 begins or completes.

Typically in an application it is expected that only the even SOCx of the pair will be used. However, it is possible to use the odd SOCx instead, or even both. In the latter case, both SOCx triggers will start a conversion.

Therefore, caution is urged as both SOCx's will store their results to the same ADCRESULTx registers, possibly overwriting each other.

The rules of priority for the SOCx's remain the same as in sequential sampling mode.

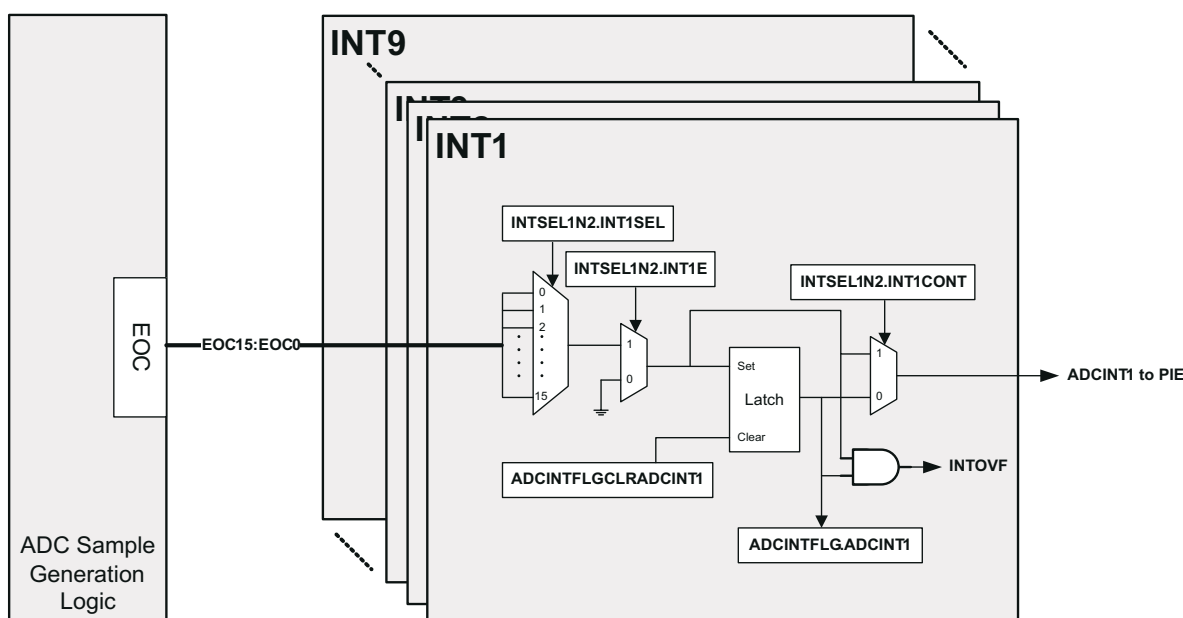
Section 6.11 shows the timing of simultaneous sampling mode.

## 6.7 EOC and Interrupt Operation

Just as there are 16 independent SOCx configuration sets, there are 16 EOCx pulses. In sequential sampling mode, the EOCx is associated directly with the SOCx. In simultaneous sampling mode, the even and the following odd EOCx pair are associated with the even and the following odd SOCx pair, as described in Section 6.6. Depending on the ADCCTL1.INTPULSEPOS setting, the EOCx pulse will occur either at the beginning of a conversion or the end. See section 1.11 for exact timings on the EOCx pulses.

The ADC contains 9 interrupts that can be flagged and/or passed on to the PIE. Each of these interrupts can be configured to accept any of the available EOCx signals as its source. The configuration of which EOCx is the source is done in the INTSELxNy registers. Additionally, the ADCINT1 and ADCINT2 signals can be configured to generate an SOCx trigger. This is beneficial to creating a continuous stream of conversions.

Figure 6-7 shows a block diagram of the interrupt structure of the ADC.



**Figure 6-7. Interrupt Structure**

### Note

Interrupt generation may be disrupted in non-continuous conversion mode when the interrupt overflow bit in ADCINTOVF is set.

## 6.8 Power-Up Sequence

The ADC resets to the ADC off state. Before writing to any of the ADC registers the ADCENCLK bit in the PCLKCR0 register must be set. For a description of the PCLKCR0 register, see the System Control and Interrupts section in this manual. When powering up the ADC, use the following sequence:

1. If an external reference is desired, enable this mode using bit 3 (ADCREFSSEL) in the ADCCTL1 register.
2. Power up the reference, bandgap, and analog circuits together by setting bits 7-5 (ADCPWDN, ADCBGPWD, ADCREFPWD) in the ADCCTL1 register.
3. Enable the ADC by setting bit 14 (ADCENABLE) of the ADCCTL1 register.
4. Before performing the first conversion, a delay of 1 millisecond after step 2 is required.

Alternatively, steps 1 through 3 can be performed simultaneously.

When powering down the ADC, all three bits in step 2 can be cleared simultaneously. The ADC power levels must be controlled via software and they are independent of the state of the device power modes.

---

### Note

This type ADC requires a 1ms delay after all of the circuits are powered up. This differs from the previous type ADC's.

---

## 6.9 ADC Calibration

Inherent in any converter is a zero offset error and a full scale gain error. The ADC is factory calibrated at 30-degrees Celsius to correct both of these while allowing the user to modify the offset correction for any application environmental effects, such as the ambient temperature. Except under certain emulation conditions, or unless a modification from the factory settings is desired, the user is not required to perform any specific action. The ADC will be properly calibrated during the device boot process.

---

### Note

If the system is reset or the ADC module is reset using Bit 15 (RESET) from the ADC Control Register 1, the Device\_cal() routine must be repeated.

---

### 6.9.1 Factory Settings and Calibration Function

During the fabrication and test process Texas Instruments calibrates several ADC settings along with a couple of internal oscillator settings. These settings are embedded into the TI reserved OTP memory as part of a C-callable function named Device\_cal(). Called during the startup boot procedure in the Boot ROM this function writes the factory settings into their respective active registers. Until this occurs, the ADC and the internal oscillators will not adhere to their specified parameters. If the boot process is skipped during emulation, the user must ensure the trim settings are written to their respective registers to ensure the ADC and the internal oscillators meet the specifications in the data sheet. This can be done either by calling this function manually or in the application itself, or by a direct write via CCS. A gel function for device calibration is included in CCS when the appropriate .ccxml file is created for the target MCU.

For more information on the Device\_cal() function refer to the Boot ROM section in this manual.

Texas Instruments cannot assure the parameters specified in the data sheet, if a value other than the factory settings contained in the TI reserved OTP memory is written into the ADC trim registers.

### 6.9.2 ADC Zero Offset Calibration

Zero offset error is defined as the resultant digital value that occurs when converting a voltage at VREFLO. This base error affects all conversions of the ADC and together with the full scale gain and linearity specifications, determine the DC accuracy of a converter. The zero offset error can be positive, meaning that a positive digital value is output when VREFLO is presented, or negative, meaning that a voltage higher than a one step above VREFLO still reads as a digital zero value. To correct this error, the two's complement of the error is written into the ADCOFFTRIM register. The value contained in this register will be applied before the results are available in the ADC result registers. This operation is fully contained within the ADC core, so the timing for the results



will not be affected and the full dynamic range of the ADC will be maintained for any trim value. Calling the `Device_cal()` function writes the `ADCOFFTRIM` register with the factory calibrated offset error correction, but the user can modify the `ADCOFFTRIM` register to compensate for additional offset error induced by the application environment. This can be done without sacrificing an ADC channel by using the `VREFLOCONV` bit in the `ADCCTRL1` register.

Use the following procedure to re-calibrate the ADC offset:

1. **Set `ADCOFFTRIM` to 80 (50h).** This adds an artificial offset to account for negative offset that may reside in the ADC core.
2. **Set `ADCCTL1.VREFLOCONV` to 1.** This internally connects `VREFLO` to input channel B5. See the `ADCCTL1` register description.
3. **Perform multiple conversions on B5 (sample `VREFLO`) and take an average to account for board noise.** See [Section 6.2](#) on how to setup and initiate the ADC to sample B5.
4. **Set `ADCOFFTRIM` to 80 (50h) minus the average obtained in step 3.** This removes the artificial offset from step 1 and creates a two's compliment of the offset error.
5. **Set `ADCCTL1.VREFLOCONV` to 0.** This connects B5 back to the external `ADCINB5` input pin.

---

#### Note

The `AdcOffsetSelfCal()` function located in `F2802x_Adc.c` in the common header files performs these steps.

---

### 6.9.3 ADC Full Scale Gain Calibration

Gain error occurs as an incremental error as the voltage input is increased. Full scale gain error occurs at the maximum input voltage. As in offset error, gain error can be positive or negative. A positive full scale gain error means that the full scale digital result is reached before the maximum voltage is input. A negative full scale error implies that the full digital result will never be achieved. The calibration function `Device_cal()` writes a factory trim value to correct the ADC full scale gain error into the `ADCREFTTRIM` register. This register should not be modified after the `Device_cal()` function is called.

### 6.9.4 ADC Bias Current Calibration

To further increase the accuracy of the ADC, the calibration function `Device_cal()` also writes a factory trim value to an ADC register for the ADC bias currents. This register should not be modified after the `Device_cal()` function is called.

## 6.10 Internal/External Reference Voltage Selection

### 6.10.1 Internal Reference Voltage

The ADC can operate in two different reference modes, selected by the `ADCCTL1.ADCREFSEL` bit. By default the internal bandgap is chosen to generate the reference voltage for the ADC. This will convert the voltage presented according to a fixed scale 0 to 3.3 V range. The equation governing conversions in this mode is:

Digital Value = 0	when Input ≤ 0 V
Digital Value = 4096 [(Input – VREFLO)/3.3 V]	when 0 V < Input < 3.3 V
Digital Value = 4095,	when Input ≥ 3.3 V

\*All fractional values are truncated

\*\*VREFLO must be tied to ground in this mode. This is done internally on some devices.



### 6.10.2 External Reference Voltage

To convert the voltage presented as a ratiometric signal, the external VREFHI/VREFLO pins should be chosen to generate the reference voltage. In contrast with the fixed 0 to 3.3 V input range of the internal bandgap mode, the ratiometric mode has an input range from VREFLO to VREFHI. Converted values will scale to this range. For instance, if VREFLO is set to 0.5 V and VREFHI is 3.0 V, a voltage of 1.75 V will be converted to the digital result of 2048. See the device data sheet for the allowable ranges of VREFLO and VREFHI. On some devices VREFLO is tied to ground internally, and hence limited to 0 V. The equation governing the conversions in this mode is:

$$\begin{aligned} \text{Digital Value} &= 0 && \text{when Input} \leq \text{VREFLO} \\ \text{Digital Value} &= 4096 \left[ \frac{\text{Input} - \text{VREFLO}}{\text{VREFHI} - \text{VREFLO}} \right] && \text{when VREFLO} < \text{Input} < \text{VREFHI} \\ \text{Digital Value} &= 4095, && \text{when Input} \geq \text{VREFHI} \end{aligned}$$

\*All fractional values are truncated

### 6.11 ADC Timings

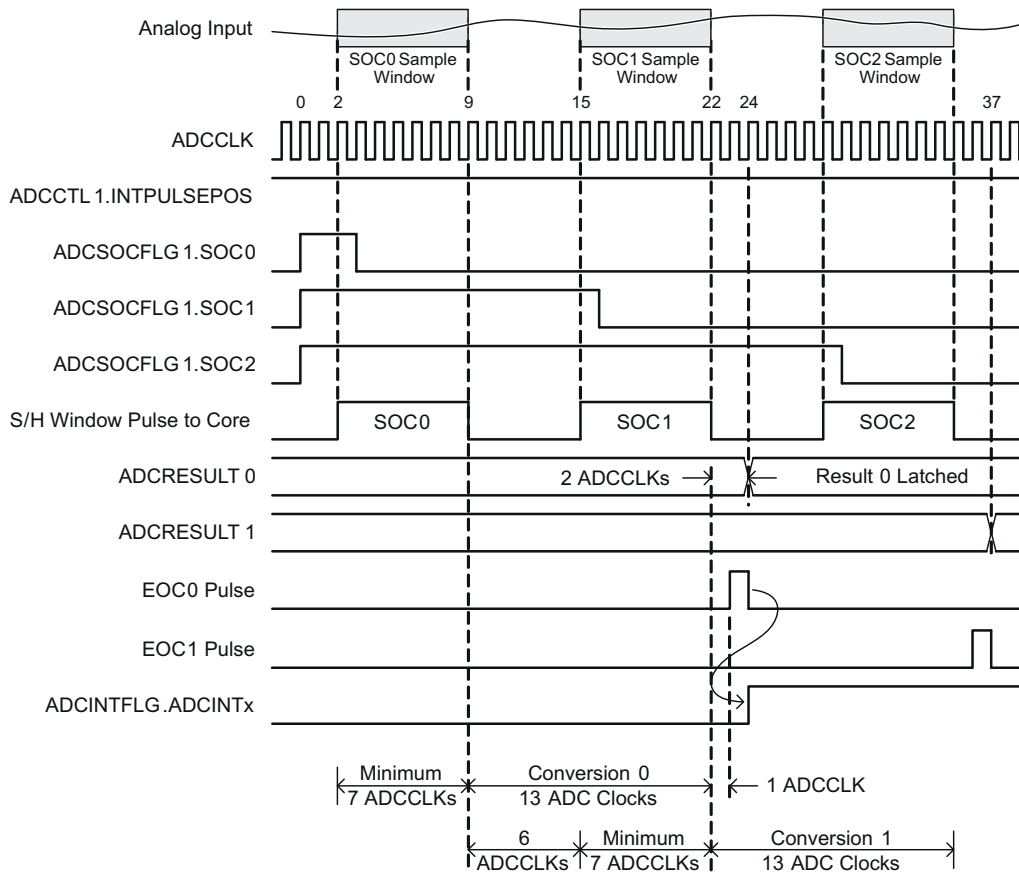


Figure 6-8. Timing Example For Sequential Mode / Late Interrupt Pulse

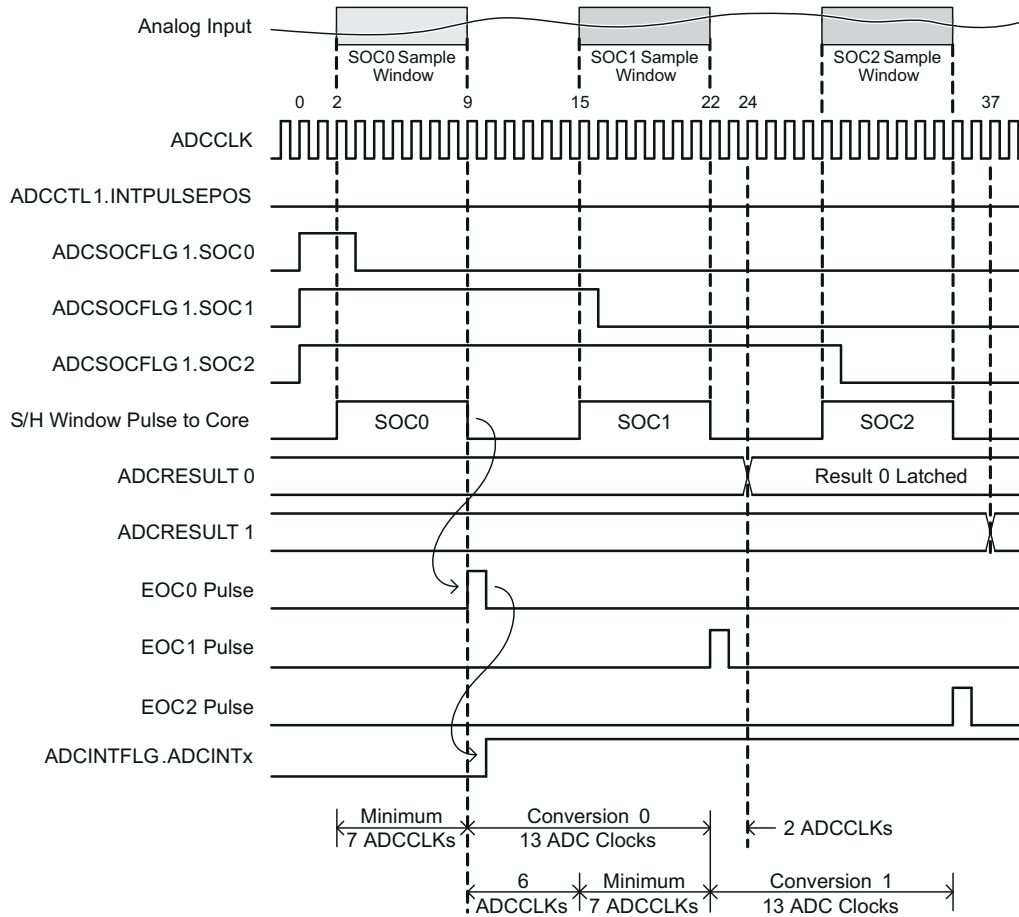


Figure 6-9. Timing Example For Sequential Mode / Early Interrupt Pulse

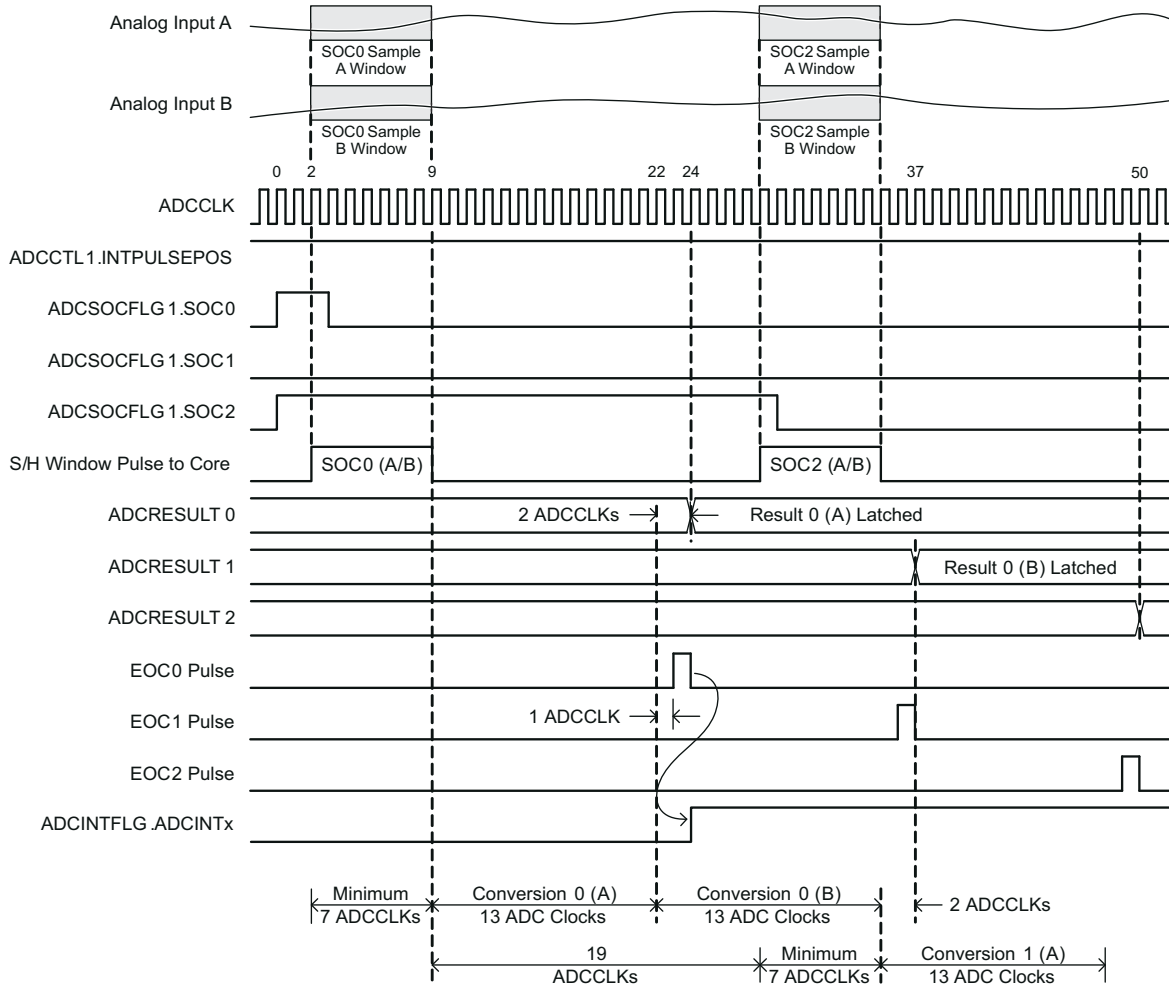


Figure 6-10. Timing Example For Simultaneous Mode / Late Interrupt Pulse

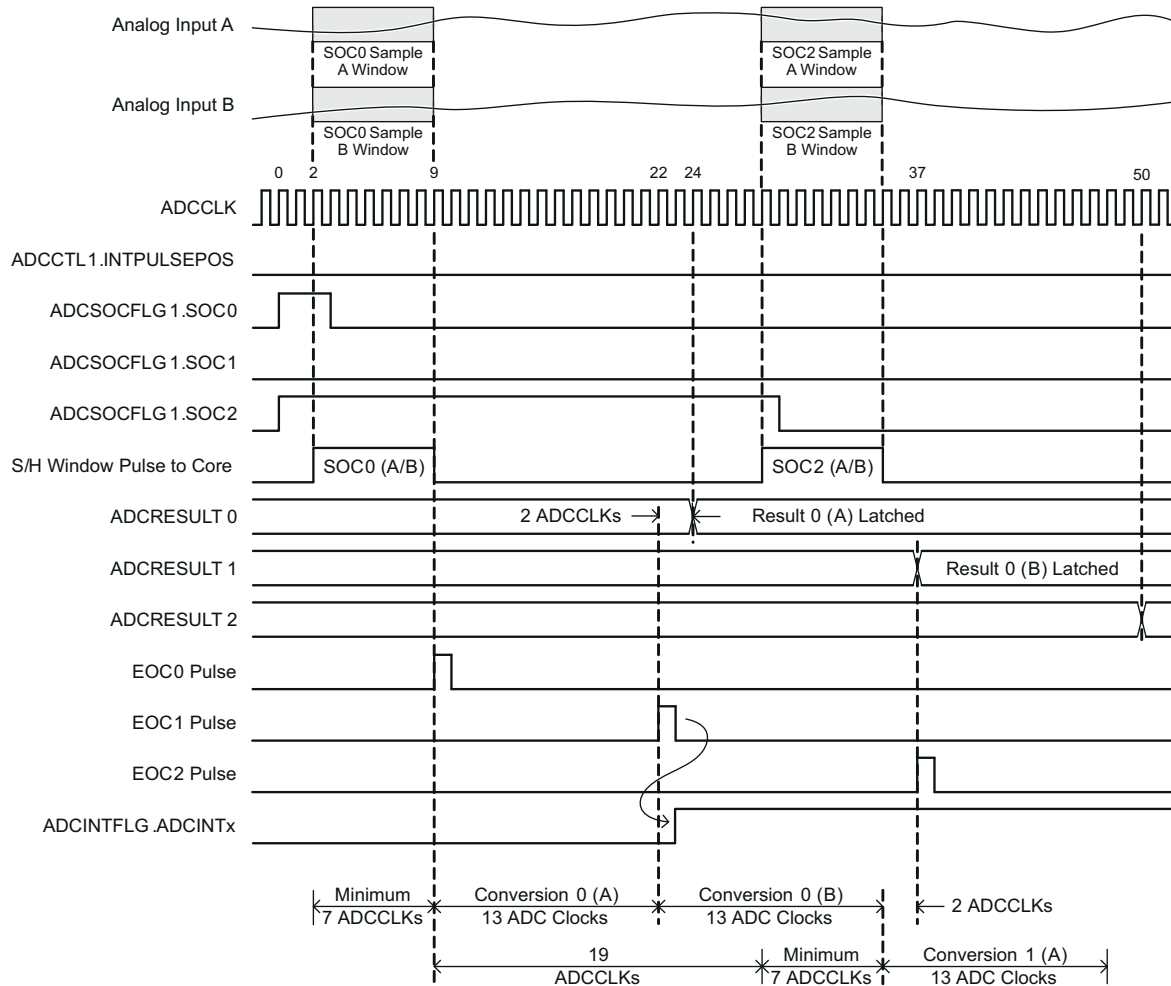
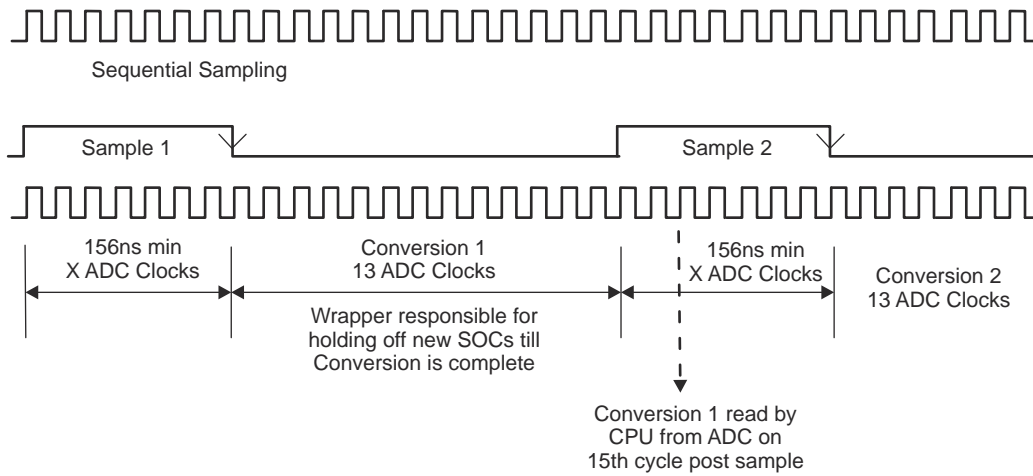


Figure 6-11. Timing Example For Simultaneous Mode / Early Interrupt Pulse



**Figure 6-12. Timing Example for NONOVERLAP Mode**

**Note**

The NONOVERLAP bit in the ADCCTL2 register, when enabled, removes the overlap of sampling and conversion stages.

## 6.12 Internal Temperature Sensor

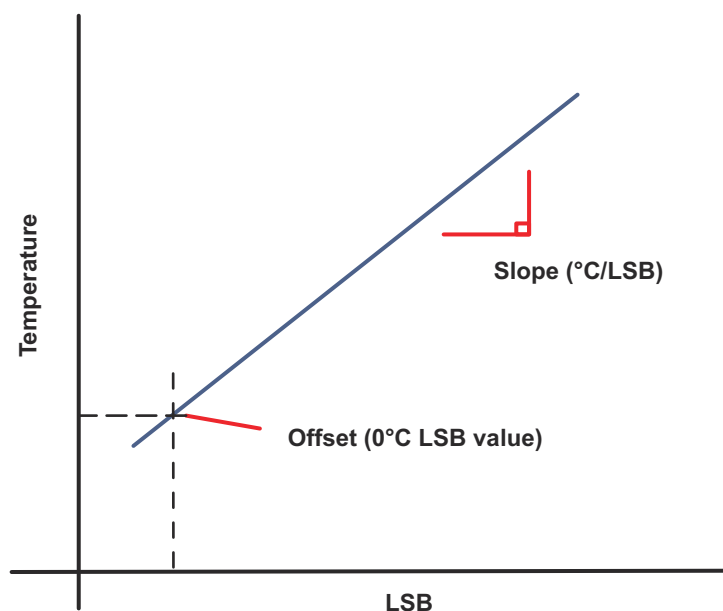
The internal temperature sensor measures the junction temperature of the device. The sensor output can be sampled with the ADC on channel A5 using a switch controlled by the ADCCTL1.TEMPCONV bit. The switch allows A5 to be used both as an external ADC input pin and the temperature sensor access point. When sampling the temperature sensor, the external circuitry on ADCINA5 has no effect on the sample. Refer to [Section 6.13.1](#) for information about switching between the external ADCINA5 input pin and the internal temperature sensor.

### 6.12.1 Transfer Function

The temperature sensor output and the resulting ADC values increase with increasing junction temperature. The offset is defined as the 0 °C LSB crossing as illustrated in [Figure 6-13](#). This information can be used to convert the ADC sensor sample into a temperature unit.

The transfer function to determine a temperature is defined as:

$$\text{Temperature} = (\text{sensor} - \text{Offset}) * \text{Slope}$$



**Figure 6-13. Temperature Sensor Transfer Function**

Refer to the electrical characteristics section in the [TMS320F2802x Microcontrollers Data Manual](#) for the slope and offset, or use the stored slope and offset calibrated per device in the factory that can be extracted by a function at the following locations.

For F2802x:

- 0x3D7E80 - Slope (°C / LSB, fixed-point Q15 format)
- 0x3D7E83 - Offset (0 °C LSB value)

The values listed are assuming a 3.3-V full-scale range. Using the internal reference mode automatically achieves this fixed range, but if using the external mode, the temperature sensor values must be adjusted accordingly to the external reference voltages.

## Example

The header files include an example project to easily sample the temperature sensor and convert the result into two different temperature units. There are three steps to using the temperature sensor:

1. Configure the ADC to sample the temperature sensor
2. Sample the temperature sensor
3. Convert the result into a temperature unit, such as °C.

Here is an example of these steps:

```
// Configure the ADC to sample the temperature sensor EALLOW;
AdcRegs.ADCCTL1.bit.TEMPCONV = 1; //Connect A5 - temp sensor
AdcRegs.ADCSOC0CTL.bit.CHSEL = 5; //Set SOC0 to sample A5
AdcRegs.ADCSOC1CTL.bit.CHSEL = 5; //Set SOC1 to sample A5
AdcRegs.ADCSOC0CTL.bit.ACQPS = 6; //Set SOC0 ACQPS to 7 ADCCLK
AdcRegs.ADCSOC1CTL.bit.ACQPS = 6; //Set SOC1 ACQPS to 7 ADCCLK
AdcRegs.INTSEL1N2.bit.INT1SEL = 1; //Connect ADCINT1 to EOC1
AdcRegs.INTSEL1N2.bit.INT1E = 1; //Enable ADCINT1
EDIS;

// Sample the temperature sensor
AdcRegs.ADCSOFRC1.all = 0x03; //Sample temp sensor
while(AdcRegs.ADCINTFLG.bit.ADCINT1 == 0){ //Wait for ADCINT1
AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //Clear ADCINT1
sensorSample = AdcResult.ADCRESULT1; //Get temp sensor sample result

//Convert raw temperature sensor output to a temperature (degC)
DegreesC = (sensorSample - TempSensorOffset) * TempSensorSlope;

For the F2806x, call the below factory stored slope and offset get functions:
//Slope of temperature sensor (deg. C / ADC code, fixed pt Q15 format)
#define getTempSlope() (*(int (*)(void))0x3D7E82)()

//ADC code corresponding to temperature sensor output at 0-degreesC
#define getTempOffset() (*(int (*)(void))0x3D7E85)()
```

## 6.13 ADC Registers

This section contains the ADC registers and bit definitions with the registers grouped by function. All of the ADC registers are located in Peripheral Frame 2 except the ADCRESULTx registers, which are found in Peripheral Frame 0. See the device data sheet for specific addresses.

**Table 6-3. ADC Configuration and Control Registers (AdcRegs and AdcResult)**

Register	Address Offset	Size (x16)	Description	Bit Description
ADCCTL1	0x00	1	Control 1 Register <sup>(1)</sup>	<a href="#">Section 6.13.1</a>
ADCCTL2	0x01	1	Control 2 Register <sup>(1)</sup>	<a href="#">Section 6.13.2</a>
ADCINTFLG	0x04	1	Interrupt Flag Register	<a href="#">Section 6.13.3.1</a>
ADCINTFLGCLR	0x05	1	Interrupt Flag Clear Register	<a href="#">Section 6.13.3.2</a>
ADCINTOVF	0x06	1	Interrupt Overflow Register	<a href="#">Section 6.13.3.3</a>
ADCINTOVFCLR	0x07	1	Interrupt Overflow Clear Register	<a href="#">Section 6.13.3.4</a>
INTSEL1N2	0x08	1	Interrupt 1 and 2 Selection Register <sup>(1)</sup>	<a href="#">Section 6.13.3.5</a>
INTSEL3N4	0x09	1	Interrupt 3 and 4 Selection Register <sup>(1)</sup>	<a href="#">Section 6.13.3.5</a>
INTSEL5N6	0x0A	1	Interrupt 5 and 6 Selection Register <sup>(1)</sup>	<a href="#">Section 6.13.3.5</a>
INTSEL7N8	0x0B	1	Interrupt 7 and 8 Selection Register <sup>(1)</sup>	<a href="#">Section 6.13.3.5</a>
INTSEL9N10	0x0C	1	Interrupt 9 Selection Register (reserved Interrupt 10 Selection) <sup>(1)</sup>	<a href="#">Section 6.13.3.5</a>
SOCPRCTL	0x10	1	Start of Conversion Priority Control Register <sup>(1)</sup>	<a href="#">Section 6.13.4</a>
ADCSAMPLEMODE	0x12	1	Sampling Mode Register <sup>(1)</sup>	<a href="#">Section 6.13.5.1</a>
ADCINTSOCSEL1	0x14	1	Interrupt SOC Selection 1 Register (for 8 channels) <sup>(1)</sup>	<a href="#">Section 6.13.5.2</a>
ADCINTSOCSEL2	0x15	1	Interrupt SOC Selection 2 Register (for 8 channels) <sup>(1)</sup>	<a href="#">Section 6.13.5.3</a>
ADCSOCFLG1	0x18	1	SOC Flag 1 Register (for 16 channels)	<a href="#">Section 6.13.5.4</a>
ADCSOCFRC1	0x1A	1	SOC Force 1 Register (for 16 channels)	<a href="#">Section 6.13.5.5</a>
ADCSOCOVF1	0x1C	1	SOC Overflow 1 Register (for 16 channels)	<a href="#">Section 6.13.5.6</a>
ADCSOCOVFCLR1	0x1E	1	SOC Overflow Clear 1 Register (for 16 channels)	<a href="#">Section 6.13.5.7</a>
ADCSOC0CTL - ADCSOC15CTL	0x20 - 0x2F	1	SOC0 Control Register to SOC15 Control Register <sup>(1)</sup>	<a href="#">Section 6.13.5.8</a>
ADCREFTTRIM	0x40	1	Reference Trim Register <sup>(1)</sup>	<a href="#">Section 6.13.6.1</a>
ADCOFFTRIM	0x41	1	Offset Trim Register <sup>(1)</sup>	<a href="#">Section 6.13.6.2</a>
COMPHYSTCTL	0x4C	1	Comp Hysteresis Control Register <sup>(1)</sup>	<a href="#">Section 6.13.7</a>
ADCREV – reserved	0x4F	1	Revision Register	<a href="#">Section 6.13.8</a>
ADCRESULT0 - ADCRESULT15	0x00 - 0x0F <sup>(2)</sup>	1	ADC Result 0 Register to ADC Result 15 Register	<a href="#">Section 6.13.9</a>

(1) This register is EALLOW protected.

(2) The base address of the ADCRESULT registers differs from the base address of the other ADC registers. In the header files, the ADCRESULT registers are found in the AdcResult register file, not in the AdcRegs.



### 6.13.1 ADC Control Register 1 (ADCCTL1)

**Figure 6-14. ADC Control Register 1 (ADCCTL1)**

15	14	13	12					8
RESET	ADCENABLE	ADCBSY	ADCBSYCHN					
R-0/W-1	R-1	R-0						R-0
7	6	5	4	3	2	1	0	
ADCPWN	ADCBGPWD	ADCREFPWD	Reserved	ADCREFSEL	INTPULSEPOS	VREFLO CONV	TEMPCONV	
R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; R-0/W-1 = always read as 0, write 1 to set; -n = value after reset

**Table 6-4. ADC Control Register 1 (ADCCTL1) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15	RESET	0 1	<p>ADC module software reset. This bit causes a master reset on the entire ADC module. All register bits and state machines are reset to the initial state as occurs when the device reset pin is pulled low (or after a power-on reset). This is a one-time-effect bit, meaning this bit is self-cleared immediately after it is set to 1. Read of this bit always returns a 0. Also, the reset of ADC has a latency of two clock cycles (that is, other ADC control register bits should not be modified until two clock cycles after the instruction that resets the ADC.</p> <p>0 no effect</p> <p>1 Resets entire ADC module (bit is then set back to 0 by ADC logic)</p> <p><b>Note:</b> The ADC module is reset during a system reset. If an ADC module reset is desired at any other time, you can do so by writing a 1 to this bit. After two clock cycles, you can then write the appropriate values to the ADCCTL1 register bits. Assembly code:</p> <pre>MOV ADCCTL1, #1xxxxxxxxxxxxxb ; Resets the ADC (RESET = 1) NOP ; Delay two cycles NOP MOV ADCCTL1, #0xxxxxxxxxxxxxb ; Set to user-desired value</pre> <p><b>Note:</b> The second MOV is not required if the default configuration is sufficient.</p> <p><b>Note:</b> If the system is reset or the ADC module is reset using Bit 15 (RESET) from the ADC Control Register 1, the Device_cal() routine must be repeated .</p>
14	ADCENABLE	0 1	<p>ADC Enable</p> <p>0 ADC disabled (<b>does not</b> power down ADC)</p> <p>1 ADC Enabled. Must set before an ADC conversion (recommend that it be set directly after setting ADC power-up bits)</p>
13	ADCBSY	0 1	<p>ADC Busy</p> <p>Set when ADC SOC is generated, cleared per below. Used by the ADC state machine to determine if ADC is available to sample.</p> <p>Sequential Mode: Cleared 4 ADC clocks after negative edge of S+H pulse</p> <p>Simultaneous Mode: Cleared 14 ADC clocks after negative edge of S+H pulse</p> <p>0 ADC is available to sample next channel</p> <p>1 ADC is busy and cannot sample another channel</p>

**Table 6-4. ADC Control Register 1 (ADCCTL1) Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
12-8	ADCB SYCHN	00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch 0Dh 0Eh 0Fh 1xh	Set when ADC SOC for current SOC is generated When ADCBSY = 0: holds the value of the last converted SOC When ADCBSY = 1: reflects SOC currently being processed SOC0 is currently processing or was last SOC converted SOC1 is currently processing or was last SOC converted SOC2 is currently processing or was last SOC converted SOC3 is currently processing or was last SOC converted SOC4 is currently processing or was last SOC converted SOC5 is currently processing or was last SOC converted SOC6 is currently processing or was last SOC converted SOC7 is currently processing or was last SOC converted SOC8 is currently processing or was last SOC converted SOC9 is currently processing or was last SOC converted SOC10 is currently processing or was last SOC converted SOC11 is currently processing or was last SOC converted SOC12 is currently processing or was last SOC converted SOC13 is currently processing or was last SOC converted SOC14 is currently processing or was last SOC converted ADCINB15 is currently processing or was last SOC converted Invalid value
7	ADCPWDN	0 1	ADC power down (active low). This bit controls the power up and power down of all the analog circuitry inside the analog core except the bandgap and reference circuitry 0 All analog circuitry inside the core except the bandgap and reference circuitry is powered down 1 The analog circuitry inside the core is powered up
6	ADCBGPWD	0 1	Bandgap circuit power down (active low) 0 Bandgap circuitry is powered down 1 Bandgap buffer's circuitry inside core is powered up
5	ADCREFPWD	0 1	Reference buffers circuit power down (active low) 0 Reference buffers circuitry is powered down 1 Reference buffers circuitry inside the core is powered up
4	Reserved	0	Reads return a zero; Writes have no effect.
3	ADCREFSSEL	0 1	Internal or external reference select 0 Internal Bandgap used for reference generation 1 External VREFHI or VREFLO pins used for reference generation. On some devices the VREFHI pin is shared with ADCINA0. In this case ADCINA0 will not be available for conversions in this mode. On some devices the VREFLO pin is shared with VSSA. In this case the VREFLO voltage cannot be varied.
2	INTPULSEPOS	0 1	INT Pulse Generation control 0 INT pulse generation occurs when ADC begins conversion (negative edge of sample pulse of the sampled signal) 1 INT pulse generation occurs 1 cycle prior to ADC result latching into its result register
1	VREFLOCONV	0 1	VREFLO Convert. When enabled, internally connects VREFLO to the ADC channel B5 and disconnects the ADCINB5 pin from the ADC. Whether the pin ADCINB5 exists on the device does not affect this function. Any external circuitry on the ADCINB5 pin is unaffected by this mode. 0 ADCINB5 is passed to the ADC module as normal, VREFLO connection to ADCINB5 is disabled 1 VREFLO internally connected to the ADC for sampling

**Table 6-4. ADC Control Register 1 (ADCCTL1) Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
0	TEMPCONV		Temperature sensor convert. When enabled internally connects the internal temperature sensor to ADC channel A5 and disconnects the ADCINA5 pin from the ADC. Whether the pin ADCINA5 exists on the device does not affect this function. Any external circuitry on the ADCINA5 pin is unaffected by this mode
		0	ADCINA5 is passed to the ADC module as normal, internal temperature sensor connection to ADCINA5 is disabled.
		1	Temperature sensor is internally connected to the ADC for sampling

(1) This register is EALLOW protected.

### 6.13.2 ADC Control Register 2 (ADCCTL2)

**Figure 6-15. ADC Control Register 2 (ADCCTL2)**

15	2	1	0
Reserved		ADCNONOVERLAP	CLKDIV2EN
R-0		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-5. ADC Control Register 2 (ADCCTL2) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-2	Reserved	0	Reads return a zero; writes have no effect.
1	ADCNONOVERLAP		ADCNONOVERLAP control bit
		0	Overlap of sample and conversion is allowed
		1	Overlap of sample is not allowed
0	CLKDIV2EN		ADC Clock Prescaler.
		0	ADCCLK = SYSCLK
		1	ADCCLK = SYSCLK / 2

(1) This register is EALLOW protected.

### 6.13.3 ADC Interrupt Registers

#### 6.13.3.1 ADC Interrupt Flag Register (ADCINTFLG)

**Figure 6-16. ADC Interrupt Flag Register (ADCINTFLG)**

15							9	8
Reserved							ADCINT9	
R-0							R-0	
7	6	5	4	3	2	1	0	
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1	
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-6. ADC Interrupt Flag Register (ADCINTFLG) Field Descriptions**

Bit	Field	Value	Description
15-9	Reserved	0	Reads return a zero; Writes have no effect.
8-0	ADCINT <sub>x</sub> (x = 9 to 1)	0	ADC Interrupt Flag Bits: Reading this bit indicates if an ADCINT pulse was generated No ADC interrupt pulse generated
		1	ADC Interrupt pulse generated If the ADC interrupt is placed in continuous mode (INTSELxNy register) then further interrupt pulses are generated whenever a selected EOC event occurs even if the flag bit is set. If the continuous mode is not enabled, then no further interrupt pulses are generated until the user clears this flag bit using the ADCINTFLGCLR register. The ADCINTOVF flag will be set if EOC events are generated while the ADCINTFLG flag is set. Both ADCINTFLG and ADCINTOVF flags must be cleared before normal interrupt operation can resume in non-continuous mode.

### 6.13.3.2 ADC Interrupt Flag Clear Register (ADCINTFLGCLR)

Figure 6-17. ADC Interrupt Flag Clear Register (ADCINTFLGCLR)

Reserved								ADCINT9
R-0								R/W-0
15							9	8
7	6	5	4	3	2	1	0	
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 6-7. ADC Interrupt Flag Clear Register (ADCINTFLGCLR) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reads return a zero; Writes have no effect.
8-0	ADCINTx (x = 9 to 1)	0	ADC interrupt Flag Clear Bit No action.
		1	Clears respective flag bit in the ADCINTFLG register. <b>Boundary condition for clearing or setting flag bits:</b> If hardware tries to set a flag bit while software tries to clear the flag bit in the same cycle, the following will take place: <ol style="list-style-type: none"> <li>1. SW has priority, and will clear the flag</li> <li>2. HW set will be discarded, no signal will propagate to the PIE from the latch</li> <li>3. Overflow flag or condition will be generated</li> </ol>

**6.13.3.3 ADC Interrupt Overflow Register (ADCINTOVF)**
**Figure 6-18. ADC Interrupt Overflow Register (ADCINTOVF)**

15							9	8
Reserved							ADCINT9	
R-0							R-0	
7	6	5	4	3	2	1	0	
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1	
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-8. ADC Interrupt Overflow Register (ADCINTOVF) Field Descriptions**

Bit	Field	Value	Description
15-9	Reserved	0	Reserved
8-0	ADCINT <sub>x</sub> (x = 9 to 1)	0 1	ADC Interrupt Overflow Bits. Indicates if an overflow occurred when generating ADCINT pulses. If the respective ADCINTFLG bit is set and a selected additional EOC trigger is generated, then an overflow condition occurs. 0 No ADC interrupt overflow event detected. 1 ADC Interrupt overflow event detected. The overflow bit does not care about the continuous mode bit state. An overflow condition is generated irrespective of this mode selection. Both ADCINTFLG and ADCINTOVF flags must be cleared before normal interrupt operation can resume in non-continuous mode.

### 6.13.3.4 ADC Interrupt Overflow Clear Register (ADCINTOVFCLR)

**Figure 6-19. ADC Interrupt Overflow Clear Register (ADCINTOVFCLR)**

Reserved								ADCINT9
R-0								R-0/W-1
15							9	8
7	6	5	4	3	2	1	0	
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1	
R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	

LEGEND: R/W = Read/Write; R = Read only; R-0/W-1 =always read 0, write 1 to set; -n = value after reset

**Table 6-9. ADC Interrupt Overflow Clear Register (ADCINTOVFCLR) Field Descriptions**

Bit	Field	Value	Description
15-9	Reserved	0	Reads return a zero; Writes have no effect.
8-0	ADCINT <sub>x</sub> (x = 9 to 1)	0	ADC Interrupt Overflow Clear Bits. No action.
		1	Clears the respective overflow bit in the ADCINTOVF register. If software tries to set this bit on the same clock cycle that hardware tries to set the overflow bit in the ADCINTOVF register, then hardware has priority and the ADCINTOVF bit will be set.

**6.13.3.5 Interrupt Select Registers (INTSELxNy)**
**Figure 6-20. Interrupt Select 1 and 2 Register (INTSEL1N2)**

15	14	13	12	8
Reserved	INT2CONT	INT2E	INT2SEL	
R-0	R/W-0	R/W-0	R/W-0	
7	6	5	4	0
Reserved	INT1CONT	INT1E	INT1SEL	
R-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Figure 6-21. Interrupt Select 3 and 4 Register (INTSEL3N4)**

15	14	13	12	8
Reserved	INT4CONT	INT4E	INT4SEL	
R-0	R/W-0	R/W-0	R/W-0	
7	6	5	4	0
Reserved	INT3CONT	INT3E	INT3SEL	
R-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Figure 6-22. Interrupt Select 5 and 6 Register (INTSEL5N6)**

15	14	13	12	8
Reserved	INT6CONT	INT6E	INT6SEL	
R-0	R/W-0	R/W-0	R/W-0	
7	6	5	4	0
Reserved	INT5CONT	INT5E	INT5SEL	
R-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset



**Figure 6-23. Interrupt Select 7 and 8 Register (INTSEL7N8)**

15	14	13	12	8
Reserved	INT8CONT	INT8E	INT8SEL	
R-0	R/W-0	R/W-0	R/W-0	
7	6	5	4	0
Reserved	INT7CONT	INT7E	INT7SEL	
R-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Figure 6-24. Interrupt Select 9 and 10 Register (INTSEL9N10)**

15				8
Reserved				
R-0				
7	6	5	4	0
Reserved	INT9CONT	INT9E	INT9SEL	
R-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-10. Interrupt Select Register (INTSELxNy) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15	Reserved	0	Reserved
14	INTyCONT	0	ADCINTy Continuous Mode Enable No further ADCINTy pulses are generated until ADCINTy flag (in ADCINTFLG register) is cleared by user.
		1	ADCINTy pulses are generated whenever an EOC pulse is generated irrespective if the flag bit is cleared or not.
13	INTyE	0	ADCINTy Interrupt Enable ADCINTy is disabled.
		1	ADCINTy is enabled.

**Table 6-10. Interrupt Select Register (INTSELxNy) Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
12-8	INTySEL		ADCINTy EOC Source Select
		00h	EOC0 is trigger for ADCINTy
		01h	EOC1 is trigger for ADCINTy
		02h	EOC2 is trigger for ADCINTy
		03h	EOC3 is trigger for ADCINTy
		04h	EOC4 is trigger for ADCINTy
		05h	EOC5 is trigger for ADCINTy
		06h	EOC6 is trigger for ADCINTy
		07h	EOC7 is trigger for ADCINTy
		08h	EOC8 is trigger for ADCINTy
		09h	EOC9 is trigger for ADCINTy
		0Ah	EOC10 is trigger for ADCINTy
		0Bh	EOC11 is trigger for ADCINTy
		0Ch	EOC12 is trigger for ADCINTy
		0Dh	EOC13 is trigger for ADCINTy
		0Eh	EOC14 is trigger for ADCINTy
0Fh	EOC15 is trigger for ADCINTy		
1xh	Invalid value.		
7	Reserved	0	Reads return a zero; Writes have no effect.
6	INTxCONT	0	ADCINTx Continuous Mode Enable. No further ADCINTx pulses are generated until ADCINTx flag (in ADCINTFLG register) is cleared by user.
		1	ADCINTx pulses are generated whenever an EOC pulse is generated irrespective if the flag bit is cleared or not.
5	INTxE	0	ADCINTx Interrupt Enable ADCINTx is disabled.
		1	ADCINTx is enabled .
4-0	INTxSEL		ADCINTx EOC Source Select
		00h	EOC0 is trigger for ADCINTx
		01h	EOC1 is trigger for ADCINTx
		02h	EOC2 is trigger for IADCNTx
		03h	EOC3 is trigger for ADCINTx
		04h	EOC4 is trigger for ADCINTx
		05h	EOC5 is trigger for ADCINTx
		06h	EOC6 is trigger for ADCINTx
		07h	EOC7 is trigger for ADCINTx
		08h	EOC8 is trigger for ADCINTx
		09h	EOC9 is trigger for ADCINTx
		0Ah	EOC10 is trigger for ADCINTx
		0Bh	EOC11 is trigger for ADCINTx
		0Ch	EOC12 is trigger for ADCINTx
		.0Dh	EOC13 is trigger for ADCINTx
		0Eh	EOC14 is trigger for ADCINTx
0Fh	EOC15 is trigger for ADCINTx		
1xh	Invalid value.		

(1) This register is EALLOW protected.

### 6.13.4 ADC Start of Conversion Priority Control Register (SOCPRICTL)

**Figure 6-25. ADC Start of Conversion Priority Control Register (SOCPRICTL)**

15	14	11	10	5	4	0
ONESHOT	Reserved		RRPOINTER			SOCPRIORITY
R/W-0	R-0	R-20h			R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-11. SOCPRICTL Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15	ONESHOT	0 1	One shot mode disabled One shot mode enabled
14-11	Reserved		Reads return a zero; Writes have no effect.
10-5	RRPOINTER	00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch 0Dh 0Eh 0Fh 1xh 20h Others	Round Robin Pointer. Holds the value of the last converted round robin SOCx to be used by the round robin scheme to determine order of conversions. SOC0 was last round robin SOC to convert. SOC1 is highest round robin priority. SOC1 was last round robin SOC to convert. SOC2 is highest round robin priority. SOC2 was last round robin SOC to convert. SOC3 is highest round robin priority. SOC3 was last round robin SOC to convert. SOC4 is highest round robin priority. SOC4 was last round robin SOC to convert. SOC5 is highest round robin priority. SOC5 was last round robin SOC to convert. SOC6 is highest round robin priority. SOC6 was last round robin SOC to convert. SOC7 is highest round robin priority. SOC7 was last round robin SOC to convert. SOC8 is highest round robin priority. SOC8 was last round robin SOC to convert. SOC9 is highest round robin priority. SOC9 was last round robin SOC to convert. SOC10 is highest round robin priority. SOC10 was last round robin SOC to convert. SOC11 is highest round robin priority. SOC11 was last round robin SOC to convert. SOC12 is highest round robin priority. SOC12 was last round robin SOC to convert. SOC13 is highest round robin priority. SOC13 was last round robin SOC to convert. SOC14 is highest round robin priority. SOC14 was last round robin SOC to convert. SOC15 is highest round robin priority. SOC15 was last round robin SOC to convert. SOC0 is highest round robin priority. Invalid value Reset value to indicate no SOC has been converted. SOC0 is highest round robin priority. Set to this value when the device is reset, when the ADCCTL1.RESET bit is set, or when the SOCPRICTL register is written. In the latter case, if a conversion is currently in progress, it will complete and then the new priority will take effect. Invalid selection.

**Table 6-11. SOCPRICTL Register Field Descriptions (continued)**

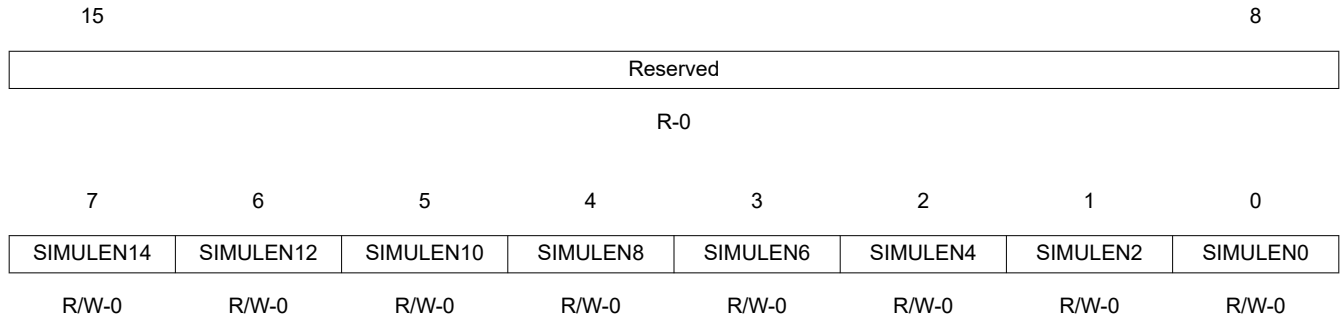
Bit	Field	Value	Description <sup>(1)</sup>
4-0	SOC PRIORITY		SOC Priority. Determines the cutoff point for priority mode and round robin arbitration for SOCx
		00h	SOC priority is handled in round robin mode for all channels.
		01h	SOC0 is high priority, rest of channels are in round robin mode.
		02h	SOC0-SOC1 are high priority, SOC2-SOC15 are in round robin mode.
		03h	SOC0-SOC2 are high priority, SOC3-SOC15 are in round robin mode.
		04h	SOC0-SOC3 are high priority, SOC4-SOC15 are in round robin mode.
		05h	SOC0-SOC4 are high priority, SOC5-SOC15 are in round robin mode.
		06h	SOC0-SOC5 are high priority, SOC6-SOC15 are in round robin mode.
		07h	SOC0-SOC6 are high priority, SOC7-SOC15 are in round robin mode.
		08h	SOC0-SOC7 are high priority, SOC8-SOC15 are in round robin mode.
		09h	SOC0-SOC8 are high priority, SOC9-SOC15 are in round robin mode.
		0Ah	SOC0-SOC9 are high priority, SOC10-SOC15 are in round robin mode.
		0Bh	SOC0-SOC10 are high priority, SOC11-SOC15 are in round robin mode.
		0Ch	SOC0-SOC11 are high priority, SOC12-SOC15 are in round robin mode.
		0Dh	SOC0-SOC12 are high priority, SOC13-SOC15 are in round robin mode.
		0Eh	SOC0-SOC13 are high priority, SOC14-SOC15 are in round robin mode.
		0Fh	SOC0-SOC14 are high priority, SOC15 is in round robin mode.
		10h	All SOCx are in high priority mode, arbitrated by SOC number
		Others	Invalid selection.

(1) This register is EALLOW protected.

## 6.13.5 ADC SOC Registers

### 6.13.5.1 ADC Sample Mode Register (ADCSAMPLEMODE)

**Figure 6-26. ADC Sample Mode Register (ADCSAMPLEMODE)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-12. ADC Sample Mode Register (ADCSAMPLEMODE) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15:8	Reserved	0	Reserved
7	SIMULEN14	0	Simultaneous sampling enable for SOC14/SOC15. Couples SOC14 and SOC15 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC14 or SOC15.
		1	Single sample mode set for SOC14 and SOC15. All bits of CHSEL field define channel to be converted. EOC14 associated with SOC14. EOC15 associated with SOC15. SOC14's result placed in ADCRESULT14 register. SOC15's result placed in ADCRESULT15.
		1	Simultaneous sample for SOC14 and SOC15. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC14 and EOC15 associated with SOC14 and SOC15 pair. SOC14's and SOC15's results will be placed in ADCRESULT14 and ADCRESULT15 registers, respectively.
6	SIMULEN12	0	Simultaneous sampling enable for SOC12/SOC13. Couples SOC12 and SOC13 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC12 or SOC13.
		1	Single sample mode set for SOC12 and SOC13. All bits of CHSEL field define channel to be converted. EOC12 associated with SOC12. EOC13 associated with SOC13. SOC12's result placed in ADCRESULT12 register. SOC13's result placed in ADCRESULT13.
		1	Simultaneous sample for SOC12 and SOC13. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC12 and EOC13 associated with SOC12 and SOC13 pair. SOC12's and SOC13's results will be placed in ADCRESULT12 and ADCRESULT13 registers, respectively.
5	SIMULEN10	0	Simultaneous sampling enable for SOC10/SOC11. Couples SOC10 and SOC11 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC10 or SOC11.
		1	Single sample mode set for SOC10 and SOC11. All bits of CHSEL field define channel to be converted. EOC10 associated with SOC10. EOC11 associated with SOC11. SOC10's result placed in ADCRESULT10 register. SOC11's result placed in ADCRESULT11.
		1	Simultaneous sample for SOC10 and SOC11. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC10 and EOC11 associated with SOC10 and SOC11 pair. SOC10's and SOC11's results will be placed in ADCRESULT10 and ADCRESULT11 registers, respectively.
4	SIMULEN8	0	Simultaneous sampling enable for SOC8/SOC9. Couples SOC8 and SOC9 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC8 or SOC9.
		1	Single sample mode set for SOC8 and SOC9. All bits of CHSEL field define channel to be converted. EOC8 associated with SOC8. EOC9 associated with SOC9. SOC8's result placed in ADCRESULT8 register. SOC9's result placed in ADCRESULT9.
		1	Simultaneous sample for SOC8 and SOC9. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC8 and EOC9 associated with SOC8 and SOC9 pair. SOC8's and SOC9's results will be placed in ADCRESULT8 and ADCRESULT9 registers, respectively.

**Table 6-12. ADC Sample Mode Register (ADCSAMPLEMODE) Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
3	SIMULEN6	0 1	<p>Simultaneous sampling enable for SOC6/SOC7. Couples SOC6 and SOC7 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC6 or SOC7.</p> <p>0 Single sample mode set for SOC6 and SOC7. All bits of CHSEL field define channel to be converted. EOC6 associated with SOC6. EOC7 associated with SOC7. SOC6's result placed in ADCRESULT6 register. SOC7's result placed in ADCRESULT7.</p> <p>1 Simultaneous sample for SOC6 and SOC7. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC6 and EOC7 associated with SOC6 and SOC7 pair. SOC6's and SOC7's results will be placed in ADCRESULT6 and ADCRESULT7 registers, respectively.</p>
2	SIMULEN4	0 1	<p>Simultaneous sampling enable for SOC4/SOC5. Couples SOC4 and SOC5 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC4 or SOC5.</p> <p>0 Single sample mode set for SOC4 and SOC5. All bits of CHSEL field define channel to be converted. EOC4 associated with SOC4. EOC5 associated with SOC5. SOC4's result placed in ADCRESULT4 register. SOC5's result placed in ADCRESULT5.</p> <p>1 Simultaneous sample for SOC4 and SOC5. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC4 and EOC5 associated with SOC4 and SOC5 pair. SOC4's and SOC5's results will be placed in ADCRESULT4 and ADCRESULT5 registers, respectively.</p>
1	SIMULEN2	0 1	<p>Simultaneous sampling enable for SOC2/SOC3. Couples SOC2 and SOC3 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC2 or SOC3.</p> <p>0 Single sample mode set for SOC2 and SOC3. All bits of CHSEL field define channel to be converted. EOC2 associated with SOC2. EOC3 associated with SOC3. SOC2's result placed in ADCRESULT2 register. SOC3's result placed in ADCRESULT3.</p> <p>1 Simultaneous sample for SOC2 and SOC3. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC2 and EOC3 associated with SOC2 and SOC3 pair. SOC2's and SOC3's results will be placed in ADCRESULT2 and ADCRESULT3 registers, respectively.</p>
0	SIMULEN0	0 1	<p>Simultaneous sampling enable for SOC0/SOC1. Couples SOC0 and SOC1 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC0 or SOC1.</p> <p>0 Single sample mode set for SOC0 and SOC1. All bits of CHSEL field define channel to be converted. EOC0 associated with SOC0. EOC1 associated with SOC1. SOC0's result placed in ADCRESULT0 register. SOC1's result placed in ADCRESULT1.</p> <p>1 Simultaneous sample for SOC0 and SOC1. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC0 and EOC1 associated with SOC0 and SOC1 pair. SOC0's and SOC1's results will be placed in ADCRESULT0 and ADCRESULT1 registers, respectively.</p>

(1) This register is EALLOW protected.

### 6.13.5.2 ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1)

**Figure 6-27. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SOC7		SOC6		SOC5		SOC4		SOC3		SOC2		SOC1		SOC0	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-13. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15--0	SOCx (x = 7 to 0)		SOCx ADC Interrupt Trigger Select. Select ADCINT to trigger SOCx. The ADCINT trigger is OR'ed with the trigger selected by the TRIGSEL field in the ADCSOCxCTL register, as well as the software force trigger signal from the ADCSOCFRC1 register.
		00	No ADCINT will trigger SOCx.
		01	ADCINT1 will trigger SOCx.
		10	ADCINT2 will trigger SOCx.
		11	Invalid selection.

(1) This register is EALLOW protected.

### 6.13.5.3 ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2)

**Figure 6-28. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SOC15		SOC14		SOC13		SOC12		SOC11		SOC10		SOC9		SOC8	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-14. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-0	SOCx (x = 15 to 8)		SOCx ADC Interrupt Trigger Select. Select ADCINT to trigger SOCx. The ADCINT trigger is OR'ed with the trigger selected by the TRIGSEL field in the ADCSOCxCTL register, as well as the software force trigger signal from the ADCSOCFRC1 register.
		00	No ADCINT will trigger SOCx.
		01	ADCINT1 will trigger SOCx.
		10	ADCINT2 will trigger SOCx.
		11	Invalid selection.

(1) This register is EALLOW protected.

**6.13.5.4 ADC SOC Flag 1 Register (ADCSOCFLG1)**
**Figure 6-29. ADC SOC Flag 1 Register (ADCSOCFLG1)**

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-15. ADC SOC Flag 1 Register (ADCSOCFLG1) Field Descriptions**

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)	0 1	SOCx Start of Conversion Flag. Indicates the state of individual SOC conversions. No sample pending for SOCx. Trigger has been received and sample is pending for SOCx. The bit will be automatically cleared when the respective SOCx conversion is started. If contention exists where this bit receives both a request to set <b>and</b> a request to clear on the same cycle, regardless of the source of either, this bit will be set and the request to clear will be ignored. In this case the overflow bit in the ADCSOCOVF1 register will not be affected regardless of whether this bit was previously set or not.



### 6.13.5.5 ADC SOC Force 1 Register (ADCSOCFRC1)

**Figure 6-30. ADC SOC Force 1 Register (ADCSOCFRC1)**

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-16. ADC SOC Force 1 Register (ADCSOCFRC1) Field Descriptions**

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)	0 1	<p>SOCx Force Start of Conversion Flag. Writing a 1 will force to 1 the respective SOCx flag bit in the ADCSOCFLG1 register. This can be used to initiate a software initiated conversion. Writes of 0 are ignored.</p> <p>No action.</p> <p>Force SOCx flag bit to 1. This will cause a conversion to start once priority is given to SOCx.</p> <p>If software tries to set this bit on the same clock cycle that hardware tries to clear the SOCx bit in the ADCSOCFLG1 register, then software has priority and the ADCSOCFLG1 bit will be set. In this case the overflow bit in the ADCSOCOVF1 register will not be affected regardless of whether the ADCSOCFLG1 bit was previously set or not.</p>

### 6.13.5.6 ADC SOC Overflow 1 Register (ADCSOCOVF1)

**Figure 6-31. ADC SOC Overflow 1 Register (ADCSOCOVF1)**

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-17. ADC SOC Overflow 1 Register (ADCSOCOVF1) Field Descriptions**

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)	0 1	SOCx Start of Conversion Overflow Flag. Indicates an SOCx event was generated while an existing SOCx event was already pending. No SOCx event overflow SOCx event overflow An overflow condition does not stop SOCx events from being processed. It simply is an indication that a trigger was missed.

### 6.13.5.7 ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1)

**Figure 6-32. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1)**

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

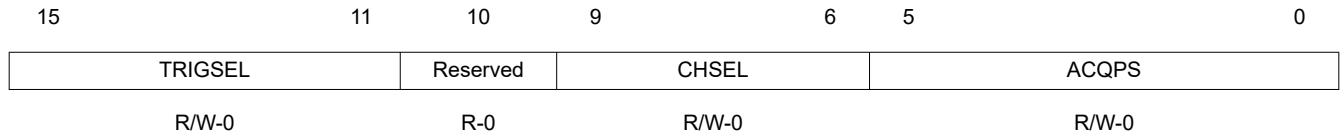
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-18. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1) Field Descriptions**

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)	0 1	SOCx Clear Start of Conversion Overflow Flag. Writing a 1 will clear the respective SOCx overflow flag in the ADCSOCOVF1 register. Writes of 0 are ignored. No action. Clear SOCx overflow flag. If software tries to set this bit on the same clock cycle that hardware tries to set the overflow bit in the ADCSOCOVF1 register, then hardware has priority and the ADCSOCOVF1 bit will be set.

### 6.13.5.8 ADC SOC0-SOC15 Control Registers (ADCSOCxCTL)

**Figure 6-33. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-19. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-11	TRIGSEL	00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch Others	SOCx Trigger Source Select. Configures which trigger will set the respective SOCx flag in the ADCSOCFLG1 register to initiate a conversion to start once priority is given to SOCx. This setting can be overridden by the respective SOCx field in the ADCINTSOCSEL1 or ADCINTSOCSEL2 register. ADCTRIG0 - Software only. ADCTRIG1 - CPU Timer 0, TINT0n ADCTRIG2 - CPU Timer 1, TINT1n ADCTRIG3 - CPU Timer 2, TINT2n ADCTRIG4 - XINT2, XINT2SOC ADCTRIG5 - ePWM1, ADCSOCA ADCTRIG6 - ePWM1, ADCSOCB ADCTRIG7 - ePWM2, ADCSOCA ADCTRIG8 - ePWM2, ADCSOCB ADCTRIG9 - ePWM3, ADCSOCA ADCTRIG10 - ePWM3, ADCSOCB ADCTRIG11 - ePWM4, ADCSOCA ADCTRIG12 - ePWM4, ADCSOCB Invalid selection.
10	Reserved		Reads return a zero; Writes have no effect.

**Table 6-19. ADC SOC0-SOC15 Control Registers (ADC SOCxCTL) Register Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
9-6	CHSEL		SOCx Channel Select. Selects the channel to be converted when SOCx is received by the ADC.
			Sequential Sampling Mode (SIMULENx = 0):
		0h	ADCINA0
		1h	ADCINA1
		2h	ADCINA2
		3h	ADCINA3
		4h	ADCINA4
		5h	ADCINA5
		6h	ADCINA6
		7h	ADCINA7
		8h	ADCINB0
		9h	ADCINB1
		Ah	ADCINB2
		Bh	ADCINB3
		Ch	ADCINB4
		Dh	ADCINB5
		Eh	ADCINB6
		Fh	ADCINB7
			Simultaneous Sampling Mode (SIMULENx = 1):
		0h	ADCINA0/ADCINB0 pair
		1h	ADCINA1/ADCINB1 pair
		2h	ADCINA2/ADCINB2 pair
		3h	ADCINA3/ADCINB3 pair
		4h	ADCINA4/ADCINB4 pair
		5h	ADCINA5/ADCINB5 pair
		6h	ADCINA6/ADCINB6 pair
		7h	ADCINA7/ADCINB7 pair
		8h	Invalid selection.
		9h	Invalid selection.
		Ah	Invalid selection.
		Bh	Invalid selection.
		Ch	Invalid selection.
		Dh	Invalid selection.
		Eh	Invalid selection.
		Fh	Invalid selection.

**Table 6-19. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions (continued)**

Bit	Field	Value	Description <sup>(1)</sup>
5-0	ACQPS		SOCx Acquisition Prescale. Controls the sample and hold window for SOCx.
		00h	Invalid selection.
		01h	Invalid selection.
		02h	Invalid selection.
		03h	Invalid selection.
		04h	Invalid selection.
		05h	Invalid selection.
		06h	Sample window is 7 cycles long (6 + 1 clock cycles).
		07h	Sample window is 8 cycles long (7 + 1 clock cycles).
		08h	Sample window is 9 cycles long (8 + 1 clock cycles).
		09h	Sample window is 10 cycles long (9 + 1 clock cycles).
...	...		
3Fh	Sample window is 64 cycles long (63 + 1 clock cycles).		
<b>Other invalid selections:</b> 10h, 11h, 12h, 13h, 14h, 1Dh, 1Eh, 1Fh, 20h, 21h, 2Ah, 2Bh, 2Ch, 2Dh, 2Eh, 37h, 38h, 39h, 3Ah, 3Bh			

(1) This register is EALLOW protected.

## 6.13.6 ADC Calibration Registers

### 6.13.6.1 ADC Reference/Gain Trim Register (ADCREFTTRIM)

**Figure 6-34. ADC Reference/Gain Trim Register (ADCREFTTRIM)**

15	14	13	9	8	5	4	0
Reserved		EXTREF_FINE_TRIM		BG_COARSE_TRIM		BG_FINE_TRIM	
R-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-20. ADC Reference/Gain Trim Register (ADCREFTTRIM) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-14	Reserved		Reads return a zero; Writes have no effect.
13-9	EXTREF_FINE_TRIM		ADC External reference Fine Trim. These bits should not be modified after device boot code loads them with the factory trim setting.
8-5	BG_COARSE_TRIM		ADC Internal Bandgap Fine Trim. These bits should not be modified after device boot code loads them with the factory trim setting.
4-0	BG_FINE_TRIM		ADC Internal Bandgap Coarse Trim. A maximum value of 30 is supported. These bits should not be modified after device boot code loads them with the factory trim setting.

(1) This register is EALLOW protected.

### 6.13.6.2 ADC Offset Trim Register (ADCOFFTRIM)

**Figure 6-35. ADC Offset Trim Register (ADCOFFTRIM)**

15	9	8	0
Reserved		OFFTRIM	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-21. ADC Offset Trim Register (ADCOFFTRIM) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-9	Reserved		Reads return a zero; Writes have no effect.
8-0	OFFTRIM		ADC Offset Trim. 2's complement of ADC offset. Range is -256 to +255. These bits are loaded by device boot code with a factory trim setting. Modification of this default setting can be made to correct any board induced offset.

(1) This register is EALLOW protected.

### 6.13.7 Comparator Hysteresis Control Register (COMPHYSTCTL)

**Figure 6-36. Comparator Hysteresis Control Register (COMPHYSTCTL)**

15	12	11	10	7	6	5	2	1	0
Reserved	COMP3_HYST_DISABLE	Reserved	COMP2_HYST_DISABLE	Reserved	COMP1_HYST_DISABLE	Reserved	Reserved	Reserved	Reserved
R-0	R/W--0	R-0	R/W--0	R-0	R/W--0	R-0	R/W--0	R-0	R-0

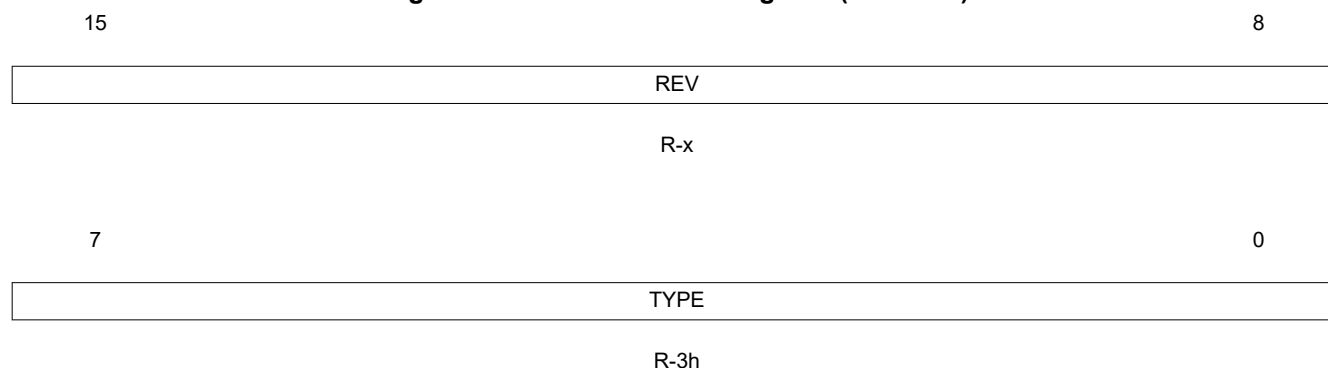
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-22. Comparator Hysteresis Control Register (COMPHYSTCTL) Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-12	Reserved		Reads return a zero; Writes have no effect.
11	COMP3_HYST_DISABLE	0 1	Comparator 3 Hysteresis disable Hysteresis enabled Hysteresis disabled
10-7	Reserved		Reserved
6	COMP2_HYST_DISABLE	0 1	Comparator 2 Hysteresis disable Hysteresis enabled Hysteresis disabled
5-2	Reserved		Reserved
1	COMP1_HYST_DISABLE	0 1	Comparator 1 Hysteresis disable Hysteresis enabled Hysteresis disabled
0	Reserved		Reserved

(1) This register is EALLOW protected.

### 6.13.8 ADC Revision Register (ADCREV)

**Figure 6-37. ADC Revision Register (ADCREV)**


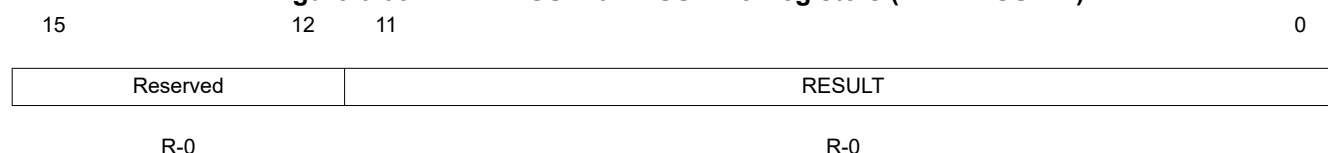
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-23. ADC Revision Register (ADCREV) Field Descriptions**

Bit	Field	Value	Description
15-8	REV		ADC Revision. To allow documentation of differences between revisions. First version is labeled as 00h.
7-0	TYPE	3	ADC Type. Always set to 3 for this type ADC.

### 6.13.9 ADC RESULT0-RESULT15 Registers (ADCRESULTx)

The ADC Result Registers are found in Peripheral Frame 0 (PF0). In the header files, the ADCRESULTx registers are located in the AdcResult register file, not AdcRegs.

**Figure 6-38. ADC RESULT0-RESULT15 Registers (ADCRESULTx)**


LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 6-24. ADC RESULT0-ADCRESULT15 Registers (ADCRESULTx) Field Descriptions**

Bit	Field	Value	Description
15-12	Reserved		Reads return a zero; Writes have no effect.
11-0	RESULT		12-bit right-justified ADC result Sequential Sampling Mode (SIMULENx = 0): After the ADC completes a conversion of an SOCx, the digital result is placed in the corresponding ADCRESULTx register. For example, if SOC4 is configured to sample ADCINA1, the completed result of that conversion will be placed in ADCRESULT4. Simultaneous Sampling Mode (SIMULENx = 1): After the ADC completes a conversion of a channel pair, the digital results are found in the corresponding ADCRESULTx and ADCRESULTx+1 registers (assuming x is even). For example, for SOC4, the completed results of those conversions will be placed in ADCRESULT4 and ADCRESULT5. See 1.11 for timings of when this register is written.



The Comparator (COMP) module described in this chapter is a Type 0 Comparator. See the [C2000 Real-Time Control Peripherals Reference Guide](#) for a list of all devices with modules of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

<b>7.1 Introduction</b> .....	<b>450</b>
<b>7.2 Comparator Function</b> .....	<b>451</b>
<b>7.3 DAC Reference</b> .....	<b>451</b>
<b>7.4 Ramp Generator Input</b> .....	<b>452</b>
<b>7.5 Initialization</b> .....	<b>453</b>
<b>7.6 Digital Domain Manipulation</b> .....	<b>453</b>
<b>7.7 Comparator Registers</b> .....	<b>454</b>

## 7.1 Introduction

The comparator module is a true analog voltage comparator in the VDDA domain. The core analog circuits include the comparator, its inputs and outputs, and the internal DAC reference. The supporting digital circuits include the DAC controls, interface to other on-chip logic, output qualification block, and the programmable control signals.

### 7.1.1 Features

The comparator block can monitor two external analog inputs, or monitor one external analog input using the internal DAC reference for the other input. The output of the comparator can be passed asynchronously, or be qualified and synchronized to the system clock period. The comparator output is routed to both the ePWM Trip Zone modules, as well as the GPIO output multiplexer.

### 7.1.2 Block Diagram

#### Note

Comparator hysteresis feedback is enabled by default and may interfere with high-impedance input signals. Use the COMPHYSTCTL register from the ADC module to configure the comparator hysteresis.

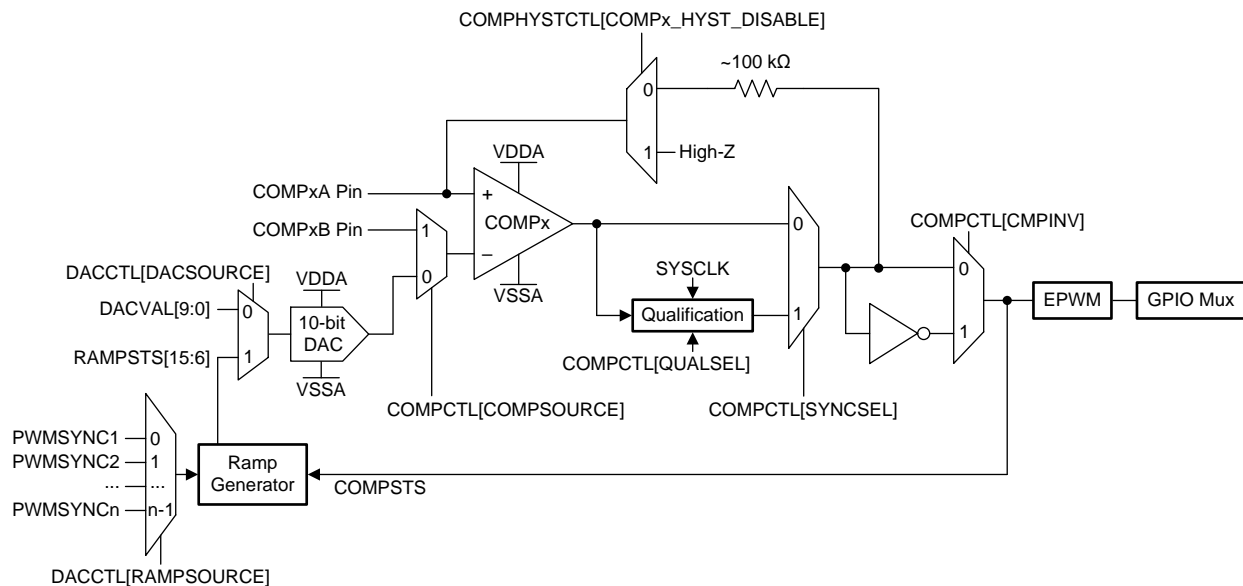
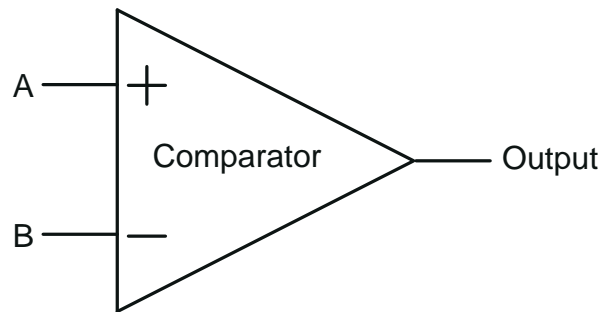


Figure 7-1. Comparator Block Diagram

## 7.2 Comparator Function

The comparator is an analog comparator module (Figure 7-2) and the output is asynchronous to the system clock. The truth table for the comparator is shown in Table 7-1.



**Figure 7-2. Comparator**

**Table 7-1. Comparator Truth Table**

Voltages	Output
Voltage A > Voltage B	1
Voltage B > Voltage A	0

There is no definition for the condition Voltage A = Voltage B, since there is hysteresis in the response of the comparator output. Refer to the device data sheet for the value of this hysteresis. This also limits the sensitivity of the comparator output to noise on the input voltages.

The output state of the comparator, after qualification, is reflected by the COMPSTS bit in the COMPSTS register. The COMPSTS register will not update if the module clock is not enabled.

## 7.3 DAC Reference

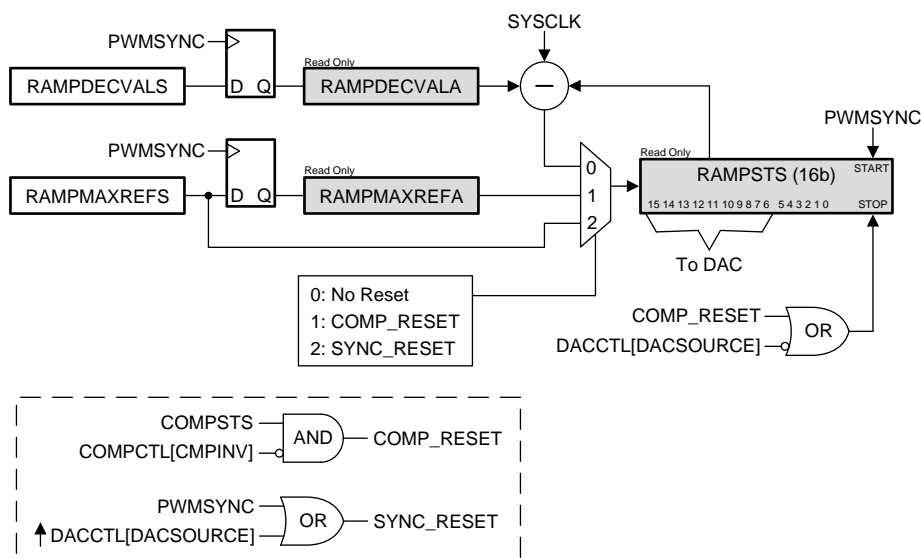
Each comparator block contains an internal 10-bit voltage DAC reference that can be used to supply the inverting input (B side input) of the comparator. The voltage output of the DAC is controlled by the DACVAL bit field in the DACVAL register. The output of the DAC is given by the equation:

$$V = \frac{\text{DACVAL} * (\text{VDDA} - \text{VSSA})}{1023}$$

Since the DAC is also in the analog domain it does not require a clock to maintain its voltage output. A clock is required, however, to modify the digital inputs that control the DAC.

## 7.4 Ramp Generator Input

When selected, the ramp generator (see [Figure 7-3](#)) can produce a falling-ramp DAC output signal. In this mode, the DAC uses the most significant 10-bits of the 16-bit RAMPSTS countdown register as its input.



**Figure 7-3. Ramp Generator Block Diagram**

### Note

The PWMSYNC signal for the Ramp Generator is derived from the HRPWM register field HRPCTL[PWMSYNCSEL]. The PWMSYNC signal is not the same as the EPWMSYNCl and EPWMSYNCO signals.

The RAMPSTS register is set to the value of RAMPMAXREF\_SHDW when a selected PWMSYNC signal is received, and the value of RAMPDECVAL\_ACTIVE is subtracted from RAMPSTS on every SYSCLK cycle thereafter. When the ramp generator is first enabled by setting DACSOURCE = 1, the value of RAMPSTS is loaded from RAMPMAXREF\_SHDW, and the register remains static until the first PWMSYNC signal is received.

If the COMPSTS bit is set by the comparator while the ramp generator is active, the RAMPSTS register will reset to the value of RAMPMAXREF\_ACTIVE and remain static until the next PWMSYNC signal is received. If the value of RAMPSTS reaches zero, the RAMPSTS register will remain static at zero until the next PWMSYNC signal is received.

To reduce the likelihood of race conditions when updating the ramp generator RAMPMAXREFA and RAMPDECVALA values, only the shadow registers RAMPMAXREF\_SHDW and RAMPDECVAL\_SHDW have write permissions. The values of the shadow registers are copied to the active registers on the next PWMSYNC signal. User software should take further steps to avoid writing to the shadow registers in the same cycle as a PWMSYNC signal or else the previous shadow register value may be lost.

The PWMSYNC signal width must be greater than SYSCLK to ensure that the ramp generator is able to detect the PWMSYNC signal.

The ramp generator behavior is further illustrated in [Figure 7-4](#).

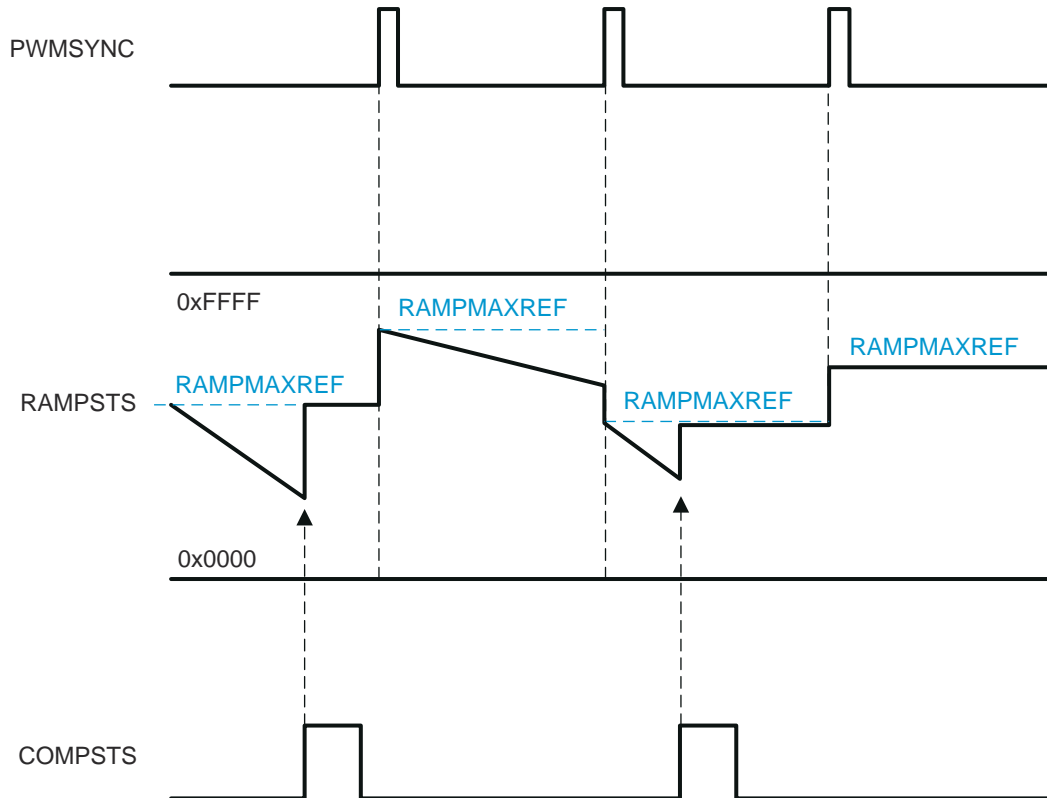


Figure 7-4. Ramp Generator Behavior

## 7.5 Initialization

There are 2 steps that must be performed prior to using the comparator block:

1. Enable the Band Gap inside the ADC by writing a 1 to the ADCBGPWD bit inside ADCCTL1.
2. Enable the comparator block by writing a 1 to the COMPDACEN bit in the COMPCTL register.

## 7.6 Digital Domain Manipulation

At the output of the comparator there are two more functional blocks that can be used to influence the behavior of the comparator output. They are:

1. Inverter circuit: Controlled by the CMPINV bit in the COMPCTL register; will apply a logical NOT to the output of the comparator. This function is asynchronous, while its control requires a clock present in order to change its value.
2. Qualification block: Controlled by the QUALSEL bit field in the COMPCTL register, and gated by the SYNCSEL bit in the COMPCTL register. This block can be used as a simple filter to only pass the output of the comparator once it is synchronized to the system clock. and qualified by the number of system clocks defined in QUALSEL bit field.

## 7.7 Comparator Registers

This device has the comparator modules listed in [Table 7-2](#) and the registers listed in [Table 7-3](#).

**Table 7-2. Comparator Modules**

Name	Address Range	Size(x16)	Description
COMP1	6400h – 641Fh	1	Comparator
COMP2	6420h – 643Fh	1	Comparator

**Table 7-3. Comparator Module Registers**

Name	Address Range (base)	Size (x16)	Description	Bit Description
COMPCTL	0x00	1	Comparator Control <sup>(1)</sup>	<a href="#">Section 7.7.1</a>
Reserved	0x01	1	Reserved	-
COMPSTS	0x02	1	Compare Output Status	<a href="#">Section 7.7.2</a>
Reserved	0x03	1	Reserved	-
DACCTL	0x04	1	DAC Control <sup>(1)</sup>	<a href="#">Section 7.7.3</a>
Reserved	0x05	1	Reserved	-
DACVAL	0x06	1	10-bit DAC Value	<a href="#">Section 7.7.4</a>
Reserved	0x07	1	Reserved	-
RAMPMAXREF_ACTIVE	0x08	1	Ramp Generator Maximum Reference (Active)	<a href="#">Section 7.7.5</a>
Reserved	0x09	1	Reserved	-
RAMPMAXREF_SHDW	0x0A	1	Ramp Generator Maximum Reference (Shadow)	<a href="#">Section 7.7.6</a>
Reserved	0x0B	1	Reserved	-
RAMPDECVAL_ACTIVE	0x0C	1	Ramp Generator Decrement Value (Active)	<a href="#">Section 7.7.7</a>
Reserved	0x0D	1	Reserved	-
RAMPDECVAL_SHDW	0x0E	1	Ramp Generator Decrement Value (Shadow)	<a href="#">Section 7.7.8</a>
Reserved	0x0F	1	Reserved	-
RAMPSTS	0x10	1	Ramp Generator Status	<a href="#">Section 7.7.9</a>
Reserved	0x11-0x1F	15	Reserved	-

(1) This register is EALLOW protected.

### 7.7.1 Comparator Control (COMPCTL) Register

**Figure 7-5. Comparator Control (COMPCTL) Register**

15	Reserved			9	8
				SYNCSEL	
R-0				R/W-0	
7	3	2	1	0	
QUALSEL			CMPINV	COMPSOURCE	COMPDACEN
R/W-0			R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 7-4. Comparator Control (COMPCTL) Register Field Descriptions**

Bit	Field	Value	Description <sup>(1)</sup>
15-9	Reserved		Reads return a 0; Writes have no effect.
8	SYNCSEL	0 1	Synchronization select for output of the comparator before being passed to EPWM/GPIO blocks Asynchronous version of Comparator output is passed Synchronous version of comparator output is passed
7-3	QUALSEL	0h 1h 2h ... 1Fh	Qualification Period for synchronized output of the comparator Synchronized value of comparator is passed through Input to the block must be consistent for 2 consecutive clocks before output of Qual block can change Input to the block must be consistent for 3 consecutive clocks before output of Qual block can change ... Input to the block must be consistent for 32 consecutive clocks before output of Qual block can change
2	CMPINV	0 1	Invert select for Comparator Output of comparator is passed Inverted output of comparator is passed
1	COMPSOURCE	0 1	Source select for comparator inverting input Inverting input of comparator connected to internal DAC Inverting input connected to external pin
0	COMPDACEN	0 1	Comparator/DAC Enable Comparator/DAC logic is powered down. Comparator/DAC logic is powered up.

(1) This register is EALLOW protected.

## 7.7.2 Compare Output Status (COMPSTS) Register

**Figure 7-6. Compare Output Status (COMPSTS) Register**

15	1	0
Reserved	COMPSTS	
R-0	R-0	

LEGEND: R = Read only; -n = value after reset

**Table 7-5. Compare Output Status (COMPSTS) Register Field Descriptions**

Bit	Field	Value	Description
15-1	Reserved		Reads return zero and writes have no effect.
0	COMPSTS		Logical latched value of the comparator.

## 7.7.3 DAC Control (DACCTL) Register

**Figure 7-7. DAC Control (DACCTL) Register**

15	14	13	8
FREE:SOFT		Reserved	
R/W-0		R-0	
7	5	4	0
Reserved		RAMPSOURCE	DACSOURCE
R-0		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 7-6. DAC Control (DACCTL) Register Field Descriptions**

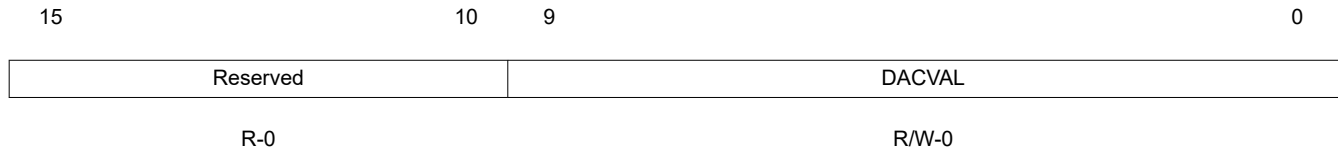
Bit	Field	Value	Description <sup>(1)</sup>
15-14	FREE:SOFT	0h 1h 2h-3h	Emulation mode behavior. Selects ramp generator behavior during emulation suspend. Stop immediately Complete current ramp, and stop on the next PWMSYNC signal Run free
13-5	Reserved		Reads return a 0; Writes have no effect.
4-1	RAMPSOURCE	0h 1h 2h ... n-1	Ramp generator source sync select. PWMSYNC is derived from the HRPWM register field HRPCTL[PWMSYNCSSEL]. PWMSYNC1 is the source sync PWMSYNC2 is the source sync PWMSYNC3 is the source sync ... PWMSYNcn is the source sync
0	DACSOURCE	0 1	DAC source control. Select DACVAL or ramp generator to control the DAC. DAC controlled by DACVAL DAC controlled by ramp generator

(1) This register is EALLOW protected.



### 7.7.4 DAC Value (DACVAL) Register

**Figure 7-8. DAC Value (DACVAL) Register**



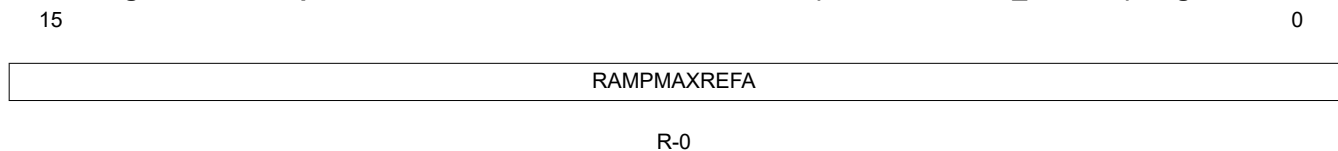
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 7-7. DAC Value (DACVAL) Register Field Descriptions**

Bit	Field	Value	Description
15-10	Reserved		Reads return zero and writes have no effect.
9-0	DACVAL	0-3FFh	DAC Value bits, scales the output of the DAC from 0 – 1023.

### 7.7.5 Ramp Generator Maximum Reference Active (RAMPMAXREF\_ACTIVE) Register

**Figure 7-9. Ramp Generator Maximum Reference Active (RAMPMAXREF\_ACTIVE) Register**



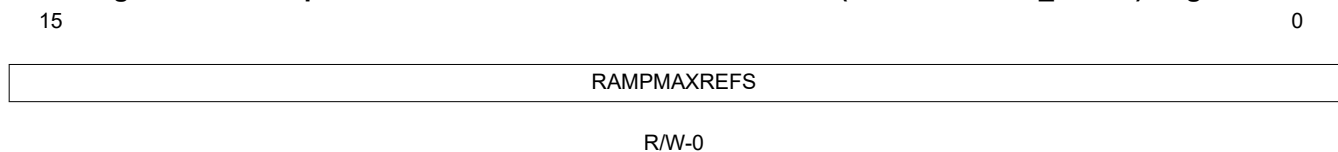
LEGEND: R = Read only; -n = value after reset

**Table 7-8. Ramp Generator Maximum Reference Active (RAMPMAXREF\_ACTIVE) Register Field Descriptions**

Bit	Field	Value	Description
15-0	RAMPMAXREFA	0-FFFFh	16-bit maximum reference active value for down ramp generator. This value is loaded from RAMPMAXREF_SHDW when the PWMSYNC signal is received.

### 7.7.6 Ramp Generator Maximum Reference Shadow (RAMPMAXREF\_SHDW) Register

**Figure 7-10. Ramp Generator Maximum Reference Shadow (RAMPMAXREF\_SHDW) Register**



LEGEND: R/W = Read/Write; -n = value after reset

**Table 7-9. Ramp Generator Maximum Reference Shadow (RAMPMAXREF\_SHDW) Register Field Descriptions**

Bit	Field	Value	Description
15-0	RAMPMAXREFS	0-FFFFh	16-bit maximum reference shadow value for down ramp generator.

### 7.7.7 Ramp Generator Decrement Value Active (RAMPDECVAL\_ACTIVE) Register

**Figure 7-11. Ramp Generator Decrement Value Active (RAMPDECVAL\_ACTIVE) Register**

15 0

RAMPDECVALA
-------------

R-0

LEGEND: R = Read only; -n = value after reset

**Table 7-10. Ramp Generator Decrement Value Active (RAMPDECVAL\_ACTIVE) Register Field Descriptions**

Bit	Field	Value	Description
15-0	RAMPDECVALA	0-FFFFh	16-bit decrement active value for down ramp generator. This value is loaded from RAMPDECVAL_SHDW when the PWMSYNC signal is received.

### 7.7.8 Ramp Generator Decrement Value Shadow (RAMPDECVAL\_SHDW) Register

**Figure 7-12. Ramp Generator Decrement Value Shadow (RAMPDECVAL\_SHDW) Register**

15 0

RAMPDECVALS
-------------

R/W-0

LEGEND: R/W = Read/Write; -n = value after reset

**Table 7-11. Ramp Generator Decrement Value Shadow (RAMPDECVAL\_SHDW) Register Field Descriptions**

Bit	Field	Value	Description
15-0	RAMPDECVALS	0-FFFFh	16-bit decrement shadow value for down ramp generator.

### 7.7.9 Ramp Generator Status (RAMPSTS) Register

**Figure 7-13. Ramp Generator Status (RAMPSTS) Register**

15 0

RAMPVALUE
-----------

R-0

LEGEND: R = Read only; -n = value after reset

**Table 7-12. Ramp Generator Status (RAMPSTS) Register Field Descriptions**

Bit	Field	Value	Description
15-0	RAMPVALUE	0-FFFFh	16-bit value of down ramp generator.

This chapter describes the serial peripheral interface (SPI) which is a high-speed synchronous serial input and output (I/O) port that allows a serial bit stream of programmed length (one to 16 bits) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI is normally used for communications between the MCU controller and external peripherals or another controller. Typical applications include external I/O or peripheral expansion using devices such as shift registers, display drivers, and analog-to-digital converters (ADCs). Multi-device communications are supported by the master or slave operation of the SPI. The port supports a 4-level, receive and transmit FIFO for reducing CPU servicing overhead.

<b>8.1 Introduction</b> .....	<b>460</b>
<b>8.2 System-Level Integration</b> .....	<b>462</b>
<b>8.3 SPI Operation</b> .....	<b>465</b>
<b>8.4 Programming Procedure</b> .....	<b>474</b>
<b>8.5 SPI Registers</b> .....	<b>477</b>

## 8.1 Introduction

### 8.1.1 Features

The SPI module features include:

- SPISOMI: SPI slave-output/master-input pin
- SPISIMO: SPI slave-input/master-output pin
- $\overline{\text{SPISTE}}$ : SPI slave transmit-enable pin
- SPICLK: SPI serial-clock pin

---

#### Note

All four pins can be used as GPIO if the SPI module is not used.

---

- Two operational modes: Master and Slave
- Baud rate: 125 different programmable rates. The maximum baud rate that can be employed is limited by the maximum speed of the I/O buffers used on the SPI pins. See the device-specific data sheet for more details.
- Data word length: 1 to 16 data bits
- Four clocking schemes (controlled by clock polarity and clock phase bits) include:
  - Falling edge without phase delay: SPICLK active-high. SPI transmits data on the falling edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.
  - Falling edge with phase delay: SPICLK active-high. SPI transmits data one half-cycle ahead of the falling edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
  - Rising edge without phase delay: SPICLK inactive-low. SPI transmits data on the rising edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
  - Rising edge with phase delay: SPICLK inactive-low. SPI transmits data one half-cycle ahead of the rising edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.
- Simultaneous receive and transmit operation (transmit function can be disabled in software)
- Transmitter and receiver operations are accomplished through either interrupt- driven or polled algorithm
- 4-level transmit/receive FIFO
- Delayed transmit control
- 3-wire SPI mode

### 8.1.2 Block Diagram

Figure 8-1 shows the SPI CPU interfaces.

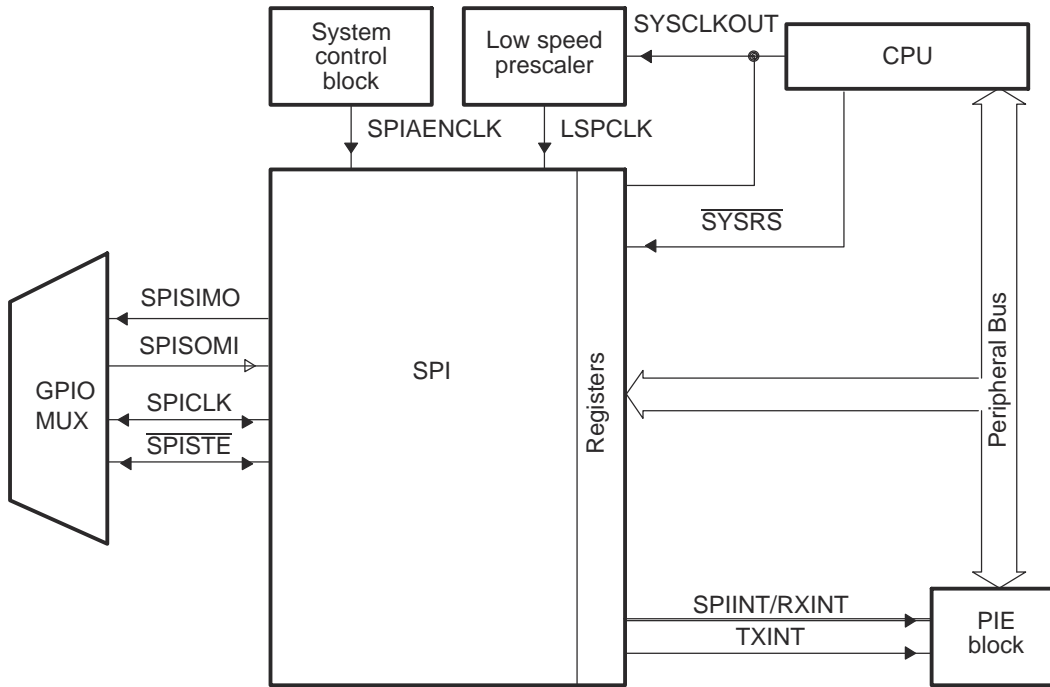


Figure 8-1. SPI CPU Interface

## 8.2 System-Level Integration

This section describes the various functionality that is applicable to the device integration. These features require configuration of other modules in the device that are not within the scope of this chapter.

### 8.2.1 SPI Module Signals

[Table 8-1](#) classifies and provides a summary of the SPI module signals.

**Table 8-1. SPI Module Signal Summary**

Signal Name	Description
<b>External Signals</b>	
SPICLK	SPI clock
SPISIMO	SPI slave in, master out
SPISOMI	SPI slave out, master in
$\overline{\text{SPISTE}}$	SPI slave transmit enable
<b>Control</b>	
SPI Clock Rate	LSPCLK
<b>Interrupt Signals</b>	
SPIINT/SPIRXINT	Transmit interrupt/ Receive Interrupt in non FIFO mode (referred to as SPIINT) Receive interrupt in FIFO mode
SPITXINT	Transmit interrupt in FIFO mode

### Special Considerations

The  $\overline{\text{SPISTE}}$  signal provides the ability to gate any spurious clock and data pulses when the SPI is in slave mode. A HIGH logic signal on  $\overline{\text{SPISTE}}$  will not allow the slave to receive data. This prevents the SPI slave from losing synchronization with the master. It is this reason that TI does not recommend that the  $\overline{\text{SPISTE}}$  always be tied to the active state.

If the SPI slave does ever lose synchronization with the master, toggling SPISWRESET resets the internal bit counter as well as the various status flags in the module. By resetting the bit counter, the SPI interprets the next clock transition as the first bit of a new transmission. The register bit fields that are reset by SPISWRESET are found in [Section 8.5](#).

### Configuring a GPIO to Emulate $\overline{\text{SPISTE}}$

In many systems, a SPI master may be connected to multiple SPI slaves using multiple instances of  $\overline{\text{SPISTE}}$ . Though this SPI module does not natively support multiple  $\overline{\text{SPISTE}}$  signals, it is possible to emulate this behavior in software using GPIOs. In this configuration, the SPI must be configured as the master. Rather than using the GPIO Mux to select  $\overline{\text{SPISTE}}$ , the application would configure pins to be GPIO outputs, one GPIO per SPI slave. Before transmitting any data, the application would drive the desired GPIO to the active state. Immediately after the transmission has been completed, the GPIO chip select would be driven to the inactive state. This process can be repeated for many slaves that share the SPICLK, SPISIMO, and SPISOMI lines.

### 8.2.2 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPYPUD register.

See the *GPIO* chapter for more details on GPIO mux and settings.

### 8.2.3 SPI Interrupts

This section includes information on the available interrupts present in the SPI module.

The SPI module contains two interrupt lines: SPIINT/SPIRXINT and SPITXINT. When the SPI is operating in non-FIFO mode, all available interrupts are routed together to generate the single SPIINT interrupt. When FIFO mode is used, both SPIRXINT and SPITXINT can be generated.

#### SPIINT/SPIRXINT

When the SPI is operating in non-FIFO mode, the interrupt generated is called SPIINT. If FIFO enhancements are enabled, the interrupt is called SPIRXINT. These interrupts share the same interrupt vector in the Peripheral Interrupt Expansion (PIE) block.

In non-FIFO mode, two conditions can trigger an interrupt: a transmission is complete (INT\_FLAG), or there is overrun in the receiver (OVERRUN\_FLAG). Both of these conditions share the same interrupt vector: SPIINT.

The transmission complete flag (INT\_FLAG) indicates that the SPI has completed sending or receiving the last bit and is ready to be serviced. At the same time this bit is set, the received character is placed in the receiver buffer (SPIRXBUF). The INT\_FLAG will generate an interrupt on the SPIINT vector if the SPIINTENA bit is set.

The receiver overrun flag (OVERRUN\_FLAG) indicates that a transmit or receive operation has completed before the previous character has been read from the buffer. The OVERRUN\_FLAG will generate an interrupt on the SPIINT vector if the OVERRUNINTENA bit is set and OVERRUN\_FLAG was previously cleared.

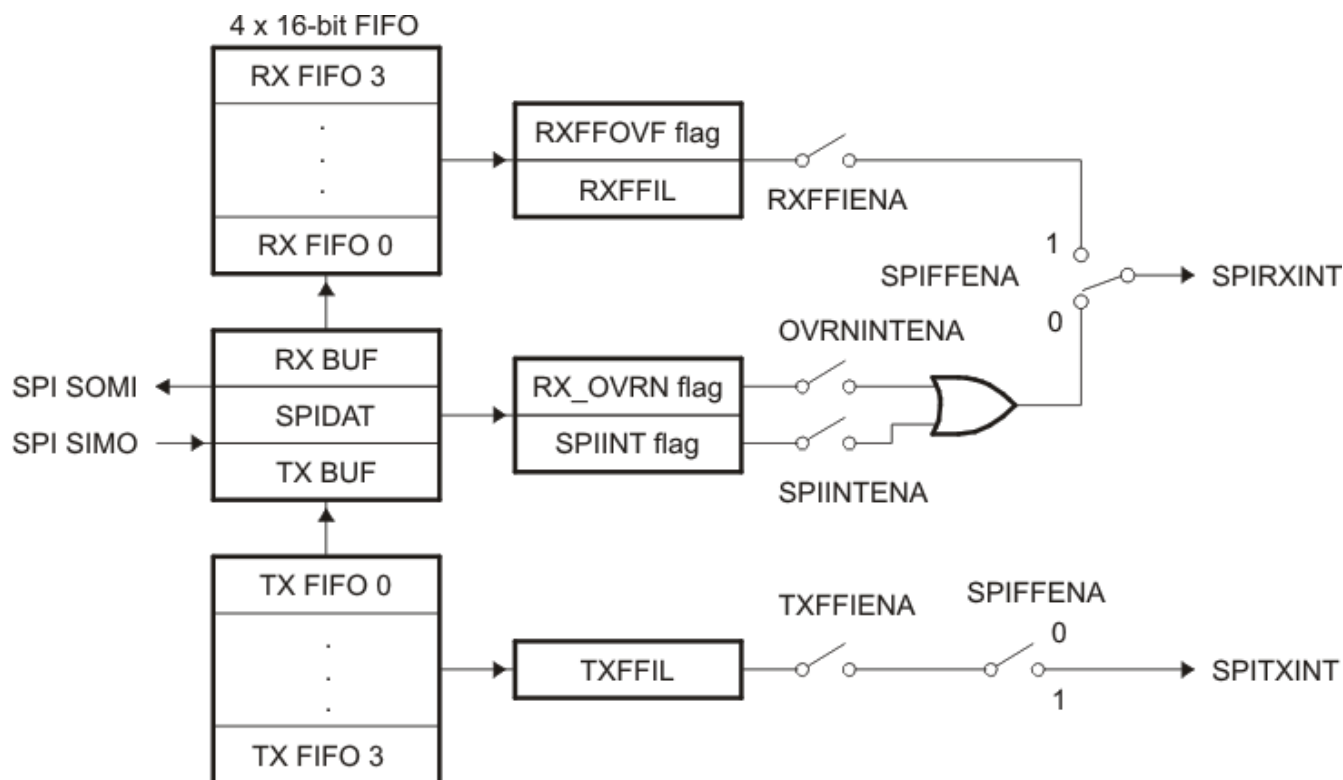
In FIFO mode, the SPI can interrupt the CPU upon a match condition between the current receive FIFO status (RXFFST) and the receive FIFO interrupt level (RXFFIL). If RXFFST is greater than or equal to RXFFIL, the receive FIFO interrupt flag (RXFFINT) will be set. SPIRXINT will be triggered in the PIE block if RXFFINT is set and the receive FIFO interrupt is enabled (RXFFIENA = 1).

#### SPITXINT

The SPITXINT interrupt is not available when the SPI is operating in non-FIFO mode.

In FIFO mode, the SPITXINT behavior is similar to the SPIRXINT. SPITXINT is generated upon a match condition between the current transmit FIFO status (TXFFST) and the transmit FIFO interrupt level (TXFFIL). If TXFFST is less than or equal to TXFFIL, the transmit FIFO interrupt flag (TXFFINT) will be set. SPITXINT will be triggered in the PIE block if TXFFINT is set and the transmit FIFO interrupt is enabled in the SPI module (TXFFIENA = 1).

[Figure 8-2](#) and [Table 8-2](#) show how these control bits influence the SPI interrupt generation.


**Figure 8-2. SPI Interrupt Flags and Enable Logic Generation**
**Table 8-2. SPI Interrupt Flag Modes**

FIFO Options	SPI Interrupt Source	Interrupt Flags	Interrupt Enables	FIFO Enable (SPIFFENA)	Interrupt Line <sup>(1)</sup>
SPI without FIFO	Receive overrun	RXOVRN	OVRNINTENA	0	SPIRXINT
	Data receive	SPIINT	SPIINTENA	0	SPIRXINT
	Transmit empty	SPIINT	SPIINTENA	0	SPIRXINT
SPI FIFO mode	FIFO receive	RXFFIL	RXFFIENA	1	SPIRXINT
	Transmit empty	TXFFIL	TXFFIENA	1	SPITXINT

(1) In non-FIFO mode, SPIRXINT is the same name as the SPIINT interrupt in C28x devices.



### 8.3 SPI Operation

This section describes the various modes of operation of the SPI. Included are explanations of the operational modes, interrupts, data format, clock sources, and initialization. Typical timing diagrams for data transfers are given.

#### 8.3.1 Introduction to Operation

Figure 8-3 shows typical connections of the SPI for communications between two controllers: a master and a slave.

The master transfers data by sending the SPICLK signal. For both the slave and the master, data is shifted out of the shift registers on one edge of the SPICLK and latched into the shift register on the opposite SPICLK clock edge. If the CLK\_PHASE bit is high, data is transmitted and received a half-cycle before the SPICLK transition. As a result, both controllers send and receive data simultaneously. The application software determines whether the data is meaningful or dummy data. There are three possible methods for data transmission:

- Master sends data; slave sends dummy data.
- Master sends data; slave sends data.
- Master sends dummy data; slave sends data.

The master can initiate data transfer at any time because it controls the SPICLK signal. The software, however, determines how the master detects when the slave is ready to broadcast data.

The SPI can operate in master or slave mode. The MASTER\_SLAVE bit selects the operating mode and the source of the SPICLK signal.

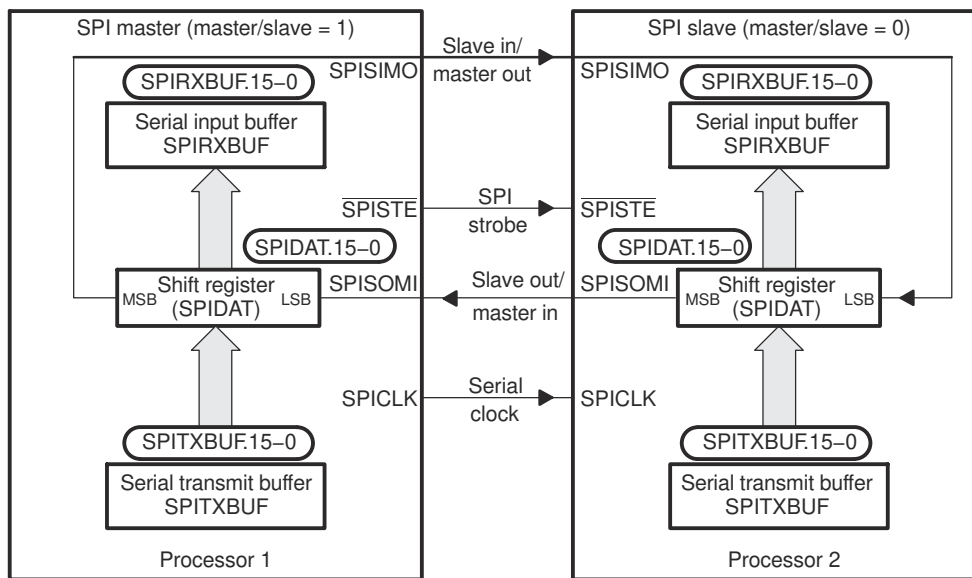


Figure 8-3. SPI Master/Slave Connection

Figure 8-4 is a block diagram of the SPI module showing all of the basic control blocks available on the SPI module.

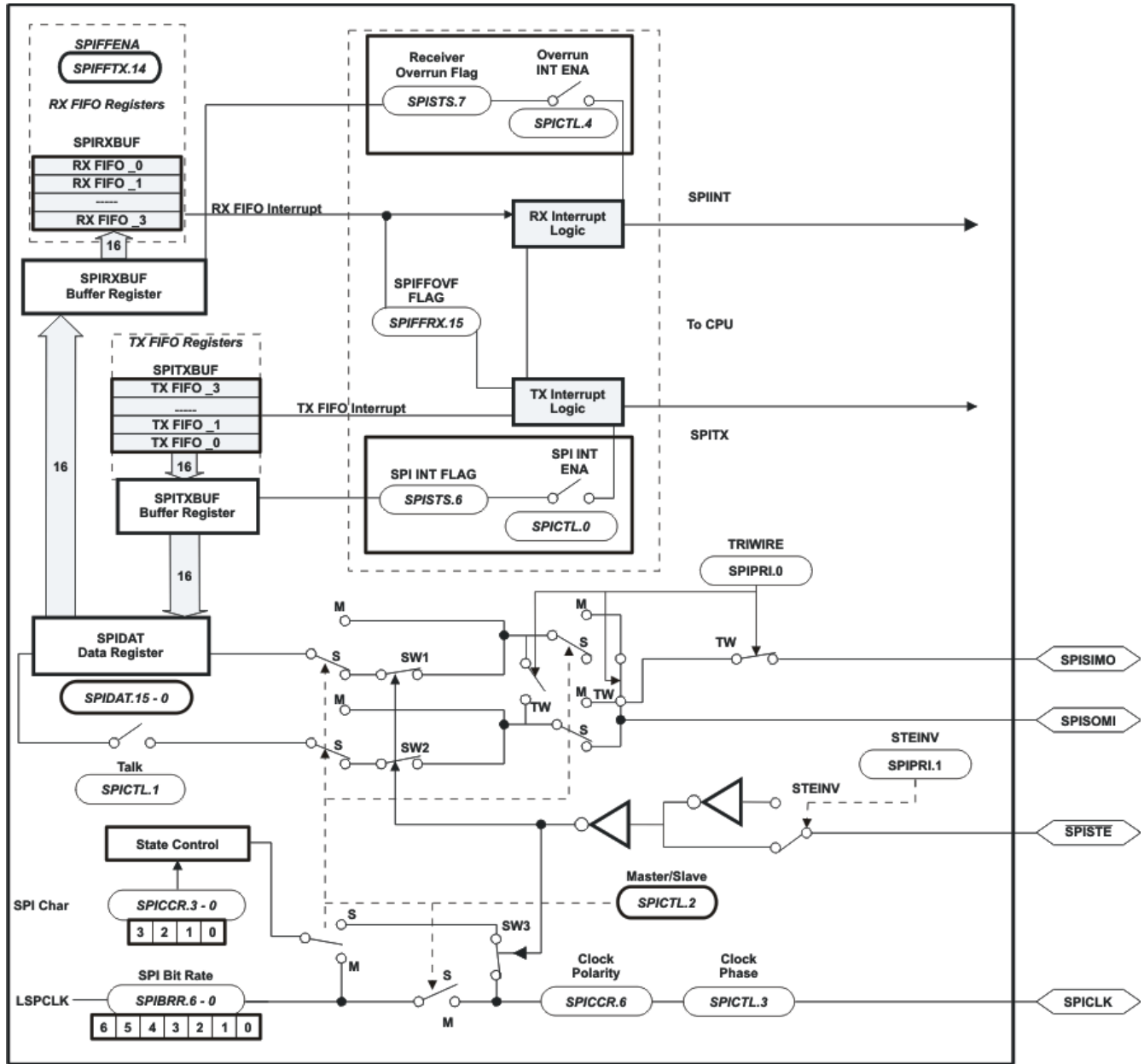


Figure 8-4. Serial Peripheral Interface Block Diagram

### 8.3.2 Master Mode

In master mode (MASTER\_SLAVE = 1), the SPI provides the serial clock on the SPICLK pin for the entire serial communications network. Data is output on the SPISIMO pin and latched from the SPISOMI pin.

The SPIBRR register determines both the transmit and receive bit transfer rate for the network. SPIBRR can select 125 different data transfer rates.

Data written to SPIDAT or SPITXBUF initiates data transmission on the SPISIMO pin, MSB (most-significant bit) first. Simultaneously, received data is shifted through the SPISOMI pin into the LSB (least-significant bit) of SPIDAT. When the selected number of bits has been transmitted, the received data is transferred to the SPIRXBUF (buffered receiver) for the CPU to read. Data is stored right-justified in SPIRXBUF.

When the specified number of data bits has been shifted through SPIDAT, the following events occur:

- SPIDAT contents are transferred to SPIRXBUF.
- INT\_FLAG bit is set to 1.
- If there is valid data in the transmit buffer SPITXBUF, as indicated by the transmit buffer full flag (BUFFULL\_FLAG), this data is transferred to SPIDAT and is transmitted; otherwise, SPICLK stops after all bits have been shifted out of SPIDAT.
- If the SPIINTENA bit is set to 1, an interrupt is asserted.

In a typical application, the  $\overline{\text{SPISTE}}$  pin serves as a chip-enable pin for a SPI slave device. This pin is driven low by the master before transmitting data to the slave and is taken high after the transmission is complete.

### 8.3.3 Slave Mode

In slave mode (MASTER\_SLAVE = 0), data shifts out on the SPISOMI pin and in on the SPISIMO pin. The SPICLK pin is used as the input for the serial shift clock, which is supplied from the external network master. The transfer rate is defined by this clock. The SPICLK input frequency should be no greater than the LSPCLK frequency divided by 4.

Data written to SPIDAT or SPITXBUF is transmitted to the network when appropriate edges of the SPICLK signal are received from the network master. A character written to the SPITXBUF register will be copied to the SPIDAT register when all bits of the current character in SPIDAT have been shifted out. If no character was previously copied to SPIDAT, then any character written to SPITXBUF will be immediately copied to SPIDAT. If a character was previously copied to SPIDAT, any data written to SPITXBUF will not be copied to SPIDAT until the current character in SPIDAT has been shifted out. To receive data, the SPI waits for the network master to send the SPICLK signal and then shifts the data on the SPISIMO pin into SPIDAT. If data is to be transmitted by the slave simultaneously, and SPIDAT has not been previously loaded, the character must be written to SPITXBUF before the beginning of the SPICLK signal.

When the TALK bit is cleared, data transmission is disabled, and the output line (SPISOMI) is put into the high-impedance state. If this occurs while a transmission is active, the current character is completely transmitted even though SPISOMI is forced into the high-impedance state. This ensures that the SPI is still able to receive incoming data correctly. This TALK bit allows many slave devices to be tied together on the network, but only one slave at a time is allowed to drive the SPISOMI line.

The  $\overline{\text{SPISTE}}$  pin operates as the slave-select pin. An active-low signal on the  $\overline{\text{SPISTE}}$  pin allows the slave SPI to transfer data to the serial data line; an inactive-high signal causes the slave SPI serial shift register to stop and its serial output pin to be put into the high-impedance state. This allows many slave devices to be tied together on the network, although only one slave device is selected at a time.

### 8.3.4 Data Format

The four-bit SPICHR register field specifies the number of bits in the data character (1 to 16). This information directs the state control logic to count the number of bits received or transmitted to determine when a complete character has been processed.

The following statements apply to characters with fewer than 16 bits:

- Data must be left-justified when written to SPIDAT and SPITXBUF.
- Data read back from SPIRXBUF is right-justified.
- SPIRXBUF contains the most recently received character, right-justified, plus any bits that remain from previous transmission(s) that have been shifted to the left (shown in [Example 8-1](#)).

#### Example 8-1. Transmission of Bit from SPIRXBUF

Conditions:

1. Transmission character length = 1 bit (specified in SPICHR bits)
2. The current value of SPIDAT = 737Bh

SPIDAT (before transmission)																	
	0	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	
SPIDAT (after transmission)																	
(TXed) 0 ←	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	x <sup>(1)</sup>	← (RXed)
SPIRXBUF (after transmission)																	
	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	x <sup>(1)</sup>	

(1) x = 1, if SPISOMI data is high; x = 0, if SPISOMI data is low; master mode is assumed.

### 8.3.5 Baud Rate Selection

The SPI module supports 125 different baud rates and four different clock schemes. Depending on whether the SPI clock is in slave or master mode, the SPICLK pin can receive an external SPI clock signal or provide the SPI clock signal, respectively.

- In the slave mode, the SPI clock is received on the SPICLK pin from the external source and can be no greater than the LSPCLK frequency divided by 4.
- In the master mode, the SPI clock is generated by the SPI and is output on the SPICLK pin and can be no greater than the LSPCLK frequency divided by 4.

---

#### Note

The baud rate should be configured to not exceed the maximum rated GPIO toggle frequency. Refer to the device data sheet for the maximum GPIO toggle frequency

---

[Example 8-2](#) shows how to determine the SPI baud rates.

#### Example 8-2. Baud Rate Determination

For SPIBRR = 3 to 127:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{(\text{SPIBRR} + 1)} \quad (1)$$

For SPIBRR = 0, 1, or 2:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{4} \quad (2)$$

where:

LSPCLK = Low-speed peripheral clock frequency of the device

SPIBRR = Contents of the SPIBRR in the master SPI device

To determine what value to load into SPIBRR, you must know the device system clock (LSPCLK) frequency (that is device-specific) and the baud rate at which you will be operating.

[Example 8-3](#) shows how to calculate the baud rate of the SPI module .

#### Example 8-3. Baud Rate Calculation

$$\begin{aligned} \text{Maximum SPI Baud Rate} &= \frac{\text{LSPCLK}}{4} \\ &= \frac{40 \times 10^6}{4} \\ &= 10 \times 10^6 \text{ bps} \end{aligned} \quad (3)$$

### 8.3.6 SPI Clocking Schemes

The clock polarity select bit (CLKPOLARITY) and the clock phase select bit (CLK\_PHASE) control four different clocking schemes on the SPICLK pin. CLKPOLARITY selects the active edge, either rising or falling, of the clock. CLK\_PHASE selects a half-cycle delay of the clock. The four different clocking schemes are as follows:

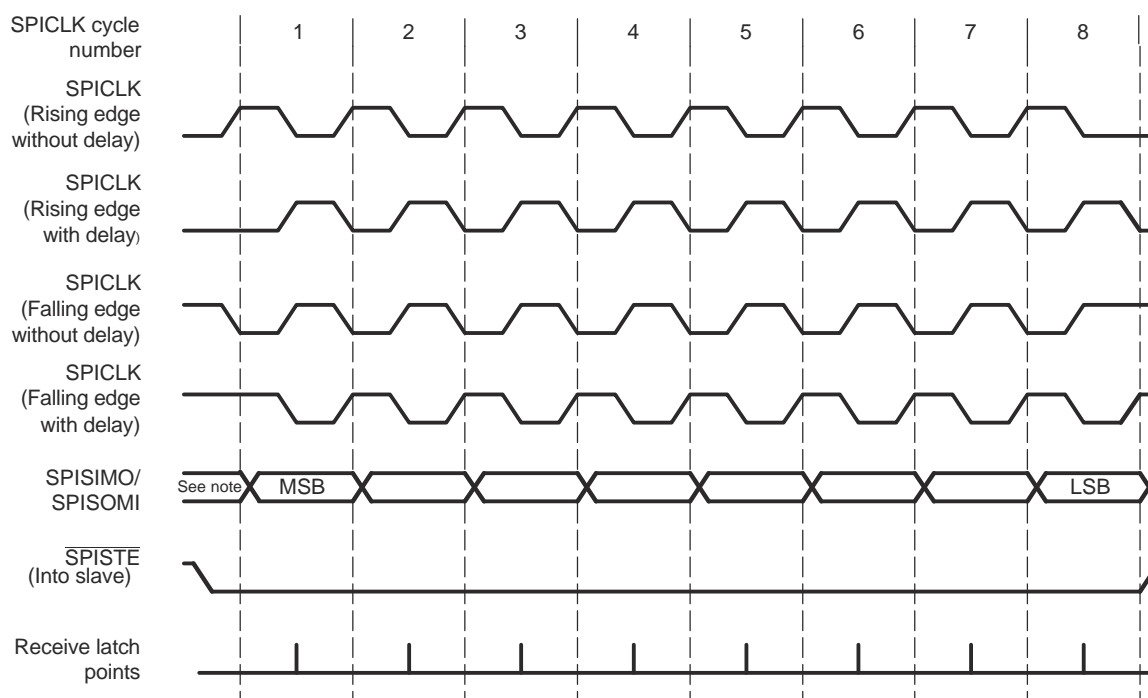
- Falling Edge Without Delay. The SPI transmits data on the falling edge of the SPICLK and receives data on the rising edge of the SPICLK.
- Falling Edge With Delay. The SPI transmits data one half-cycle ahead of the falling edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
- Rising Edge Without Delay. The SPI transmits data on the rising edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
- Rising Edge With Delay. The SPI transmits data one half-cycle ahead of the rising edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.

The selection procedure for the SPI clocking scheme is shown in [Table 8-3](#). Examples of these four clocking schemes relative to transmitted and received data are shown in [Figure 8-5](#).

**Table 8-3. SPI Clocking Scheme Selection Guide**

SPICLK Scheme	CLKPOLARITY	CLK_PHASE <sup>(1)</sup>
Rising edge without delay	0	0
Rising edge with delay	0	1
Falling edge without delay	1	0
Falling edge with delay	1	1

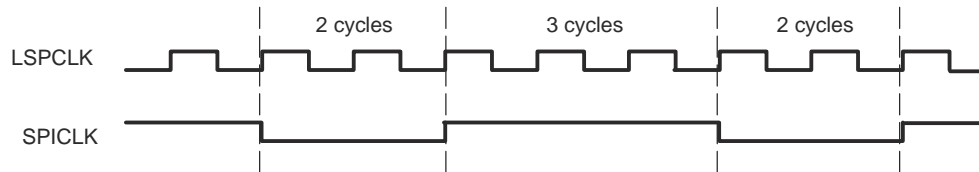
(1) The description of CLK\_PHASE and CLKPOLARITY differs between manufacturers. For proper operation, select the desired waveform to determine the clock phase and clock polarity settings.



**Note:** Previous data bit

**Figure 8-5. SPICLK Signal Options**

SPICLK symmetry is retained only when the result of  $(SPIBRR+1)$  is an even value. When  $(SPIBRR + 1)$  is an odd value and  $SPIBRR$  is greater than 3, SPICLK becomes asymmetrical. The low pulse of SPICLK is one LSPCLK cycle longer than the high pulse when  $CLKPOLARITY$  bit is clear (0). When  $CLKPOLARITY$  bit is set to 1, the high pulse of the SPICLK is one LSPCLK cycle longer than the low pulse, as shown in [Figure 8-6](#).



**Figure 8-6. SPI: SPICLK-LSPCLK Characteristic When  $(BRR + 1)$  is Odd,  $BRR > 3$ , and  $CLKPOLARITY = 1$**

### 8.3.7 SPI FIFO Description

The following steps explain the FIFO features and help with programming the SPI FIFOs:

1. **Reset.** At reset the SPI powers up in standard SPI mode and the FIFO function is disabled. The FIFO registers SPIFFTX, SPIFFRX and SPIFFCT remain inactive.
2. **Standard SPI.** The standard 28x SPI mode will work with SPIINT/SPIRXINT as the interrupt source.
3. **Mode change.** FIFO mode is enabled by setting the SPIFFENA bit to 1 in the SPIFFTX register. SPIRST can reset the FIFO mode at any stage of its operation.
4. **Active registers.** All the SPI registers and SPI FIFO registers SPIFFTX, SPIFFRX, and SPIFFCT will be active.
5. **Interrupts.** FIFO mode has two interrupts one for the transmit FIFO, SPITXINT and one for the receive FIFO, SPIRXINT. SPIRXINT is the common interrupt for SPI FIFO receive, receive error and receive FIFO overflow conditions. The single SPIINT for both transmit and receive sections of the standard SPI will be disabled and this interrupt will service as SPI receive FIFO interrupt. For more information, refer to [Section 8.2.3](#).
6. **Buffers.** Transmit and receive buffers are each supplemented with a 4 word FIFO. The one-word transmit buffer (SPITXBUF) of the standard SPI functions as a transition buffer between the transmit FIFO and shift register. The one-word transmit buffer will be loaded from transmit FIFO only after the last bit of the shift register is shifted out.
7. **Delayed transfer.** The rate at which transmit words in the FIFO are transferred to transmit shift register is programmable. The SPIFFCT register bits (7–0) FFFXDLY7–FFFXDLY0 define the delay between the word transfer. The delay is defined in number SPI serial clock cycles. The 8-bit register could define a minimum delay of 0 SPICLK cycles and a maximum of 255 SPICLK cycles. With zero delay, the SPI module can transmit data in continuous mode with the FIFO words shifting out back to back. With the 255 clock delay, the SPI module can transmit data in a maximum delayed mode with the FIFO words shifting out with a delay of 255 SPICLK cycles between each word. The programmable delay facilitates glueless interface to various slow SPI peripherals, such as EEPROMs, ADC, DAC, and so on.
8. **FIFO status bits.** Both transmit and receive FIFOs have status bits TXFFST or RXFFST that define the number of words available in the FIFOs at any time. The transmit FIFO reset bit (TXFIFO) and receive reset bit (RXFIFO) will reset the FIFO pointers to zero when these bits are set to 1. The FIFOs will resume operation from start once these bits are cleared to zero.
9. **Programmable interrupt levels.** Both transmit and receive FIFOs can generate CPU interrupts. The transmit interrupt (SPITXINT) is generated whenever the transmit FIFO status bits (TXFFST) match (less than or equal to) the interrupt trigger level bits (TXFFIL). The receive interrupt (SPIRXINT) is generated whenever the receive FIFO status bits (RXFFST) match (greater than or equal to) the interrupt trigger level RXFFIL. This provides a programmable interrupt trigger for transmit and receive sections of the SPI. The default value for these trigger level bits will be 0x11111 for receive FIFO and 0x00000 for transmit FIFO, respectively.

### 8.3.8 SPI 3-Wire Mode Description

SPI 3-wire mode allows for SPI communication over three pins instead of the normal four pins.

In master mode, if the TRIWIRE bit is set, enabling 3-wire SPI mode, SPISIMOMx becomes the bi-directional SPIMOMIx (SPI master out, master in) pin, and SPISOMIx is no longer used by the SPI. In slave mode, if the TRIWIRE bit is set, SPISOMIx becomes the bi-directional SPISISOx (SPI slave in, slave out) pin, and SPISIMOMx is no longer used by the SPI.

Table 8-4 indicates the pin function differences between 3-wire and 4-wire SPI mode for a master and slave SPI.

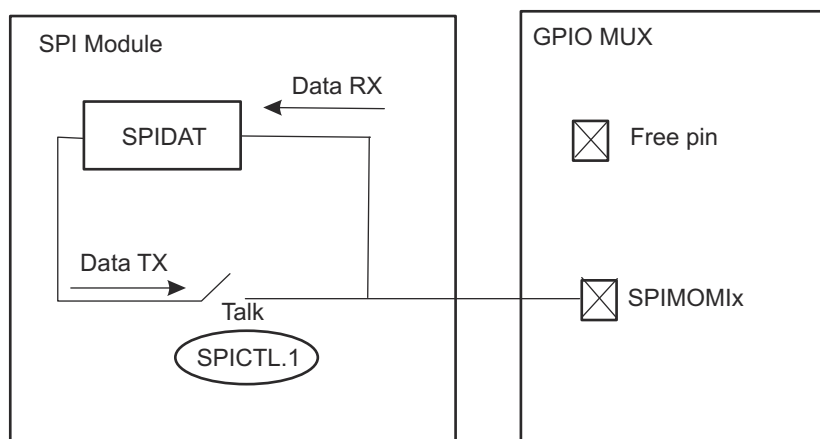
**Table 8-4. 4-wire vs. 3-wire SPI Pin Functions**

4-wire SPI	3-wire SPI (Master)	3-wire SPI (Slave)
SPICLKx	SPICLKx	SPICLKx
SPISTEx	SPISTEx	SPISTEx
SPISIMOMx	SPIMOMIx	Free
SPISOMIx	Free	SPISISOx

Because in 3-wire mode, the receive and transmit paths within the SPI are connected, any data transmitted by the SPI module is also received by itself. The application software must take care to perform a dummy read to clear the SPI data register of the additional received data.

The TALK bit plays an important role in 3-wire SPI mode. The bit must be set to transmit data and cleared prior to reading data. In master mode, in order to initiate a read, the application software must write dummy data to the SPI data register (SPIDAT or SPIRXBUF) while the TALK bit is cleared (no data is transmitted out the SPIMOMI pin) before reading from the data register.

Figure 8-7 and Figure 8-8 illustrate 3-wire master and slave mode.



**Figure 8-7. SPI 3-wire Master Mode**



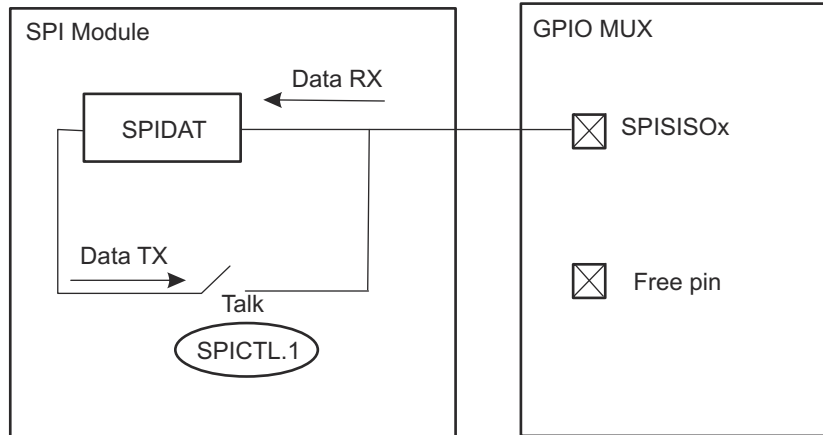


Figure 8-8. SPI 3-wire Slave Mode

Table 8-5 indicates how data is received or transmitted in the various SPI modes while the TALK bit is set or cleared.

Table 8-5. 3-Wire SPI Pin Configuration

Pin Mode	SPIPRI[TRIWIRE]	SPICTL[TALK]	SPISIMO	SPISOMI
<b>Master Mode</b>				
4-wire	0	X	TX	RX
3-pin mode	1	0	RX	Disconnect from SPI
		1	TX/RX	
<b>Slave Mode</b>				
4-wire	0	X	RX	TX
3-pin mode	1	0	Disconnect from SPI	RX
		1		TX/RX

## 8.4 Programming Procedure

This section describes the procedure for configuring the SPI for the various modes of operation.

### 8.4.1 Initialization Upon Reset

A system reset forces the SPI peripheral into the following default configuration:

- Unit is configured as a slave module (MASTER\_SLAVE = 0)
- Transmit capability is disabled (TALK = 0)
- Data is latched at the input on the falling edge of the SPICLK signal
- Character length is assumed to be one bit
- SPI interrupts are disabled
- Data in SPIDAT is reset to 0000h

### 8.4.2 Configuring the SPI

This section describes the procedure in which to configure the SPI module for operation. To prevent unwanted and unforeseen events from occurring during or as a result of initialization changes, clear the SPISWRESET bit before making initialization changes, and then set this bit after initialization is complete. While the SPI is held in reset (SPISWRESET = 0), configuration may be changed in any order. The following list shows the SPI configuration procedure in a logical order. However, the SPI registers can be written with single 16-bit writes, so the order is not required with the exception of SPISWRESET.

To change the SPI configuration:

1. Clear the SPI Software Reset bit (SPISWRESET) to 0 to force the SPI to the reset state.
2. Configure the SPI as desired:
  - Select either master or slave mode (MASTER\_SLAVE).
  - Choose SPICLK polarity and phase (CLKPOLARITY and CLK\_PHASE).
  - Set the desired baud rate (SPIBRR).
  - Set the SPI character length (SPICHR).
  - Clear the SPI Flags (OVERRUN\_FLAG, INT\_FLAG).
  - Enable 3-wire mode (TRIWIRE), if needed.
  - If using FIFO enhancements:
    - Enable the FIFO enhancements (SPIFFENA).
    - Clear the FIFO Flags (TXFFINTCLR, RXFFOVFCLR, and RXFFINTCLR).
    - Release transmit and receive FIFO resets (TXFIFO and RXFIFORESET).
    - Release SPI FIFO channels from reset (SPIRST).
3. If interrupts are used:
  - In non-FIFO mode, enable the receiver overrun and/or SPI interrupts (OVERRUNINTENA and SPIINTENA).
  - In FIFO mode, set the transmit and receive interrupt levels (TXFFIL and RXFFIL) then enable the interrupts (TXFFIENA and RXFFIENA).
4. Set SPISWRESET to 1 to release the SPI from the reset state.

---

#### Note

Do not change the SPI configuration when communication is in progress.

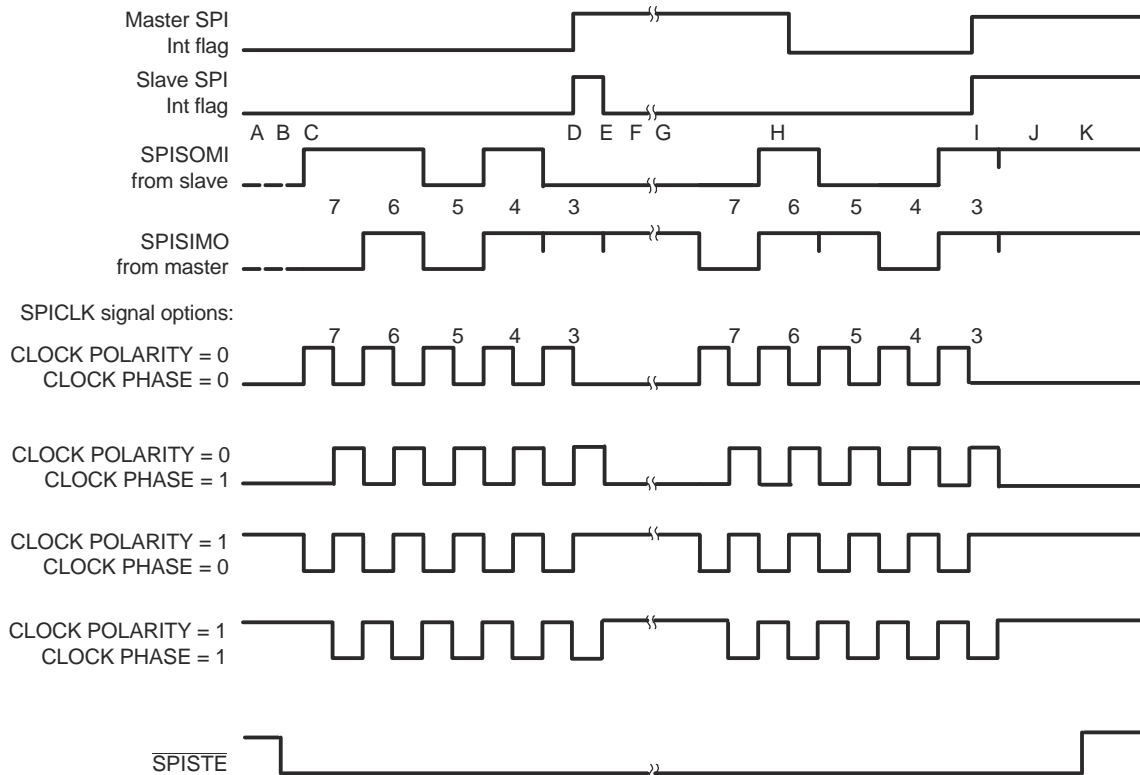
---

### 8.4.3 Data Transfer Example

The timing diagram shown in Figure 8-9 shows an SPI data transfer between two devices using a character length of five bits with the SPICLK being symmetrical.

The timing diagram with SPICLK asymmetrical (Figure 8-6) shares similar characterizations with Figure 8-9 except that the data transfer is one LSPCLK cycle longer per bit during the low pulse (CLKPOLARITY = 0) or during the high pulse (CLKPOLARITY = 1) of the SPICLK.

Figure 8-9 is applicable for 8-bit SPI only and is not for C28x devices that are capable of working with 16-bit data. The figure is shown for illustrative purposes only.



- A. Slave writes 0D0h to SPIDAT and waits for the master to shift out the data.
- B. Master sets the slave  $\overline{\text{SPISTE}}$  signal low (active).
- C. Master writes 058h to SPIDAT, which starts the transmission procedure.
- D. First byte is finished and sets the interrupt flags.
- E. Slave reads 0Bh from its SPIRXBUF (right-justified).
- F. Slave writes 04Ch to SPIDAT and waits for the master to shift out the data.
- G. Master writes 06Ch to SPIDAT, which starts the transmission procedure.
- H. Master reads 01Ah from the SPIRXBUF (right-justified).
- I. Second byte is finished and sets the interrupt flags.
- J. Master reads 89h and the slave reads 8Dh from their respective SPIRXBUF. After the user software masks off the unused bits, the master receives 09h and the slave receives 0Dh.
- K. Master clears the slave  $\overline{\text{SPISTE}}$  signal high (inactive).

Figure 8-9. Five Bits per Character

### 8.4.4 SPI 3-Wire Mode Code Examples

In addition to the normal SPI initialization, to configure the SPI module for 3-wire mode, the TRIWIRE bit (SPIPRI.0) must be set to 1. After initialization, there are several considerations to take into account when transmitting and receiving data in 3-wire master and slave mode. The following examples demonstrate these considerations.

In 3-wire master mode, SPICLKx, SPISTEx, and SPISIMOX pins must be configured as SPI pins (SPISOMIx pin can be configured as non-SPI pin). When the master transmits, it receives the data it transmits (because SPISIMOX and SPISOMIx are connected internally in 3-wire mode). Therefore, the junk data received must be cleared from the receive buffer every time data is transmitted.

#### Example 8-4. 3-Wire Master Mode Transmit

```

Uint16 data;
Uint16 dummy;
    SpiaRegs.SPICTL.bit.TALK = 1;           // Enable Transmit path
    SpiaRegs.SPITXBUF = data; // Master transmits data
    while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Waits until data rx'd
    dummy = SpiaRegs.SPIRXBUF;             // Clears junk data from itself
                                           // bc it rx'd same data tx'd

```

To receive data in 3-wire master mode, the master must clear the TALK (SPICTL.1) bit to 0 to close the transmit path and then transmit dummy data in order to initiate the transfer from the slave. Because the TALK bit is 0, unlike in transmit mode, the master dummy data does not appear on the SPISIMOX pin, and the master does not receive its own dummy data. Instead, the data from the slave is received by the master.

#### Example 8-5. 3-Wire Master Mode Receive

```

Uint16 rdata;
Uint16 dummy;
    SpiaRegs.SPICTL.bit.TALK = 0;           // Disable Transmit path
    SpiaRegs.SPITXBUF = dummy;             // Send dummy to start tx
    // NOTE: because TALK = 0, data does not tx onto SPISIMOX pin
    while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Wait until data received
    rdata = SpiaRegs.SPIRXBUF;             // Master reads data

```

In 3-wire slave mode, SPICLKx, SPISTEx, and SPISOMIx pins must be configured as SPI pins (SPISIMOX pin can be configured as non-SPI pin). Like in master mode, when transmitting, the slave receives the data it transmits and must clear this junk data from its receive buffer.

#### Example 8-6. 3-Wire Slave Mode Transmit

```

Uint16 data;
Uint16 dummy;
    SpiaRegs.SPICTL.bit.TALK = 1;           // Enable Transmit path
    SpiaRegs.SPITXBUF = data; // Slave transmits data
    while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Wait until data rx'd
    dummy = SpiaRegs.SPIRXBUF;             // Clears junk data from itself

```

As in 3-wire master mode, the TALK bit must be cleared to 0. Otherwise, the slave receives data normally.

### Example 8-7. 3-Wire Slave Mode Receive

```

Uint16 rdata;
SpiaRegs.SPICTL.bit.TALK = 0;           // Disable Transmit path
while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Waits until data rx'd
rdata = SpiaRegs.SPIRXBUF;             // Slave reads data

```

## 8.5 SPI Registers

This section describes the Serial Peripheral Interface registers. It is important to note that the SPI registers only allow 16-bit accesses.

### 8.5.1 SPI Base Addresses

**Table 8-6. SPI Base Address Table (C28)**

Bit Field Name		Base Address
Instance	Structure	
SpiaRegs	SPI_REGS	0x0000_7040

### 8.5.2 SPI\_REGS Registers

[Table 8-7](#) lists the SPI\_REGS registers. All register offset addresses not listed in [Table 8-7](#) should be considered as reserved locations and the register contents should not be modified.

**Table 8-7. SPI\_REGS Registers**

Offset	Acronym	Register Name	Section
0h	SPICCR	SPI Configuration Control Register	<a href="#">Section 8.5.2.1</a>
1h	SPICTL	SPI Operation Control Register	<a href="#">Section 8.5.2.2</a>
2h	SPISTS	SPI Status Register	<a href="#">Section 8.5.2.3</a>
4h	SPIBRR	SPI Baud Rate Register	<a href="#">Section 8.5.2.4</a>
6h	SPIRXEMU	SPI Emulation Buffer Register	<a href="#">Section 8.5.2.5</a>
7h	SPIRXBUF	SPI Serial Input Buffer Register	<a href="#">Section 8.5.2.6</a>
8h	SPITXBUF	SPI Serial Output Buffer Register	<a href="#">Section 8.5.2.7</a>
9h	SPIDAT	SPI Serial Data Register	<a href="#">Section 8.5.2.8</a>
Ah	SPIFFTX	SPI FIFO Transmit Register	<a href="#">Section 8.5.2.9</a>
Bh	SPIFFRX	SPI FIFO Receive Register	<a href="#">Section 8.5.2.10</a>
Ch	SPIFFCT	SPI FIFO Control Register	<a href="#">Section 8.5.2.11</a>
Fh	SPIPRI	SPI Priority Control Register	<a href="#">Section 8.5.2.12</a>

Complex bit access types are encoded to fit into small table cells. [Table 8-8](#) shows the codes that are used for access types in this section.

**Table 8-8. SPI\_REGS Access Type Codes**

Access Type	Code	Description
Read Type		
R	R	Read
RC	R C	Read to Clear
Write Type		
W	W	Write
W1C	W 1C	Write 1 to clear
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

### 8.5.2.1 SPI Configuration Control Register (SPICCR) (Offset = 0h) [reset = 0h]

SPICCR controls the setup of the SPI for operation.

**Figure 8-10. SPI Configuration Control Register (SPICCR)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
SPISWRESET	CLKPOLARITY	RESERVED	SPILBK	SPICCHAR			
R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h			

**Table 8-9. SPI Configuration Control Register (SPICCR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	SPISWRESET	R/W	0h	<p>SPI Software Reset</p> <p>When changing configuration, you should clear this bit before the changes and set this bit before resuming operation.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Initializes the SPI operating flags to the reset condition. Specifically, the RECEIVER OVERRUN Flag bit (SPISTS.7), the SPI INT FLAG bit (SPISTS.6), and the TXBUF FULL Flag bit (SPISTS.5) are cleared. SPISTE will become inactive. SPICLK will be immediately driven to 0 regardless of the clock polarity. The SPI configuration remains unchanged.</p> <p>1h (R/W) = SPI is ready to transmit or receive the next character. When the SPI SW RESET bit is a 0, a character written to the transmitter will not be shifted out when this bit is set. A new character must be written to the serial data register. SPICLK will be returned to its inactive state one SPICLK cycle after this bit is set.</p>

**Table 8-9. SPI Configuration Control Register (SPICCR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
6	CLKPOLARITY	R/W	0h	<p>Shift Clock Polarity</p> <p>This bit controls the polarity of the SPICLK signal. CLOCK POLARITY and POLARITY CLOCK PHASE (SPICTL.3) control four clocking schemes on the SPICLK pin.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Data is output on rising edge and input on falling edge. When no SPI data is sent, SPICLK is at low level. The data input and output edges depend on the value of the CLOCK PHASE bit (SPICTL.3) as follows:</p> <ul style="list-style-type: none"> <li>- CLOCK PHASE = 0: Data is output on the rising edge of the SPICLK signal. Input data is latched on the falling edge of the SPICLK signal.</li> <li>- CLOCK PHASE = 1: Data is output one half-cycle before the first rising edge of the SPICLK signal and on subsequent falling edges of the SPICLK signal. Input data is latched on the rising edge of the SPICLK signal.</li> </ul> <p>1h (R/W) = Data is output on falling edge and input on rising edge. When no SPI data is sent, SPICLK is at high level. The data input and output edges depend on the value of the CLOCK PHASE bit (SPICTL.3) as follows:</p> <ul style="list-style-type: none"> <li>- CLOCK PHASE = 0: Data is output on the falling edge of the SPICLK signal. Input data is latched on the rising edge of the SPICLK signal.</li> <li>- CLOCK PHASE = 1: Data is output one half-cycle before the first falling edge of the SPICLK signal and on subsequent rising edges of the SPICLK signal. Input data is latched on the falling edge of the SPICLK signal.</li> </ul>
5	RESERVED	R	0h	Reserved
4	SPILBK	R/W	0h	<p>SPI Loopback Mode Select</p> <p>Loopback mode allows module validation during device testing. This mode is valid only in master mode of the SPI.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = SPI loopback mode disabled. This is the default value after reset.</p> <p>1h (R/W) = SPI loopback mode enabled, SIMO/SOMI lines are connected internally. Used for module self-tests.</p>



**Table 8-9. SPI Configuration Control Register (SPICCR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
3-0	SPICCHAR	R/W	0h	Character Length Control Bits These four bits determine the number of bits to be shifted in or SPI CHAR0 out as a single character during one shift sequence. SPICCHAR = Word length - 1 Reset type: SYSRSn 0h (R/W) = 1-bit word 1h (R/W) = 2-bit word 2h (R/W) = 3-bit word 3h (R/W) = 4-bit word 4h (R/W) = 5-bit word 5h (R/W) = 6-bit word 6h (R/W) = 7-bit word 7h (R/W) = 8-bit word 8h (R/W) = 9-bit word 9h (R/W) = 10-bit word Ah (R/W) = 11-bit word Bh (R/W) = 12-bit word Ch (R/W) = 13-bit word Dh (R/W) = 14-bit word Eh (R/W) = 15-bit word Fh (R/W) = 16-bit word

### 8.5.2.2 SPI Operation Control (SPICTL) Register (Offset = 1h) [reset = 0h]

SPICTL controls data transmission, the SPI's ability to generate interrupts, the SPICLK phase, and the operational mode (slave or master).

**Figure 8-11. SPI Operation Control (SPICTL) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			OVERRUNINT ENA	CLK_PHASE	MASTER_SLAVE	TALK	SPIINTENA
R-0h			R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

**Table 8-10. SPI Operation Control (SPICTL) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-5	RESERVED	R	0h	Reserved
4	OVERRUNINTENA	R/W	0h	Overrun Interrupt Enable Overrun Interrupt Enable. Setting this bit causes an interrupt to be generated when the RECEIVER OVERRUN Flag bit (SPISTS.7) is set by hardware. Interrupts generated by the RECEIVER OVERRUN Flag bit and the SPI INT FLAG bit (SPISTS.6) share the same interrupt vector. Reset type: SYSRSn 0h (R/W) = Disable RECEIVER OVERRUN interrupts. 1h (R/W) = Enable RECEIVER_OVERRUN interrupts.
3	CLK_PHASE	R/W	0h	SPI Clock Phase Select This bit controls the phase of the SPICLK signal. CLOCK PHASE and CLOCK POLARITY (SPICCR.6) make four different clocking schemes possible (see clocking figures in SPI chapter). When operating with CLOCK PHASE high, the SPI (master or slave) makes the first bit of data available after SPIDAT is written and before the first edge of the SPICLK signal, regardless of which SPI mode is being used. Reset type: SYSRSn 0h (R/W) = Normal SPI clocking scheme, depending on the CLOCK POLARITY bit (SPICCR.6). 1h (R/W) = SPICLK signal delayed by one half-cycle. Polarity determined by the CLOCK POLARITY bit.
2	MASTER_SLAVE	R/W	0h	SPI Network Mode Control This bit determines whether the SPI is a network master or slave. SLAVE During reset initialization, the SPI is automatically configured as a network slave. Reset type: SYSRSn 0h (R/W) = SPI is configured as a slave. 1h (R/W) = SPI is configured as a master.

**Table 8-10. SPI Operation Control (SPICTL) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
1	TALK	R/W	0h	<p><b>Transmit Enable</b>            The TALK bit can disable data transmission (master or slave) by placing the serial data output in the high-impedance state. If this bit is disabled during a transmission, the transmit shift register continues to operate until the previous character is shifted out. When the TALK bit is disabled, the SPI is still able to receive characters and update the status flags. TALK is cleared (disabled) by a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Disables transmission:</p> <ul style="list-style-type: none"> <li>- Slave mode operation: If not previously configured as a general-purpose I/O pin, the SPISOMI pin will be put in the high-impedance state.</li> <li>- Master mode operation: If not previously configured as a general-purpose I/O pin, the SPISIMO pin will be put in the high-impedance state.</li> </ul> <p>1h (R/W) = Enables transmission For the 4-pin option, ensure to enable the receiver's SPISTEn input pin.</p>
0	SPIINTENA	R/W	0h	<p><b>SPI Interrupt Enable</b>            This bit controls the SPI's ability to generate a transmit/receive interrupt. The SPI INT FLAG bit (SPISTS.6) is unaffected by this bit.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Disables the interrupt.</p> <p>1h (R/W) = Enables the interrupt.</p>

### 8.5.2.3 SPI Status (SPISTS) Register (Offset = 2h) [reset = 0h]

SPISTS contains interrupt and status bits.

**Figure 8-12. SPI Status (SPISTS) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
OVERRUN_FLAG	INT_FLAG	BUFFULL_FLAG	RESERVED				
W1C-0h	RC-0h	R-0h	R-0h				

**Table 8-11. SPI Status (SPISTS) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	OVERRUN_FLAG	W1C	0h	<p><b>SPI Receiver Overrun Flag</b>                      This bit is a read/clear-only flag. The SPI hardware sets this bit when a receive or transmit operation completes before the previous character has been read from the buffer. The bit is cleared in one of three ways:</p> <ul style="list-style-type: none"> <li>- Writing a 1 to this bit</li> <li>- Writing a 0 to SPI SW RESET (SPICCR.7)</li> <li>- Resetting the system</li> </ul> <p>If the OVERRUN INT ENA bit (SPICTL.4) is set, the SPI requests only one interrupt upon the first occurrence of setting the RECEIVER OVERRUN Flag bit. Subsequent overruns will not request additional interrupts if this flag bit is already set. This means that in order to allow new overrun interrupt requests the user must clear this flag bit by writing a 1 to SPISTS.7 each time an overrun condition occurs. In other words, if the RECEIVER OVERRUN Flag bit is left set (not cleared) by the interrupt service routine, another overrun interrupt will not be immediately re-entered when the interrupt service routine is exited.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = A receive overrun condition has not occurred.                      1h (R/W) = The last received character has been overwritten and therefore lost (when the SPIRXBUF was overwritten by the SPI module before the previous character was read by the user application).</p> <p>Writing a '1' will clear this bit. The RECEIVER OVERRUN Flag bit should be cleared during the interrupt service routine because the RECEIVER OVERRUN Flag bit and SPI INT FLAG bit (SPISTS.6) share the same interrupt vector. This will alleviate any possible doubt as to the source of the interrupt when the next byte is received.</p>

**Table 8-11. SPI Status (SPISTS) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
6	INT_FLAG	RC	0h	<p>SPI Interrupt Flag</p> <p>SPI INT FLAG is a read-only flag. Hardware sets this bit to indicate that the SPI has completed sending or receiving the last bit and is ready to be serviced. This flag causes an interrupt to be requested if the SPI INT ENA bit (SPICTL.0) is set. The received character is placed in the receiver buffer at the same time this bit is set. This bit is cleared in one of three ways:</p> <ul style="list-style-type: none"> <li>- Reading SPIRXBUF</li> <li>- Writing a 0 to SPI SW RESET (SPICCR.7)</li> <li>- Resetting the system</li> </ul> <p>Note: This bit should not be used if FIFO mode is enabled. The internal process of copying the received word from SPIRXBUF to the Receive FIFO will clear this bit. Use the FIFO status, or FIFO interrupt bits for similar functionality.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No full words have been received or transmitted. 1h (R/W) = Indicates that the SPI has completed sending or receiving the last bit and is ready to be serviced.</p>
5	BUFFULL_FLAG	R	0h	<p>SPI Transmit Buffer Full Flag</p> <p>This read-only bit gets set to 1 when a character is written to the SPI Transmit buffer SPITXBUF. It is cleared when the character is automatically loaded into SPIDAT when the shifting out of a previous character is complete.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Transmit buffer is not full. 1h (R/W) = Transmit buffer is full.</p>
4-0	RESERVED	R	0h	Reserved

### 8.5.2.4 SPI Baud Rate Register (SPIBRR) (Offset = 4h) [reset = 0h]

SPIBRR contains the bits used for baud-rate selection.

**Figure 8-13. SPI Baud Rate Register (SPIBRR)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED		SPI_BIT_RATE					
R-0h		R/W-0h					

**Table 8-12. SPI Baud Rate Register (SPIBRR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6-0	SPI_BIT_RATE	R/W	0h	<p>SPI Baud Rate Control</p> <p>These bits determine the bit transfer rate if the SPI is the network SPI BIT RATE 0 master. There are 125 data-transfer rates (each a function of the CPU clock, LSPCLK) that can be selected. One data bit is shifted per SPICLK cycle. (SPICLK is the baud rate clock output on the SPICLK pin.)</p> <p>If the SPI is a network slave, the module receives a clock on the SPICLK pin from the network master. Therefore, these bits have no effect on the SPICLK signal. The frequency of the input clock from the master should not exceed the slave SPI's LSPCLK signal divided by 4.</p> <p>In master mode, the SPI clock is generated by the SPI and is output on the SPICLK pin. The SPI baud rates are determined by the following formula:</p> <p>For SPIBRR = 3 to 127: SPI Baud Rate = LSPCLK / (SPIBRR + 1)</p> <p>For SPIBRR = 0, 1, or 2: SPI Baud Rate = LSPCLK / 4</p> <p>Reset type: SYSRSn</p> <p>3h (R/W) = SPI Baud Rate = LSPCLK/4</p> <p>4h (R/W) = SPI Baud Rate = LSPCLK/5</p> <p>7Eh (R/W) = SPI Baud Rate = LSPCLK/127</p> <p>7Fh (R/W) = SPI Baud Rate = LSPCLK/128</p>

### 8.5.2.5 SPI Emulation Buffer (SPIRXEMU) Register (Offset = 6h) [reset = 0h]

SPIRXEMU contains the received data. Reading SPIRXEMU does not clear the SPI INT FLAG bit of SPISTS. This is not a real register but a dummy address from which the contents of SPIRXBUF can be read by the debug probe connection without clearing the SPI INT FLAG.

**Figure 8-14. SPI Emulation Buffer (SPIRXEMU) Register**

15	14	13	12	11	10	9	8
ERXBn							
R-0h							
7	6	5	4	3	2	1	0
ERXBn							
R-0h							

**Table 8-13. SPI Emulation Buffer (SPIRXEMU) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-0	ERXBn	R	0h	<p>Emulation Buffer Received Data</p> <p>SPIRXEMU functions almost identically to SPIRXBUF, except that reading SPIRXEMU does not clear the SPI INT FLAG bit (SPISTS.6). Once the SPIDAT has received the complete character, the character is transferred to SPIRXEMU and SPIRXBUF, where it can be read. At the same time, SPI INT FLAG is set.</p> <p>This mirror register was created to support emulation. Reading SPIRXBUF clears the SPI INT FLAG bit (SPISTS.6). In the normal operation with a debug probe connection, the control registers are read to continually update the contents of these registers on the display screen. SPIRXEMU was created so that the debug probe connection can read this register and properly update the contents on the display screen. Reading SPIRXEMU does not clear the SPI INT FLAG bit, but reading SPIRXBUF clears this flag. In other words, SPIRXEMU enables the debug probe connection to emulate the true operation of the SPI more accurately.</p> <p>It is recommended that you view SPIRXEMU when the debug probe is connected.</p> <p>Reset type: SYSRSn</p>

### 8.5.2.6 SPI Serial Input Buffer (SPIRXBUF) Register (Offset = 7h) [reset = 0h]

SPIRXBUF contains the received data. Reading SPIRXBUF clears the SPI INT FLAG bit in SPISTS. If FIFO mode is enabled, reading this register will also decrement the RXFFST counter in SPIFFRX.

**Figure 8-15. SPI Serial Input Buffer (SPIRXBUF) Register**

15	14	13	12	11	10	9	8
RXBn							
R-0h							
7	6	5	4	3	2	1	0
RXBn							
R-0h							

**Table 8-14. SPI Serial Input Buffer (SPIRXBUF) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-0	RXBn	R	0h	Received Data Once SPIDAT has received the complete character, the character is transferred to SPIRXBUF, where it can be read. At the same time, the SPI INT FLAG bit (SPISTS.6) is set. Since data is shifted into the SPI's most significant bit first, it is stored right-justified in this register. Reset type: SYSRSn

### 8.5.2.7 SPI Serial Output Buffer (SPITXBUF) Register (Offset = 8h) [reset = 0h]

SPITXBUF stores the next character to be transmitted. Writing to this register sets the TX BUF FULL Flag bit in SPISTS. When the transmission of the current character is complete, the contents of this register are automatically loaded in SPIDAT and the TX BUF FULL Flag is cleared. If no transmission is currently active, data written to this register falls through into the SPIDAT register and the TX BUF FULL Flag is not set. In master mode, if no transmission is currently active, writing to this register initiates a transmission in the same manner that writing to SPIDAT does.

**Figure 8-16. SPI Serial Output Buffer (SPITXBUF) Register**

15	14	13	12	11	10	9	8
TXBn							
R/W-0h							
7	6	5	4	3	2	1	0
TXBn							
R/W-0h							

**Table 8-15. SPI Serial Output Buffer (SPITXBUF) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-0	TXBn	R/W	0h	Transmit Data Buffer This is where the next character to be transmitted is stored. When the transmission of the current character has completed, if the TX BUF FULL Flag bit is set, the contents of this register is automatically transferred to SPIDAT, and the TX BUF FULL Flag is cleared. Writes to SPITXBUF must be left-justified. Reset type: SYSRSn



### 8.5.2.8 SPI Serial Data (SPIDAT) Register (Offset = 9h) [reset = 0h]

SPIDAT is the transmit and receive shift register. Data written to SPIDAT is shifted out (MSB) on subsequent SPICLK cycles. For every bit (MSB) shifted out of the SPI, a bit is shifted into the LSB end of the shift register.

**Figure 8-17. SPI Serial Data (SPIDAT) Register**

15	14	13	12	11	10	9	8
SDATn							
R/W-0h							
7	6	5	4	3	2	1	0
SDATn							
R/W-0h							

**Table 8-16. SPI Serial Data (SPIDAT) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-0	SDATn	R/W	0h	Serial Data Shift Register - It provides data to be output on the serial output pin if the TALK bit (SPICTL.1) is set. - When the SPI is operating as a master, a data transfer is initiated. When initiating a transfer, check the CLOCK POLARITY bit (SPICCR.6) described in Section 10.2.1.1 and the CLOCK PHASE bit (SPICTL.3) described in Section 10.2.1.2, for the requirements. In master mode, writing dummy data to SPIDAT initiates a receiver sequence. Since the data is not hardware-justified for characters shorter than sixteen bits, transmit data must be written in left-justified form, and received data read in right-justified form. Reset type: SYSRSn

### 8.5.2.9 SPI FIFO Transmit (SPIFFTX) Register (Offset = Ah) [reset = A000h]

SPIFFTX contains both control and status bits related to the output FIFO buffer. This includes FIFO reset control, FIFO interrupt level control, FIFO level status, as well as FIFO interrupt enable and clear bits.

**Figure 8-18. SPI FIFO Transmit (SPIFFTX) Register**

15		14		13		12		11		10		9		8	
SPIRST		SPIFFENA		TXFIFO						TXFFST					
R/W-1h		R/W-0h		R/W-1h						R-0h					
7		6		5		4		3		2		1		0	
TXFFINT		TXFFINTCLR		TXFFIENA						TXFFIL					
R-0h		W-0h		R/W-0h						R/W-0h					

**Table 8-17. SPI FIFO Transmit (SPIFFTX) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15	SPIRST	R/W	1h	SPI Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the SPI transmit and receive channels. The SPI FIFO register configuration bits will be left as is. 1h (R/W) = SPI FIFO can resume transmit or receive. No effect to the SPI registers bits.
14	SPIFFENA	R/W	0h	SPI FIFO Enhancements Enable Reset type: SYSRSn 0h (R/W) = SPI FIFO enhancements are disabled. 1h (R/W) = SPI FIFO enhancements are enabled.
13	TXFIFO	R/W	1h	TX FIFO Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Release transmit FIFO from reset.
12-8	TXFFST	R	0h	Transmit FIFO Status Reset type: SYSRSn 0h (R/W) = Transmit FIFO is empty. 1h (R/W) = Transmit FIFO has 1 word. 2h (R/W) = Transmit FIFO has 2 words. 3h (R/W) = Transmit FIFO has 3 words. 4h (R/W) = Transmit FIFO has 4 words, which is the maximum. 1Fh (R/W) = Reserved.
7	TXFFINT	R	0h	TX FIFO Interrupt Flag Reset type: SYSRSn 0h (R/W) = TXFIFO interrupt has not occurred, This is a read-only bit. 1h (R/W) = TXFIFO interrupt has occurred, This is a read-only bit.
6	TXFFINTCLR	W	0h	TXFIFO Interrupt Clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFTX[TXFFINT] flag.

**Table 8-17. SPI FIFO Transmit (SPIFFTX) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
5	TXFFIENA	R/W	0h	<p>TX FIFO Interrupt Enable</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = TX FIFO interrupt based on TXFFIL match (less than or equal to) will be disabled.</p> <p>1h (R/W) = TX FIFO interrupt based on TXFFIL match (less than or equal to) will be enabled.</p>
4-0	TXFFIL	R/W	0h	<p>Transmit FIFO Interrupt Level Bits</p> <p>Transmit FIFO will generate interrupt when the FIFO status bits (TXFFST4-0) and FIFO level bits (TXFFIL4-0) match (less than or equal to).</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = A TX FIFO interrupt request is generated when there are no words remaining in the TX buffer.</p> <p>1h (R/W) = A TX FIFO interrupt request is generated when there is 1 word or no words remaining in the TX buffer.</p> <p>2h (R/W) = A TX FIFO interrupt request is generated when there is 2 words or fewer remaining in the TX buffer.</p> <p>3h (R/W) = A TX FIFO interrupt request is generated when there are 3 words or fewer remaining in the TX buffer.</p> <p>4h (R/W) = A TX FIFO interrupt request is generated when there are 4 words or fewer remaining in the TX buffer.</p> <p>1Fh (R/W) = Reserved.</p>

### 8.5.2.10 SPI FIFO Receive (SPIFFRX) Register (Offset = Bh) [reset = 201Fh]

SPIFFRX contains both control and status bits related to the input FIFO buffer. This includes FIFO reset control, FIFO interrupt level control, FIFO level status, as well as FIFO interrupt enable and clear bits.

**Figure 8-19. SPI FIFO Receive (SPIFFRX) Register**

15	14	13	12	11	10	9	8
RXFFOVF	RXFFOVFCLR	RXFIFORESET	RXFFST				
R-0h	W-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL				
R-0h	W-0h	R/W-0h	R/W-1Fh				

**Table 8-18. SPI FIFO Receive (SPIFFRX) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15	RXFFOVF	R	0h	Receive FIFO Overflow Flag Reset type: SYSRSn 0h (R/W) = Receive FIFO has not overflowed. This is a read-only bit. 1h (R/W) = Receive FIFO has overflowed, read-only bit. More than 4 words have been received in to the FIFO, and the first received word is lost.
14	RXFFOVFCLR	W	0h	Receive FIFO Overflow Clear Reset type: SYSRSn 0h (R/W) = Write 0 does not affect RXFFOVF flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFRX[RXFFOVF].
13	RXFIFORESET	R/W	1h	Receive FIFO Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Re-enable receive FIFO operation.
12-8	RXFFST	R	0h	Receive FIFO Status Reset type: SYSRSn 0h (R/W) = Receive FIFO is empty. 1h (R/W) = Receive FIFO has 1 word. 2h (R/W) = Receive FIFO has 2 words. 3h (R/W) = Receive FIFO has 3 words. 4h (R/W) = Receive FIFO has 4 words, which is the maximum. 1Fh (R/W) = Reserved.
7	RXFFINT	R	0h	Receive FIFO Interrupt Flag Reset type: SYSRSn 0h (R/W) = RXFIFO interrupt has not occurred. This is a read-only bit. 1h (R/W) = RXFIFO interrupt has occurred. This is a read-only bit.
6	RXFFINTCLR	W	0h	Receive FIFO Interrupt Clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFIFINT flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFRX[RXFFINT] flag

**Table 8-18. SPI FIFO Receive (SPIFFRX) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
5	RXFFIENA	R/W	0h	<p>RX FIFO Interrupt Enable</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = RX FIFO interrupt based on RXFFIL match (greater than or equal to) will be disabled.</p> <p>1h (R/W) = RX FIFO interrupt based on RXFFIL match (greater than or equal to) will be enabled.</p>
4-0	RXFFIL	R/W	1Fh	<p>Receive FIFO Interrupt Level Bits</p> <p>Receive FIFO generates an interrupt when the FIFO status bits (RXFFST4-0) are greater than or equal to the FIFO level bits (RXFFIL4-0). The default value of these bits after reset is 11111.</p> <p>This avoids frequent interrupts after reset, as the receive FIFO will be empty most of the time.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = A RX FIFO interrupt request is generated when there is 0 or more words in the RX buffer.</p> <p>1h (R/W) = A RX FIFO interrupt request is generated when there are 1 or more words in the RX buffer.</p> <p>2h (R/W) = A RX FIFO interrupt request is generated when there are 2 or more words in the RX buffer.</p> <p>3h (R/W) = A RX FIFO interrupt request is generated when there are 3 or more words in the RX buffer.</p> <p>4h (R/W) = A RX FIFO interrupt request is generated when there are 4 words in the RX buffer.</p> <p>1Fh (R/W) = Reserved.</p>

### 8.5.2.11 SPI FIFO Control (SPIFFCT) Register (Offset = Ch) [reset = 0h]

SPIFFCT controls the FIFO transmit delay bits.

**Figure 8-20. SPI FIFO Control (SPIFFCT) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXDLY							
R/W-0h							

**Table 8-19. SPI FIFO Control (SPIFFCT) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	TXDLY	R/W	0h	<p>FIFO Transmit Delay Bits</p> <p>These bits define the delay between every transfer from FIFO transmit buffer to transmit shift register. The delay is defined in number SPI serial clock cycles. The 8-bit register could define a minimum delay of 0 serial clock cycles and a maximum of 255 serial clock cycles. In FIFO mode, the buffer (TXBUF) between the shift register and the FIFO should be filled only after the shift register has completed shifting of the last bit. This is required to pass on the delay between transfers to the data stream. In the FIFO mode TXBUF should not be treated as one additional level of buffer.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF immediately upon completion of transmission of the previous word.</p> <p>1h (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF1 serial clock cycle after completion of transmission of the previous word.</p> <p>2h (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF 2 serial clock cycles after completion of transmission of the previous word.</p> <p>FFh (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF 255 serial clock cycles after completion of transmission of the previous word.</p>

### 8.5.2.12 SPI Priority Control (SPIPRI) Register (Offset = Fh) [reset = 0h]

SPIPRI controls auxiliary functions for the SPI including emulation control, and 3-wire control.

**Figure 8-21. SPI Priority Control (SPIPRI) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	RESERVED	SOFT	FREE	RESERVED			TRIWIRES
R-0h	R/W-0h	R/W-0h	R/W-0h	R-0h			R/W-0h

**Table 8-20. SPI Priority Control (SPIPRI) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	RESERVED	R/W	0h	Reserved
5	SOFT	R/W	0h	<p>Emulation Soft Run</p> <p>This bit only has an effect when the FREE bit is 0.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Transmission stops midway in the bit stream while TSUSPEND is asserted. Once TSUSPEND is deasserted without a system reset, the remainder of the bits pending in the DATBUF are shifted. Example: If SPIDAT has shifted 3 out of 8 bits, the communication freezes right there. However, if TSUSPEND is later deasserted without resetting the SPI, SPI starts transmitting from where it had stopped (fourth bit in this case) and will transmit 8 bits from that point.</p> <p>1h (R/W) = If the emulation suspend occurs before the start of a transmission, (that is, before the first SPICLK pulse) then the transmission will not occur. If the emulation suspend occurs after the start of a transmission, then the data will be shifted out to completion. When the start of transmission occurs is dependent on the baud rate used.</p> <p>Standard SPI mode: Stop after transmitting the words in the shift register and buffer. That is, after TXBUF and SPIDAT are empty.</p> <p>In FIFO mode: Stop after transmitting the words in the shift register and buffer. That is, after TX FIFO and SPIDAT are empty.</p>
4	FREE	R/W	0h	<p>Emulation Free Run</p> <p>These bits determine what occurs when an emulation suspend occurs (for example, when the debugger hits a breakpoint). The peripheral can continue whatever it is doing (free-run mode) or, if in stop mode, it can either stop immediately or stop when the current operation (the current receive/transmit sequence) is complete.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Emulation mode is selected by the SOFT bit</p> <p>1h (R/W) = Free run, continue SPI operation regardless of suspend or when the suspend occurred.</p>
3-1	RESERVED	R	0h	Reserved

**Table 8-20. SPI Priority Control (SPIPRI) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
0	TRIWIRE	R/W	0h	SPI 3-wire Mode Enable Reset type: SYSRSn 0h (R/W) = Normal 4-wire SPI mode. 1h (R/W) = 3-wire SPI mode enabled. The unused pin becomes a GPIO pin. In master mode, the SPISIMO pin becomes the SPIMOMI (master receive and transmit) pin and SPISOMI is free for non-SPI use. In slave mode, the SPISOMI pin becomes the SPISISO (slave receive and transmit) pin and SPISIMO is free for non-SPI use.



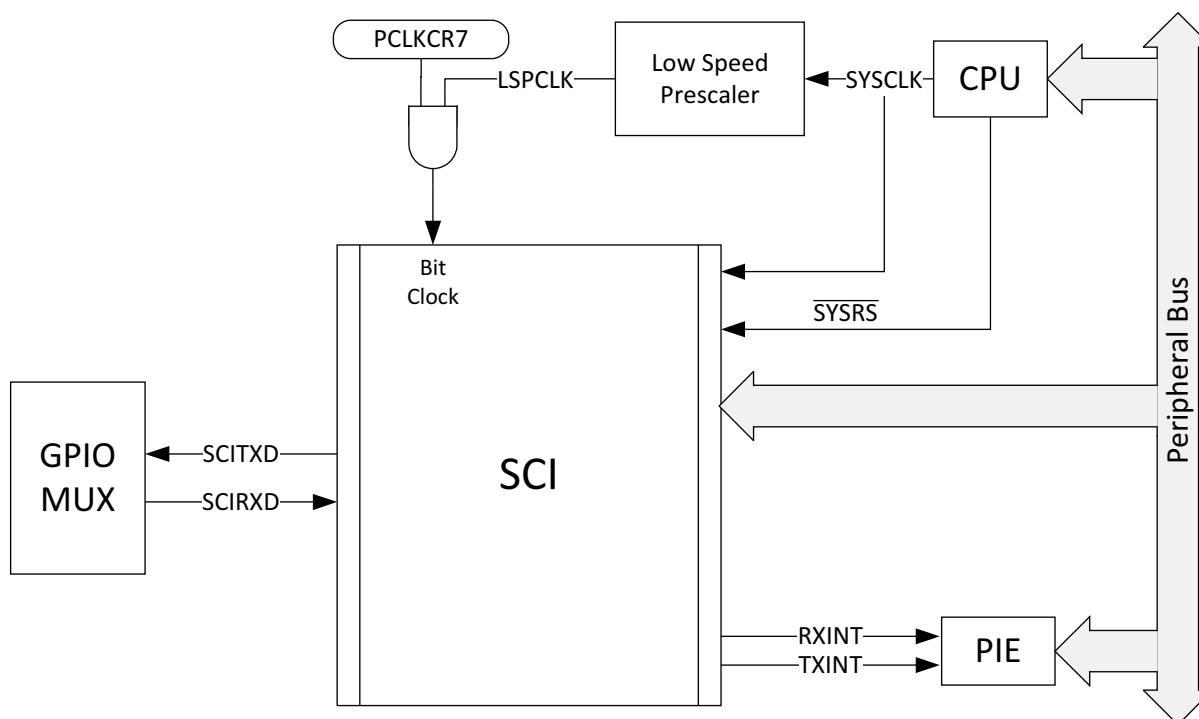
This chapter describes the features and operation of the serial communication interface (SCI) module. SCI is a two-wire asynchronous serial port, commonly known as a UART. The SCI modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format. The SCI receiver and transmitter each have a 4-level deep FIFO for reducing servicing overhead, and each has its own separate enable and interrupt bits. Both can be operated independently for half-duplex communication, or simultaneously for full-duplex communication.

To specify data integrity, the SCI checks received data for break detection, parity, overrun, and framing errors. The bit rate is programmable to different speeds through a 16-bit baud-select register.

<b>9.1 Introduction</b> .....	<b>498</b>
<b>9.2 Architecture</b> .....	<b>499</b>
<b>9.3 SCI Module Signal Summary</b> .....	<b>499</b>
<b>9.4 Configuring Device Pins</b> .....	<b>500</b>
<b>9.5 Multiprocessor and Asynchronous Communication Modes</b> .....	<b>501</b>
<b>9.6 SCI Programmable Data Format</b> .....	<b>501</b>
<b>9.7 SCI Multiprocessor Communication</b> .....	<b>502</b>
<b>9.8 Idle-Line Multiprocessor Mode</b> .....	<b>503</b>
<b>9.9 Address-Bit Multiprocessor Mode</b> .....	<b>505</b>
<b>9.10 SCI Communication Format</b> .....	<b>506</b>
<b>9.11 SCI Port Interrupts</b> .....	<b>508</b>
<b>9.12 SCI Baud Rate Calculations</b> .....	<b>508</b>
<b>9.13 SCI Enhanced Features</b> .....	<b>509</b>
<b>9.14 SCI Registers</b> .....	<b>512</b>

## 9.1 Introduction

The SCI interfaces are shown in [Figure 9-1](#).



**Figure 9-1. SCI CPU Interface**

### 9.1.1 Features

Features of the SCI module include:

- Two external pins:
  - SCITXD: SCI transmit-output pin
  - SCIRXD: SCI receive-input pin

Both pins can be used as GPIO if not used for SCI.

- Baud rate programmable to 64K different rates
- Data-word format
  - One start bit
  - Data-word length programmable from one to eight bits
  - Optional even/odd/no parity bit
  - One or two stop bits
  - An extra bit to distinguish addresses from data (address bit mode only)
- Four error-detection flags: parity, overrun, framing, and break detection
- Two wake-up multiprocessor modes: idle-line and address bit
- Half- or full-duplex operation
- Double-buffered receive and transmit functions
- Transmitter and receiver operations can be accomplished through interrupt-driven or polled algorithms with status flags
- Separate enable bits for transmitter and receiver interrupts (except BRKDT)
- NRZ (non-return-to-zero) format

Enhanced features include:

- Auto-baud-detect hardware logic
- 4-level transmit/receive FIFO

## 9.1.2 SCI Related Collateral

### Foundational Materials

- [One Minute RS-485 Introduction](#) (Video)
- [RS-232, RS-422, RS-485: What Are the Differences?](#) (Video)

### Getting Started Materials

- [\[FAQ\] My C2000 SCI is not Transmitting and/or Receiving data correctly, how do I fix this?](#)

## 9.1.3 Block Diagram

Figure 9-2 shows the SCI module block diagram. The SCI port operation is configured and controlled by the registers listed in [Section 9.14](#).

## 9.2 Architecture

The major elements used in full-duplex operation are shown in [Figure 9-2](#) and include:

- A transmitter (TX) and its major registers (upper half of [Figure 9-2](#))
  - SCITXBUF — transmitter data buffer register. Contains data (loaded by the CPU) to be transmitted
  - TXSHF register — transmitter shift register. Accepts data from register SCITXBUF and shifts data onto the SCITXD pin, one bit at a time
- A receiver (RX) and its major registers (lower half of [Figure 9-2](#))
  - RXSHF register — receiver shift register. Shifts data in from SCIRXD pin, one bit at a time
  - SCIRXBUF — receiver data buffer register. Contains data to be read by the CPU. Data from a remote processor is loaded into register RXSHF and then into registers SCIRXBUF and SCIRXEMU
- A programmable baud generator
- Control and status registers

The SCI receiver and transmitter can operate either independently or simultaneously.

## 9.3 SCI Module Signal Summary

A summarized description of each SCI signal name is shown in [Table 9-1](#).

**Table 9-1. SCI Module Signal Summary**

Signal Name	Description
<b>External signals</b>	
SCIRXD	SCI Asynchronous Serial Port receive data
SCITXD	SCI Asynchronous Serial Port transmit data
<b>Control</b>	
Baud clock	LSPCLK Prescaled clock
<b>Interrupt signals</b>	
TXINT	Transmit interrupt
RXINT	Receive Interrupt

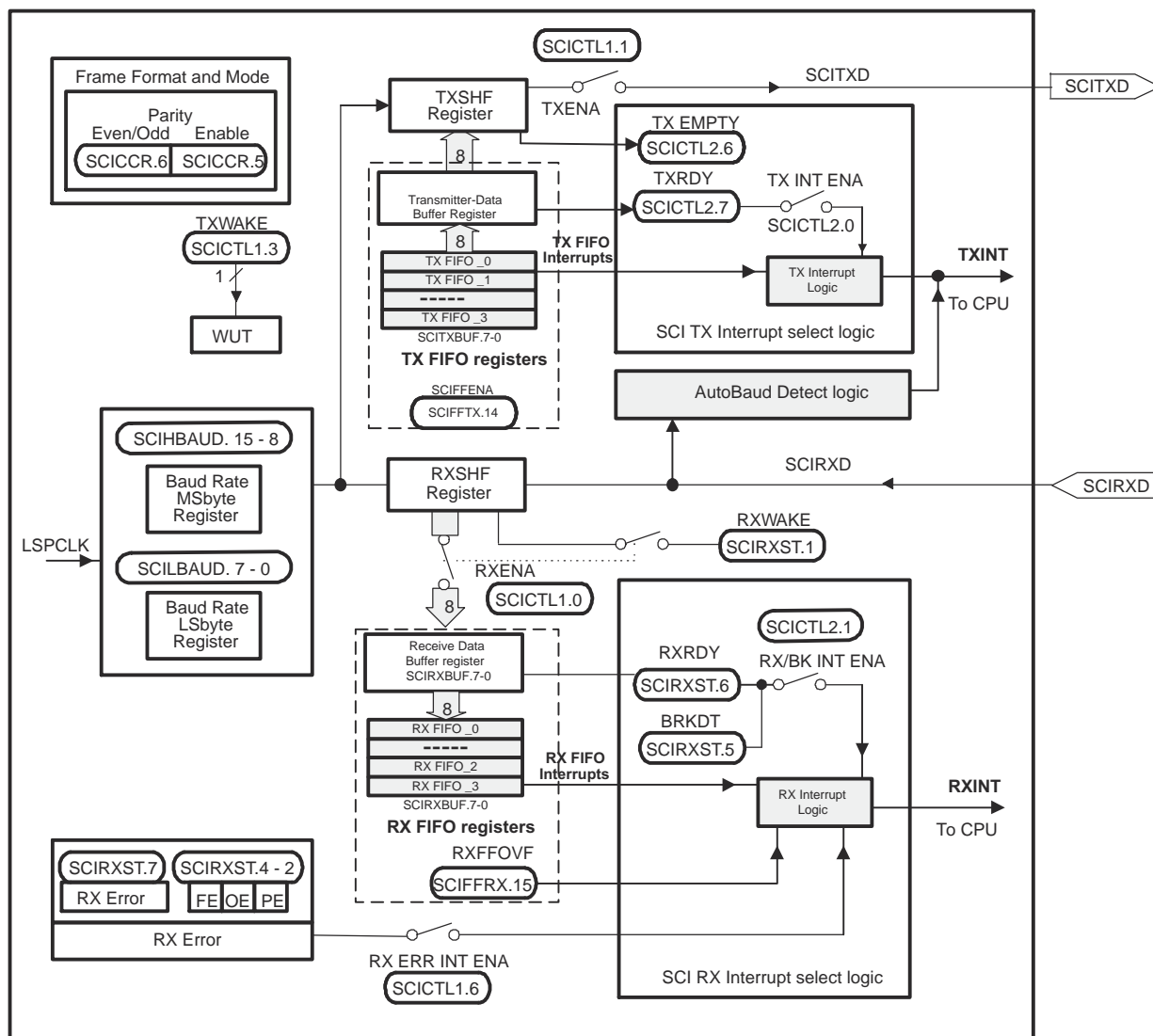


Figure 9-2. Serial Communications Interface (SCI) Module Block Diagram

### 9.4 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins. To avoid glitches on the pins, the GPyMUX bits must be configured first (while keeping the corresponding GPyMUX bits at the default of zero), followed by writing the GPyMUX register to the desired value.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPyPUD register.

See the *General-Purpose Input/Output (GPIO)* chapter for more details on GPIO mux and settings.

### 9.5 Multiprocessor and Asynchronous Communication Modes

The SCI has two multiprocessor protocols, the idle-line multiprocessor mode (see Section 9.8) and the address-bit multiprocessor mode (see Section 9.9). These protocols allow efficient data transfer between multiple processors.

The SCI offers the universal asynchronous receiver/transmitter (UART) communications mode for interfacing with many popular peripherals. The asynchronous mode (see Section 9.10) requires two lines to interface with many standard devices such as terminals and printers that use RS-232-C formats. Data transmission characteristics include:

- One start bit
- One to eight data bits
- An even/odd parity bit or no parity bit
- One or two stop bits

### 9.6 SCI Programmable Data Format

SCI data, both receive and transmit, is in NRZ (non-return-to-zero) format. The NRZ data format, shown in Figure 9-3, consists of:

- One start bit
- One to eight data bits
- An even/odd parity bit (optional)
- One or two stop bits
- An extra bit to distinguish addresses from data (address-bit mode only)

The basic unit of data is called a character and is one to eight bits in length. Each character of data is formatted with a start bit, one or two stop bits, and optional parity and address bits. A character of data with its formatting information is called a frame and is shown in Figure 9-3.

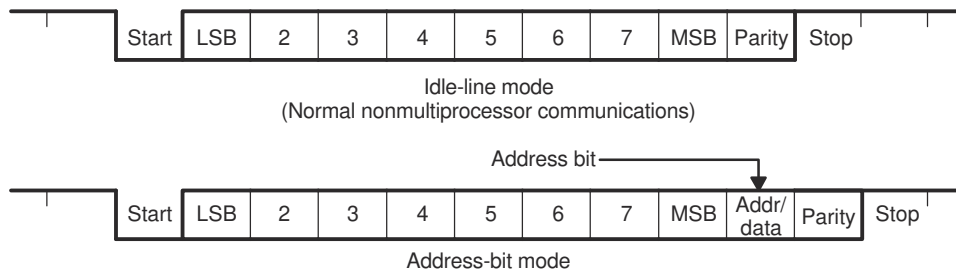


Figure 9-3. Typical SCI Data Frame Formats

To program the data format, use the SCICCR register. The bits used to program the data format are shown in Table 9-2.

Table 9-2. Programming the Data Format Using SCICCR

Bit(s)	Bit Name	Designation	Functions
2-0	SCICHAR	SCICCR.2:0	Select the character (data) length (one to eight bits).
5	PARITYENA (ENABLE)	SCICCR.5	Enables the parity function if set to 1, or disables the parity function if cleared to 0.
6	PARITY (EVEN/ODD)	SCICCR.6	If parity is enabled, selects odd parity if cleared to 0; even parity if set to 1.
7	STOPBITS	SCICCR.7	Determines the number of stop bits transmitted—one stop bit if cleared to 0 or two stop bits if set to 1.

## 9.7 SCI Multiprocessor Communication

The multiprocessor communication format allows one processor to efficiently send blocks of data to other processors on the same serial link. On one serial line, there should be only one transfer at a time. In other words, there can be only one talker on a serial line at a time.

### Address Byte

The first byte of a block of information that the talker sends contains an address byte that is read by all listeners. Only listeners with the correct address can be interrupted by the data bytes that follow the address byte. The listeners with an incorrect address remain uninterrupted until the next address byte.

### Sleep Bit

All processors on the serial link set the SCI SLEEP bit (bit 2 of SCICTL1) to 1 so that they are interrupted only when the address byte is detected. When a processor reads a block address that corresponds to the CPU device address as set by your application software, your program must clear the SLEEP bit to enable the SCI to generate an interrupt on receipt of each data byte.

Although the receiver still operates when the SLEEP bit is 1, it does not set RXRDY, RXINT, or any of the receiver error status bits to 1 unless the address byte is detected and the address bit in the received frame is a 1 (applicable to address-bit mode). The SCI does not alter the SLEEP bit; your software must alter the SLEEP bit.

### 9.7.1 Recognizing the Address Byte

A processor recognizes an address byte differently, depending on the multiprocessor mode used. For example:

- The idle-line mode ([Section 9.8](#)) leaves a quiet space before the address byte. This mode does not have an extra address/data bit and is more efficient than the address-bit mode for handling blocks that contain more than ten bytes of data. The idle-line mode should be used for typical non-multiprocessor SCI communication.
- The address-bit mode ([Section 9.9](#)) adds an extra bit (that is, an address bit) into every byte to distinguish addresses from data. This mode is more efficient in handling many small blocks of data because, unlike the idle mode, it does not have to wait between blocks of data. However, at a high transmit speed, the program is not fast enough to avoid a 10-bit idle in the transmission stream.

### 9.7.2 Controlling the SCI TX and RX Features

The multiprocessor mode is software selectable via the ADDR/IDLE MODE bit (SCICCR, bit 3). Both modes use the TXWAKE flag bit (SCICTL1, bit 3), RXWAKE flag bit (SCIRXST, bit1), and the SLEEP flag bit (SCICTL1, bit 2) to control the SCI transmitter and receiver features of these modes.

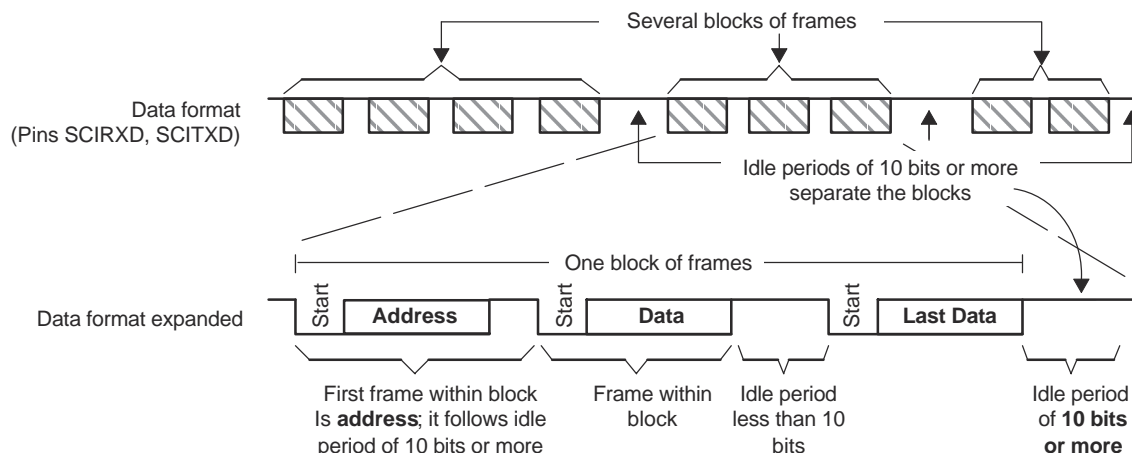
### 9.7.3 Receipt Sequence

In both multiprocessor modes, the receive sequence is as follows:

1. At the receipt of an address block, the SCI port wakes up and requests an interrupt (bit number 1 RX/BK INT ENA-of SCICTL2 must be enabled to request an interrupt). It reads the first frame of the block, which contains the destination address.
2. A software routine is entered through the interrupt and checks the incoming address. This address byte is checked against its device address byte stored in memory.
3. If the check shows that the block is addressed to the device CPU, the CPU clears the SLEEP bit and reads the rest of the block. If not, the software routine exits with the SLEEP bit still set, and does not receive interrupts until the next block start.

## 9.8 Idle-Line Multiprocessor Mode

In the idle-line multiprocessor protocol (ADDR/IDLE MODE bit=0), blocks are separated by having a longer idle time between the blocks than between frames in the blocks. An idle time of ten or more high-level bits after a frame indicates the start of a new block. The time of a single bit is calculated directly from the baud value (bits per second). The idle-line multiprocessor communication format is shown in Figure 9-4 (ADDR/IDLE MODE bit is bit 3 of SCICCR).



**Figure 9-4. Idle-Line Multiprocessor Communication Format**

### 9.8.1 Idle-Line Mode Steps

The steps followed by the idle-line mode:

1. SCI wakes up after receipt of the block-start signal.
2. The processor recognizes the next SCI interrupt.
3. The interrupt service routine compares the received address (sent by a remote transmitter) to its own.
4. If the CPU is being addressed, the service routine clears the SLEEP bit and receives the rest of the data block.
5. If the CPU is not being addressed, the SLEEP bit remains set. This lets the CPU continue to execute its main program without being interrupted by the SCI port until the next detection of a block start.

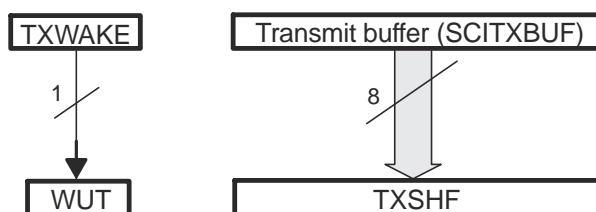
### 9.8.2 Block Start Signal

There are two ways to send a block-start signal:

- **Method 1:** Deliberately leave an idle time of ten bits or more by delaying the time between the transmission of the last frame of data in the previous block and the transmission of the address frame of the new block.
- **Method 2:** The SCI port first sets the TXWAKE bit (SCICTL1, bit 3) to 1 before writing to the SCITXBUF register. This sends an idle time of exactly 11 bits. In this method, the serial communications line is not idle any longer than necessary. (A don't care byte has to be written to SCITXBUF after setting TXWAKE, and before sending the address, so as to transmit the idle time.)

### 9.8.3 Wake-UP Temporary (WUT) Flag

Associated with the TXWAKE bit is the wake-up temporary (WUT) flag. WUT is an internal flag, double-buffered with TXWAKE. When TXSHF is loaded from SCITXBUF, WUT is loaded from TXWAKE, and the TXWAKE bit is cleared to 0. This arrangement is shown in [Figure 9-5](#).



**Figure 9-5. Double-Buffered WUT and TXSHF**

#### 9.8.3.1 Sending a Block Start Signal

To send out a block-start signal of exactly one frame time during a sequence of block transmissions:

1. Write a 1 to the TXWAKE bit.
2. Write a data word (content not important: a don't care) to the SCITXBUF register (transmit data buffer) to send a block-start signal. (The first data word written is suppressed while the block-start signal is sent out and ignored after that.) When the TXSHF (transmit shift register) is free again, SCITXBUF contents are shifted to TXSHF, the TXWAKE value is shifted to WUT, and then TXWAKE is cleared.

Because TXWAKE was set to a 1, the start, data, and parity bits are replaced by an idle period of 11 bits transmitted following the last stop bit of the previous frame.

3. Write a new address value to SCITXBUF.

A don't-care data word must first be written to register SCITXBUF so that the TXWAKE bit value can be shifted to WUT. After the don't-care data word is shifted to the TXSHF register, the SCITXBUF (and TXWAKE, if necessary) can be written to again because TXSHF and WUT are both double-buffered.

### 9.8.4 Receiver Operation

The receiver operates regardless of the SLEEP bit. However, the receiver neither sets RXRDY nor the error status bits, nor does it request a receive interrupt until an address frame is detected.



## 9.9 Address-Bit Multiprocessor Mode

In the address-bit protocol (ADDR/IDLE MODE bit=1), frames have an extra bit called an address bit that immediately follows the last data bit. The address bit is set to 1 in the first frame of the block and to 0 in all other frames. The idle period timing is irrelevant (see Figure 9-6).

### 9.9.1 Sending an Address

The TXWAKE bit value is placed in the address bit. During transmission, when the SCITXBUF register and TXWAKE are loaded into the TXSHF register and WUT respectively, TXWAKE is reset to 0 and WUT becomes the value of the address bit of the current frame. Thus, to send an address:

1. Set the TXWAKE bit to 1 and write the appropriate address value to the SCITXBUF register.

When this address value is transferred to the TXSHF register and shifted out, its address bit is sent as a 1. This flags the other processors on the serial link to read the address.

2. Write to SCITXBUF and TXWAKE after TXSHF and WUT are loaded. (Can be written to immediately since both TXSHF and WUT are both double-buffered.)
3. Leave the TXWAKE bit set to 0 to transmit non-address frames in the block.

#### Note

As a general rule, the address-bit format is typically used for data frames of 11 bytes or less. This format adds one bit value (1 for an address frame, 0 for a data frame) to all data bytes transmitted. The idle-line format is typically used for data frames of 12 bytes or more.

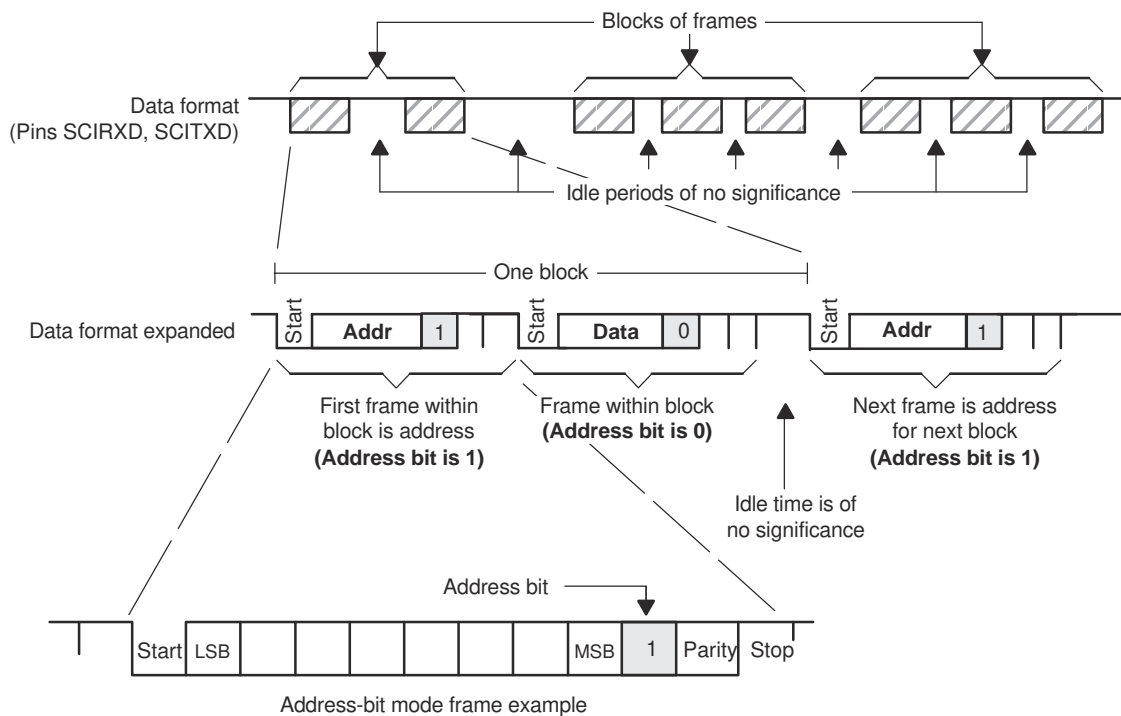


Figure 9-6. Address-Bit Multiprocessor Communication Format

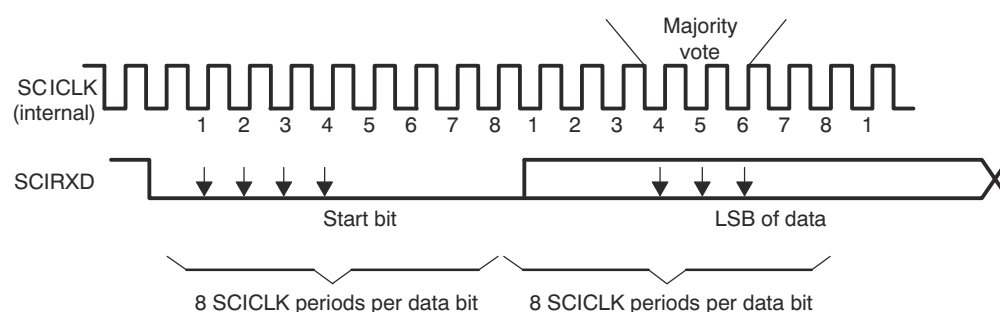
## 9.10 SCI Communication Format

The SCI asynchronous communication format uses either single line (one way) or two line (two way) communications. In this mode, the frame consists of a start bit, one to eight data bits, an optional even/odd parity bit, and one or two stop bits (shown in Figure 9-7). There are eight SCICLK periods per data bit.

The receiver begins operation on receipt of a valid start bit. A valid start bit is identified by four consecutive internal SCICLK periods of zero bits as shown in Figure 9-7. If any bit is not zero, then the processor starts over and begins looking for another start bit.

For the bits following the start bit, the processor determines the bit value by making three samples in the middle of the bits. These samples occur on the fourth, fifth, and sixth SCICLK periods, and bit-value determination is on a majority (two out of three) basis. Figure 9-7 illustrates the asynchronous communication format for this with a start bit showing where a majority vote is taken.

Since the receiver synchronizes itself to frames, the external transmitting and receiving devices do not have to use a synchronized serial clock. The clock can be generated locally.

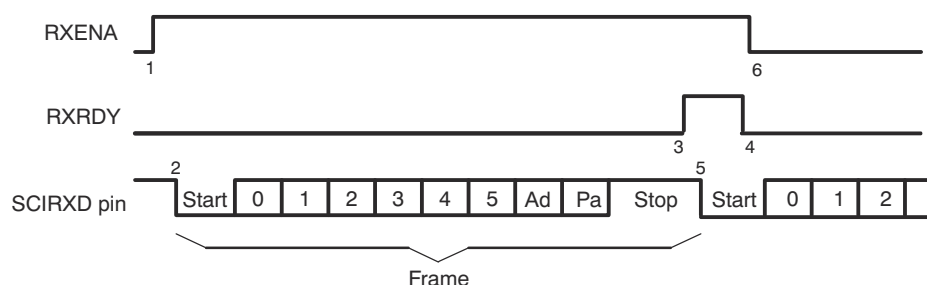


**Figure 9-7. SCI Asynchronous Communications Format**

### 9.10.1 Receiver Signals in Communication Modes

Figure 9-8 illustrates an example of receiver signal timing that assumes the following conditions:

- Address-bit wake-up mode (address bit does not appear in idle-line mode)
- Six bits per character



**Figure 9-8. SCI RX Signals in Communication Modes**

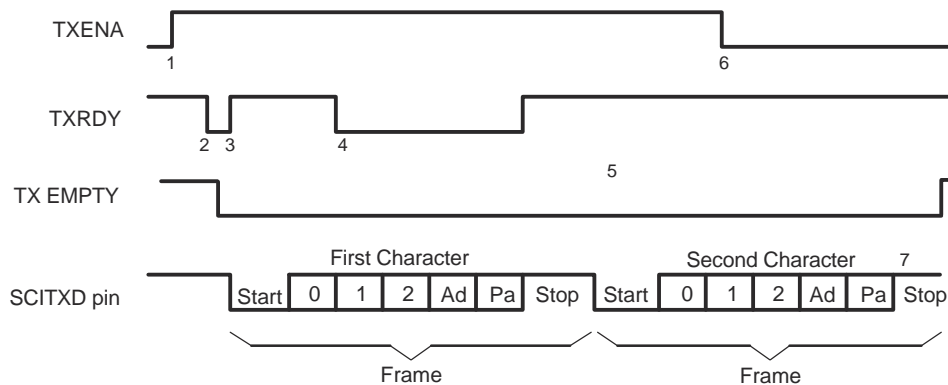
Notes:

1. Flag bit RXENA (SCICTL1, bit 0) goes high to enable the receiver.
2. Data arrives on the SCIRXD pin, start bit detected.
3. Data is shifted from RXSHF to the receiver buffer register (SCIRXBUF); an interrupt is requested. Flag bit RXRDY (SCIRXST, bit 6) goes high to signal that a new character has been received.
4. The program reads SCIRXBUF; flag RXRDY is automatically cleared.
5. The next byte of data arrives on the SCIRXD pin; the start bit is detected, then cleared.
6. Bit RXENA is brought low to disable the receiver. Data continues to be assembled in RXSHF but is not transferred to the receiver buffer register.

### 9.10.2 Transmitter Signals in Communication Modes

Figure 9-9 illustrates an example of transmitter signal timing that assumes the following conditions:

- Address-bit wake-up mode (address bit does not appear in idle-line mode)
- Three bits per character



**Figure 9-9. SCI TX Signals in Communications Mode**

**Notes:**

1. Bit TXENA (SCICTL1, bit 1) goes high, enabling the transmitter to send data.
2. SCITXBUF is written to; thus, (1) the transmitter is no longer empty, and (2) TXRDY goes low.
3. The SCI transfers data to the shift register (TXSHF). The transmitter is ready for a second character (TXRDY goes high), and it requests an interrupt (to enable an interrupt, bit TX INT ENA — SCICTL2, bit 0 — must be set).
4. The program writes a second character to SCITXBUF after TXRDY goes high (item 3). (TXRDY goes low again after the second character is written to SCITXBUF.)
5. Transmission of the first character is complete. Transfer of the second character to shift register TXSHF begins.
6. Bit TXENA goes low to disable the transmitter; the SCI finishes transmitting the current character.
7. Transmission of the second character is complete; transmitter is empty and ready for new character.

## 9.11 SCI Port Interrupts

The SCI receiver and transmitter can be interrupt controlled. The SCICTL2 register has one flag bit (TXRDY) that indicates active interrupt conditions, and the SCIRXST register has two interrupt flag bits (RXRDY and BRKDT), plus the RX ERROR interrupt flag that is a logical-OR of the FE, OE, BRKDT, and PE conditions. The transmitter and receiver have separate interrupt-enable bits. When not enabled, the interrupts are not asserted; however, the condition flags remain active, reflecting transmission and receipt status.

The SCI has independent peripheral interrupt vectors for the receiver and transmitter. Peripheral interrupt requests can be either high priority or low priority. This is indicated by the priority bits that are output from the peripheral to the PIE controller. When both RX and TX interrupt requests are made at the same priority level, the receiver always has higher priority than the transmitter, reducing the possibility of receiver overrun.

The operation of peripheral interrupts is described in the Peripheral Interrupts section of the *System Control and Interrupts* chapter.

- If the RX/BK INT ENA bit (SCICTL2, bit 1) is set, the receiver peripheral interrupt request is asserted when one of the following events occurs:
  - The SCI receives a complete frame and transfers the data in the RXSHF register to the SCIRXBUF register. This action sets the RXRDY flag (SCIRXST, bit 6) and initiates an interrupt.
  - A break detect condition occurs (the SCIRXD is low for 9.625 bit periods following a missing stop bit). This action sets the BRKDT flag bit (SCIRXST, bit 5) and initiates an interrupt.
- If the TX INT ENA bit (SCICTL2.0) is set, the transmitter peripheral interrupt request is asserted whenever the data in the SCITXBUF register is transferred to the TXSHF register, indicating that the CPU can write to SCITXBUF; this action sets the TXRDY flag bit (SCICTL2, bit 7) and initiates an interrupt.

---

### Note

Interrupt generation due to the RXRDY and BRKDT bits is controlled by the RX/BK INT ENA bit (SCICTL2, bit 1). Interrupt generation due to the RX ERROR bit is controlled by the RX ERR INT ENA bit (SCICTL1, bit 6).

---

## 9.12 SCI Baud Rate Calculations

The internally generated serial clock is determined by the low-speed peripheral clock (LSPCLK) and the baud-select registers. The SCI uses the 16-bit value of the baud-select registers to select one of the 64K different serial clock rates possible for a given LSPCLK.

See the bit descriptions in the baud-select registers, for the formula to use when calculating the SCI asynchronous baud. [Table 9-3](#) shows the baud-select values for common SCI bit rates. LSPCLK/16 is the maximum baud rate. For example, if LSPCLK is 100 MHz, then the maximum baud rate is 6.25 Mbps.

**Table 9-3. Asynchronous Baud Register Values for Common SCI Bit Rates**

Ideal Baud	LSPCLK Clock Frequency, 100 MHz		
	BRR	Actual Baud	% Error
2400	5207 (1457h)	2400	0
4800	2603 (A2Bh)	4800	0
9600	1301 (515h)	9601	0.01
19200	650 (28Ah)	19201	0.01
38400	324 (144h)	38462	0.16

## 9.13 SCI Enhanced Features

The C28x SCI features autobaud detection and transmit/receive FIFO. The following section explains the FIFO operation.

### 9.13.1 SCI FIFO Description

The following steps explain the FIFO features and help with programming the SCI with FIFOs.

1. **Reset.** At reset the SCI powers up in standard SCI mode and the FIFO function is disabled. The FIFO registers SCIFFTX, SCIFFRX, and SCIFFCT remain inactive.
2. **Standard SCI.** The standard SCI modes will work normally with TXINT/RXINT interrupts as the interrupt source for the module.
3. **FIFO enable.** FIFO mode is enabled by setting the SCIFFEN bit in the SCIFFTX register. SCIRST can reset the FIFO mode at any stage of its operation.
4. **Active registers.** All the SCI registers and SCI FIFO registers (SCIFFTX, SCIFFRX, and SCIFFCT) are active.
5. **Interrupts.** FIFO mode has two interrupts; one for transmit FIFO, TXINT and one for receive FIFO, RXINT. RXINT is the common interrupt for SCI FIFO receive, receive error, and receive FIFO overflow conditions. The TXINT of the standard SCI will be disabled and this interrupt will service as SCI transmit FIFO interrupt.
6. **Buffers.** Transmit and receive buffers are supplemented with two 4-level FIFOs. The transmit FIFO registers are 8 bits wide and receive FIFO registers are 10 bits wide. The one-word transmit buffer (SCITXBUF) of the standard SCI functions as a transition buffer before the transmit FIFO and shift register. SCITXBUF is loaded into either the FIFO (when FIFO is enabled) or TXSHF (when FIFO is disabled). When FIFO is enabled, SCITXBUF loads into the FIFO only after the last bit of the shift register is shifted out, so SCITXBUF should not be treated as an additional level of buffer. With the FIFO enabled, TXSHF is directly loaded from the FIFO (not TXBUF) after an optional delay value (SCIFFCT). When FIFO mode is enabled for SCI, characters written to SCITXBUF are queued in to SCI-TXFIFO and the characters received in SCI-RXFIFO can be read using SCIRXBUF.
7. **Delayed transfer.** The rate at which words in the FIFO are transferred to the transmit shift register is programmable. The SCIFFCT register bits (7–0) FFTXDLY7–FFTXDLY0 define the delay between the word transfer. The delay is defined in the number SCI baud clock cycles. The 8 bit register can define a minimum delay of 0 baud clock cycles and a maximum of 256-baud clock cycles. With zero delay, the SCI module can transmit data in continuous mode with the FIFO words shifting out back to back. With the 256 clock delay the SCI module can transmit data in a maximum delayed mode with the FIFO words shifting out with a delay of 256 baud clocks between each words. The programmable delay facilitates communication with slow SCI/UARTs with little CPU intervention.
8. **FIFO status bits.** Both the transmit and receive FIFOs have status bits TXFFST or RXFFST (bits 12–8) that define the number of words available in the FIFOs at any time. The transmit FIFO reset bit TXFIFO and receive reset bit RXFIFO reset the FIFO pointers to zero when these bits are cleared to 0. The FIFOs resumes operation from start once these bits are set to 1.
9. **Programmable interrupt levels.** Both transmit and receive FIFO can generate CPU interrupts. The interrupt trigger is generated whenever the transmit FIFO status bits TXFFST (bits 12–8) match (less than or equal to) the interrupt trigger level bits TXFFIL (bits 4–0). This provides a programmable interrupt trigger for transmit and receive sections of the SCI. Default value for these trigger level bits will be 0x1111 for receive FIFO and 0x0000 for transmit FIFO, respectively.

Figure 9-10 and Table 9-4 explain the operation/configuration of SCI interrupts in nonFIFO/FFO mode.

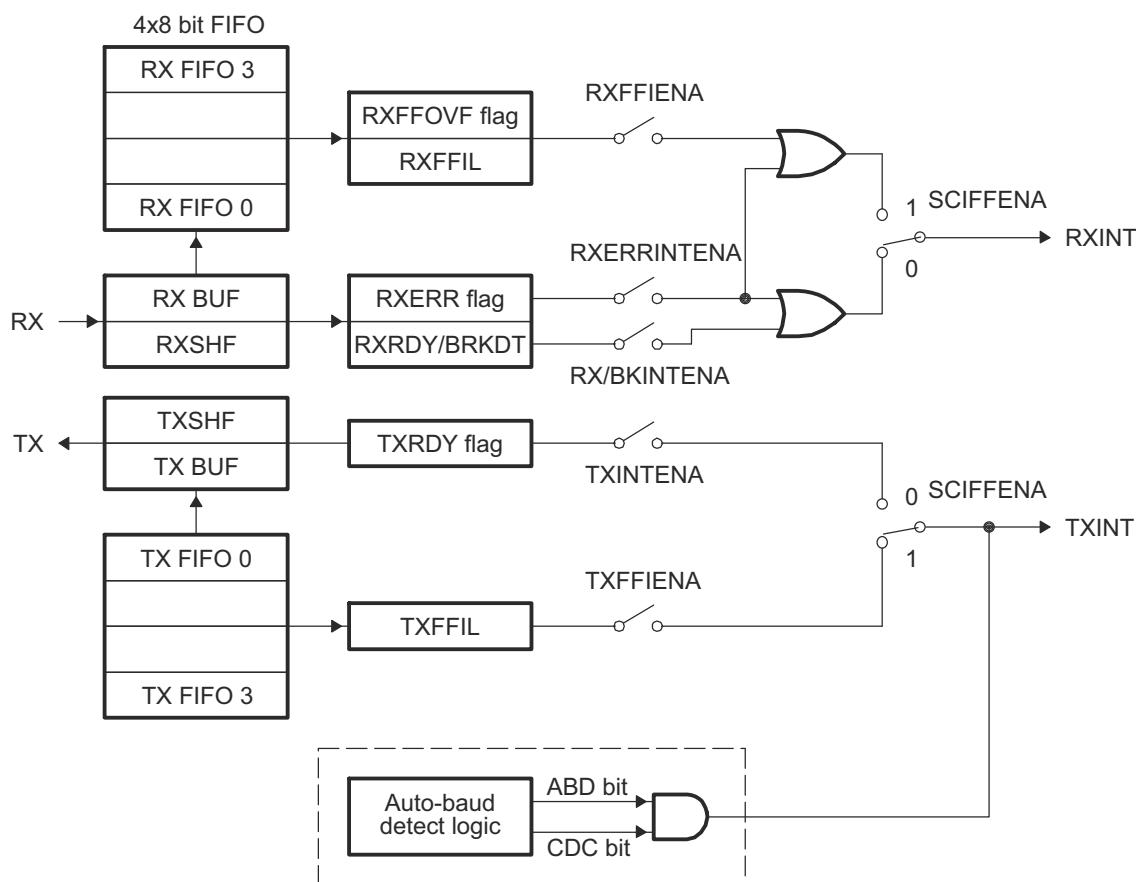


Figure 9-10. SCI FIFO Interrupt Flags and Enable Logic

Table 9-4. SCI Interrupt Flags

FIFO Options <sup>(1)</sup>	SCI Interrupt Source	Interrupt Flags	Interrupt Enables	FIFO Enable SCIFFENA	Interrupt Line
SCI without FIFO	Receive error	RXERR <sup>(2)</sup>	RXERRINTENA	0	RXINT
	Receive break	BRKDT	RX/BKINTENA	0	RXINT
	Data receive	RXRDY	RX/BKINTENA	0	RXINT
	Transmit empty	TXRDY	TXINTENA	0	TXINT
SCI with FIFO	Receive error and receive break	RXERR	RXERRINTENA	1	RXINT
	FIFO receive	RXFFIL	RXFFIENA	1	RXINT
	Transmit empty	TXFFIL	TXFFIENA	1	TXINT
Auto-baud	Auto-baud detected	ABD	Don't care	x	TXINT

(1) FIFO mode TXSHF is directly loaded after delay value, TXBUF is not used.

(2) RXERR can be set by BRKDT, FE, OE, PE flags. In FIFO mode, BRKDT interrupt is only through RXERR flag.

### 9.13.2 SCI Auto-Baud

Most SCI modules do not have an auto-baud detect logic built-in hardware. These SCI modules are integrated with embedded controllers whose clock rates are dependent on PLL reset values. Often embedded controller clocks change after final design. In the enhanced feature set this module supports an autobaud-detect logic in hardware. The following section explains the enabling sequence for autobaud-detect feature.

#### 9.13.3 Autobaud-Detect Sequence

Bits ABD and CDC in SCIFFCT control the autobaud logic. The SCIRST bit should be enabled to make autobaud logic work.

If ABD is set while CDC is 1, which indicates auto-baud alignment, SCI transmit FIFO interrupt will occur (TXINT). After the interrupt service CDC bit has to be cleared by software. If CDC remains set even after interrupt service, there should be no repeat interrupts.

1. Enable autobaud-detect mode for the SCI by setting the CDC bit (bit 13) in SCIFFCT and clearing the ABD bit (Bit 15) by writing a 1 to ABDCLR bit (bit 14).
2. Initialize the baud register to be 1 or less than a baud rate limit of 500 Kbps.
3. Allow SCI to receive either character "A" or "a" from a host at the desired baud rate. If the first character is either "A" or "a", the autobaud-detect hardware will detect the incoming baud rate and set the ABD bit.
4. The auto-detect hardware will update the baud rate register with the equivalent baud value hex. The logic will also generate an interrupt to the CPU.
5. Respond to the interrupt clear ADB bit by writing a 1 to ABD CLR (bit 14) of SCIFFCT register and disable further autobaud locking by clearing CDC bit by writing a 0.
6. Read the receive buffer for character "A" or "a" to empty the buffer and buffer status.
7. If ABD is set while CDC is 1, which indicates autobaud alignment, the SCI transmit FIFO interrupt will occur (TXINT). After the interrupt service CDC bit must be cleared by software.

---

#### Note

At higher baud rates, the slew rate of the incoming data bits can be affected by transceiver and connector performance. While normal serial communications may work well, this slew rate may limit reliable autobaud detection at higher baud rates (typically beyond 100k baud) and cause the auto-baudlock feature to fail.

To avoid this, the following is recommended:

- Achieve a baud-lock between the host and C28x SCI boot loader using a lower baud rate.
  - The host may then handshake with the loaded C28x application to set the SCI baud rate register to the desired higher baud rate.
-

## 9.14 SCI Registers

The section describes the Serial Communication Interface module registers.

### 9.14.1 SCI Base Addresses

**Table 9-5. SCI Base Address Table (C28)**

Bit Field Name		Base Address
Instance	Structure	
SciaRegs	SCI_REGS	0x0000_7050

### 9.14.2 SCI\_REGS Registers

[Table 9-6](#) lists the SCI\_REGS registers. All register offset addresses not listed in [Table 9-6](#) should be considered as reserved locations and the register contents should not be modified.

**Table 9-6. SCI\_REGS Registers**

Offset	Acronym	Register Name	Section
0h	SCICCR	SCI Communications Control Register	<a href="#">Section 9.14.2.1</a>
1h	SCICTL1	SCI Control Register 1	<a href="#">Section 9.14.2.2</a>
2h	SCIHBAUD	SCI Baud Rate (high) Register	<a href="#">Section 9.14.2.3</a>
3h	SCILBAUD	SCI Baud Rate (low) Register	<a href="#">Section 9.14.2.4</a>
4h	SCICTL2	SCI Control Register 2	<a href="#">Section 9.14.2.5</a>
5h	SCIRXST	SCI Receive Status Register	<a href="#">Section 9.14.2.6</a>
6h	SCIRXEMU	SCI Receive Emulation Buffer Register	<a href="#">Section 9.14.2.7</a>
7h	SCIRXBUF	SCI Receive Data Buffer Register	<a href="#">Section 9.14.2.8</a>
9h	SCITXBUF	SCI Transmit Data Buffer Register	<a href="#">Section 9.14.2.9</a>
Ah	SCIFFTX	SCI FIFO Transmit Register	<a href="#">Section 9.14.2.10</a>
Bh	SCIFFRX	SCI FIFO Receive Register	<a href="#">Section 9.14.2.11</a>
Ch	SCIFFCT	SCI FIFO Control Register	<a href="#">Section 9.14.2.12</a>
Fh	SCIPRI	SCI Priority Control Register	<a href="#">Section 9.14.2.13</a>



Complex bit access types are encoded to fit into small table cells. [Table 9-7](#) shows the codes that are used for access types in this section.

**Table 9-7. SCI\_REGS Access Type Codes**

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R-0	Read Returns 0s
Write Type		
W	W	Write
W1S	W1S	Write 1 to set
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

### 9.14.2.1 SCI Communications Control Register (SCICCR) (Offset = 0h) [reset = 0h]

SCICCR defines the character format, protocol, and communications mode used by the SCI.

**Figure 9-11. SCI Communications Control Register (SCICCR)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
STOPBITS	PARITY	PARITYENA	LOOPBKENA	ADDRIDLE_MODE	SCICHAR		
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h		

**Table 9-8. SCI Communications Control Register (SCICCR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	STOPBITS	R/W	0h	SCI number of stop bits. This bit specifies the number of stop bits transmitted. The receiver checks for only one stop bit. Reset type: SYSRSn 0h (R/W) = One stop bit 1h (R/W) = Two stop bits
6	PARITY	R/W	0h	SCI parity odd/even selection. If the PARITY ENABLE bit (SCICCR, bit 5) is set, PARITY (bit 6) designates odd or even parity (odd or even number of bits with the value of 1 in both transmitted and received characters). Reset type: SYSRSn 0h (R/W) = Odd parity 1h (R/W) = Even parity
5	PARITYENA	R/W	0h	SCI parity enable. This bit enables or disables the parity function. If the SCI is in the address bit multiprocessor mode (set using bit 3 of this register), the address bit is included in the parity calculation (if parity is enabled). For characters of less than eight bits, the remaining unused bits should be masked out of the parity calculation. Reset type: SYSRSn 0h (R/W) = Parity disabled no parity bit is generated during transmission or is expected during reception 1h (R/W) = Parity is enabled
4	LOOPBKENA	R/W	0h	Loop Back test mode enable. This bit enables the Loop Back test mode where the Tx pin is internally connected to the Rx pin. Reset type: SYSRSn 0h (R/W) = Loop Back test mode disabled 1h (R/W) = Loop Back test mode enabled
3	ADDRIDLE_MODE	R/W	0h	SCI multiprocessor mode control bit. This bit selects one of the multiprocessor protocols. Multiprocessor communication is different from the other communication modes because it uses SLEEP and TXWAKE functions (bits SCICTL1, bit 2 and SCICTL1, bit 3, respectively). The idle-line mode is usually used for normal communications because the address-bit mode adds an extra bit to the frame. The idle-line mode does not add this extra bit and is compatible with RS-232 type communications. Reset type: SYSRSn 0h (R/W) = Idle-line mode protocol selected 1h (R/W) = Address-bit mode protocol selected

**Table 9-8. SCI Communications Control Register (SCICCR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
2-0	SCICCHAR	R/W	0h	Character-length control bits 2-0. These bits select the SCI character length from one to eight bits. Characters of less than eight bits are right-justified in SCIRXBUF and SCIRXEMU and are padded with leading zeros in SCIRXBUF. SCITXBUF doesn't need to be padded with leading zeros. Reset type: SYSRSn 0h (R/W) = SCICCHAR_LENGTH_1 1h (R/W) = SCICCHAR_LENGTH_2 2h (R/W) = SCICCHAR_LENGTH_3 3h (R/W) = SCICCHAR_LENGTH_4 4h (R/W) = SCICCHAR_LENGTH_5 5h (R/W) = SCICCHAR_LENGTH_6 6h (R/W) = SCICCHAR_LENGTH_7 7h (R/W) = SCICCHAR_LENGTH_8

### 9.14.2.2 SCI Control Register 1 (SCICTL1) (Offset = 1h) [reset = 0h]

SCICTL1 controls the receiver/transmitter enable, TXWAKE and SLEEP functions, and the SCI software reset.

**Figure 9-12. SCI Control Register 1 (SCICTL1)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	RXERRINTENA	SWRESET	RESERVED	TXWAKE	SLEEP	TXENA	RXENA
R-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

**Table 9-9. SCI Control Register 1 (SCICTL1) Field Descriptions**

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	RXERRINTENA	R/W	0h	SCI receive error interrupt enable. Setting this bit enables an interrupt if the RX ERROR bit (SCIRXST, bit 7) becomes set because of errors occurring. Reset type: SYSRSn 0h (R/W) = Receive error interrupt disabled 1h (R/W) = Receive error interrupt enabled
5	SWRESET	R/W	0h	SCI software reset (active low). Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition. The SW RESET bit does not affect any of the configuration bits. All affected logic is held in the specified reset state until a 1 is written to SW RESET (the bit values following a reset are shown beneath each register diagram in this section). Thus, after a system reset, re-enable the SCI by writing a 1 to this bit. Clear this bit after a receiver break detect (BRKDT flag, bit SCIRXST, bit 5). SW RESET affects the operating flags of the SCI, but it neither affects the configuration bits nor restores the reset values. Once SW RESET is asserted, the flags are frozen until the bit is deasserted. The affected flags are as follows: Value After SW SCI Flag Register Bit RESET 1 TXRDY SCICTL2, bit 7 1 TX EMPTY SCICTL2, bit 6 0 RXWAKE SCIRXST, bit 1 0 PE SCIRXST, bit 2 0 OE SCIRXST, bit 3 0 FE SCIRXST, bit 4 0 BRKDT SCIRXST, bit 5 0 RXRDY SCIRXST, bit 6 0 RX ERROR SCIRXST, bit 7 Reset type: SYSRSn 0h (R/W) = Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition. 1h (R/W) = After a system reset, re-enable the SCI by writing a 1 to this bit.
4	RESERVED	R	0h	Reserved

**Table 9-9. SCI Control Register 1 (SCICTL1) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
3	TXWAKE	R/W	0h	<p>SCI transmitter wake-up method select.</p> <p>The TXWAKE bit controls selection of the data-transmit feature, depending on which transmit mode (idle-line or address-bit) is specified at the ADDR/IDLE MODE bit (SCICCR, bit 3)</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Transmit feature is not selected. In idle-line mode: write a 1 to TXWAKE, then write data to register SCITXBUF to generate an idle period of 11 data bits In address-bit mode: write a 1 to TXWAKE, then write data to SCITXBUF to set the address bit for that frame to 1</p> <p>1h (R/W) = Transmit feature selected is dependent on the mode, idle-line or address-bit: TXWAKE is not cleared by the SW RESET bit (SCICTL1, bit 5)</p> <p>it is cleared by a system reset or the transfer of TXWAKE to the WUT flag.</p>
2	SLEEP	R/W	0h	<p>SCI sleep.</p> <p>The TXWAKE bit controls selection of the data-transmit feature, depending on which transmit mode (idle-line or address-bit) is specified at the ADDR/IDLE MODE bit (SCICCR, bit 3). In a multiprocessor configuration, this bit controls the receiver sleep function. Clearing this bit brings the SCI out of the sleep mode. The receiver still operates when the SLEEP bit is set however, operation does not update the receiver buffer ready bit (SCIRXST, bit 6, RXRDY) or the error status bits (SCIRXST, bit 5-2: BRKDT, FE, OE, and PE) unless the address byte is detected. SLEEP is not cleared when the address byte is detected.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Sleep mode disabled</p> <p>1h (R/W) = Sleep mode enabled</p>
1	TXENA	R/W	0h	<p>SCI transmitter enable.</p> <p>Data is transmitted through the SCITXD pin only when TXENA is set. If reset, transmission is halted but only after all data previously written to SCITXBUF has been sent. Data written into SCITXBUF when TXENA is disabled will not be transmitted even if the TXENA is enabled later.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Transmitter disabled</p> <p>1h (R/W) = Transmitter enabled</p>
0	RXENA	R/W	0h	<p>SCI receiver enable.</p> <p>Data is received on the SCIRXD pin and is sent to the receiver shift register and then the receiver buffers. This bit enables or disables the receiver (transfer to the buffers).</p> <p>Clearing RXENA stops received characters from being transferred to the two receiver buffers and also stops the generation of receiver interrupts. However, the receiver shift register can continue to assemble characters. Thus, if RXENA is set during the reception of a character, the complete character will be transferred into the receiver buffer registers, SCIRXEMU and SCIRXBUF.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Prevent received characters from transfer into the SCIRXEMU and SCIRXBUF receiver buffers</p> <p>1h (R/W) = Send received characters to SCIRXEMU and SCIRXBUF</p>

### 9.14.2.3 SCI Baud Rate (high) (SCIHBAUD) Register (Offset = 2h) [reset = 0h]

The values in SCIHBAUD and SCILBAUD specify the baud rate for the SCI.

**Figure 9-13. SCI Baud Rate (high) (SCIHBAUD) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
BAUD							
R/W-0h							

**Table 9-10. SCI Baud Rate (high) (SCIHBAUD) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	BAUD	R/W	0h	SCI 16-bit baud selection Registers SCIHBAUD (MSbyte). The internally-generated serial clock is determined by the low speed peripheral clock (LSPCLK) signal and the two baud-select registers. The SCI uses the 16-bit value of these registers to select one of 64K serial clock rates for the communication modes. $BRR = (SCIHBAUD \ll 8) + (SCILBAUD)$ The SCI baud rate is calculated using the following equation: SCI Asynchronous Baud = $LSPCLK / ((BRR + 1) * 8)$ Alternatively, $BRR = LSPCLK / (SCI \text{ Asynchronous Baud} * 8) - 1$ Note that the above formulas are applicable only when $0 < BRR < 65536$ . If $BRR = 0$ , then SCI Asynchronous Baud = $LSPCLK / 16$ Where: BRR = the 16-bit value (in decimal) in the baud-select registers Reset type: SYSRStn

#### 9.14.2.4 SCI Baud Rate (low) (SCILBAUD) Register (Offset = 3h) [reset = 0h]

The values in SCIHBAUD and SCILBAUD specify the baud rate for the SCI.

**Figure 9-14. SCI Baud Rate (low) (SCILBAUD) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
BAUD							
R/W-0h							

**Table 9-11. SCI Baud Rate (low) (SCILBAUD) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	BAUD	R/W	0h	See SCIHBAUD Detailed Description Reset type: SYSRSn

#### 9.14.2.5 SCI Control Register 2 (SCICTL2) (Offset = 4h) [reset = C0h]

SCICTL2 enables the receive-ready, break-detect, and transmit-ready interrupts as well as transmitter-ready and -empty flags.

**Figure 9-15. SCI Control Register 2 (SCICTL2)**

15	14	13	12	11	10	9	8	
RESERVED								
R-0h								
7	6	5	4	3	2	1	0	
TXRDY	TXEMPTY	RESERVED				RXBKINTENA	TXINTENA	
R-1h	R-1h	R-0h				R/W-0h	R/W-0h	

**Table 9-12. SCI Control Register 2 (SCICTL2) Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	TXRDY	R	1h	<p>Transmitter buffer register ready flag. When set, this bit indicates that the transmit data buffer register, SCITXBUF, is ready to receive another character. Writing data to the SCITXBUF automatically clears this bit. When set, this flag asserts a transmitter interrupt request if the interrupt-enable bit, TX INT ENA (SCICTL2.0), is also set. TXRDY is set to 1 by enabling the SW RESET bit (SCICTL1.5) or by a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = SCITXBUF is full</p> <p>1h (R/W) = SCITXBUF is ready to receive the next character</p>
6	TXEMPTY	R	1h	<p>Transmitter empty flag. This flag's value indicates the contents of the transmitter's buffer register (SCITXBUF) and shift register (TXSHF). An active SW RESET (SCICTL1.5), or a system reset, sets this bit. This bit does not cause an interrupt request.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Transmitter buffer or shift register or both are loaded with data</p> <p>1h (R/W) = Transmitter buffer and shift registers are both empty</p>
5-2	RESERVED	R	0h	Reserved
1	RXBKINTENA	R/W	0h	<p>Receiver-buffer/break interrupt enable. This bit controls the interrupt request caused by either the RXRDY flag or the BRKDT flag (bits SCIRXST.6 and .5) being set. However, RX/BK INT ENA does not prevent the setting of these flags.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Disable RXRDY/BRKDT interrupt</p> <p>1h (R/W) = Enable RXRDY/BRKDT interrupt</p>
0	TXINTENA	R/W	0h	<p>SCITXBUF-register interrupt enable. This bit controls the interrupt request caused by the setting of TXRDY flag bit (SCICTL2.7). However, it does not prevent the TXRDY flag from being set (which indicates SCITXBUF is ready to receive another character).</p> <p>0 Disable TXRDY interrupt</p> <p>1 Enable TXRDY interrupt.</p> <p>In non-FIFO mode, a dummy (or a valid) data has to be written to SCITXBUF for the first transmit interrupt to occur. This is the case when you enable the transmit interrupt for the first time and also when you re-enable (disable and then enable) the transmit interrupt. If TXINTENA is enabled after writing the data to SCITXBUF, it will not generate an interrupt.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Disable TXRDY interrupt</p> <p>1h (R/W) = Enable TXRDY interrupt</p>



### 9.14.2.6 SCI Receive Status (SCIRXST) Register (Offset = 5h) [reset = 0h]

SCIRXST contains seven bits that are receiver status flags (two of which can generate interrupt requests). Each time a complete character is transferred to the receiver buffers (SCIRXEMU and SCIRXBUF), the status flags are updated.

**Figure 9-16. SCI Receive Status (SCIRXST) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RXERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	RESERVED
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

**Table 9-13. SCI Receive Status (SCIRXST) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	RXERROR	R	0h	<p>SCI receiver error flag.</p> <p>The RX ERROR flag indicates that one of the error flags in the receiver status register is set. RX ERROR is a logical OR of the break detect, framing error, overrun, and parity error enable flags (bits 5-2: BRKDT, FE, OE, and PE).</p> <p>A 1 on this bit will cause an interrupt if the RX ERR INT ENA bit (SCICTL1.6) is set. This bit can be used for fast error-condition checking during the interrupt service routine. This error flag cannot be cleared directly</p> <p>it is cleared by an active SW RESET or by a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No error flags set</p> <p>1h (R/W) = Error flag(s) set</p>
6	RXRDY	R	0h	<p>SCI receiver-ready flag.</p> <p>When a new character is ready to be read from the SCIRXBUF register, the receiver sets this bit, and a receiver interrupt is generated if the RX/BK INT ENA bit (SCICTL2.1) is a 1. RXRDY is cleared by a reading of the SCIRXBUF register, by an active SW RESET, or by a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No new character in SCIRXBUF</p> <p>1h (R/W) = Character ready to be read from SCIRXBUF</p>
5	BRKDT	R	0h	<p>SCI break-detect flag.</p> <p>The SCI sets this bit when a break condition occurs. A break condition occurs when the SCI receiver data line (SCIRXD) remains continuously low for at least ten bits, beginning after a missing first stop bit. The occurrence of a break causes a receiver interrupt to be generated if the RX/BK INT ENA bit is a 1, but it does not cause the receiver buffer to be loaded. A BRKDT interrupt can occur even if the receiver SLEEP bit is set to 1. BRKDT is cleared by an active SW RESET or by a system reset. It is not cleared by receipt of a character after the break is detected. In order to receive more characters, the SCI must be reset by toggling the SW RESET bit or by a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No break condition</p> <p>1h (R/W) = Break condition occurred</p>

**Table 9-13. SCI Receive Status (SCIRXST) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
4	FE	R	0h	<p>SCI framing-error flag.</p> <p>The SCI sets this bit when an expected stop bit is not found. Only the first stop bit is checked. The missing stop bit indicates that synchronization with the start bit has been lost and that the character is incorrectly framed. The FE bit is reset by a clearing of the SW RESET bit or by a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No framing error detected 1h (R/W) = Framing error detected</p>
3	OE	R	0h	<p>SCI overrun-error flag.</p> <p>The SCI sets this bit when a character is transferred into registers SCIRXEMU and SCIRXBUF before the previous character is fully read by the CPU or DMAC. The previous character is overwritten and lost. The OE flag bit is reset by an active SW RESET or by a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No overrun error detected 1h (R/W) = Overrun error detected</p>
2	PE	R	0h	<p>SCI parity-error flag.</p> <p>This flag bit is set when a character is received with a mismatch between the number of 1s and its parity bit. The address bit is included in the calculation. If parity generation and detection is not enabled, the PE flag is disabled and read as 0. The PE bit is reset by an active SW RESET or a system reset.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No parity error or parity is disabled 1h (R/W) = Parity error is detected</p>
1	RXWAKE	R	0h	<p>Receiver wake-up-detect flag</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No detection of a receiver wake-up condition 1h (R/W) = A value of 1 in this bit indicates detection of a receiver wake-up condition. In the address-bit multiprocessor mode (SCICCR.3 = 1), RXWAKE reflects the value of the address bit for the character contained in SCIRXBUF. In the idle-line multiprocessor mode, RXWAKE is set if the SCIRXD data line is detected as idle.</p> <p>RXWAKE is a read-only flag, cleared by one of the following:</p> <ul style="list-style-type: none"> <li>- The transfer of the first byte after the address byte to SCIRXBUF (only in non-FIFO mode)</li> <li>- The reading of SCIRXBUF</li> <li>- An active SW RESET</li> <li>- A system reset</li> </ul>
0	RESERVED	R	0h	Reserved

### 9.14.2.7 SCI Receive Emulation Buffer (SCIRXEMU) Register (Offset = 6h) [reset = 0h]

Normal SCI data-receive operations read the data received from the SCIRXBUF register. The SCIRXEMU register is used principally during a debug connection because it can continuously read the data received for screen updates without clearing the RXRDY flag. SCIRXEMU is cleared by a system reset. This is the register that should be used in the CCS watch window to view the contents of the SCIRXBUF register. SCIRXEMU is not physically implemented

it is just a different address location to access the SCIRXBUF register without clearing the RXRDY flag.

**Figure 9-17. SCI Receive Emulation Buffer (SCIRXEMU) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
ERXDT							
R-0h							

**Table 9-14. SCI Receive Emulation Buffer (SCIRXEMU) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	ERXDT	R	0h	Receive emulation buffer data Reset type: SYSRSn

### 9.14.2.8 SCI Receive Data Buffer (SCIRXBUF) Register (Offset = 7h) [reset = 0h]

When the current data received is shifted from RXSHF to the receiver buffer, flag bit RXRDY is set and the data is ready to be read. If the RXBKINTENA bit (SCICTL2.1) is set, this shift also causes an interrupt. When SCIRXBUF is read, the RXRDY flag is reset. SCIRXBUF is cleared by a system reset.

**Figure 9-18. SCI Receive Data Buffer (SCIRXBUF) Register**

15	14	13	12	11	10	9	8
SCIFFFE	SCIFFPE	RESERVED					
R-0h	R-0h	R-0h					
7	6	5	4	3	2	1	0
SAR							
R-0h							

**Table 9-15. SCI Receive Data Buffer (SCIRXBUF) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15	SCIFFFE	R	0h	SCIFFFE. SCI FIFO Framing error flag bit (applicable only if the FIFO is enabled) Reset type: SYSRSn 0h (R/W) = No frame error occurred while receiving the character, in bits 7-0. This bit is associated with the character on the top of the FIFO. 1h (R/W) = A frame error occurred while receiving the character in bits 7-0. This bit is associated with the character on the top of the FIFO.
14	SCIFFPE	R	0h	SCIFFPE. SCI FIFO parity error flag bit (applicable only if the FIFO is enabled) Reset type: SYSRSn 0h (R/W) = No parity error occurred while receiving the character, in bits 7-0. This bit is associated with the character on the top of the FIFO. 1h (R/W) = A parity error occurred while receiving the character in bits 7-0. This bit is associated with the character on the top of the FIFO.
13-8	RESERVED	R	0h	Reserved
7-0	SAR	R	0h	Receive Character bits Reset type: SYSRSn

#### 9.14.2.9 SCI Transmit Data Buffer (SCITXBUF) Register (Offset = 9h) [reset = 0h]

Data bits to be transmitted are written to SCITXBUF. These bits must be right-justified because the leftmost bits are ignored for characters less than eight bits long. The transfer of data from this register to the TXSHF transmitter shift register sets the TXRDY flag (SCICTL2.7), indicating that SCITXBUF is ready to receive another set of data. If bit TXINTENA (SCICTL2.0) is set, this data transfer also causes an interrupt.

**Figure 9-19. SCI Transmit Data Buffer (SCITXBUF) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXDT							
R/W-0h							

**Table 9-16. SCI Transmit Data Buffer (SCITXBUF) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	TXDT	R/W	0h	Transmit data buffer Reset type: SYSRSn

#### 9.14.2.10 SCI FIFO Transmit (SCIFFTX) Register (Offset = Ah) [reset = A000h]

SCIFFTX controls the transmit FIFO interrupt, FIFO enhancements, and reset for the SCI transmit and receive channels.

**Figure 9-20. SCI FIFO Transmit (SCIFFTX) Register**

15	14	13	12	11	10	9	8
SCIRST	SCIFFENA	TXFIFORESET					TXFFST
R/W-1h	R/W-0h	R/W-1h					R-0h
7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA				TXFFIL	
R-0h	R-0/W1S-0h	R/W-0h				R/W-0h	

**Table 9-17. SCI FIFO Transmit (SCIFFTX) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15	SCIRST	R/W	1h	SCI Reset 0 Write 0 to reset the SCI transmit and receive channels. SCI FIFO register configuration bits will be left as is. 1 SCI FIFO can resume transmit or receive. SCIRST should be 1 even for Autobaud logic to work. Reset type: SYSRSn
14	SCIFFENA	R/W	0h	SCI FIFO enable Reset type: SYSRSn 0h (R/W) = SCI FIFO enhancements are disabled 1h (R/W) = SCI FIFO enhancements are enabled
13	TXFIFORESET	R/W	1h	Transmit FIFO reset Reset type: SYSRSn 0h (R/W) = Reset the FIFO pointer to zero and hold in reset 1h (R/W) = Re-enable transmit FIFO operation
12-8	TXFFST	R	0h	FIFO status Reset type: SYSRSn 0h (R/W) = Transmit FIFO is empty 1h (R/W) = Transmit FIFO has 1 words 2h (R/W) = Transmit FIFO has 2 words 3h (R/W) = Transmit FIFO has 3 words 4h (R/W) = Transmit FIFO has 4 words
7	TXFFINT	R	0h	Transmit FIFO interrupt Reset type: SYSRSn 0h (R/W) = TXFIFO interrupt has not occurred, read-only bit 1h (R/W) = TXFIFO interrupt has occurred, read-only bit
6	TXFFINTCLR	R-0/W1S	0h	Transmit FIFO clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero 1h (R/W) = Write 1 to clear TXFFINT flag in bit 7
5	TXFFIENA	R/W	0h	Transmit FIFO interrupt enable Reset type: SYSRSn 0h (R/W) = TX FIFO interrupt is disabled 1h (R/W) = TX FIFO interrupt is enabled. This interrupt is triggered whenever the transmit FIFO status (TXFFST) bits match (equal to or less than) the interrupt trigger level bits TXFFIL (bits 4-0).
4-0	TXFFIL	R/W	0h	TXFFIL4-0 Transmit FIFO interrupt level bits. The transmit FIFO generates an interrupt whenever the FIFO status bits (TXFFST4-0) are less than or equal to the FIFO level bits (TXFFIL4-0). The maximum value that can be assigned to these bits to generate an interrupt cannot be more than the depth of the TX FIFO. The default value of these bits after reset is 00000b. Users should set TXFFIL to best fit their application needs by weighing between the CPU overhead to service the ISR and the best possible usage of SCI bus bandwidth. Reset type: SYSRSn

### 9.14.2.11 SCI FIFO Receive (SCIFFRX) Register (Offset = Bh) [reset = 201Fh]

SCIFFTX controls the receive FIFO interrupt, receive FIFO reset, and status of the receive FIFO overflow.

**Figure 9-21. SCI FIFO Receive (SCIFFRX) Register**

15		14		13		12		11		10		9		8	
RXFFOVF		RXFFOVRCLR		RXFIFORESET						RXFFST					
R-0h		R-0/W1S-0h		R/W-1h						R-0h					
7		6		5		4		3		2		1		0	
RXFFINT		RXFFINTCLR		RXFFIENA						RXFFIL					
R-0h		W-0h		R/W-0h						R/W-1Fh					

**Table 9-18. SCI FIFO Receive (SCIFFRX) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15	RXFFOVF	R	0h	Receive FIFO overflow. This will function as flag, but cannot generate interrupt by itself. This condition will occur while receive interrupt is active. Receive interrupts should service this flag condition. Reset type: SYSRSn 0h (R/W) = Receive FIFO has not overflowed, read-only bit 1h (R/W) = Receive FIFO has overflowed, read-only bit. More than 16 words have been received in to the FIFO, and the first received word is lost
14	RXFFOVRCLR	R-0/W1S	0h	RXFFOVF clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFFOVF flag bit, Bit reads back a zero 1h (R/W) = Write 1 to clear RXFFOVF flag in bit 15
13	RXFIFORESET	R/W	1h	Receive FIFO reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Re-enable receive FIFO operation
12-8	RXFFST	R	0h	FIFO status Reset type: SYSRSn 0h (R/W) = Receive FIFO is empty 1h (R/W) = Receive FIFO has 1 words 2h (R/W) = Receive FIFO has 2 words 3h (R/W) = Receive FIFO has 3 words 4h (R/W) = Receive FIFO has 4 words
7	RXFFINT	R	0h	Receive FIFO interrupt Reset type: SYSRSn 0h (R/W) = RXFIFO interrupt has not occurred, read-only bit 1h (R/W) = RXFIFO interrupt has occurred, read-only bit
6	RXFFINTCLR	W	0h	Receive FIFO interrupt clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFIFINT flag bit. Bit reads back a zero. 1h (R/W) = Write 1 to clear RXFFINT flag in bit 7

**Table 9-18. SCI FIFO Receive (SCIFFRX) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
5	RXFFIENA	R/W	0h	Receive FIFO interrupt enable Reset type: SYSRSn 0h (R/W) = RX FIFO interrupt is disabled 1h (R/W) = RX FIFO interrupt is enabled. This interrupt is triggered whenever the receive FIFO status (RXFFST) bits match (equal to or greater than) the interrupt trigger level bits RXFFIL (bits 4-0).
4-0	RXFFIL	R/W	1Fh	Receive FIFO interrupt level bits The receive FIFO generates an interrupt whenever the FIFO status bits (RXFFST4-0) are greater than or equal to the FIFO level bits (RXFFIL4-0). The maximum value that can be assigned to these bits to generate an interrupt cannot be more than the depth of the RX FIFO. The default value of these bits after reset is 11111b. Users should set RXFFIL to best fit their application needs by weighing between the CPU overhead to service the ISR and the best possible usage of received SCI data. Reset type: SYSRSn



### 9.14.2.12 SCI FIFO Control (SCIFFCT) Register (Offset = Ch) [reset = 0h]

SCIFFCT contains the status of auto-baud detect, clears the auto-baud flag, and calibrate for A-detect bit.

**Figure 9-22. SCI FIFO Control (SCIFFCT) Register**

15	14	13	12	11	10	9	8
ABD	ABDCLR	CDC	RESERVED				
R-0h	W-0h	R/W-0h	R-0h				
7	6	5	4	3	2	1	0
FFTXDLY							
R/W-0h							

**Table 9-19. SCI FIFO Control (SCIFFCT) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15	ABD	R	0h	Auto-baud detect (ABD) bit Reset type: SYSRSn 0h (R/W) = Auto-baud detection is not complete. "A", "a" character has not been received successfully. 1h (R/W) = Auto-baud hardware has detected "A" or "a" character on the SCI receive register. Auto-detect is complete.
14	ABDCLR	W	0h	ABD-clear bit Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on ABD flag bit. Bit reads back a zero. 1h (R/W) = Write 1 to clear ABD flag in bit 15.
13	CDC	R/W	0h	CDC calibrate A-detect bit Reset type: SYSRSn 0h (R/W) = Disables auto-baud alignment 1h (R/W) = Enables auto-baud alignment
12-8	RESERVED	R	0h	Reserved
7-0	FFTXDLY	R/W	0h	FIFO transfer delay. These bits define the delay between every transfer from FIFO transmit buffer to transmit shift register. The delay is defined in the number of SCI serial baud clock cycles. The 8 bit register could define a minimum delay of 0 baud clock cycles and a maximum of 256 baud clock cycles. In FIFO mode, the buffer (TXBUF) between the shift register and the FIFO should be filled only after the shift register has completed shifting of the last bit. This is required to pass on the delay between transfers to the data stream. In FIFO mode, TXBUF should not be treated as one additional level of buffer. The delayed transmit feature will help to create an auto-flow scheme without RTS/CTS controls as in standard UARTS. When SCI is configured for one stop-bit, delay introduced by FFTXDLY between one frame and the next frame is equal to number of baud clock cycles that FFTXDLY is set to. When SCI is configured for two stop-bits, delay introduced by FFTXDLY between one frame and the next frame is equal to number of baud clock cycles that FFTXDLY is set to minus 1. Reset type: SYSRSn

### 9.14.2.13 SCI Priority Control (SCIPRI) Register (Offset = Fh) [reset = 0h]

SCIPRI determines what happens when an emulation suspend event occurs.

**Figure 9-23. SCI Priority Control (SCIPRI) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			FREESOFT			RESERVED	
R-0h			R/W-0h			R-0h	

**Table 9-20. SCI Priority Control (SCIPRI) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-5	RESERVED	R	0h	Reserved
4-3	FREESOFT	R/W	0h	These bits determine what occurs when an emulation suspend event occurs (for example, when the debugger hits a break point). The peripheral can continue whatever it is doing (free-run mode), or if in stop mode, it can either stop immediately or stop when the current operation (the current receive/transmit sequence) is complete. Reset type: SYSRSn 0h (R/W) = Immediate stop on suspend 1h (R/W) = Complete current receive/transmit sequence before stopping 2h (R/W) = Free run 3h (R/W) = Free run
2-0	RESERVED	R	0h	Reserved

This chapter describes the features and operation of the inter-integrated circuit (I2C) module. The I2C module provides an interface between one of these devices and devices compliant with the NXP Semiconductors Inter-IC bus (I2C bus) specification version 2.1, and connected by way of an I2C bus. External components attached to this 2-wire serial bus can transmit/receive 1- to 8-bit data to/from the device through the I2C module. This chapter assumes the reader is familiar with the I2C bus specification.

---

### Note

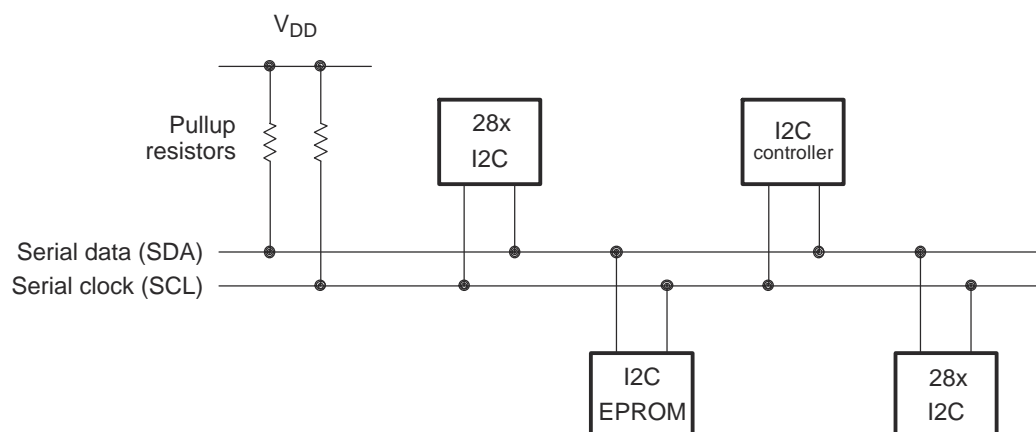
A unit of data transmitted or received by the I2C module can have fewer than 8 bits; however, for convenience, a unit of data is called a data byte throughout this document. The number of bits in a data byte is selectable by way of the BC bits of the mode register, I2CMDR.

---

<b>10.1 Introduction</b> .....	<b>532</b>
<b>10.2 Configuring Device Pins</b> .....	<b>537</b>
<b>10.3 I2C Module Operational Details</b> .....	<b>537</b>
<b>10.4 Interrupt Requests Generated by the I2C Module</b> .....	<b>549</b>
<b>10.5 Resetting or Disabling the I2C Module</b> .....	<b>551</b>
<b>10.6 I2C Registers</b> .....	<b>552</b>

## 10.1 Introduction

The I2C module supports any slave or master I2C-compatible device. [Figure 10-1](#) shows an example of multiple I2C modules connected for a two-way transfer from one device to other devices.



**Figure 10-1. Multiple I2C Modules Connected**

### 10.1.1 I2C Related Collateral

#### Foundational Materials

- [I2C Hardware Overview](#) (Video)
- [I2C Protocol Overview](#) (Video)
- [Understanding the I2C Bus Application Report](#)

#### Getting Started Materials

- [Configuring the TMS320F280x DSP as an I2C Processor Application Report](#)
- [I2C Buffers Overview](#) (Video)
- [I2C Dynamic Addressing Application Report](#)
- [I2C translators overview](#) (Video)
- [Why, When, and How to use I2C Buffers Application Report](#)

#### Expert Materials

- [I2C Bus Pull-Up Resistor Calculation Application Report](#)
- [Maximum Clock Frequency of I2C Bus Using Repeaters Application Report](#)

### 10.1.2 Features

The I2C module has the following features:

- Compliance with the NXP Semiconductors I2C bus specification (version 2.1):
  - Support for 8-bit format transfers
  - 7-bit and 10-bit addressing modes
  - General call
  - START byte mode
  - Support for multiple master-transmitters and slave-receivers
  - Support for multiple slave-transmitters and master-receivers
  - Combined master transmit/receive and receive/transmit mode
  - Data transfer rate from 10 kbps up to 400 kbps (Fast-mode)
- Receive FIFO and Transmitter FIFO (4-deep x 8-bit FIFO)
- Supports two ePIE interrupts:
  - I2Cx Interrupt – Any of the following events can be configured to generate an I2Cx interrupt:
    - Transmit-data ready
    - Receive-data ready
    - Register-access ready
    - No-acknowledgment received
    - Arbitration lost
    - Stop condition detected
    - Addressed as slave
  - I2Cx\_FIFO interrupts:
    - Transmit FIFO interrupt
    - Receive FIFO interrupt
- Module enable and disable capability
- Free data format mode

### 10.1.3 Features Not Supported

The I2C module does not support:

- High-speed mode (Hs-mode)
- CBUS-compatibility mode

### 10.1.4 Functional Overview

Each device connected to an I2C bus is recognized by a unique address. Each device can operate as either a transmitter or a receiver, depending on the function of the device. A device connected to the I2C bus can also be considered as the master or the slave when performing data transfers. A master device is the device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. During this transfer, any device addressed by this master is considered a slave. The I2C module supports the multi-master mode, in which one or more devices capable of controlling an I2C bus can be connected to the same I2C bus.

For data communication, the I2C module has a serial data pin (SDA) and a serial clock pin (SCL), as shown in [Figure 10-2](#). These two pins carry information between the C28x device and other devices connected to the I2C bus. The SDA and SCL pins both are bidirectional. They each must be connected to a positive supply voltage using a pull-up resistor. When the bus is free, both pins are high. The driver of these two pins has an open-drain configuration to perform the required wired-AND function.

There are two major transfer techniques:

- Standard Mode: Send exactly n data values, where n is a value you program in an I2C module register. See the I2CCNT register in [Section 10.6](#) for more information.
- Repeat Mode: Keep sending data values until you use software to initiate a STOP condition or a new START condition. See the I2CMDR register in [Section 10.6](#) for RM bit information.

The I2C module consists of the following primary blocks:

- A serial interface: one data pin (SDA) and one clock pin (SCL)
- Data registers and FIFOs to temporarily hold receive data and transmit data traveling between the SDA pin and the CPU
- Control and status registers
- A peripheral bus interface to enable the CPU to access the I2C module registers and FIFOs.
- A clock synchronizer to synchronize the I2C input clock (from the device clock generator) and the clock on the SCL pin, and to synchronize data transfers with masters of different clock speeds
- A prescaler to divide down the input clock that is driven to the I2C module
- A noise filter on each of the two pins, SDA and SCL
- An arbitrator to handle arbitration between the I2C module (when it is a master) and another master
- Interrupt generation logic, so that an interrupt can be sent to the CPU
- FIFO interrupt generation logic, so that FIFO access can be synchronized to data reception and data transmission in the I2C module

[Figure 10-2](#) shows the four registers used for transmission and reception in non-FIFO mode. The CPU writes data for transmission to I2CDXR and reads received data from I2CDRR. When the I2C module is configured as a transmitter, data written to I2CDXR is copied to I2CXSR and shifted out on the SDA pin one bit at a time. When the I2C module is configured as a receiver, received data is shifted into I2CRSR and then copied to I2CDRR.

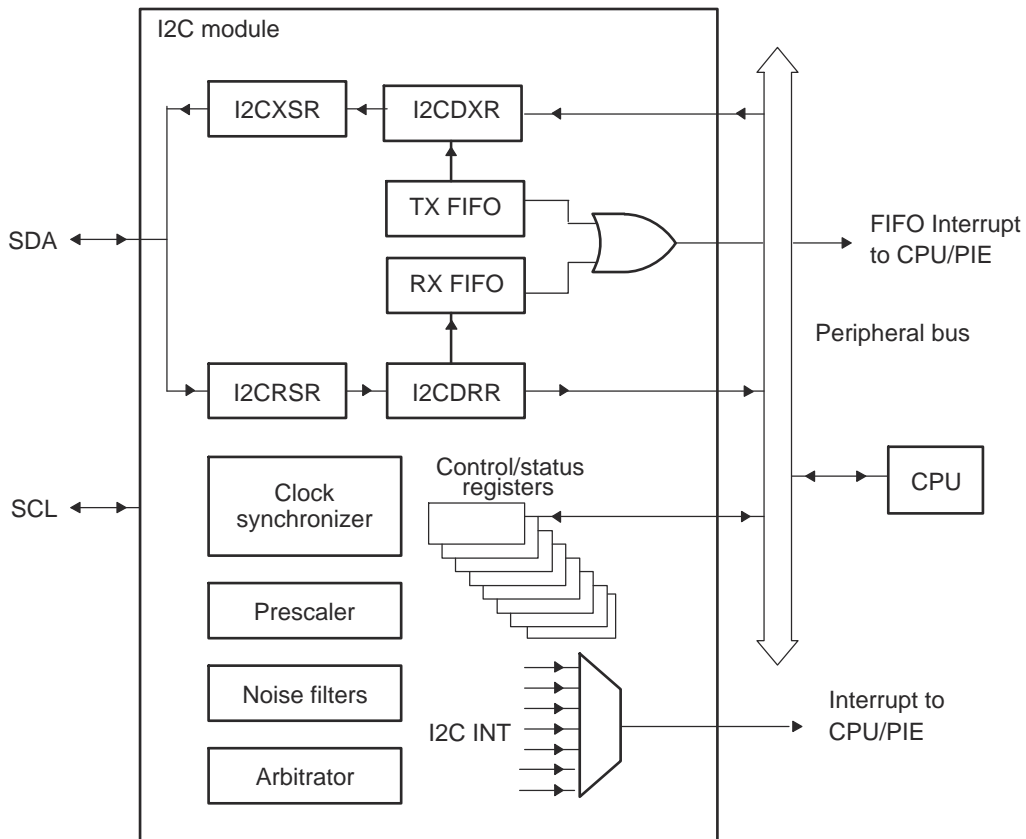


Figure 10-2. I2C Module Conceptual Block Diagram

### 10.1.5 Clock Generation

The I2C module clock determines the frequency at which the I2C module operates. A programmable prescaler in the I2C module divides down the SYSCLK to produce the I2C module clock and this I2C module clock is divided further to produce the I2C master clock on the SCL pin. Figure 10-3 shows the clock generation diagram for I2C module.

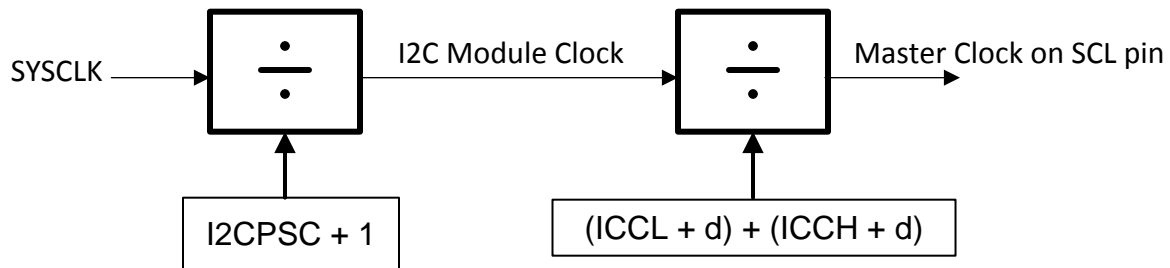


Figure 10-3. Clocking Diagram for the I2C Module

#### Note

To meet all of the I2C protocol timing specifications, the I2C module clock must be between 7-12 MHz.

To specify the divide-down value, initialize the IPSC field of the prescaler register, I2CPSC. The resulting frequency is:

$$\text{I2C Module Clock (Fmod)} = \frac{\text{SYSCLK}}{(\text{I2CPSC} + 1)} \quad (4)$$

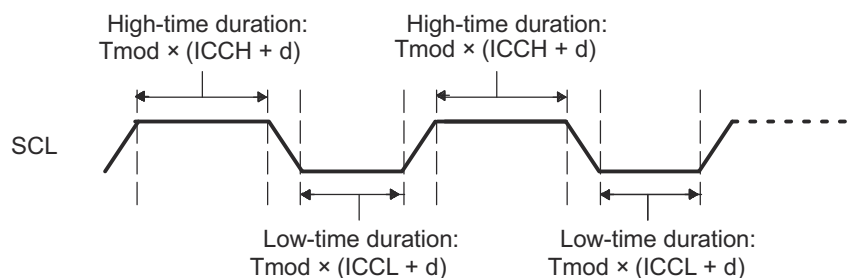
The prescaler must be initialized only while the I2C module is in the reset state (IRS = 0 in I2CMDR). The prescaled frequency takes effect only when IRS is changed to 1. Changing the IPSC value while IRS = 1 has no effect.

The master clock appears on the SCL pin when the I2C module is configured to be a master on the I2C bus. This clock controls the timing of communication between the I2C module and a slave. As shown in Figure 10-3, a second clock divider in the I2C module divides down the module clock to produce the master clock. The clock divider uses the ICCL value of I2CCLKL to divide down the low portion of the module clock signal and uses the ICCH value of I2CCLKH to divide down the high portion of the module clock signal. See Section 10.1.6 for the master clock frequency equation.

### 10.1.6 I2C Clock Divider Registers (I2CCLKL and I2CCLKH)

As explained in Section 10.1.5, when the I2C module is a master, the I2C module clock is divided down further to use as the master clock on the SCL pin. As shown in Figure 10-4, the shape of the master clock depends on two divide-down values:

- ICCL in I2CCLKL. For each master clock cycle, ICCL determines the amount of time the signal is low.
- ICCH in I2CCLKH. For each master clock cycle, ICCH determines the amount of time the signal is high.



**Figure 10-4. Roles of the Clock Divide-Down Values (ICCL and ICCH)**

#### 10.1.6.1 Formula for the Master Clock Period

The master clock period ( $T_{\text{mst}}$ ) is a multiple of the period of the I2C Module Clock ( $T_{\text{mod}}$ ):

$$\text{Master Clock period (Tmst)} = \frac{[(\text{ICCH} + d) + (\text{ICCL} + d)]}{\text{I2C Module Clock (Fmod)}} \quad (5)$$

where  $d$  depends on the divide-down value IPSC, as shown in Table 10-1. IPSC is described in the I2CPSC register.

**Table 10-1. Dependency of Delay  $d$  on the Divide-Down Value IPSC**

IPSC	$d$
0	7
1	6
Greater than 1	5



## 10.2 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPyPUD register.

See the *GPIO* chapter for more details on GPIO mux and settings.

## 10.3 I2C Module Operational Details

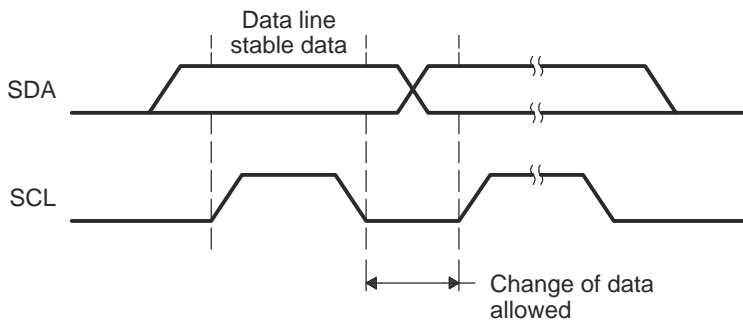
This section provides an overview of the I2C bus protocol and how it is implemented.

### 10.3.1 Input and Output Voltage Levels

One clock pulse is generated by the master device for each data bit transferred. Due to a variety of different technology devices that can be connected to the I2C bus, the levels of logic 0 (low) and logic 1 (high) are not fixed and depend on the associated level of  $V_{DD}$ . For details, see your device-specific data sheet.

### 10.3.2 Data Validity

The data on SDA must be stable during the high period of the clock (see [Figure 10-5](#)). The high or low state of the data line, SDA, should change only when the clock signal on SCL is low.



**Figure 10-5. Bit Transfer on the I2C bus**

### 10.3.3 Operating Modes

The I2C module has four basic operating modes to support data transfers as a master and as a slave. See [Table 10-2](#) for the names and descriptions of the modes.

If the I2C module is a master, it begins as a master-transmitter and typically transmits an address for a particular slave. When giving data to the slave, the I2C module must remain a master-transmitter. To receive data from a slave, the I2C module must be changed to the master-receiver mode.

If the I2C module is a slave, it begins as a slave-receiver and typically sends acknowledgment when it recognizes its slave address from a master. If the master will be sending data to the I2C module, the module must remain a slave-receiver. If the master has requested data from the I2C module, the module must be changed to the slave-transmitter mode.

**Table 10-2. Operating Modes of the I2C Module**

Operating Mode	Description
Slave-receiver mode	The I2C module is a slave and receives data from a master.  All slaves begin in this mode. In this mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master. As a slave, the I2C module does not generate the clock signal, but it can hold SCL low while the intervention of the device is required (RSFULL = 1 in I2CSTR) after a byte has been received. See <a href="#">Section 10.3.7</a> for more details.
Slave-transmitter mode	The I2C module is a slave and transmits data to a master.  This mode can be entered only from the slave-receiver mode; the I2C module must first receive a command from the master. When you are using any of the 7-bit/10-bit addressing formats, the I2C module enters its slave-transmitter mode if the slave address byte is the same as its own address (in I2COAR) and the master has transmitted R/ $\bar{W}$ = 1. As a slave-transmitter, the I2C module then shifts the serial data out on SDA with the clock pulses that are generated by the master. While a slave, the I2C module does not generate the clock signal, but it can hold SCL low while the intervention of the device is required (XSMT = 0 in I2CSTR) after a byte has been transmitted. See <a href="#">Section 10.3.7</a> for more details.
Master-receiver mode	The I2C module is a master and receives data from a slave.  This mode can be entered only from the master-transmitter mode; the I2C module must first transmit a command to the slave. When you are using any of the 7-bit/10-bit addressing formats, the I2C module enters its master-receiver mode after transmitting the slave address byte and R/ $\bar{W}$ = 1. Serial data bits on SDA are shifted into the I2C module with the clock pulses generated by the I2C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the device is required (RSFULL = 1 in I2CSTR) after a byte has been received.
Master-transmitter mode	The I2C module is a master and transmits control information and data to a slave.  All masters begin in this mode. In this mode, data assembled in any of the 7-bit/10-bit addressing formats is shifted out on SDA. The bit shifting is synchronized with the clock pulses generated by the I2C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the device is required (XSMT = 0 in I2CSTR) after a byte has been transmitted.

To summarize, SCL will be held low in the following conditions:

- When an overrun condition is detected (RSFULL = 1), in Slave-receiver mode.
- When an underflow condition is detected (XSMT = 0), in Slave-transmitter mode.

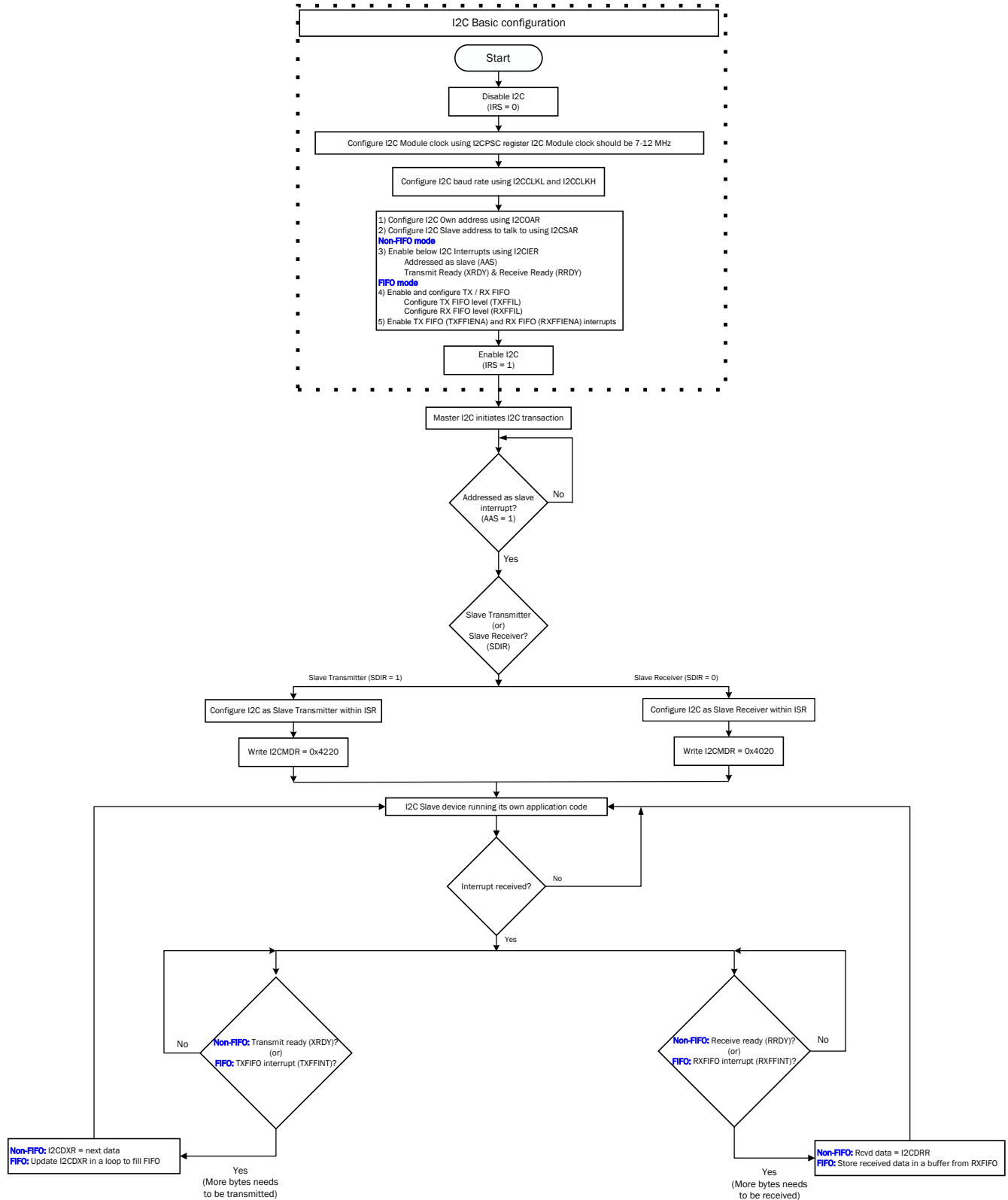
I2C slave nodes have to accept and provide data when the I2C master node requests it.

- To release SCL in slave-receiver mode, read data from I2CDRR.
- To release SCL in slave-transmitter mode, write data to I2CDXR.
- To force a release without handling the data, reset the module using the I2CMDR.IRS bit.

**Table 10-3. Master-Transmitter/Receiver Bus Activity Defined by the RM, STT, and STP Bits of I2CMDR**

RM	STT	STP	Bus Activity <sup>(1)</sup>	Description
0	0	0	None	No activity
0	0	1	P	STOP condition
0	1	0	S-A-D..(n)..D.	START condition, slave address, n data bytes (n = value in I2CCNT)
0	1	1	S-A-D..(n)..D-P	START condition, slave address, n data bytes, STOP condition (n = value in I2CCNT)
1	0	0	None	No activity
1	0	1	P	STOP condition
1	1	0	S-A-D-D-D.	Repeat mode transfer: START condition, slave address, continuous data transfers until STOP condition or next START condition
1	1	1	None	Reserved bit combination (No activity)

(1) S = START condition; A = Address; D = Data byte; P = STOP condition;



**Figure 10-6. I2C Slave TX / RX Flowchart**

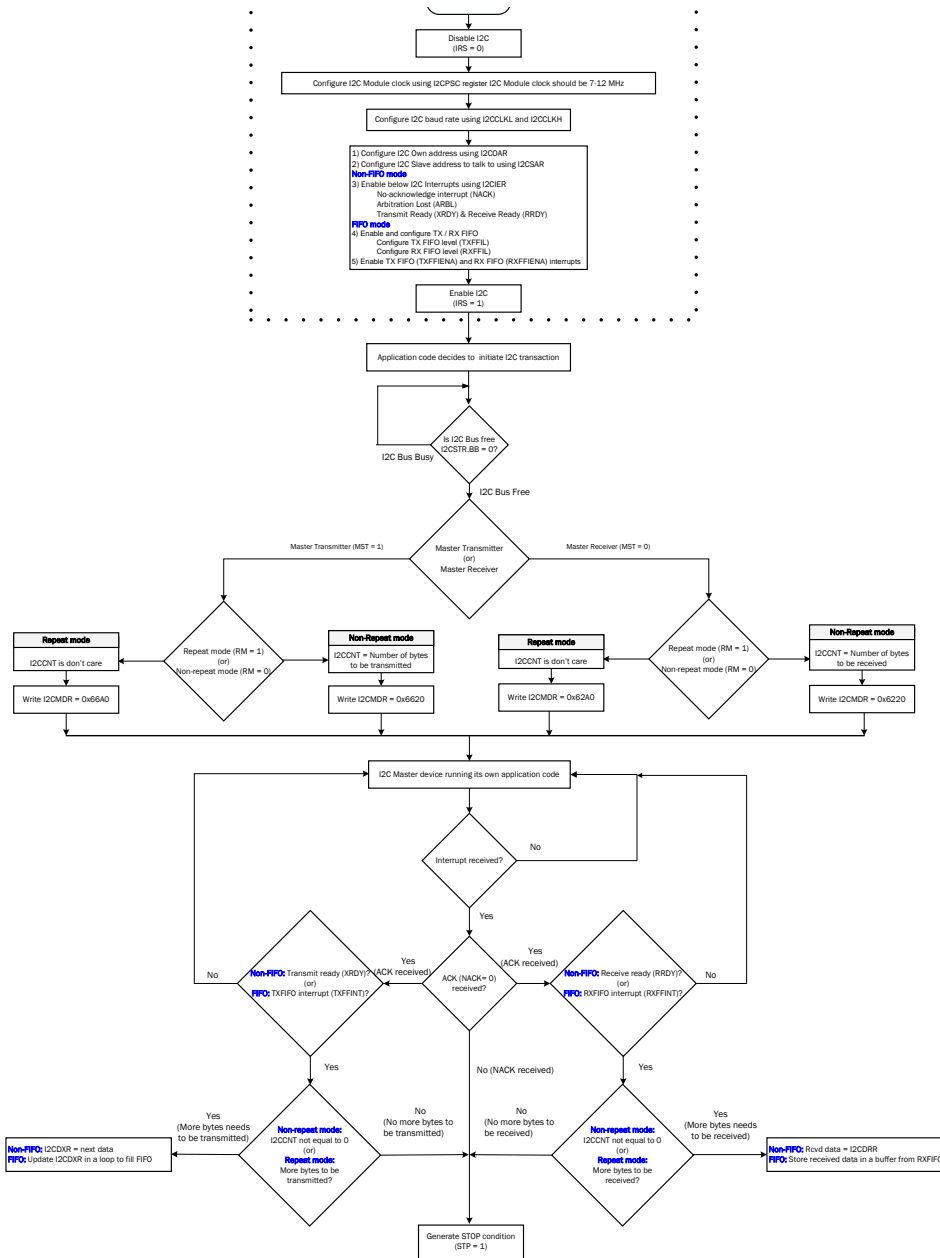
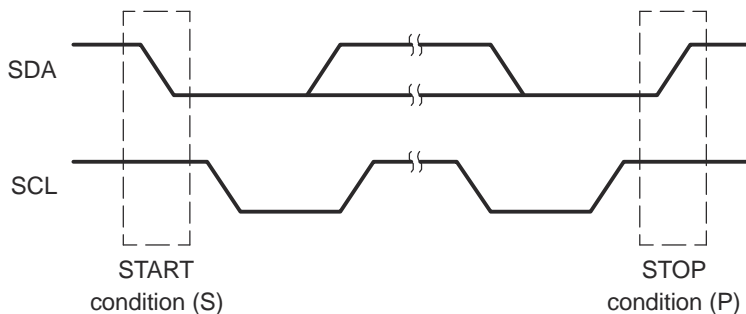


Figure 10-7. I2C Master TX / RX Flowchart

### 10.3.4 I2C Module START and STOP Conditions

START and STOP conditions can be generated by the I2C module when the module is configured to be a master on the I2C bus. As shown in [Figure 10-8](#):

- The START condition is defined as a high-to-low transition on the SDA line while SCL is high. A master drives this condition to indicate the start of a data transfer.
- The STOP condition is defined as a low-to-high transition on the SDA line while SCL is high. A master drives this condition to indicate the end of a data transfer.



**Figure 10-8. I2C Module START and STOP Conditions**

After a START condition and before a subsequent STOP condition, the I2C bus is considered busy, and the bus busy (BB) bit of I2CSTR is 1. Between a STOP condition and the next START condition, the bus is considered free, and BB is 0.

For the I2C module to start a data transfer with a START condition, the master mode bit (MST) and the START condition bit (STT) in I2CMDR must both be 1. For the I2C module to end a data transfer with a STOP condition, the STOP condition bit (STP) must be set to 1. When the BB bit is set to 1 and the STT bit is set to 1, a repeated START condition is generated. For a description of I2CMDR and its bits (including MST, STT, and STP), see [Section 10.6](#).

The I2C peripheral cannot detect a START or STOP condition while it is in reset (IRS = 0). The BB bit will remain in the cleared state (BB = 0) while the I2C peripheral is in reset (IRS = 0). When the I2C peripheral is taken out of reset (IRS set to 1) the BB bit will not correctly reflect the I2C bus status until a START or STOP condition is detected.

Follow these steps before initiating the first data transfer with I2C:

1. After taking the I2C peripheral out of reset by setting the IRS bit to 1, wait a period larger than the total time taken for the longest data transfer in the application. By waiting for a period of time after I2C comes out of reset, users can ensure that at least one START or STOP condition will have occurred on the I2C bus and been captured by the BB bit. After this period, the BB bit will correctly reflect the state of the I2C bus.
2. Check the BB bit and verify that BB = 0 (bus not busy) before proceeding.
3. Begin data transfers.

Not resetting the I2C peripheral in between transfers ensures that the BB bit reflects the actual bus status. If users must reset the I2C peripheral in between transfers, repeat steps 1 through 3 every time the I2C peripheral is taken out of reset.

### 10.3.5 Non-repeat Mode versus Repeat Mode

#### Non-repeat mode:

- When I2CMDR.RM = 0, I2C module is configured in non-repeat mode.
- I2CCNT register determines the number of bytes to be transmitted (or) received.
- If STP = 0 in I2CMDR, the ARDY bit is set when the internal data counter counts down to 0.
- If STP = 1, ARDY bit doesn't get set and I2C module generates a STOP condition when the internal data counter counts down to 0.

---

#### Note

In non-repeat mode (RM = 0), if I2CCNT is set to 0, I2C state machine expects to transmit (or) receive 65536 bytes and not 0 bytes.

---

#### Repeat mode:

- When I2CMDR.RM = 1, I2C module is configured in repeat mode.
- I2CCNT register contents don't determine the number of bytes to be transmitted (or) received.
- Number of bytes to be transmitted (or) received can be controlled by software.
- ARDY bit gets set at end of transmission and reception of each byte.

---

#### Note

Once you start I2C transaction in non-repeat mode (or) repeat mode, you cannot switch into another mode until the I2C transaction is completed with a STOP condition.

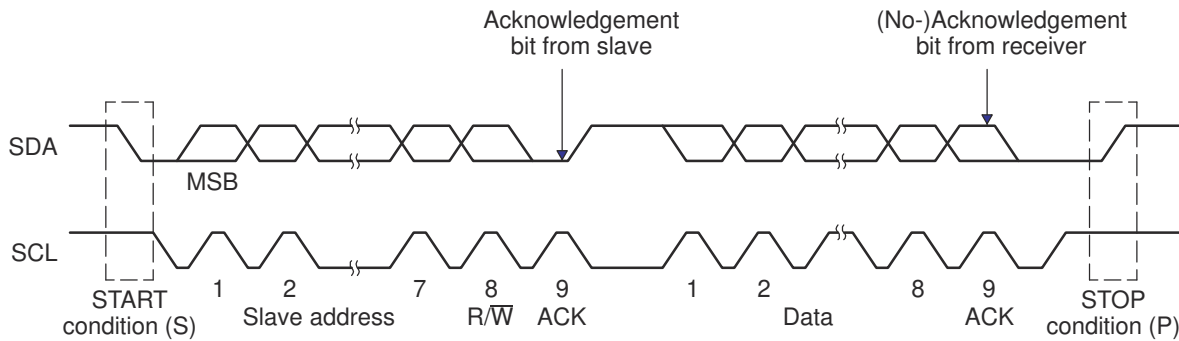
---

### 10.3.6 Serial Data Formats

Figure 10-9 shows an example of a data transfer on the I2C bus. The I2C module supports 1 to 8-bit data values. In Figure 10-9, 8-bit data is transferred. Each bit put on the SDA line equates to 1 pulse on the SCL line, and the values are always transferred with the most significant bit (MSB) first. The number of data values that can be transmitted or received is unrestricted. The serial data format used in Figure 10-9 is the 7-bit addressing format. The I2C module supports the formats shown in Figure 10-10 through Figure 10-12 and described in the paragraphs that follow the figures.

**Note**

In Figure 10-9 through Figure 10-12, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.



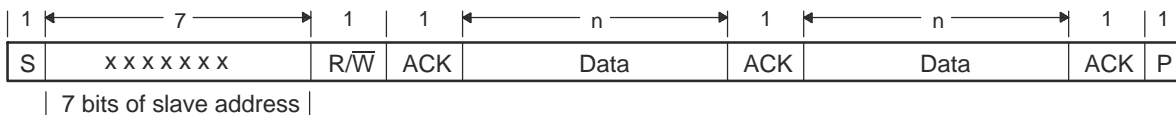
**Figure 10-9. I2C Module Data Transfer (7-Bit Addressing with 8-bit Data Configuration Shown)**

#### 10.3.6.1 7-Bit Addressing Format

The 7-bit addressing format is the default format after reset. Disabling expanded address (I2CMDR.XA = 0) and free data format (I2CMDR.FDF = 0) enables 7-bit addressing format.

In this format (see Figure 10-10), the first byte after a START condition (S) consists of a 7-bit slave address followed by a R/  $\bar{W}$  bit. R/  $\bar{W}$  determines the direction of the data:

- R/  $\bar{W}$  = 0: The I2C master writes (transmits) data to the addressed slave. This can be achieved by setting I2CMDR.TRX = 1 (Transmitter mode)
- R/  $\bar{W}$  = 1: The I2C master reads (receives) data from the slave. This can be achieved by setting I2CMDR.TRX = 0 (Receiver mode)



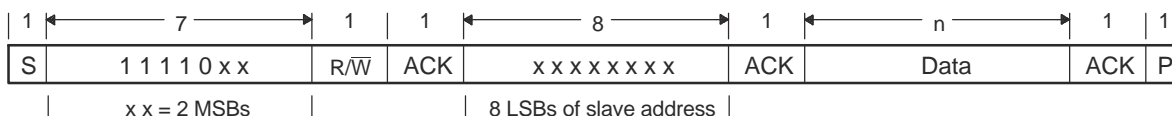
**Figure 10-10. I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMDR)**

An extra clock cycle dedicated for acknowledgment (ACK) is inserted after each byte. If the ACK bit is inserted by the slave after the first byte from the master, it is followed by n bits of data from the transmitter (master or slave, depending on the R/  $\bar{W}$  bit). n is a number from 1 to 8 determined by the bit count (BC) field of I2CMDR. After the data bits have been transferred, the receiver inserts an ACK bit.

### 10.3.6.2 10-Bit Addressing Format

The 10-bit addressing format can be enabled by setting expanded address (I2CMDR.XA = 1) and disabling free data format (I2CMDR.FDF = 0).

The 10-bit addressing format (see Figure 10-11) is similar to the 7-bit addressing format, but the master sends the slave address in two separate byte transfers. The first byte consists of 11110b, the two MSBs of the 10-bit slave address, and R/W. The second byte is the remaining 8 bits of the 10-bit slave address. The slave must send acknowledgment after each of the two byte transfers. Once the master has written the second byte to the slave, the master can either write data or use a repeated START condition to change the data direction. For more details about using 10-bit addressing, see the NXP Semiconductors I2C bus specification.

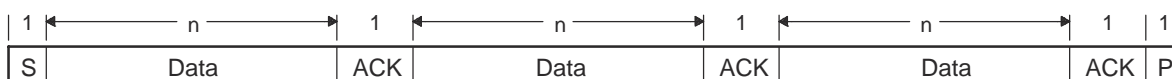


**Figure 10-11. I2C Module 10-Bit Addressing Format (FDF = 0, XA = 1 in I2CMDR)**

### 10.3.6.3 Free Data Format

The free data format can be enabled by setting I2CMDR.FDF = 1.

In this format (see Figure 10-12), the first byte after a START condition (S) is a data byte. An ACK bit is inserted after each data byte, which can be from 1 to 8 bits, depending on the BC field of I2CMDR. No address or data-direction bit is sent. Therefore, the transmitter and the receiver must both support the free data format, and the direction of the data must be constant throughout the transfer.



**Figure 10-12. I2C Module Free Data Format (FDF = 1 in I2CMDR)**

#### Note

The free data format is not supported in the digital loopback mode (I2CMDR.DLB = 1).

**Table 10-4. How the MST and FDF Bits of I2CMDR Affect the Role of the TRX Bit of I2CMDR**

MST	FDF	I2C Module State	Function of TRX
0	0	In slave mode but not free data format mode	TRX is a don't care. Depending on the command from the master, the I2C module responds as a receiver or a transmitter.
0	1	In slave mode and free data format mode	The free data format mode requires that the I2C module remains the transmitter or the receiver throughout the transfer. TRX identifies the role of the I2C module: TRX = 1: The I2C module is a transmitter. TRX = 0: The I2C module is a receiver.
1	0	In master mode but not free data format mode	TRX = 1: The I2C module is a transmitter. TRX = 0: The I2C module is a receiver.
1	1	In master mode and free data format mode	TRX = 0: The I2C module is a receiver. TRX = 1: The I2C module is a transmitter.



### 10.3.6.4 Using a Repeated START Condition

I2C master can communicate with multiple slave addresses without having to give up control of the I2C bus by driving a STOP condition. This can be achieved by driving another START condition at the end of each data type. The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. Figure 10-13 shows a repeated START condition in the 7-bit addressing format.

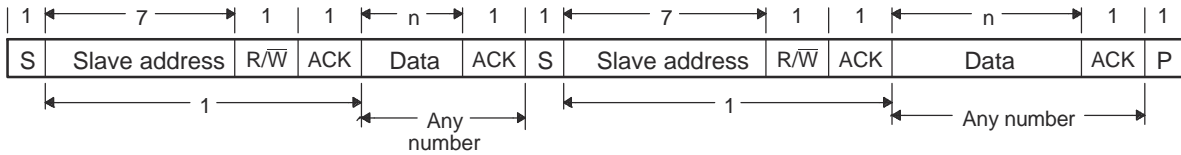


Figure 10-13. Repeated START Condition (in This Case, 7-Bit Addressing Format)

#### Note

In Figure 10-13, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

### 10.3.7 Clock Synchronization

Under normal conditions, only one master device generates the clock signal, SCL. During the arbitration procedure, however, there are two or more masters and the clock must be synchronized so that the data output can be compared. Figure 10-14 illustrates the clock synchronization. The wired-AND property of SCL means that a device that first generates a low period on SCL overrules the other devices. At this high-to-low transition, the clock generators of the other devices are forced to start their own low period. The SCL is held low by the device with the longest low period. The other devices that finish their low periods must wait for SCL to be released, before starting their high periods. A synchronized signal on SCL is obtained, where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

If a device pulls down the clock line for a longer time, the result is that all clock generators must enter the wait state. In this way, a slave slows down a fast master and the slow device creates enough time to store a received byte or to prepare a byte to be transmitted.

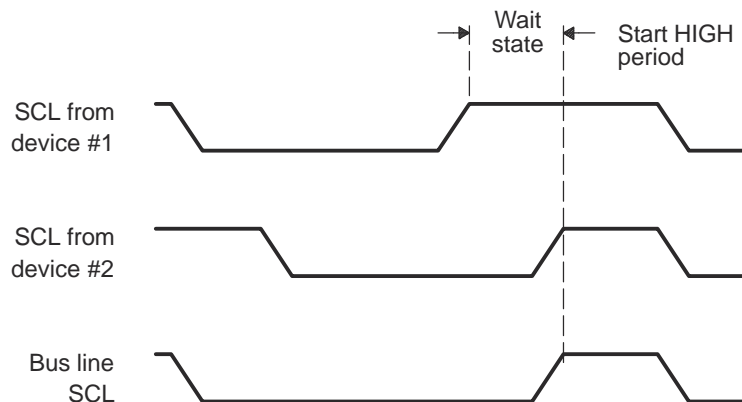


Figure 10-14. Synchronization of Two I2C Clock Generators During Arbitration

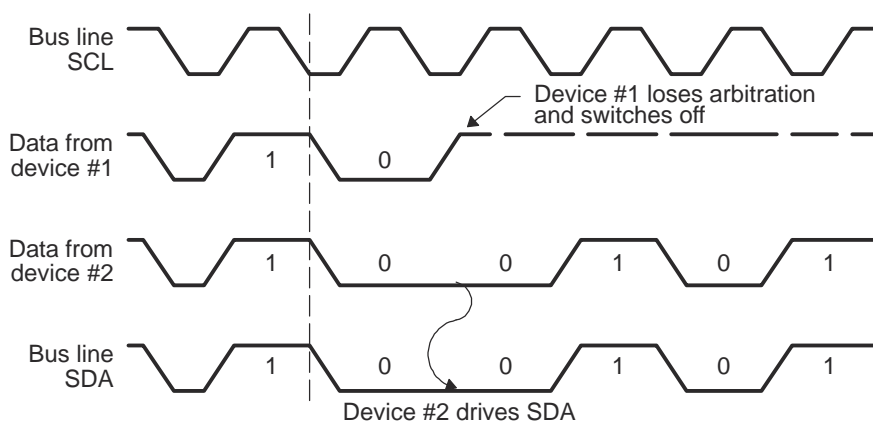
### 10.3.8 Arbitration

If two or more master-transmitters attempt to start a transmission on the same bus at approximately the same time, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. [Figure 10-15](#) illustrates the arbitration procedure between two devices. The first master-transmitter that releases the SDA line high is overruled by another master-transmitter that drives the SDA low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If the I2C module is the losing master, it switches to the slave-receiver mode, sets the arbitration lost (ARBL) flag, and generates the arbitration-lost interrupt request.

If during a serial transfer the arbitration procedure is still in progress when a repeated START condition or a STOP condition is transmitted to SDA, the master-transmitters involved must send the repeated START condition or the 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



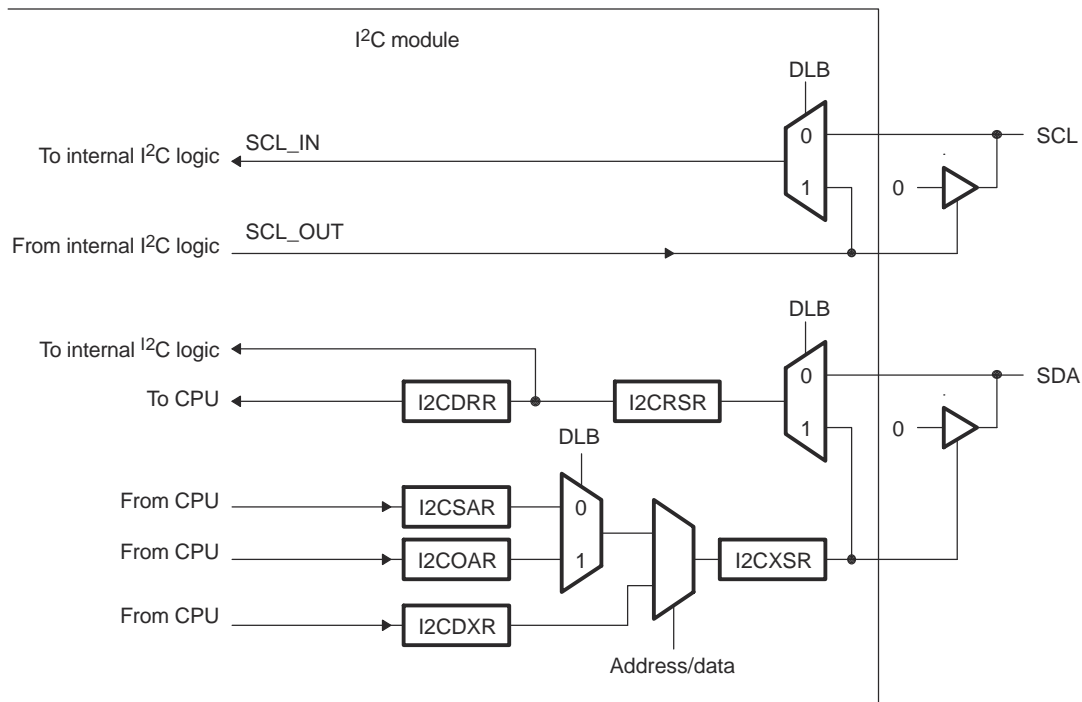
**Figure 10-15. Arbitration Procedure Between Two Master-Transmitters**

### 10.3.9 Digital Loopback Mode

The I2C module support a self-test mode called digital loopback, which is enabled by setting the DLB bit in the I2CMDR register. In this mode, data transmitted out of the I2CDXR register is received in the I2CDRR register. The data follows an internal path, and takes n cycles to reach I2CDRR, where:

$$n = 8 * (\text{SYSCLK}) / (\text{I2C module clock (Fmod)})$$

The transmit clock and the receive clock are the same. The address seen on the external SDA pin is the address in the I2COAR register. Figure 10-16 shows the signal routing in digital loopback mode.



**Figure 10-16. Pin Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit**

#### Note

The free data format (I2CMDR.FDF = 1) is not supported in digital loopback mode.

### 10.3.10 NACK Bit Generation

When the I2C module is a receiver (master or slave), it can acknowledge or ignore bits sent by the transmitter. To ignore any new bits, the I2C module must send a no-acknowledge (NACK) bit during the acknowledge cycle on the bus. [Table 10-5](#) summarizes the various ways you can allow the I2C module to send a NACK bit.

**Table 10-5. Ways to Generate a NACK Bit**

I2C Module Condition	NACK Bit Generation Options
Slave-receiver modes	Allow an overrun condition (RSFULL = 1 in I2CSTR) Reset the module (IRS = 0 in I2CMDR) Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive
Master-receiver mode AND Repeat mode (RM = 1 in I2CMDR)	Generate a STOP condition (STP = 1 in I2CMDR) Reset the module (IRS = 0 in I2CMDR) Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive
Master-receiver mode AND Nonrepeat mode (RM = 0 in I2CMDR)	If STP = 1 in I2CMDR, allow the internal data counter to count down to 0 and thus force a STOP condition If STP = 0, make STP = 1 to generate a STOP condition Reset the module (IRS = 0 in I2CMDR). = 1 to generate a STOP condition Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive

## 10.4 Interrupt Requests Generated by the I2C Module

Each I2C module can generate two CPU interrupts.

1. Basic I2C interrupt: Possible basic I2C interrupt sources that can trigger this interrupt are described in [Section 10.4.1](#).
2. I2C FIFO interrupt: Possible I2C FIFO interrupt sources that can trigger this interrupt are described in [Section 10.4.2](#)

### 10.4.1 Basic I2C Interrupt Requests

The I2C module generates the interrupt requests described in [Table 10-6](#). As shown in [Figure 10-17](#), all requests are multiplexed through an arbiter to a single I2C interrupt request to the CPU. Each interrupt request has a flag bit in the status register (I2CSTR) and an enable bit in the interrupt enable register (I2CIER). When one of the specified events occurs, its flag bit is set. If the corresponding enable bit is 0, the interrupt request is blocked. If the enable bit is 1, the request is forwarded to the CPU as an I2C interrupt.

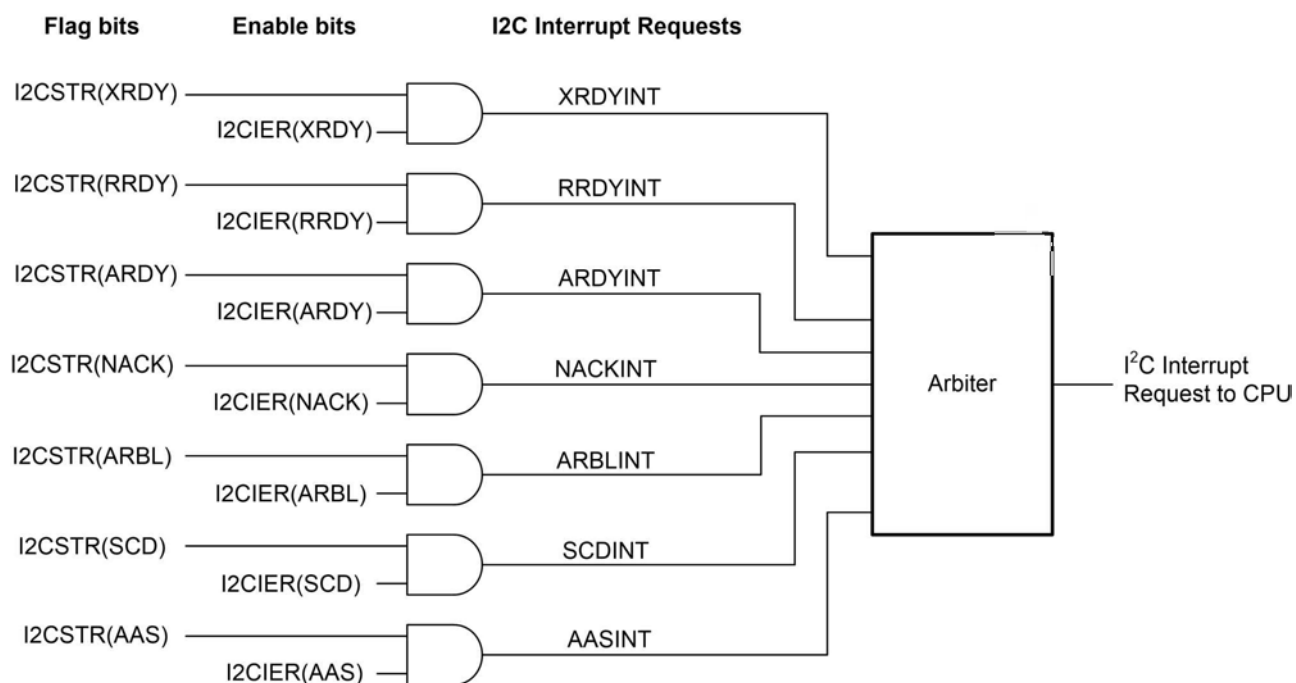
The I2C interrupt is one of the maskable interrupts of the CPU. As with any maskable interrupt request, if it is properly enabled in the CPU, the CPU executes the corresponding interrupt service routine (I2CINT1A\_ISR). The I2CINT1A\_ISR for the I2C interrupt can determine the interrupt source by reading the interrupt source register, I2CISRC. Then the I2CINT1A\_ISR can branch to the appropriate subroutine.

After the CPU reads I2CISRC, the following events occur:

1. The flag for the source interrupt is cleared in I2CSTR. Exception: The ARDY, RRDY, and XRDY bits in I2CSTR are not cleared when I2CISRC is read. To clear one of these bits, write a 1 to it.
2. The arbiter determines which of the remaining interrupt requests has the highest priority, writes the code for that interrupt to I2CISRC, and forwards the interrupt request to the CPU.

**Table 10-6. Descriptions of the Basic I2C Interrupt Requests**

I2C Interrupt Request	Interrupt Source
XRDYINT	<p>Transmit ready condition: The data transmit register (I2CDXR) is ready to accept new data because the previous data has been copied from I2CDXR to the transmit shift register (I2CXSR).</p> <p>As an alternative to using XRDYINT, the CPU can poll the XRDY bit of the status register, I2CSTR. XRDYINT should not be used when in FIFO mode. Use the FIFO interrupts instead.</p>
RRDYINT	<p>Receive ready condition: The data receive register (I2CDRR) is ready to be read because data has been copied from the receive shift register (I2CRSR) to I2CDRR.</p> <p>As an alternative to using RRDYINT, the CPU can poll the RRDY bit of I2CSTR. RRDYINT should not be used when in FIFO mode. Use the FIFO interrupts instead.</p>
ARDYINT	<p>Register-access ready condition: The I2C module registers are ready to be accessed because the previously programmed address, data, and command values have been used.</p> <p>The specific events that generate ARDYINT are the same events that set the ARDY bit of I2CSTR.</p> <p>As an alternative to using ARDYINT, the CPU can poll the ARDY bit.</p>
NACKINT	<p>No-acknowledgment condition: The I2C module is configured as a master-transmitter and did not received acknowledgment from the slave-receiver.</p> <p>As an alternative to using NACKINT, the CPU can poll the NACK bit of I2CSTR.</p>
ARBLINT	<p>Arbitration-lost condition: The I2C module has lost an arbitration contest with another master-transmitter.</p> <p>As an alternative to using ARBLINT, the CPU can poll the ARBL bit of I2CSTR.</p>
SCDINT	<p>Stop condition detected: A STOP condition was detected on the I2C bus.</p> <p>As an alternative to using SCDINT, the CPU can poll the SCD bit of the status register, I2CSTR.</p>
AASINT	<p>Addressed as slave condition: The I2C has been addressed as a slave device by another master on the I2C bus.</p> <p>As an alternative to using AASINT, the CPU can poll the AAS bit of the status register, I2CSTR.</p>



**Figure 10-17. Enable Paths of the I2C Interrupt Requests**

The priorities of the basic I2C interrupt requests are listed in order of highest priority to lowest priority:

1. ARBLINT
2. NACKINT
3. ARDYINT
4. RRDYINT
5. XRDYINT
6. SCDINT
7. AASINT

The I2C module has a backwards compatibility bit (BC) in the I2CEMDR register. The timing diagram in demonstrates the effect the backwards compatibility bit has on I2C module registers and interrupts when configured as a slave-transmitter.

### 10.4.2 I2C FIFO Interrupts

In addition to the seven basic I2C interrupts, the transmit and receive FIFOs each contain the ability to generate an interrupt (I2CINT2A). The transmit FIFO can be configured to generate an interrupt after transmitting a defined number of bytes, up to 4. The receive FIFO can be configured to generate an interrupt after receiving a defined number of bytes, up to 4. These two interrupt sources are ORed together into a single maskable CPU interrupt. Figure 10-18 shows the structure of I2C FIFO interrupt. The interrupt service routine can then read the FIFO interrupt status flags to determine from which source the interrupt came. See the I2C transmit FIFO register (I2CFFTX) and the I2C receive FIFO register (I2CFFRX) descriptions.

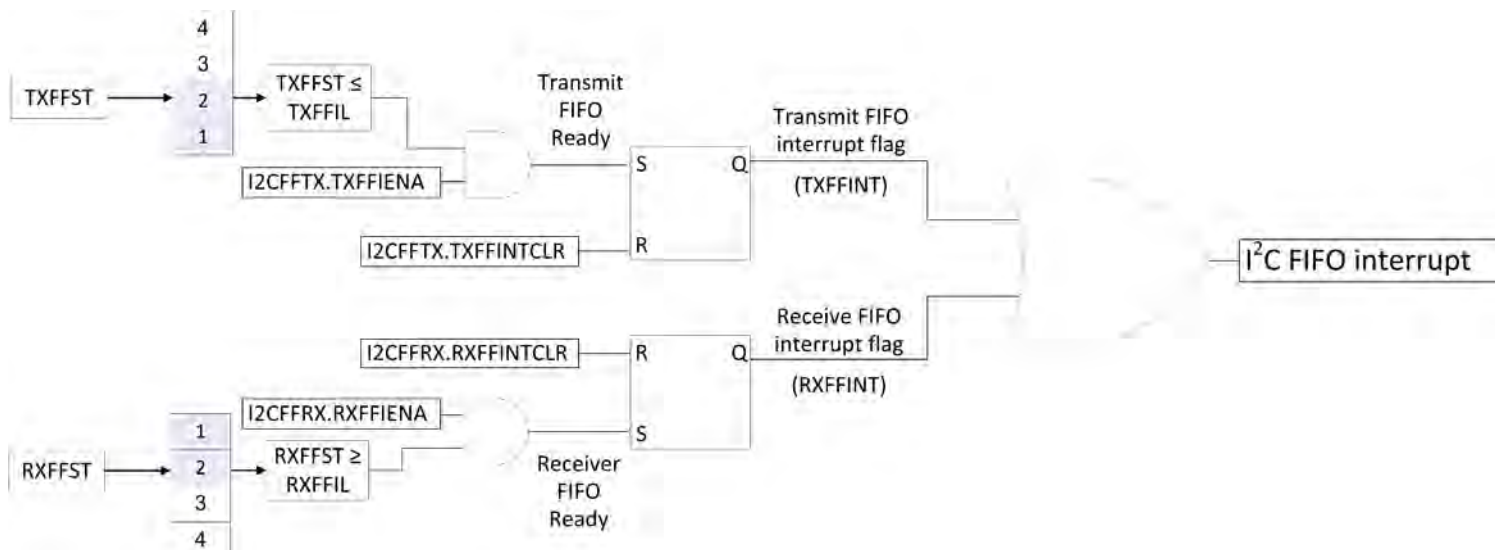


Figure 10-18. I2C FIFO Interrupt

### 10.5 Resetting or Disabling the I2C Module

You can reset or disable the I2C module in two ways:

- Write 0 to the I2C reset bit (IRS) in the I2C mode register (I2CMDR). All status bits (in I2CSTR) are forced to their default values, and the I2C module remains disabled until IRS is changed to 1. The SDA and SCL pins are in the high-impedance state.
- Initiate a device reset by driving the  $\overline{\text{XRS}}$  pin low. The entire device is reset and is held in the reset state until you drive the pin high. When the  $\overline{\text{XRS}}$  pin is released, all I2C module registers are reset to their default values. The IRS bit is forced to 0, which resets the I2C module. The I2C module stays in the reset state until you write 1 to IRS.

The IRS must be 0 while you configure or reconfigure the I2C module. Forcing IRS to 0 can be used to save power and to clear error conditions.

## 10.6 I2C Registers

This section describes the C28x I2C Module Registers.

### 10.6.1 I2C Base Address Table (C28)

**Table 10-7. I2C Base Address Table (C28)**

Bit Field Name		Base Address
Instance	Structure	
I2caRegs	I2C_REGS	0x0000_7900

### 10.6.2 I2C\_REGS Registers

[Table 10-8](#) lists the I2C\_REGS registers. All register offset addresses not listed in [Table 10-8](#) should be considered as reserved locations and the register contents should not be modified.

**Table 10-8. I2C\_REGS Registers**

Offset	Acronym	Register Name	Write Protection	Section
0h	I2COAR	I2C Own Address		<a href="#">Section 10.6.2.1</a>
1h	I2CIER	I2C Interrupt Enable		<a href="#">Section 10.6.2.2</a>
2h	I2CSTR	I2C Status		<a href="#">Section 10.6.2.3</a>
3h	I2CCLKL	I2C Clock Low-time Divider		<a href="#">Section 10.6.2.4</a>
4h	I2CCLKH	I2C Clock High-time Divider		<a href="#">Section 10.6.2.5</a>
5h	I2CCNT	I2C Data Count		<a href="#">Section 10.6.2.6</a>
6h	I2CDRR	I2C Data Receive		<a href="#">Section 10.6.2.7</a>
7h	I2CSAR	I2C Slave Address		<a href="#">Section 10.6.2.8</a>
8h	I2CDXR	I2C Data Transmit		<a href="#">Section 10.6.2.9</a>
9h	I2CMDR	I2C Mode		<a href="#">Section 10.6.2.10</a>
Ah	I2CISRC	I2C Interrupt Source		<a href="#">Section 10.6.2.11</a>
Bh	I2CEMDR	I2C Extended Mode		<a href="#">Section 10.6.2.12</a>
Ch	I2CPSC	I2C Prescaler		<a href="#">Section 10.6.2.13</a>
20h	I2CFFTX	I2C FIFO Transmit		<a href="#">Section 10.6.2.14</a>
21h	I2CFFRX	I2C FIFO Receive		<a href="#">Section 10.6.2.15</a>



Complex bit access types are encoded to fit into small table cells. [Table 10-9](#) shows the codes that are used for access types in this section.

**Table 10-9. I2C\_REGS Access Type Codes**

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R -0	Read Returns 0s
Write Type		
W	W	Write
W1C	W 1C	Write 1 to clear
W1S	W 1S	Write 1 to set
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

### 10.6.2.1 I2C Own Address Register (I2COAR) (Offset = 0h) [reset = 0h]

The I2C own address register (I2COAR) is a 16-bit register. The I2C module uses this register to specify its own slave address, which distinguishes it from other slaves connected to the I2C-bus. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6-0 are used write 0s to bits 9-7.

**Figure 10-19. I2C Own Address Register (I2COAR)**

15	14	13	12	11	10	9	8
RESERVED							OAR
R-0h							R/W-0h
7	6	5	4	3	2	1	0
OAR							
R/W-0h							

**Table 10-10. I2C Own Address Register (I2COAR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-10	RESERVED	R	0h	Reserved
9-0	OAR	R/W	0h	In 7-bit addressing mode (XA = 0 in I2CMDR): 00h-7Fh Bits 6-0 provide the 7-bit slave address of the I2C module. Write 0s to bits 9-7. In 10-bit addressing mode (XA = 1 in I2CMDR): 000h-3FFh Bits 9-0 provide the 10-bit slave address of the I2C module. Reset type: SYSRSn

### 10.6.2.2 I2C Interrupt Enable Register (I2CIER) (Offset = 1h) [reset = 0h]

I2CIER is used by the CPU to individually enable or disable I2C interrupt requests.

**Figure 10-20. I2C Interrupt Enable Register (I2CIER)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	AAS	SCD	XRDY	RRDY	ARDY	NACK	ARBL
R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

**Table 10-11. I2C Interrupt Enable Register (I2CIER) Field Descriptions**

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	AAS	R/W	0h	Addressed as slave interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
5	SCD	R/W	0h	Stop condition detected interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
4	XRDY	R/W	0h	Transmit-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
3	RRDY	R/W	0h	Receive-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
2	ARDY	R/W	0h	Register-access-ready interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
1	NACK	R/W	0h	No-acknowledgment interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
0	ARBL	R/W	0h	Arbitration-lost interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled

### 10.6.2.3 I2C Status Register (I2CSTR) (Offset = 2h) [reset = 410h]

The I2C status register (I2CSTR) is a 16-bit register used to determine which interrupt has occurred and to read status information.

**Figure 10-21. I2C Status Register (I2CSTR)**

15	14	13	12	11	10	9	8
RESERVED	SDIR	NACKSNT	BB	RSFULL	XSMT	AAS	AD0
R-0h	R/W1C-0h	R/W1C-0h	R-0h	R-0h	R-1h	R-0h	R-0h
7	6	5	4	3	2	1	0
RESERVED	SCD	XRDY	RRDY	ARDY	NACK	ARBL	
R/W-0h	R/W1C-0h	R-1h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h

**Table 10-12. I2C Status Register (I2CSTR) Field Descriptions**

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reserved
14	SDIR	R/W1C	0h	Slave direction bit Reset type: SYSRSn 0h (R/W) = I2C is not addressed as a slave transmitter. SDIR is cleared by one of the following events: - It is manually cleared. To clear this bit, write a 1 to it. - Digital loopback mode is enabled. - A START or STOP condition occurs on the I2C bus. 1h (R/W) = I2C is addressed as a slave transmitter.
13	NACKSNT	R/W1C	0h	NACK sent bit. This bit is used when the I2C module is in the receiver mode. One instance in which NACKSNT is affected is when the NACK mode is used (see the description for NACKMOD in Reset type: SYSRSn 0h (R/W) = NACK not sent. NACKSNT bit is cleared by any one of the following events: - It is manually cleared. To clear this bit, write a 1 to it. - The I2C module is reset (either when 0 is written to the IRS bit of I2CMDR or when the whole device is reset). 1h (R/W) = NACK sent: A no-acknowledge bit was sent during the acknowledge cycle on the I2C-bus.
12	BB	R	0h	Bus busy bit. BB indicates whether the I2C-bus is busy or is free for another data transfer. See the paragraph following the table for more information Reset type: SYSRSn 0h (R/W) = Bus free. BB is cleared by any one of the following events: - The I2C module receives or transmits a STOP bit (bus free). - The I2C module is reset. 1h (R/W) = Bus busy: The I2C module has received or transmitted a START bit on the bus.

**Table 10-12. I2C Status Register (I2CSTR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
11	RSFULL	R	0h	<p>Receive shift register full bit.</p> <p>RSFULL indicates an overrun condition during reception. Overrun occurs when new data is received into the shift register (I2CRSR) and the old data has not been read from the receive register (I2CDRR). As new bits arrive from the SDA pin, they overwrite the bits in I2CRSR. The new data will not be copied to ICDRR until the previous data is read.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = No overrun detected. RSFULL is cleared by any one of the following events:</p> <ul style="list-style-type: none"> <li>- I2CDRR is read by the CPU. Debug probe reads of the I2CDRR do not affect this bit.</li> <li>- The I2C module is reset.</li> </ul> <p>1h (R/W) = Overrun detected</p>
10	XSMT	R	1h	<p>Transmit shift register empty bit.</p> <p>XSMT = 0 indicates that the transmitter has experienced underflow. Underflow occurs when the transmit shift register (I2CXSR) is empty but the data transmit register (I2CDXR) has not been loaded since the last I2CDXR-to-I2CXSR transfer. The next I2CDXR-to-I2CXSR transfer will not occur until new data is in I2CDXR. If new data is not transferred in time, the previous data may be re-transmitted on the SDA pin.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Underflow detected (empty)</p> <p>1h (R/W) = No underflow detected (not empty). XSMT is set by one of the following events:</p> <ul style="list-style-type: none"> <li>- Data is written to I2CDXR.</li> <li>- The I2C module is reset</li> </ul>
9	AAS	R	0h	<p>Addressed-as-slave bit</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = In the 7-bit addressing mode, the AAS bit is cleared when receiving a NACK, a STOP condition, or a repeated START condition. In the 10-bit addressing mode, the AAS bit is cleared when receiving a NACK, a STOP condition, or by a slave address different from the I2C peripheral's own slave address.</p> <p>1h (R/W) = The I2C module has recognized its own slave address or an address of all zeros (general call).</p>
8	AD0	R	0h	<p>Address 0 bits</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = AD0 has been cleared by a START or STOP condition.</p> <p>1h (R/W) = An address of all zeros (general call) is detected.</p>
7-6	RESERVED	R/W	0h	Reserved
5	SCD	R/W1C	0h	<p>Stop condition detected bit.</p> <p>SCD is set when the I2C sends or receives a STOP condition. The I2C module delays clearing of the I2CMDR[STP] bit until the SCD bit is set.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = STOP condition not detected since SCD was last cleared. SCD is cleared by any one of the following events:</p> <ul style="list-style-type: none"> <li>- I2CISRC is read by the CPU when it contains the value 110b (stop condition detected). Debug probe reads of the I2CISRC do not affect this bit.</li> <li>- SCD is manually cleared. To clear this bit, write a 1 to it.</li> <li>- The I2C module is reset.</li> </ul> <p>1h (R/W) = A STOP condition has been detected on the I2C bus.</p>

**Table 10-12. I2C Status Register (I2CSTR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
4	XRDY	R	1h	<p>Transmit-data-ready interrupt flag bit.</p> <p>When not in FIFO mode, XRDY indicates that the data transmit register (I2CDXR) is ready to accept new data because the previous data has been copied from I2CDXR to the transmit shift register (I2CXSR). The CPU can poll XRDY or use the XRDY interrupt request. When in FIFO mode, use TXFFINT instead.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = I2CDXR not ready. XRDY is cleared when data is written to I2CDXR.</p> <p>1h (R/W) = I2CDXR ready: Data has been copied from I2CDXR to I2CXSR.</p> <p>XRDY is also forced to 1 when the I2C module is reset.</p>
3	RRDY	R/W1C	0h	<p>Receive-data-ready interrupt flag bit.</p> <p>When not in FIFO mode, RRDY indicates that the data receive register (I2CDRR) is ready to be read because data has been copied from the receive shift register (I2CRSR) to I2CDRR. The CPU can poll RRDY or use the RRDY interrupt request. When in FIFO mode, use RXFFINT instead.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = I2CDRR not ready. RRDY is cleared by any one of the following events:</p> <ul style="list-style-type: none"> <li>- I2CDRR is read by the CPU. Debug probe reads of the I2CDRR do not affect this bit.</li> <li>- RRDY is manually cleared. To clear this bit, write a 1 to it.</li> <li>- The I2C module is reset.</li> </ul> <p>1h (R/W) = I2CDRR ready: Data has been copied from I2CRSR to I2CDRR.</p>
2	ARDY	R/W1C	0h	<p>Register-access-ready interrupt flag bit (only applicable when the I2C module is in the master mode).</p> <p>ARDY indicates that the I2C module registers are ready to be accessed because the previously programmed address, data, and command values have been used. The CPU can poll ARDY or use the ARDY interrupt request.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = The registers are not ready to be accessed. ARDY is cleared by any one of the following events:</p> <ul style="list-style-type: none"> <li>- The I2C module starts using the current register contents.</li> <li>- ARDY is manually cleared. To clear this bit, write a 1 to it.</li> <li>- The I2C module is reset.</li> </ul> <p>1h (R/W) = The registers are ready to be accessed.</p> <p>In the nonrepeat mode (RM = 0 in I2CMDR): If STP = 0 in I2CMDR, the ARDY bit is set when the internal data counter counts down to 0. If STP = 1, ARDY is not affected (instead, the I2C module generates a STOP condition when the counter reaches 0).</p> <p>In the repeat mode (RM = 1): ARDY is set at the end of each byte transmitted from I2CDXR.</p>

**Table 10-12. I2C Status Register (I2CSTR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
1	NACK	R/W1C	0h	<p>No-acknowledgment interrupt flag bit. NACK applies when the I2C module is a master transmitter. NACK indicates whether the I2C module has detected an acknowledge bit (ACK) or a noacknowledge bit (NACK) from the slave receiver. The CPU can poll NACK or use the NACK interrupt request.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = ACK received/NACK not received. This bit is cleared by any one of the following events:</p> <ul style="list-style-type: none"> <li>- An acknowledge bit (ACK) has been sent by the slave receiver.</li> <li>- NACK is manually cleared. To clear this bit, write a 1 to it.</li> <li>- The CPU reads the interrupt source register (I2CISRC) and the register contains the code for a NACK interrupt. Debug probe reads of the I2CISRC do not affect this bit.</li> <li>- The I2C module is reset.</li> </ul> <p>1h (R/W) = NACK bit received. The hardware detects that a no-acknowledge (NACK) bit has been received.</p> <p>Note: While the I2C module performs a general call transfer, NACK is 1, even if one or more slaves send acknowledgment.</p>
0	ARBL	R/W1C	0h	<p>Arbitration-lost interrupt flag bit (only applicable when the I2C module is a master-transmitter). ARBL primarily indicates when the I2C module has lost an arbitration contest with another master-transmitter. The CPU can poll ARBL or use the ARBL interrupt request.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Arbitration not lost. AL is cleared by any one of the following events:</p> <ul style="list-style-type: none"> <li>- AL is manually cleared. To clear this bit, write a 1 to it.</li> <li>- The CPU reads the interrupt source register (I2CISRC) and the register contains the code for an</li> </ul> <p>AL interrupt. Debug probe reads of the I2CISRC do not affect this bit.</p> <ul style="list-style-type: none"> <li>- The I2C module is reset.</li> </ul> <p>1h (R/W) = Arbitration lost. AL is set by any one of the following events:</p> <ul style="list-style-type: none"> <li>- The I2C module senses that it has lost an arbitration with two or more competing transmitters that started a transmission almost simultaneously.</li> <li>- The I2C module attempts to start a transfer while the BB (bus busy) bit is set to 1.</li> </ul> <p>When AL becomes 1, the MST and STP bits of I2CMDR are cleared, and the I2C module becomes a slave-receiver.</p>

### 10.6.2.4 I2C Clock Low-time Divider (I2CCLKL) Register (Offset = 3h) [reset = 0h]

I2C Clock low-time divider

**Figure 10-22. I2C Clock Low-time Divider (I2CCLKL) Register**

15	14	13	12	11	10	9	8
I2CCLKL							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCLKL							
R/W-0h							

**Table 10-13. I2C Clock Low-time Divider (I2CCLKL) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-0	I2CCLKL	R/W	0h	Clock low-time divide-down value. To produce the low time duration of the master clock, the period of the module clock is multiplied by (ICCL + d). d is an adjustment factor based on the prescaler. See the Clock Divider Registers section of the Introduction for details. Note: These bits must be set to a non-zero value for proper I2C clock generation. Reset type: SYSRSn

### 10.6.2.5 I2C Clock High-time Divider (I2CCLKH) Register (Offset = 4h) [reset = 0h]

I2C Clock high-time divider

**Figure 10-23. I2C Clock High-time Divider (I2CCLKH) Register**

15	14	13	12	11	10	9	8
I2CCLKH							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCLKH							
R/W-0h							

**Table 10-14. I2C Clock High-time Divider (I2CCLKH) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-0	I2CCLKH	R/W	0h	Clock high-time divide-down value. To produce the high time duration of the master clock, the period of the module clock is multiplied by (ICCL + d). d is an adjustment factor based on the prescaler. See the Clock Divider Registers section of the Introduction for details. Note: These bits must be set to a non-zero value for proper I2C clock generation. Reset type: SYSRSn



### 10.6.2.6 I2C Data Count (I2CCNT) Register (Offset = 5h) [reset = 0h]

I2CCNT is a 16-bit register used to indicate how many data bytes to transfer when the I2C module is configured as a transmitter, or to receive when configured as a master receiver. In the repeat mode (RM = 1), I2CCNT is not used. The value written to I2CCNT is copied to an internal data counter. The internal data counter is decremented by 1 for each byte transferred (I2CCNT remains unchanged). If a STOP condition is requested in the master mode (STP = 1 in I2CMDR), the I2C module terminates the transfer with a STOP condition when the countdown is complete (that is, when the last byte has been transferred).

**Figure 10-24. I2C Data Count (I2CCNT) Register**

15	14	13	12	11	10	9	8
I2CCNT							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCNT							
R/W-0h							

**Table 10-15. I2C Data Count (I2CCNT) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-0	I2CCNT	R/W	0h	<p>Data count value. I2CCNT indicates the number of data bytes to transfer or receive.</p> <p>If a STOP condition is specified (STP=1) then I2CCNT will decrease after each byte is sent until it reaches zero, which in turn will generate a STOP condition.</p> <p>The value in I2CCNT is a don't care when the RM bit in I2CMDR is set to 1.</p> <p>The start value loaded to the internal data counter is 65536.</p> <p>The start value loaded to internal data counter is 1-65535.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = data count value is 65536</p> <p>1h (R/W) = data count value is 1</p> <p>2h (R/W) = data count value is 2</p> <p>FFFFh (R/W) = data count value is 65535</p>

### 10.6.2.7 I2C Data Receive Register (I2CDRR) (Offset = 6h) [reset = 0h]

I2CDRR is a 16-bit register used by the CPU to read received data. The I2C module can receive a data byte with 1 to 8 bits. The number of bits is selected with the bit count (BC) bits in I2CMADR. One bit at a time is shifted in from the SDA pin to the receive shift register (I2CRSR). When a complete data byte has been received, the I2C module copies the data byte from I2CRSR to I2CDRR. The CPU cannot access I2CRSR directly. If a data byte with fewer than 8 bits is in I2CDRR, the data value is right-justified, and the other bits of I2CDRR(7-0) are undefined. For example, if BC = 011 (3-bit data size), the receive data is in I2CDRR(2-0), and the content of I2CDRR(7-3) is undefined. When in the receive FIFO mode, the I2CDRR register acts as the receive FIFO buffer.

**Figure 10-25. I2C Data Receive Register (I2CDRR)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
DATA							
R-0h							

**Table 10-16. I2C Data Receive Register (I2CDRR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	DATA	R	0h	Receive data Reset type: SYSRSn

### 10.6.2.8 I2C Slave Address Register (I2CSAR) (Offset = 7h) [reset = 3FFh]

The I2C slave address register (I2CSAR) is a 16-bit register for storing the next slave address that will be transmitted by the I2C module when it is a master. The SAR field of I2CSAR contains a 7-bit or 10-bit slave address. When the I2C module is not using the free data format (FDF = 0 in I2CMDR), it uses this address to initiate data transfers with a slave, or slaves. When the address is nonzero, the address is for a particular slave. When the address is 0, the address is a general call to all slaves. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6-0 of I2CSAR are used write 0s to bits 9-7.

**Figure 10-26. I2C Slave Address Register (I2CSAR)**

15	14	13	12	11	10	9	8
RESERVED						SAR	
R-0h						R/W-3FFh	
7	6	5	4	3	2	1	0
SAR							
R/W-3FFh							

**Table 10-17. I2C Slave Address Register (I2CSAR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-10	RESERVED	R	0h	Reserved
9-0	SAR	R/W	3FFh	In 7-bit addressing mode (XA = 0 in I2CMDR): 00h-7Fh Bits 6-0 provide the 7-bit slave address that the I2C module transmits when it is in the master-transmitter mode. Write 0s to bits 9-7. In 10-bit addressing mode (XA = 1 in I2CMDR): 000h-3FFh Bits 9-0 provide the 10-bit slave address that the I2C module transmits when it is in the master transmitter mode. Reset type: SYSRSn

### 10.6.2.9 I2C Data Transmit Register (I2CDXR) (Offset = 8h) [reset = 0h]

The CPU writes transmit data to I2CDXR. This 16-bit register accepts a data byte with 1 to 8 bits. Before writing to I2CDXR, specify how many bits are in a data byte by loading the appropriate value into the bit count (BC) bits of I2CMDR. When writing a data byte with fewer than 8 bits, make sure the value is right-aligned in I2CDXR. After a data byte is written to I2CDXR, the I2C module copies the data byte to the transmit shift register (I2CXSR). The CPU cannot access I2CXSR directly. From I2CXSR, the I2C module shifts the data byte out on the SDA pin, one bit at a time. When in the transmit FIFO mode, the I2CDXR register acts as the transmit FIFO buffer.

**Figure 10-27. I2C Data Transmit Register (I2CDXR)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
DATA							
R/W-0h							

**Table 10-18. I2C Data Transmit Register (I2CDXR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	DATA	R/W	0h	Transmit data Reset type: SYSRSn

### 10.6.2.10 I2C Mode Register (I2CMDR) (Offset = 9h) [reset = 0h]

The I2C mode register (I2CMDR) is a 16-bit register that contains the control bits of the I2C module.

**Figure 10-28. I2C Mode Register (I2CMDR)**

15	14	13	12	11	10	9	8
NACKMOD	FREE	STT	RESERVED	STP	MST	TRX	XA
R/W-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
RM	DLB	IRS	STB	FDf	BC		
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h		R/W-0h	

**Table 10-19. I2C Mode Register (I2CMDR) Field Descriptions**

Bit	Field	Type	Reset	Description
15	NACKMOD	R/W	0h	<p>NACK mode bit. This bit is only applicable when the I2C module is acting as a receiver.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = In the slave-receiver mode: The I2C module sends an acknowledge (ACK) bit to the transmitter during each acknowledge cycle on the bus. The I2C module only sends a no-acknowledge (NACK) bit if you set the NACKMOD bit.</p> <p>In the master-receiver mode: The I2C module sends an ACK bit during each acknowledge cycle until the internal data counter counts down to 0. At that point, the I2C module sends a NACK bit to the transmitter. To have a NACK bit sent earlier, you must set the NACKMOD bit</p> <p>1h (R/W) = In either slave-receiver or master-receiver mode: The I2C module sends a NACK bit to the transmitter during the next acknowledge cycle on the bus. Once the NACK bit has been sent, NACKMOD is cleared.</p> <p>Important: To send a NACK bit in the next acknowledge cycle, you must set NACKMOD before the rising edge of the last data bit.</p>
14	FREE	R/W	0h	<p>This bit controls the action taken by the I2C module when a debugger breakpoint is encountered.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = When I2C module is master:</p> <p>If SCL is low when the breakpoint occurs, the I2C module stops immediately and keeps driving SCL low, whether the I2C module is the transmitter or the receiver. If SCL is high, the I2C module waits until SCL becomes low and then stops.</p> <p>When I2C module is slave:</p> <p>A breakpoint forces the I2C module to stop when the current transmission/reception is complete.</p> <p>1h (R/W) = The I2C module runs free that is, it continues to operate when a breakpoint occurs.</p>
13	STT	R/W	0h	<p>START condition bit (only applicable when the I2C module is a master). The RM, STT, and STP bits determine when the I2C module starts and stops data transmissions (see Table 9-6). Note that the STT and STP bits can be used to terminate the repeat mode, and that this bit is not writable when IRS = 0.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = In the master mode, STT is automatically cleared after the START condition has been generated.</p> <p>1h (R/W) = In the master mode, setting STT to 1 causes the I2C module to generate a START condition on the I2C-bus</p>

**Table 10-19. I2C Mode Register (I2CMDR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
12	RESERVED	R	0h	Reserved
11	STP	R/W	0h	<p>STOP condition bit (only applicable when the I2C module is a master).</p> <p>In the master mode, the RM,STT, and STP bits determine when the I2C module starts and stops data transmissions.</p> <p>Note that the STT and STP bits can be used to terminate the repeat mode, and that this bit is not writable when IRS=0. When in non-repeat mode, at least one byte must be transferred before a stop condition can be generated. The I2C module delays clearing of this bit until after the I2CSTR[SCD] bit is set. To avoid disrupting the I2C state machine, the user must wait until this bit is clear before initiating a new message.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = STP is automatically cleared after the STOP condition has been generated</p> <p>1h (R/W) = STP has been set by the device to generate a STOP condition when the internal data counter of the I2C module counts down to 0.</p>
10	MST	R/W	0h	<p>Master mode bit.</p> <p>MST determines whether the I2C module is in the slave mode or the master mode. MST is automatically changed from 1 to 0 when the I2C master generates a STOP condition</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Slave mode. The I2C module is a slave and receives the serial clock from the master.</p> <p>1h (R/W) = Master mode. The I2C module is a master and generates the serial clock on the SCL pin.</p>
9	TRX	R/W	0h	<p>Transmitter mode bit.</p> <p>When relevant, TRX selects whether the I2C module is in the transmitter mode or the receiver mode.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Receiver mode. The I2C module is a receiver and receives data on the SDA pin.</p> <p>1h (R/W) = Transmitter mode. The I2C module is a transmitter and transmits data on the SDA pin.</p>
8	XA	R/W	0h	<p>Expanded address enable bit.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = 7-bit addressing mode (normal address mode). The I2C module transmits 7-bit slave addresses (from bits 6-0 of I2CSAR), and its own slave address has 7 bits (bits 6-0 of I2COAR).</p> <p>1h (R/W) = 10-bit addressing mode (expanded address mode). The I2C module transmits 10-bit slave addresses (from bits 9-0 of I2CSAR), and its own slave address has 10 bits (bits 9-0 of I2COAR).</p>

**Table 10-19. I2C Mode Register (I2CMDR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
7	RM	R/W	0h	<p>Repeat mode bit (only applicable when the I2C module is a master-transmitter).</p> <p>The RM, STT, and STP bits determine when the I2C module starts and stops data transmissions</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Nonrepeat mode. The value in the data count register (I2CCNT) determines how many bytes are received/transmitted by the I2C module.</p> <p>1h (R/W) = Repeat mode. A data byte is transmitted each time the I2CDXR register is written to (or until the transmit FIFO is empty when in FIFO mode) until the STP bit is manually set. The value of I2CCNT is ignored. The ARDY bit/interrupt can be used to determine when the I2CDXR (or FIFO) is ready for more data, or when the data has all been sent and the CPU is allowed to write to the STP bit.</p>
6	DLB	R/W	0h	<p>Digital loopback mode bit.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = Digital loopback mode is disabled.</p> <p>1h (R/W) = Digital loopback mode is enabled. For proper operation in this mode, the MST bit must be 1.</p> <p>In the digital loopback mode, data transmitted out of I2CDXR is received in I2CDRR after n device cycles by an internal path, where:  <math>n = ((I2C \text{ input clock frequency} / \text{module clock frequency}) \times 8)</math></p> <p>The transmit clock is also the receive clock. The address transmitted on the SDA pin is the address in I2COAR.</p> <p>Note: The free data format (FDF = 1) is not supported in the digital loopback mode.</p>
5	IRS	R/W	0h	<p>I2C module reset bit.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = The I2C module is in reset/disabled. When this bit is cleared to 0, all status bits (in I2CSTR) are set to their default values.</p> <p>1h (R/W) = The I2C module is enabled. This has the effect of releasing the I2C bus if the I2C peripheral is holding it.</p>
4	STB	R/W	0h	<p>START byte mode bit. This bit is only applicable when the I2C module is a master. As described in version 2.1 of the Philips Semiconductors I2C-bus specification, the START byte can be used to help a slave that needs extra time to detect a START condition. When the I2C module is a slave, it ignores a START byte from a master, regardless of the value of the STB bit.</p> <p>Reset type: SYSRSn</p> <p>0h (R/W) = The I2C module is not in the START byte mode.</p> <p>1h (R/W) = The I2C module is in the START byte mode. When you set the START condition bit (STT), the I2C module begins the transfer with more than just a START condition. Specifically, it generates:</p> <ol style="list-style-type: none"> <li>1. A START condition</li> <li>2. A START byte (0000 0001b)</li> <li>3. A dummy acknowledge clock pulse</li> <li>4. A repeated START condition</li> </ol> <p>Then, as normal, the I2C module sends the slave address that is in I2CSAR.</p>

**Table 10-19. I2C Mode Register (I2CMDR) Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
3	FDF	R/W	0h	Free data format mode bit. Reset type: SYSRSn 0h (R/W) = Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit. 1h (R/W) = Free data format mode is enabled. Transfers have the free data (no address) format described in Section 9.2.5. The free data format is not supported in the digital loopback mode (DLB=1).
2-0	BC	R/W	0h	Bit count bits. BC defines the number of bits (1 to 8) in the next data byte that is to be received or transmitted by the I2C module. The number of bits selected with BC must match the data size of the other device. Notice that when BC = 000b, a data byte has 8 bits. BC does not affect address bytes, which always have 8 bits. Note: If the bit count is less than 8, receive data is right-justified in I2CDRR(7-0), and the other bits of I2CDRR(7-0) are undefined. Also, transmit data written to I2CDXR must be right-justified Reset type: SYSRSn 0h (R/W) = 8 bits per data byte 1h (R/W) = 1 bit per data byte 2h (R/W) = 2 bits per data byte 3h (R/W) = 3 bits per data byte 4h (R/W) = 4 bits per data byte 5h (R/W) = 5 bits per data byte 6h (R/W) = 6 bits per data byte 7h (R/W) = 7 bits per data byte



### 10.6.2.11 I2C Interrupt Source (I2CISRC) Register (Offset = Ah) [reset = 0h]

The I2C interrupt source register (I2CISRC) is a 16-bit register used by the CPU to determine which event generated the I2C interrupt.

**Figure 10-29. I2C Interrupt Source (I2CISRC) Register**

15	14	13	12	11	10	9	8
RESERVED				WRITE_ZEROS			
R-0h				R/W-0h			
7	6	5	4	3	2	1	0
RESERVED					INTCODE		
R-0h					R-0h		

**Table 10-20. I2C Interrupt Source (I2CISRC) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reserved
11-8	WRITE_ZEROS	R/W	0h	TI internal testing bits These reserved bit locations should always be written as zeros. Reset type: SYSRSn
7-3	RESERVED	R	0h	Reserved
2-0	INTCODE	R	0h	Interrupt code bits. The binary code in INTCODE indicates the event that generated an I2C interrupt. A CPU read will clear this field. If another lower priority interrupt is pending and enabled, the value corresponding to that interrupt will then be loaded. Otherwise, the value will stay cleared. In the case of an arbitration lost, a no-acknowledgment condition detected, or a stop condition detected, a CPU read will also clear the associated interrupt flag bit in the I2CSTR register. Inversely, in the case of a Register-access-ready (ARDY), a Receive-data-ready (RRDY), a Transmit-data-ready (XRDY), or an Addressed-as-slave (AAS) condition, a CPU read will NOT clear the associated interrupt flag bit in the I2CSTR register Debug probe reads will not affect the state of this field or of the status bits in the I2CSTR register. Reset type: SYSRSn 0h (R/W) = None 1h (R/W) = Arbitration lost 2h (R/W) = No-acknowledgment condition detected 3h (R/W) = Registers ready to be accessed 4h (R/W) = Receive data ready 5h (R/W) = Transmit data ready 6h (R/W) = Stop condition detected 7h (R/W) = Addressed as slave

**10.6.2.12 I2C Extended Mode Register (I2CEMDR) (Offset = Bh) [reset = 1h]**

I2C Extended Mode

**Figure 10-30. I2C Extended Mode Register (I2CEMDR)**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							BC
R-0h							R/W-1h

**Table 10-21. I2C Extended Mode Register (I2CEMDR) Field Descriptions**

Bit	Field	Type	Reset	Description
15-1	RESERVED	R	0h	Reserved
0	BC	R/W	1h	Backwards compatibility mode. This bit affects the timing of the transmit status bits (XRDY and XSMT) in the I2CSTR register when in slave transmitter mode. Check Backwards Compatibility Mode Bit, Slave Transmitter diagram for more details. Reset type: SYSRSn 0h (R/W) = See the "Backwards Compatibility Mode Bit, Slave Transmitter" Figure for details. 1h (R/W) = See the "Backwards Compatibility Mode Bit, Slave Transmitter" Figure for details.

### 10.6.2.13 I2C Prescaler (I2CPSC) Register (Offset = Ch) [reset = 0h]

The I2C prescaler register (I2CPSC) is a 16-bit register used for dividing down the I2C input clock to obtain the desired module clock for the operation of the I2C module. See the device-specific data manual for the supported range of values for the module clock frequency. IPSC must be initialized while the I2C module is in reset (IRS = 0 in I2CMR). The prescaled frequency takes effect only when IRS is changed to 1. Changing the IPSC value while IRS = 1 has no effect.

**Figure 10-31. I2C Prescaler (I2CPSC) Register**

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
IPSC							
R/W-0h							

**Table 10-22. I2C Prescaler (I2CPSC) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	IPSC	R/W	0h	I2C prescaler divide-down value. IPSC determines how much the CPU clock is divided to create the module clock of the I2C module: module clock frequency = I2C input clock frequency / (IPSC + 1) Note: IPSC must be initialized while the I2C module is in reset (IRS = 0 in I2CMR). Reset type: SYSRSn

### 10.6.2.14 I2C FIFO Transmit (I2CFFTX) Register (Offset = 20h) [reset = 0h]

The I2C transmit FIFO register (I2CFFTX) is a 16-bit register that contains the I2C FIFO mode enable bit as well as the control and status bits for the transmit FIFO mode of operation on the I2C peripheral.

**Figure 10-32. I2C FIFO Transmit (I2CFFTX) Register**

15	14	13	12	11	10	9	8	
RESERVED	I2CFFEN	TXFFRST	TXFFST					
R-0h	R/W-0h	R/W-0h	R-0h					
7	6	5	4	3	2	1	0	
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL					
R-0h	R-0/W1S-0h	R/W-0h	R/W-0h					

**Table 10-23. I2C FIFO Transmit (I2CFFTX) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reserved
14	I2CFFEN	R/W	0h	I2C FIFO mode enable bit. This bit must be enabled for either the transmit or the receive FIFO to operate correctly. Reset type: SYSRSn 0h (R/W) = Disable the I2C FIFO mode. 1h (R/W) = Enable the I2C FIFO mode.
13	TXFFRST	R/W	0h	Transmit FIFO Reset Reset type: SYSRSn 0h (R/W) = Reset the transmit FIFO pointer to 0000 and hold the transmit FIFO in the reset state. 1h (R/W) = Enable the transmit FIFO operation.
12-8	TXFFST	R	0h	Contains the status of the transmit FIFO: xxxxx Transmit FIFO contains xxxxx bytes. 00000 Transmit FIFO is empty. Note: Since these bits are reset to zero, the transmit FIFO interrupt flag will be set when the transmit FIFO operation is enabled and the I2C is taken out of reset. This will generate a transmit FIFO interrupt if enabled. To avoid any detrimental effects from this, write a one to the TXFFINTCLR once the transmit FIFO operation is enabled and the I2C is taken out of reset. Reset type: SYSRSn
7	TXFFINT	R	0h	Transmit FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the TXFFINTCLR bit. If the TXFFIENA bit is set, this bit will generate an interrupt when it is set. Reset type: SYSRSn 0h (R/W) = Transmit FIFO interrupt condition has not occurred. 1h (R/W) = Transmit FIFO interrupt condition has occurred.
6	TXFFINTCLR	R-0/W1S	0h	Transmit FIFO Interrupt Flag Clear Reset type: SYSRSn 0h (R/W) = Writes of zeros have no effect. Reads return a 0. 1h (R/W) = Writing a 1 to this bit clears the TXFFINT flag.
5	TXFFIENA	R/W	0h	Transmit FIFO Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disabled. TXFFINT flag does not generate an interrupt when set. 1h (R/W) = Enabled. TXFFINT flag does generate an interrupt when set.

**Table 10-23. I2C FIFO Transmit (I2CFFTX) Register Field Descriptions (continued)**

Bit	Field	Type	Reset	Description
4-0	TXFFIL	R/W	0h	Transmit FIFO interrupt level. These bits set the status level that will set the transmit interrupt flag. When the TXFFST4-0 bits reach a value equal to or less than these bits, the TXFFINT flag will be set. This will generate an interrupt if the TXFFIENA bit is set. Because the I2C on this device has a 4-level transmit FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels. Reset type: SYSRSn

### 10.6.2.15 I2C FIFO Receive (I2CFFRX) Register (Offset = 21h) [reset = 0h]

The I2C receive FIFO register (I2CFFRX) is a 16-bit register that contains the control and status bits for the receive FIFO mode of operation on the I2C peripheral.

**Figure 10-33. I2C FIFO Receive (I2CFFRX) Register**

15		14		13		12		11		10		9		8	
RESERVED				RXFFRST				RXFFST							
R-0h				R/W-0h				R-0h							
7		6		5		4		3		2		1		0	
RXFFINT		RXFFINTCLR		RXFFIENA		RXFFIL									
R-0h		R-0/W1S-0h		R/W-0h		R/W-0h									

**Table 10-24. I2C FIFO Receive (I2CFFRX) Register Field Descriptions**

Bit	Field	Type	Reset	Description
15-14	RESERVED	R	0h	Reserved
13	RXFFRST	R/W	0h	I2C receive FIFO reset bit Reset type: SYSRSn 0h (R/W) = Reset the receive FIFO pointer to 0000 and hold the receive FIFO in the reset state. 1h (R/W) = Enable the receive FIFO operation.
12-8	RXFFST	R	0h	Contains the status of the receive FIFO: xxxxx Receive FIFO contains xxxxx bytes 00000 Receive FIFO is empty. Reset type: SYSRSn
7	RXFFINT	R	0h	Receive FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the RXFFINTCLR bit. If the RXFFIENA bit is set, this bit will generate an interrupt when it is set Reset type: SYSRSn 0h (R/W) = Receive FIFO interrupt condition has not occurred. 1h (R/W) = Receive FIFO interrupt condition has occurred.
6	RXFFINTCLR	R-0/W1S	0h	Receive FIFO interrupt flag clear bit. Reset type: SYSRSn 0h (R/W) = Writes of zeros have no effect. Reads return a zero. 1h (R/W) = Writing a 1 to this bit clears the RXFFINT flag.
5	RXFFIENA	R/W	0h	Receive FIFO interrupt enable bit. Reset type: SYSRSn 0h (R/W) = Disabled. RXFFINT flag does not generate an interrupt when set. 1h (R/W) = Enabled. RXFFINT flag does generate an interrupt when set.
4-0	RXFFIL	R/W	0h	Receive FIFO interrupt level. These bits set the status level that will set the receive interrupt flag. When the RXFFST4-0 bits reach a value equal to or greater than these bits, the RXFFINT flag is set. This will generate an interrupt if the RXFFIENA bit is set. Note: Since these bits are reset to zero, the receive FIFO interrupt flag will be set if the receive FIFO operation is enabled and the I2C is taken out of reset. This will generate a receive FIFO interrupt if enabled. To avoid this, modify these bits on the same instruction as or prior to setting the RXFFRST bit. Because the I2C on this device has a 4-level receive FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels. Reset type: SYSRSn

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

**Changes from December 19, 2018 to May 31, 2022 (from Revision \* (December 2018) to Revision A (June 2022))**

	<b>Page</b>
• <b>Chapter 1 System Control and Interrupts:</b> .....	23
• Added <a href="#">Section 1.1.4.1</a> .....	31
• Added <a href="#">Section 1.1.4.2</a> .....	31
• Added <a href="#">Section 1.1.4.3</a> .....	32
• Added <a href="#">Section 1.1.4.4</a> .....	33
• Added <a href="#">Section 1.1.4.5</a> .....	33
• Added <a href="#">Section 1.1.4.6</a> .....	34
• Added <a href="#">Section 1.1.4.7</a> .....	35
• Added <a href="#">Section 1.2.6</a> .....	45
• Added <a href="#">Section 1.3.2.1.1</a> .....	53
• Added <a href="#">Section 1.3.2.4.1.2</a> .....	61
• Added <a href="#">Section 1.3.2.4.1.3</a> .....	63
• Added <a href="#">Section 1.3.2.7.1</a> .....	68
• Added <a href="#">Section 1.3.2.7.1.1</a> .....	68
• Added <a href="#">Section 1.3.2.7.1.2</a> .....	69
• Added <a href="#">Section 1.3.2.7.1.3</a> .....	69
• Added <a href="#">Section 1.3.2.7.1.4</a> .....	70
• Added <a href="#">Section 1.3.2.7.1.5</a> .....	70
• Added <a href="#">Section 1.3.2.7.1.6</a> .....	71
• Added <a href="#">Section 1.3.3.1</a> .....	74
• Added <a href="#">Section 1.3.4.5.1</a> .....	78
• Added <a href="#">Section 1.3.4.5.2</a> .....	79
• Added <a href="#">Section 1.3.4.5.3</a> .....	79
• Added <a href="#">Section 1.3.4.5.4</a> .....	80
• Added <a href="#">Section 1.3.5.1</a> .....	82
• Added <a href="#">Section 1.3.5.2</a> .....	83
• Added <a href="#">Section 1.3.5.3</a> .....	83
• Added <a href="#">Section 1.3.5.4</a> .....	83
• Added <a href="#">Section 1.3.5.5</a> .....	84
• Added <a href="#">Section 1.3.5.6</a> .....	85
• Added <a href="#">Section 1.3.5.7</a> .....	86
• Added <a href="#">Section 1.4.6.1</a> .....	103
• Added <a href="#">Section 1.4.6.2</a> .....	104
• Added <a href="#">Section 1.4.6.3</a> .....	105
• Added <a href="#">Section 1.4.6.4</a> .....	105
• Added <a href="#">Section 1.4.6.5</a> .....	107
• Added <a href="#">Section 1.4.6.6</a> .....	109
• Added <a href="#">Section 1.4.6.7</a> .....	110
• Added <a href="#">Section 1.4.6.8</a> .....	111
• Added <a href="#">Section 1.4.6.9</a> .....	112
• Added <a href="#">Section 1.4.6.10</a> .....	112

• Added <a href="#">Section 1.4.6.11</a> .....	114
• Added <a href="#">Section 1.4.6.12</a> .....	115
• Added <a href="#">Section 1.4.6.13</a> .....	116
• Added <a href="#">Section 1.4.6.14</a> .....	117
• Added <a href="#">Section 1.4.6.15</a> .....	118
• Added <a href="#">Section 1.4.6.16</a> .....	119
• Added <a href="#">Section 1.4.6.17</a> .....	120
• Added <a href="#">Section 1.4.6.18</a> .....	121
• Added <a href="#">Section 1.4.6.19</a> .....	122
• Added <a href="#">Section 1.4.6.20</a> .....	123
• Added <a href="#">Section 1.4.6.21</a> .....	124
• Added <a href="#">Section 1.4.6.22</a> .....	125
• Added <a href="#">Section 1.5.3.1</a> .....	132
• Added <a href="#">Section 1.5.3.2</a> .....	133
• Added <a href="#">Section 1.5.3.3</a> .....	134
• Added <a href="#">Section 1.5.3.4</a> .....	134
• Added <a href="#">Section 1.6.4.1</a> .....	151
• Added <a href="#">Section 1.6.4.2</a> .....	151
• Added <a href="#">Section 1.6.4.7</a> .....	159
• Added <a href="#">Section 1.6.5.1</a> .....	161
• Added <a href="#">Section 1.6.5.2</a> .....	162
• <b>Chapter 2 Boot ROM:</b> .....	165
• <b>Chapter 3 Enhanced Pulse Width Modulator (ePWM):</b> .....	207
• Changed <a href="#">Section 3.2.8.1</a> .....	258
• Added <a href="#">Section 3.2.8.2</a> .....	260
• Added <a href="#">Section 3.2.8.3</a> .....	260
• Changed <a href="#">Section 3.4.1</a> .....	294
• Changed <a href="#">Section 3.4.2</a> .....	302
• Changed <a href="#">Section 3.4.3</a> .....	308
• Changed <a href="#">Section 3.4.4</a> .....	313
• Changed <a href="#">Section 3.4.5</a> .....	316
• Changed <a href="#">Section 3.4.6</a> .....	325
• Changed <a href="#">Section 3.4.7</a> .....	331
• Changed <a href="#">Section 3.4.8</a> .....	333
• <b>Chapter 4 High-Resolution Pulse Width Modulator (HRPWM):</b> .....	343
• Changed the equation for autoconversion calculation in <a href="#">Section 4.2.3.4</a> .....	355
• Added <a href="#">Section 4.4.1</a> .....	367
• Added <a href="#">Section 4.4.2</a> .....	369
• Added <a href="#">Section 4.4.3</a> .....	369
• Added <a href="#">Section 4.4.4</a> .....	370
• <b>Chapter 5 Enhanced Capture (eCAP):</b> .....	371
• <b>Chapter 6 Analog-to-Digital Converter (ADC):</b> .....	399
• Added guidelines for estimating ACQPS values in <a href="#">Section 6.2.1</a> .....	404
• Added <a href="#">Section 6.5</a> .....	413
• Changed <a href="#">Section 6.13.3</a> .....	428
• Changed <a href="#">Section 6.13.5</a> .....	437
• Changed <a href="#">Section 6.13.6</a> .....	446
• <b>Chapter 7 Comparator (COMP):</b> .....	449
• <b>Chapter 8 Serial Peripheral Interface (SPI):</b> .....	459
• <b>Chapter 9 Serial Communications Interface (SCI):</b> .....	497
• Changed <a href="#">Section 9.1</a> .....	498
• Added <a href="#">Section 9.1.1</a> .....	498
• Added <a href="#">Section 9.1.3</a> .....	499
• Changed <a href="#">Figure 9-10</a> .....	509
• <b>Chapter 10 Inter-Integrated Circuit Module (I2C):</b> .....	531



---

• Added <a href="#">Figure 10-6</a> .....	537
• Added <a href="#">Figure 10-7</a> . Subsequent figures renumbered.....	537
• Moved <a href="#">Section 10.3.10</a> .....	548

---

This page intentionally left blank.

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](http://ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2022, Texas Instruments Incorporated