Piccolo TMS320F28069 Isolated controlCARD

Part No. TMDSCNCD28069ISO

The C2000 controlCARDs from Texas Instruments are ideal products for initial software development and short run builds for system prototypes, test stands, and many other projects that require easy access to high-performance controllers. The controlCARDs are complete board-level modules that utilize an industry-standard DIMM form factor to provide a low-profile single-board controller solution. C2000 controlCARDs typically use the DIM100 connector footprint providing access to the analog and digital I/Os of the C2000 MCU, while likewise providing a standard for easy interchange of C2000 MCUs in an application.

All software, documentation, and hardware documents can be accessed by installing controlSUITE software.

Features

- Piccolo TMS320F28069U MCU
- Isolated USB JTAG Interface (XDS100v1)
- Standard 100-pin DIMM Interface
- Analog I/O, digital I/O and JTAG signals at DIMM interface
- Single 5V power rail for full operation

What's Included

- TMDXCNCD28069ISO Isolated controlCARD
- Software & hardware support available in controlSUITE™ software.

TMS320x2806x Piccolo

Technical Reference Manual

Literature Number: SPRUH18E
January 2011–Revised March 2014

Contents

List of Figures

List of Tables

Read This First

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.

This TRM has been designated *Preliminary* because the documentation is in the formative or design phase of development. Texas Instruments reserves the right to change this TRM without notice. Visit the Texas Instruments website at http://www.ti.com to determine the latest version of this TRM.

Notational Conventions

These documents use the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.

Related Documentation From Texas Instruments

For a complete listing of related documentation and development-support tools for the TMS320x2806x Piccolo devices, visit the Texas Instruments website at http://www.ti.com. Additionally, the *TMS320C28x CPU and Instruction Set Reference Guide* (SPRU430) and *TMS320C28x Floating Point Unit and Instruction Set Reference Guide* (SPRUEO2) must be used in conjunction with this TRM.

System Control and Interrupts

This chapter is applicable for the System Control and Interrupts found on the Piccolo™ microcontrollers (MCUs). This guide 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 incorporated in TMS320x28x devices.
- 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.

Topic **Page**

1.1 Flash and OTP Memory Blocks

This chapter 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 then 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 affect 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.

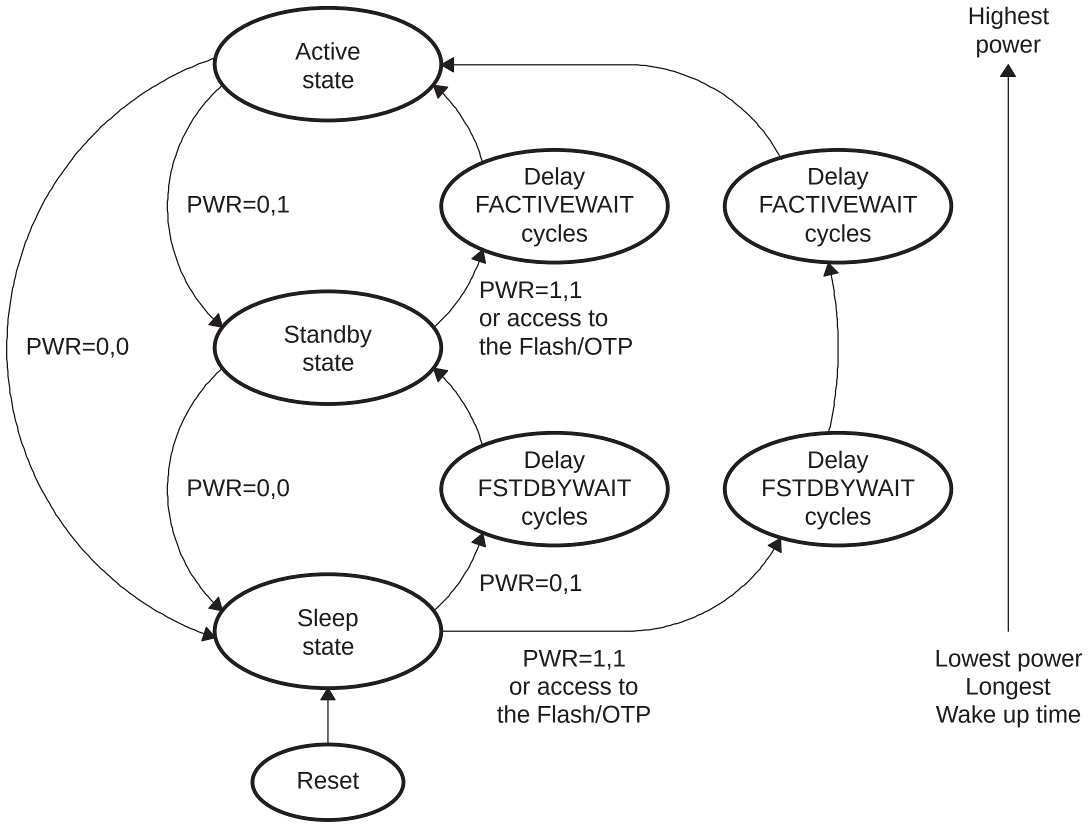
- **To move to a higher power state**

  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.

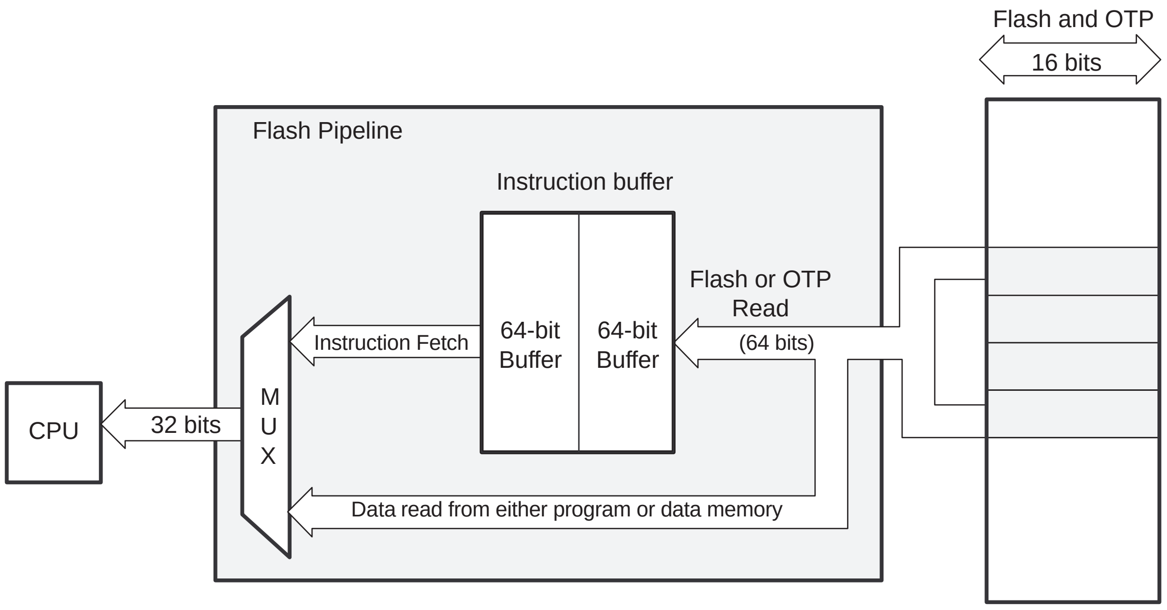
1.1.3.2 Flash Pipeline Mode

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 pre-fetch buffer. The contents of this pre-fetch 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 pre-fetch 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 pre-fetch the next 64 bits. In this manner, the flash pipeline mode works in the background to keep the instruction pre-fetch 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 pre-fetch is aborted only on a PC discontinuity caused by executing an instruction such as a branch, BANZ, call, or loop. When this occurs, the pre-fetch is aborted and the contents of the pre-fetch buffer are flushed. There are two possible scenarios when this occurs:

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

1.1.3.3 Reserved Locations Within Flash and OTP

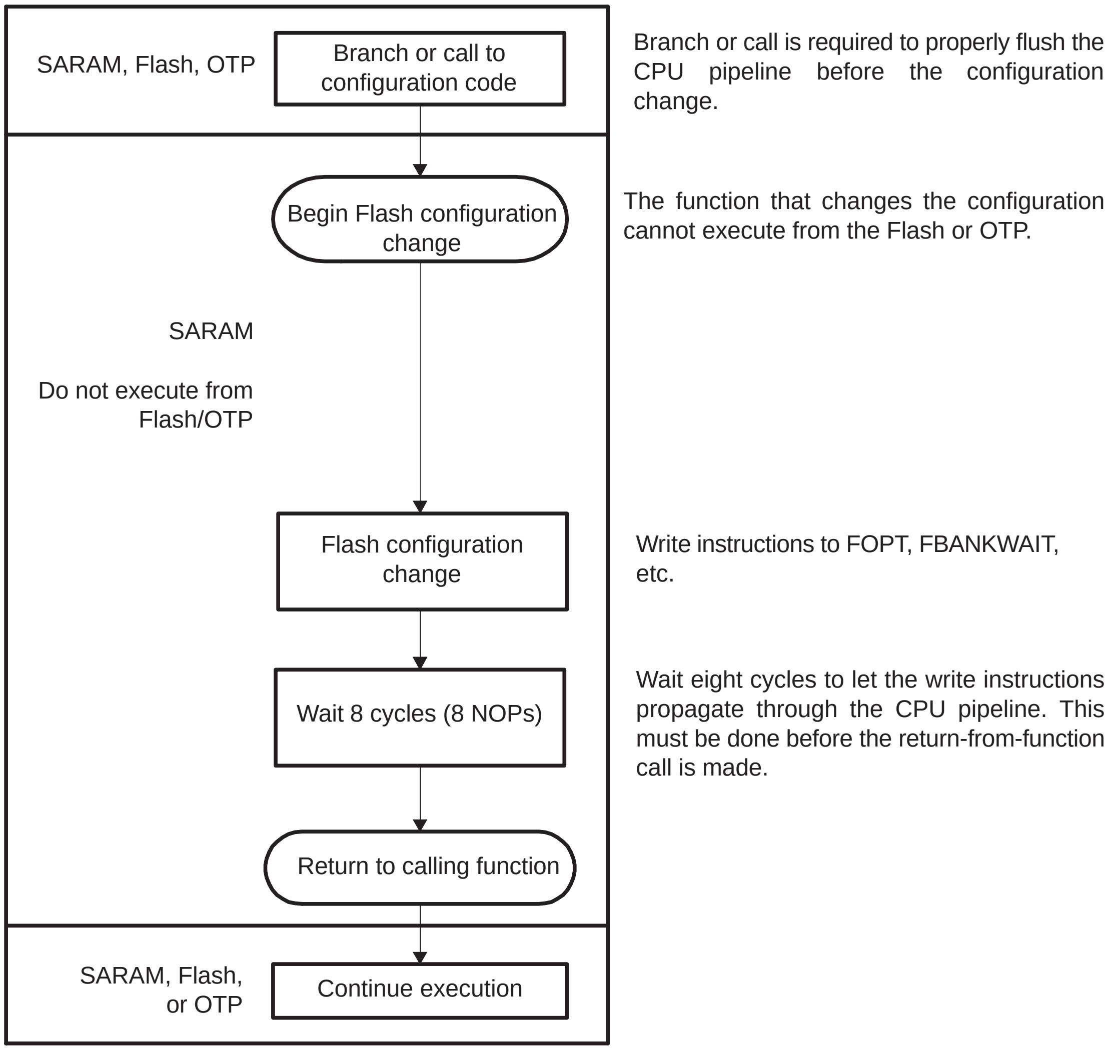
When allocating code and data to flash and OTP memory, keep the following in mind:

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 re-direct code execution to the entry point of the application.
2. Addresses from 0x3F 7FF0 to 0x3F 7FF5 are reserved for data variables and should not contain program code.

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 pre-fetch 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


SARAM, Flash, OTP
Branch or call to configuration code
Branch or call is required to properly flush the CPU pipeline before the configuration change.
Begin Flash configuration change
The function that changes the configuration cannot execute from the Flash or OTP.
SARAM
Do not execute from Flash/OTP
Flash configuration change
Write instructions to FOPT, FBANKWAIT, etc.
Wait 8 cycles (8 NOPs)
Wait eight cycles to let the write instructions propagate through the CPU pipeline. This must be done before the return-from-function call is made.
Return to calling function
SARAM, Flash, or OTP
Continue execution

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. The bit descriptions are in Figure 1-4 through Figure 1-10.

Table 1-1. Flash/OTP Configuration Registers

Name[1] [2]	Address	Size (x16)	Description	Bit Description
FOPT	0x0A80	1	Flash Option Register	Figure 1-4
Reserved	0x0A81	1	Reserved	
FPWR	0x0A82	1	Flash Power Modes Register	Figure 1-5
FSTATUS	0x0A83	1	Status Register	Figure 1-6
FSTDBYWAIT [3]	0x0A84	1	Flash Sleep To Standby Wait Register	Figure 1-7
FACTIVEWAIT [3]	0x0A85	1	Flash Standby To Active Wait Register	Figure 1-8
FBANKWAIT	0x0A86	1	Flash Read Access Wait State Register	Figure 1-9
FOTPWAIT	0x0A87	1	OTP Read Access Wait State Register	Figure 1-10

(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.

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 [(1)] [(2)] [(3)]
15-1	Reserved		Any writes to these bits(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 pre-fetching instructions. See Section 1.1.3.2 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.

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 [(1)] [(2)]
15-2	Reserved		Any writes to these bits(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 section Section 1.1.3 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.

Figure 1-6. Flash Status Register (FSTATUS)

15	9	8
Reserved		3VSTAT
R-0		R/W1C-0

7	4	3	2	1	0
Reserved		ACTIVEWAITS	STDBYWAITS	PWRS	
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 [(1)] [(2)]
15-9	Reserved		Any writes to these bits(s) must always have a value of 0.
8	3VSTAT		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.
		0	Writes of 0 are ignored.
		1	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 bits(s) must always have a value of 0.
3	ACTIVEWAITS		Bank and Pump Standby To Active Wait Counter Status Bit. This bit indicates whether the respective wait counter is timing out an access.
		0	The counter is not counting.
		1	The counter is counting.
2	STDBYWAITS		Bank and Pump Sleep To Standby Wait Counter Status Bit. This bit indicates whether the respective wait counter is timing out an access.
		0	The counter is not counting.
		1	The counter is counting.
1-0	PWRS		Power M odes 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.
		00	Pump and bank in sleep mode (lowest power)
		01	Pump and bank in standby mode
		10	Reserved
		11	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.

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 [1] [2]
15-9	Reserved		Any writes to these bits(s) must always have a value of 0.
8-0	STDBYWAIT		**This register should be left in its default state.**
			Bank and Pump Sleep To Standby Wait Count.
		111111111	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.

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

7 - 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 [1] [2]
15-9	Reserved		Any writes to these bits(s) must always have a value of 0.
8-0	ACTIVEWAIT		**This register should be left in its default state.**
			Bank and Pump Standby To Active Wait Count:
		111111111	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.

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 [(1)] [(2)] [(3)]
15-12	Reserved		Any writes to these bits(s) must always have a value of 0.
11-8	PAGEWAIT		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 Section 1.1.3.1 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 then RANDWAIT.
		0000	Zero wait-state per paged flash access or one SYSCLKOUT cycle per access
		0001	One wait state per paged flash access or a total of two SYSCLKOUT cycles per access
		0010	Two wait states per paged flash access or a total of three SYSCLKOUT cycles per access
		0011	Three wait states per paged flash access or a total of four SYSCLKOUT cycles per access
		. . .	. . .
		1111	15 wait states per paged flash access or a total of 16 SYSCLKOUT cycles per access. (default)
7-4	Reserved		Any writes to these bits(s) must always have a value of 0.
3-0	RANDWAIT		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 Section 1.1.3.1 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 then RANDWAIT.
		0000	Illegal value. RANDWAIT must be set greater then 0.
		0001	One wait state per random flash access or a total of two SYSCLKOUT cycles per access.
		0010	Two wait states per random flash access or a total of three SYSCLKOUT cycles per access.
		0011	Three wait states per random flash access or a total of four SYSCLKOUT cycles per access.
		. . .	. . .
		1111	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.

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

15 5	4 0
Reserved	OTPWAIT
R-0	R/W-0x1F

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

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

Bit(s)	Field	Value	Description [(1)] [(2)] [(3)]
15-5	Reserved		Any writes to these bits(s) must always have a value of 0.
4-0	OTPWAIT		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.
			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.
		00000	Illegal value. OTPWAIT must be set to 1 or greater.
		00001	One wait state will be used each OTP access for a total of two SYSCLKOUT cycles per access.
		00010	Two wait states will be used for each OTP access for a total of three SYSCLKOUT cycles per access.
		00011	Three wait states will be used for each OTP access for a total of four SYSCLKOUT cycles per access.
		. . .	. . .
		11111	31 wait states will be used for an OTP access for a total of 32 SYSCLKOUT cycles per access.

(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/visibility to on-chip memory to unauthorized persons — that is, it prevents duplication/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 (through a debugging tool such as Code Composer Studio™, for example).

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 emulator). 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. Table 1-9 shows the levels of security.

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. Care must be taken 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 emulator 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 emulator 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 emulator 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 emulator takes control. The emulator 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 emulator is connected by re-mapping the PC to another address or by changing the boot mode selection pin to the desired boot mode.

NOTE: 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.

Disclaimer: 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 87FF	L0 SARAM (2K x 16)
0x00 8800 - 0x00 8BFF	L1 DPSARAM - CLA Data RAM 0 (1K x 16)
0x00 8C00 - 0x00 8FFF	L2 DPSARAM - CLA Data RAM 1 (1K x 16)
0x00 9000 - 0x00 9FFF	L3 DPSARAM - CLA Program RAM (4K x 16)
0x3E 8000 - 0x3F 7FFF or 0x3F 0000 - 0x3F 7FFF	Flash (64K 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)(1)(1K x 16)
0x00 A000 - 0x00 BFFF	L4 SARAM

(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)
0x00 C000 - 0x00DFFF	LS DSPARAM
0x3F 8000 - 0x3F FFFF	Boot ROM (32K x 16)

To summarize, it is possible to load code onto the unprotected on-chip program SARAM via 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 (via 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-customer 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 a different secure zone or 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.

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

Memory Address	Register Name	Reset Values	Register Description
KEY Registers			
0x00 - 0AE0	KEY0 [1]	0xFFFF	Low word of the 128-bit KEY register
0x00 - 0AE1	KEY1 [1]	0xFFFF	Second word of the 128-bit KEY register
0x00 - 0AE2	KEY2 [1]	0xFFFF	Third word of the 128-bit KEY register
0x00 - 0AE3	KEY3 [1]	0xFFFF	Fourth word of the 128-bit key
0x00 - 0AE4	KEY4 [1]	0xFFFF	Fifth word of the 128-bit key
0x00 - 0AE5	KEY5 [1]	0xFFFF	Sixth word of the 128-bit key
0x00 - 0AE6	KEY6 [1]	0xFFFF	Seventh word of the 128-bit key
0x00 - 0AE7	KEY7 [1]	0xFFFF	High word of the 128-bit KEY register
0x00 - 0AEF	CSMSCR [1]	0x002F	CSM status and control register
Password Locations (PWL) in Flash Memory - Reserved for the CSM password only			
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.

Figure 1-11. 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 [1]
15	FORCESEC		Writing a 1 clears the KEY registers and secures the device.
		0	A read always returns a zero.
		1	Clears the KEY registers and secures the device. The password match flow described in Section 1.2.3.2 must be followed to unsecure the device again.
14-1	Reserved		Reserved
0	SECURE		Read-only bit that reflects the security state of the device.
		0	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.

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™ F28xx On-Chip Flash Programmer plug-in.

  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.

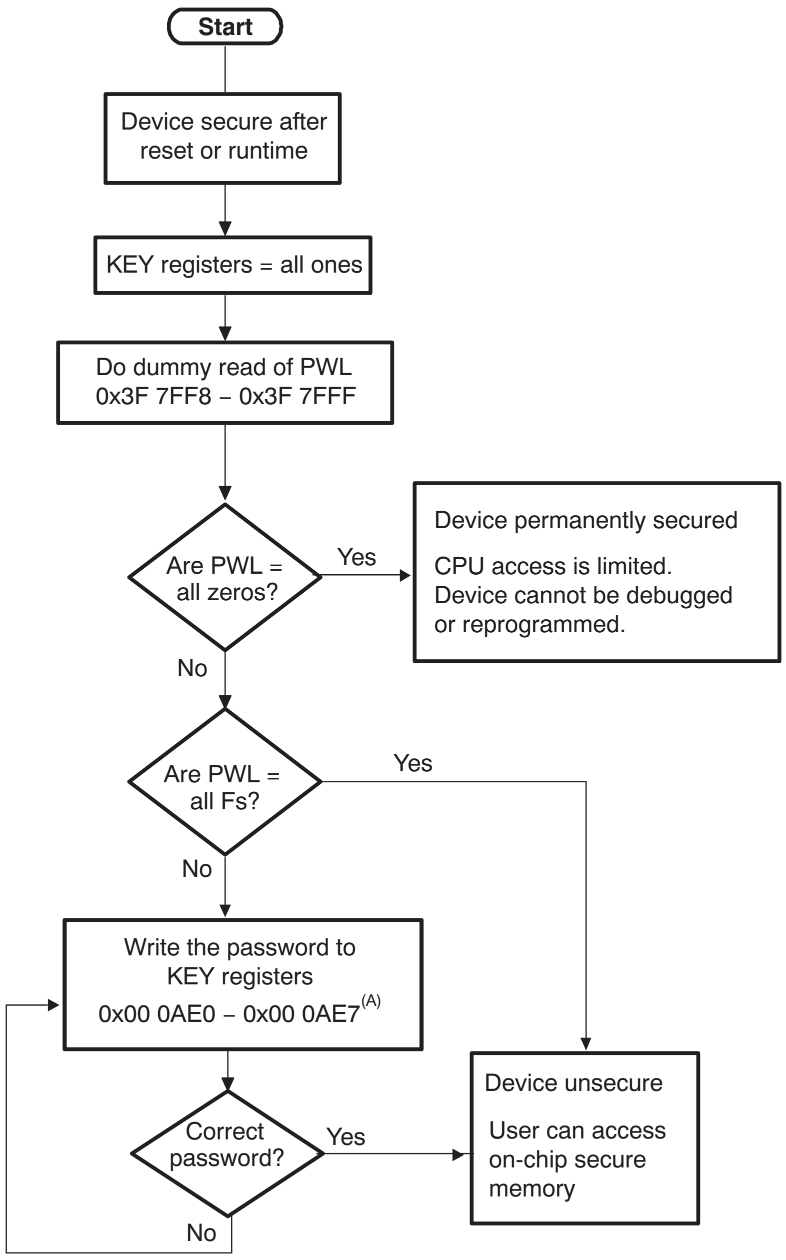
The unsecuring sequence is identical in all the above situations. This sequence is referred to as the *password match flow (PMF)* for simplicity. Figure 1-12 explains the sequence of operation that is required every time the user attempts to unsecure a device. A code example is listed for clarity.

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 1-12 shows how the PMF helps to initialize the security logic registers and disable security logic.

Figure 1-12. Password Match Flow (PMF)


Start
Device secure after reset or runtime
KEY registers = all ones
Do dummy read of PWL 0x3F 7FF8 – 0x3F 7FFF
Are PWL = all zeros?
Yes
Device permanently secured
CPU access is limited.
Device cannot be debugged or reprogrammed.
No
Are PWL = all Fs?
Yes
No
Write the password to KEY registers
0x00 0AE0 – 0x00 0AE7(A)
Correct password?
Yes
Device unsecure
User can access on-chip secure memory
No

A The KEY registers are EALLOW protected.

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, one 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/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. The CSM blocks the OTP reads to password location. Hence the dummy reads to password location can be done only from RAM (secure/unsecure) or Flash.
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. The CSM blocks the OTP reads to password location. Hence the dummy reads to password location can be done only from RAM (secure/unsecure) or Flash.
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 ones 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

1. The flash is secured after a reset until the password match flow described in Section 1.2.3.2 is executed.
2. The standard way of running code out of the flash is to program the flash with the code and power up the DSP. 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.
3. Secure memory cannot be modified by code executing from unsecure memory while the device is secured.
4. Secure memory cannot be read from any code running from unsecure memory while the device is secured.
5. Secure memory cannot be read or written to by the debugger (Code Composer Studio™) at any time that the device is secured.
6. Complete access to secure memory from both the CPU code and the debugger is granted while the device is unsecured.

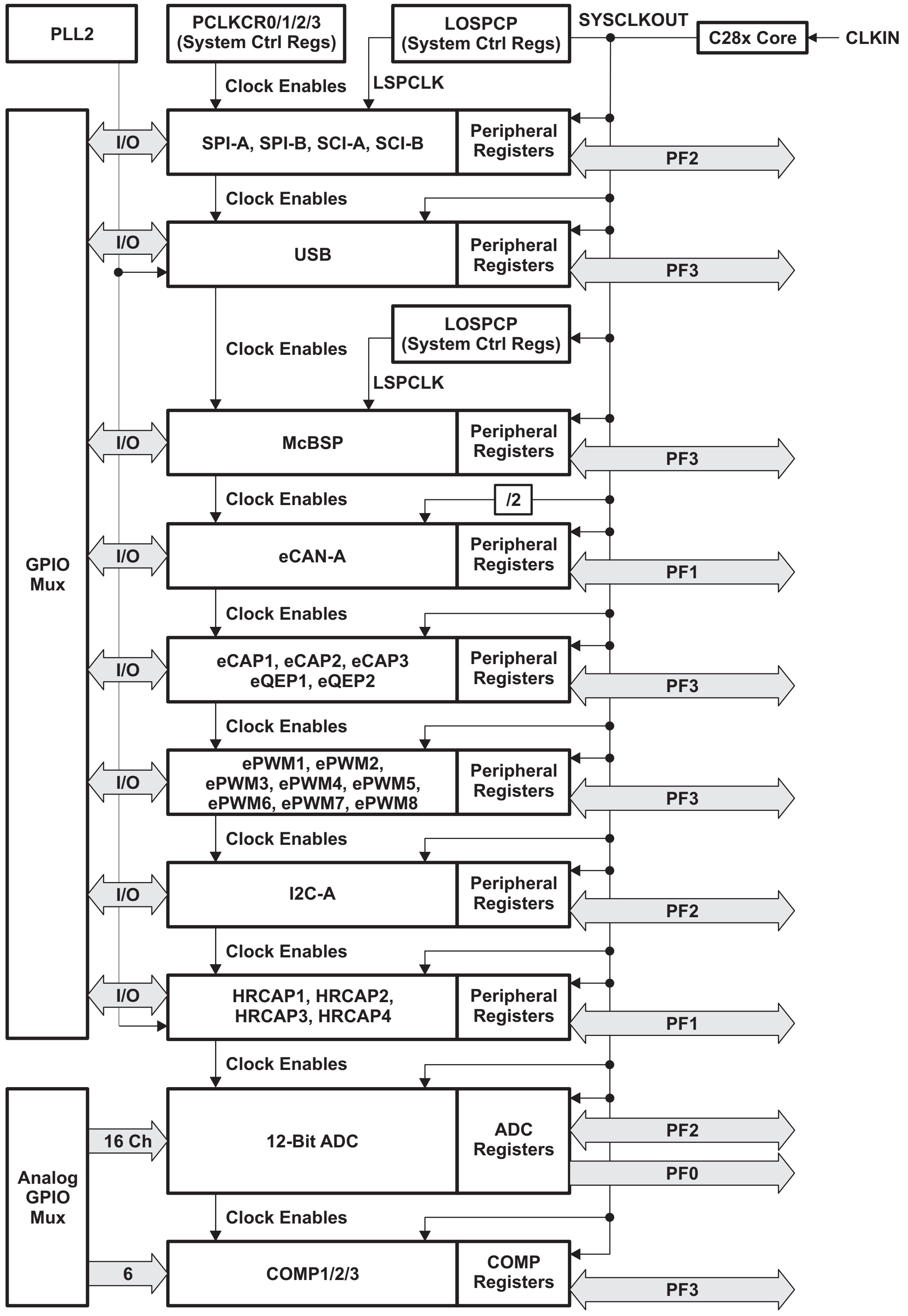
1.3 Clocking

This section describes the oscillator, PLL and clocking mechanisms, the watchdog function, and the low-power modes.

1.3.1 Clocking and System Control

Figure 1-13 shows the various clock and reset domains.

Figure 1-13. Clock and Reset Domains


PLL2
PCLKCR0/1/2/3 (System Ctrl Regs)
LOSPCP (System Ctrl Regs)
SYSCLKOUT
C28x Core
CLKIN
Clock Enables
LSPCLK
GPIO Mux
I/O
SPI-A, SPI-B, SCI-A, SCI-B
Peripheral Registers
PF2
USB
PF3
McBSP
/2
eCAN-A
PF1
eCAP1, eCAP2, eCAP3 eQEP1, eQEP2
ePWM1, ePWM2, ePWM3, ePWM4, ePWM5, ePWM6, ePWM7, ePWM8
I2C-A
HRCAP1, HRCAP2, HRCAP3, HRCAP4
Analog GPIO Mux
16 Ch
12-Bit ADC
ADC Registers
PF0
6
COMP1/2/3
COMP Registers

The PLL, clocking, watchdog and low-power modes, are controlled by the registers listed in Table 1-14.

Table 1-14. PLL, Clocking, Watchdog, and Low-Power Mode Registers

NAME	ADDRESS	SIZE (x16)	DESCRIPTION
BORCFG	0x00 0985	1	BOR Configuration Register
XCLK	0x00 7010	1	XCLKOUT Control
PLLSTS	0x00 7011	1	PLL Status Register
CLKCTL	0x00 7012	1	Clock Control Register
PLLLOCKPRD	0x00 7013	1	PLL Lock Period
INTOSC1TRIM	0x00 7014	1	Internal Oscillator 1 Trim Register
INTOSC2TRIM	0x00 7016	1	Internal Oscillator 2 Trim Register
PCLKCR2	0x00 7019	1	Peripheral Clock Control Register 2
LOSPCP	0x00 701B	1	Low-Speed Peripheral Clock Prescaler Register
PCLKCR0	0x00 701C	1	Peripheral Clock Control Register 0
PCLKCR1	0x00 701D	1	Peripheral Clock Control Register 1
LPMCR0	0x00 701E	1	Low Power Mode Control Register 0
PCLKCR3	0x00 7020	1	Peripheral Clock Control Register 3
PLLCR	0x00 7021	1	PLL Control Register
SCSR	0x00 7022	1	System Control and Status Register
WDCNTR	0x00 7023	1	Watchdog Counter Register
WDKEY	0x00 7025	1	Watchdog Reset Key Register
WDCR	0x00 7029	1	Watchdog Control Register
PLL2CTL	0x00 7030	1	PLL2 Configuration Register
PLL2MULT	0x00 7032	1	PLL2 Multiplier Register
PLL2STS	0x00 7034	1	PLL2 Lock Status Register
SYSCLK2CNTR	0x00 7036	1	SYSCLK2 Clock Counter Register
EPWMCFG	0x00 703A	1	ePWM DMA/CLA Configuration Register

1.3.1.1 Enabling/Disabling Clocks to the Peripheral Modules

The PCLKCR0/1/3 registers enable/disable clocks to the various peripheral modules. There is a 2-SYSCLKOUT 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	14	13	12	11	10	9	8
Reserved	ECANAENCLK	Reserved	MCBSPAENCLK	SCIBENCLK	SCIAENCLK	SPIBENCLK	SPIAENCLK
R-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7-5	4	3	2	1	0
Reserved	I2CAENCLK	ADCENCLK	TBCLKSYNC	Reserved	HRPWMENCLK
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	Reserved		Any writes to these bits(s) must always have a value of 0.
14	ECANAENCLK		ECAN-A clock enable
		0	The eCAN-A module is not clocked. (default) [(1)]
		1	The eCAN-A module is clocked (SYSCLKOUT/2).
13	Reserved		Any writes to these bits(s) must always have a value of 0.
12	MCBSPAENCLK		MCBSP clock enable
		0	The McBSP module is not clocked.
		1	The McBSP module is clocked.
11	SCIBENCLK		SCI-B clock enable
		0	The SCI-B module is not clocked.
		1	The SCI-B module is clocked.
10	SCIAENCLK		SCI-A clock enable
		0	The SCI-A module is not clocked. (default) [(1)]
		1	The SCI-A module is clocked by the low-speed clock (LSPCLK).
9	SPIBENCLK		SPI-B clock enable
		0	The SPI-B module is not clocked. (default) [(1)]
		1	The SPI-B module is clocked by the low-speed clock (LSPCLK).
8	SPIAENCLK		SPI-A clock enable
		0	The SPI-A module is not clocked. (default) [(2)]
		1	The SPI-A module is clocked by the low-speed clock (LSPCLK).
7-5	Reserved		Any writes to these bits(s) must always have a value of 0.
4	I2CAENCLK		I^2C clock enable
		0	The I^2C module is not clocked. (default)[(2)]
		1	The I^2C module is clocked.
3	ADCENCLK		ADC clock enable
		0	The ADC is not clocked. (default) [(2)]
		1	The ADC module is clocked
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	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: • Enable ePWM module clocks in the PCLKCR1 register. • Set TBCLKSYNC to 0. • Configure prescaler values and ePWM modes. • Set TBCLKSYNC to 1.
1	Reserved		Any writes to these bits(s) must always have a value of 0.
0	HRPWMENCLK		HRPWM clock enable
		0	HRPWM is not enabled.
		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.
(2) 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	14	13	11	10	9	8
EQEP2ENCLK	EQEP1ENCLK	Reserved		ECAP3ENCLK	ECAP2ENCLK	ECAP1ENCLK
R/W-0	R/W-0	R-0		R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
EPWM8ENCLK	EPWM7ENCLK	EPWM6ENCLK	EPWM5ENCLK	EPWM4ENCLK	EPWM3ENCLK	EPWM2ENCLK	EPWM1ENCLK
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 1-16. Peripheral Clock Control 1 Register (PCLKCR1) Field Descriptions

Bits	Field	Value	Description [(1)]
15	EQEP2ENCLK		eQEP2 clock enable
		0	The eQEP2 module is not clocked. (default) [(2)]
		1	The eQEP2 module is clocked by the system clock (SYSCLKOUT).
14	EQEP1ENCLK		eQEP1 clock enable
		0	The eQEP1 module is not clocked. (default) [(2)]
		1	The eQEP1 module is clocked by the system clock (SYSCLKOUT).
13-11	Reserved		Any writes to these bits(s) must always have a value of 0.
10	ECAP3ENCLK		eCAP3 clock enable
		0	The eCAP3 module is not clocked. (default) [(2)]
		1	The eCAP3 module is clocked by the system clock (SYSCLKOUT).
9	ECAP2ENCLK		eCAP2 clock enable
		0	The eCAP2 module is not clocked. (default) [(2)]
		1	The eCAP2 module is clocked by the system clock (SYSCLKOUT).
8	ECAP1ENCLK		eCAP1 clock enable
		0	The eCAP1 module is not clocked. (default) [(2)]
		1	The eCAP1 module is clocked by the system clock (SYSCLKOUT).
7	EPWM8ENCLK		ePWM8 clock enable. [(3)]
		0	The ePWM8 module is not clocked. (default) [(2)]
		1	The ePWM8 module is clocked by the system clock (SYSCLKOUT).
6	EPWM7ENCLK		ePWM7 clock enable. [(3)]
		0	The ePWM7 module is not clocked. (default) [(2)]
		1	The ePWM7 module is clocked by the system clock (SYSCLKOUT).
5	EPWM6ENCLK		ePWM6 clock enable. [(3)]
		0	The ePWM6 module is not clocked. (default) [(2)]
		1	The ePWM6 module is clocked by the system clock (SYSCLKOUT).
4	EPWM5ENCLK		ePWM5 clock enable [(3)]
		0	The ePWM5 module is not clocked. (default) [(2)]
		1	The ePWM5 module is clocked by the system clock (SYSCLKOUT).
3	EPWM4ENCLK		ePWM4 clock enable. [(3)]
		0	The ePWM4 module is not clocked. (default) [(2)]
		1	The ePWM4 module is clocked by the system clock (SYSCLKOUT).
2	EPWM3ENCLK		ePWM3 clock enable. [(3)]
		0	The ePWM3 module is not clocked. (default) [(2)]
		1	The ePWM3 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.

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

Bits	Field	Value	Description [1]
1	EPWM2ENCLK		ePWM2 clock enable. [3]
		0	The ePWM2 module is not clocked. (default) [2]
		1	The ePWM2 module is clocked by the system clock (SYSCLKOUT).
0	EPWM1ENCLK		ePWM1 clock enable. [3]
		0	The ePWM1 module is not clocked. (default) [2]
		1	The ePWM1 module is clocked by the system clock (SYSCLKOUT).

Figure 1-16. Peripheral Clock Control 2 Register (PCLKCR2)

15–12	11	10	9	8
Reserved	HRCAP4ENCLK	HRCAP3ENCLK	HRCAP2ENCLK	HRCAP1ENCLK
R-0	R/W-0	R/W-0	R/W-0	R/W-0

7–0
Reserved
R-0

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

Table 1-17. Peripheral Clock Control 2 Register (PCLKCR2) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved	0	Any writes to these bits(s) must always have a value of 0.
11	HRCAP4ENCLK	0	The HRCAP4 module is not clocked. (default) [1]
		1	The HRCAP4 module is clocked by the system clock (SYSCLKOUT).
10	HRCAP3ENCLK	0	The HRCAP3 module is not clocked. (default) [1]
		1	The HRCAP3 module is clocked by the system clock (SYSCLKOUT).
9	HRCAP2ENCLK	0	The HRCAP2 module is not clocked. (default) [1]
		1	The HRCAP2 module is clocked by the system clock (SYSCLKOUT).
8	HRCAP1ENCLK	0	The HRCAP1 module is not clocked. (default)[1]
		1	The HRCAP1 module is clocked by the system clock (SYSCLKOUT).
0-7	Reserved		Any writes to these bits(s) must always have a value of 0.

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

Figure 1-17. Peripheral Clock Control 3 Register (PCLKCR3)

15	14	13	12	11	10	9	8
USB0ENCLK	CLA1ENCLK	Reserved	Reserved	DMAENCLK	CPUTIMER2ENCLK	CPUTIMER1ENCLK	CPUTIMER0ENCLK
R-0	R/W-0	R-1	R-0	R/W-0	R/W-1	R/W-1	R/W-1

7–3	2	1	0
Reserved	COMP3ENCLK	COMP2ENCLK	COMP1ENCLK
R-0	R/W-0	R/W-0	R/W-0

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

Table 1-18. Peripheral Clock Control 3 Register (PCLKCR3) Field Descriptions

Bit	Field	Value	Description
15	USB0ENCLK		USB module clock enable
		0	Clock is disabled
		1	Clock is enabled
14	CLA1ENCLK		CLA module clock enable
		0	CLA is not clocked
		1	CLA is clocked
13	Reserved		Reserved
12	Reserved		Any writes to these bits(s) must always have a value of 0.
11	DMAENCLK		DMA module clock enable
		0	DMA is not clocked
		1	DMA is clocked
10	CPUTIMER2ENCLK		CPU Timer 2 Clock Enable
		0	The CPU Timer 2 is not clocked.
		1	The CPU Timer 2 is clocked.
9	CPUTIMER1ENCLK		CPU Timer 1 Clock Enable
		0	The CPU Timer 1 is not clocked.
		1	The CPU Timer 1 is clocked.
8	CPUTIMER0ENCLK		CPU Timer 0 Clock Enable
		0	The CPU Timer 0 is not clocked.
		1	The CPU Timer 0 is clocked.
7:3	Reserved		Any writes to these bits(s) must always have a value of 0.
2	COMP3ENCLK		Comparator3 clock enable
		0	Comparator3 is not clocked
		1	Comparator3 is clocked
1	COMP2ENCLK		Comparator2 clock enable
		0	Comparator2 is not clocked
		1	Comparator2 is clocked
0	COMP1ENCLK		Comparator1 clock enable
		0	Comparator1 is not clocked
		1	Comparator1 is clocked

1.3.1.2 Configuring the Low-Speed Peripheral Clock Prescaler

The low-speed peripheral clock prescale (LOSPCP) registers are used to configure the low-speed peripheral clocks. See Figure 1-18 for the LOSPCP layout.

Figure 1-18. 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 1-19. Low-Speed Peripheral Clock Prescaler Register (LOSPCP) Field Descriptions

Bits	Field	Value	Description [(1)]
15-3	Reserved		Reserved
2-0	LSPCLK		These bits configure the low-speed peripheral clock (LSPCLK) rate relative to SYSCLKOUT:
			If LOSPCP[(2)] ≠ 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

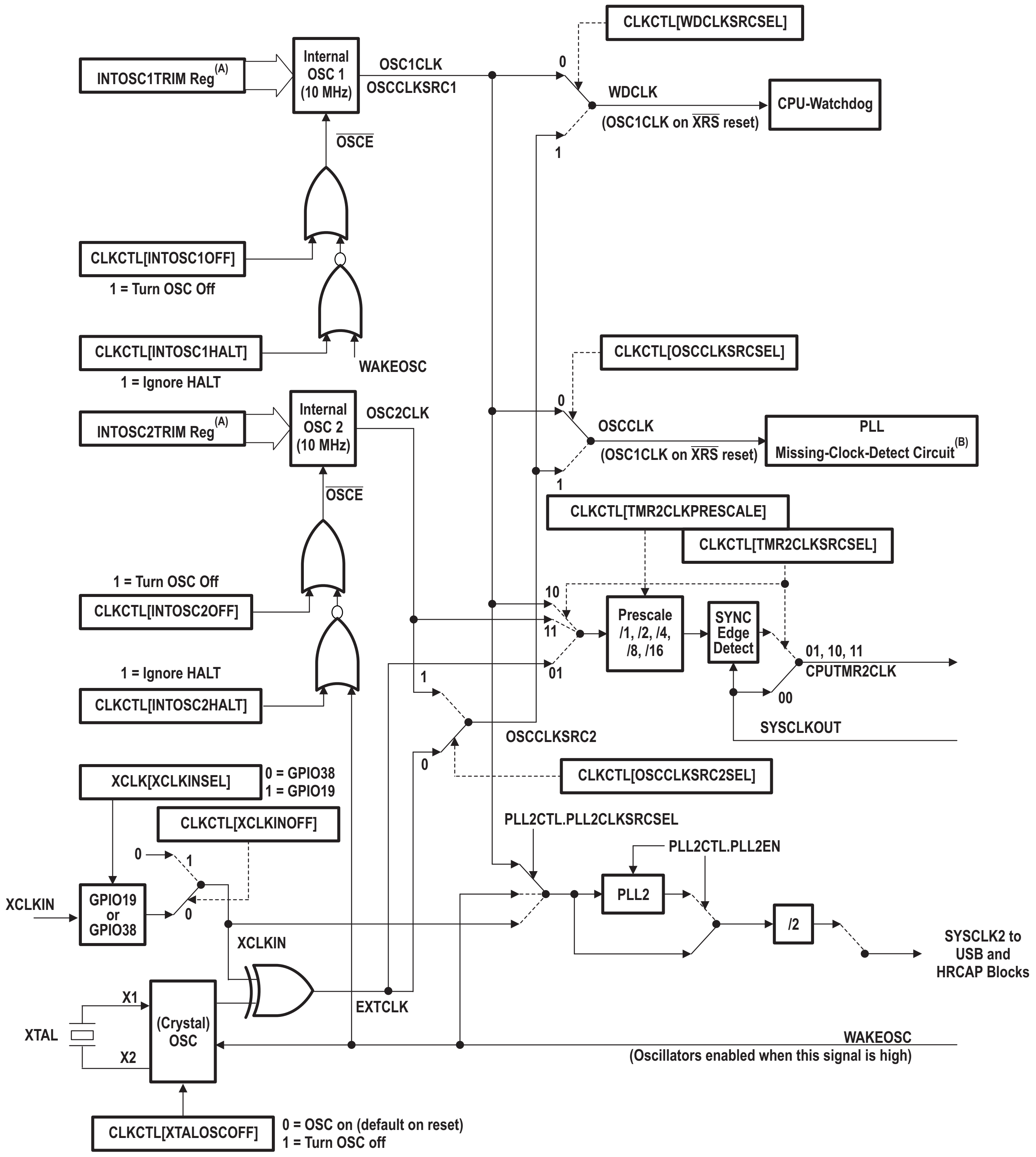
The on-chip oscillator and phase-locked loop (PLL) block provide the clocking signals for the device, as well as control for low-power mode (LPM) entry or exit.

1.3.2.1 Input Clock Options

The device has two internal oscillators (INTOSC1 and INTOSC2) that need no external components. They also have an on-chip, PLL-based clock module. Figure 1-19 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. This can provide the clock for the Watchdog block, 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. This can provide the clock for the Watchdog block, core and CPU-Timer 2. Both INTOSC1 and INTOSC2 can be independently chosen for the Watchdog block, core and CPU-Timer 2.
- **Crystal/Resonator Operation:** The on-chip crystal oscillator enables the use of an external crystal/resonator attached to the device to provide the time base. The crystal/resonator is connected to the X1/X2 pins.
- **External clock source operation:** 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 at boot time.

Figure 1-19. Clocking Options



A Register loaded from TI OTP-based calibration function.

B See the device-specific datasheet for details on missing clock detection.

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 provide you with a way to trim the oscillator using software (called fine trim). Both registers are the same so only one is shown with "n" in place of the numbers 1 or 2.

Figure 1-20. 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 1-20. Internal Oscillator n Trim (INTOSCnTRIM) Register Field Descriptions

Bit	Field	Value	Description[1]
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 bit(s) must always have a value of 0.
7-0	COARSETRIM		8-bit Coarse Trim Value: Signed magnitude value (- 127 to + 127)

[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.

1.3.2.1.2 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 the *C2806x C/C++ Header Files and Peripheral Examples*.

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 2-5. 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 2-5. 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

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

Figure 1-21. Clocking (XCLK) Register

15–8
Reserved
R-0

7	6	5–2	1–0
Reserved	XCLKINSEL	Reserved	XCLKOUTDIV
R-0	R/W-0	R-0	R/W-0

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

Table 1-21. Clocking (XCLK) Field Descriptions

Bit	Field	Value	Description[1]
15-7	Reserved		Any writes to these bits(s) must always have a value of 0.
6	XCLKINSEL		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 bits(s) must always have a value of 0.
1-0	XCLKOUTDIV[2]		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 $\overline{\text{XRS}}$ input signal.
[2] Refer to the device datasheet for the maximum permissible XCLKOUT frequency.

1.3.2.3 Configuring Device Clock Domains

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

Figure 1-22. 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 1-22. Clock Control (CLKCTL) Register Field Descriptions

Bit	Field	Value	Description
15	NMIRESETSEL		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 Watcdog Reset ($\overline{\text{NMIRS}}$) initiates $\overline{\text{MCLKRS}}$
			Note: The $\overline{\text{CLOCKFAIL}}$ signal is generated regardless of this mode selection.

Table 1-22. Clock Control (CLKCTL) Register Field Descriptions (continued)

Bit	Field	Value	Description
14	XTALOSCOFF		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		XCLKIN Off Bit: This bit turns external XCLKIN oscillator input off:
		0	XCLKIN oscillator input on (default on reset)
		1	XCLKIN oscillator input off
			Note: 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		Watchdog HALT Mode Ignore Bit: This bit selects if the watchdog is automatically turned on/off by the HALT mode or not. This feature can be used to allow the selected WDCLK 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 On/Off By HALT (default on reset)
		1	Watchdog Ignores HALT Mode
11	INTOSC2HALTI		Internal Oscillator 2 HALT Mode Ignore Bit: This bit selects if the internal oscillator 2 is automatically turned on/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 On/Off By HALT (default on reset)
		1	Internal Oscillator 2 Ignores HALT Mode
10	INTOSC2OFF		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		Internal Oscillator 1 HALT Mode Ignore Bit: This bit selects if the internal oscillator 1 is automatically turned on/off by the HALT mode or not:
		0	Internal Oscillator 1 Automatically Turned On/Off By HALT (default on reset)
		1	Internal Oscillator 1 Ignores 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		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		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.
		000	/1 (default on reset)
		001	/2
		010	/4
		011	/8
		100	/16
		101	Reserved
		110	Reserved
		111	Reserved
4-3	TMR2CLKSRCSEL		CPU Timer 2 Clock Source Select Bit: This bit selects the source for CPU Timer 2:
		00	SYSCLKOUT 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.

Table 1-22. Clock Control (CLKCTL) Register Field Descriptions (continued)

Bit	Field	Value	Description
2	WDCLKSRCSEL		Watchdog Clock Source Select Bit: This bit selects the source for WDCLK. 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		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
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 divisor ratio. 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.

1.3.2.3.1 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 will be issued to exit Limp Mode.

If External Oscillator or XCLKIN or Internal Oscillator 2 (OSCCLKSRC2) is selected 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.OSCCLKLSRCSEL=1 and CLKCTL.OSCCLKSRC2SEL=1

The second write should not alter the values of XTALOSCOFF and XCLKINOFF bits. If the DSP28 header files (SPRC823) supplied by Texas Intruments are used, clock switching can be achieved with the following code snip:

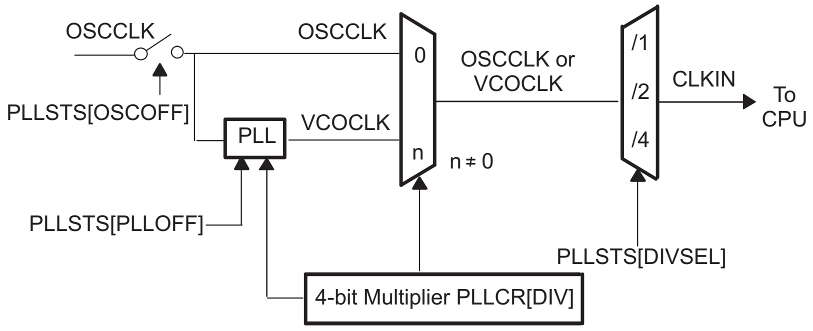
```
SysCtrlRegs.CLKCTL.all = 0x6000; // Set XTALOSCOFF=1 & XCLKINOFF=1
SysCtrlRegs.CLKCTL.all = 0x6003; // Set OSCCLKLSRCSEL=1 & OSCCLKSRC2SEL=1
```

The system initialization file (F2806x_SysCtrl.c) provided as part of the header files 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

This device has two PLL modules. “PLL” refers to the main PLL that generats the clock for the core and all peripherals. PLL2 refers to the PLL that generates the clock for the USB and HRCAP modules. The figure below shows the OSC and PLL block diagram.

Figure 1-23. OSC and PLL Block



The following is applicable for devices that have X1 and X2 pins:

When using XCLKIN as the external clock source, you must tie X1 low and leave X2 disconnected.

Table 1-23. Possible PLL Configuration Modes

PLL Mode	Remarks	PLLSTS[DIVSEL][(1)]	CLKIN and SYSCLKOUT[(2)]
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, X1 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 ($\overline{\text{XRS}}$). 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 1-24.

(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.

1.3.2.5 PLL Control (PLLCR) Register

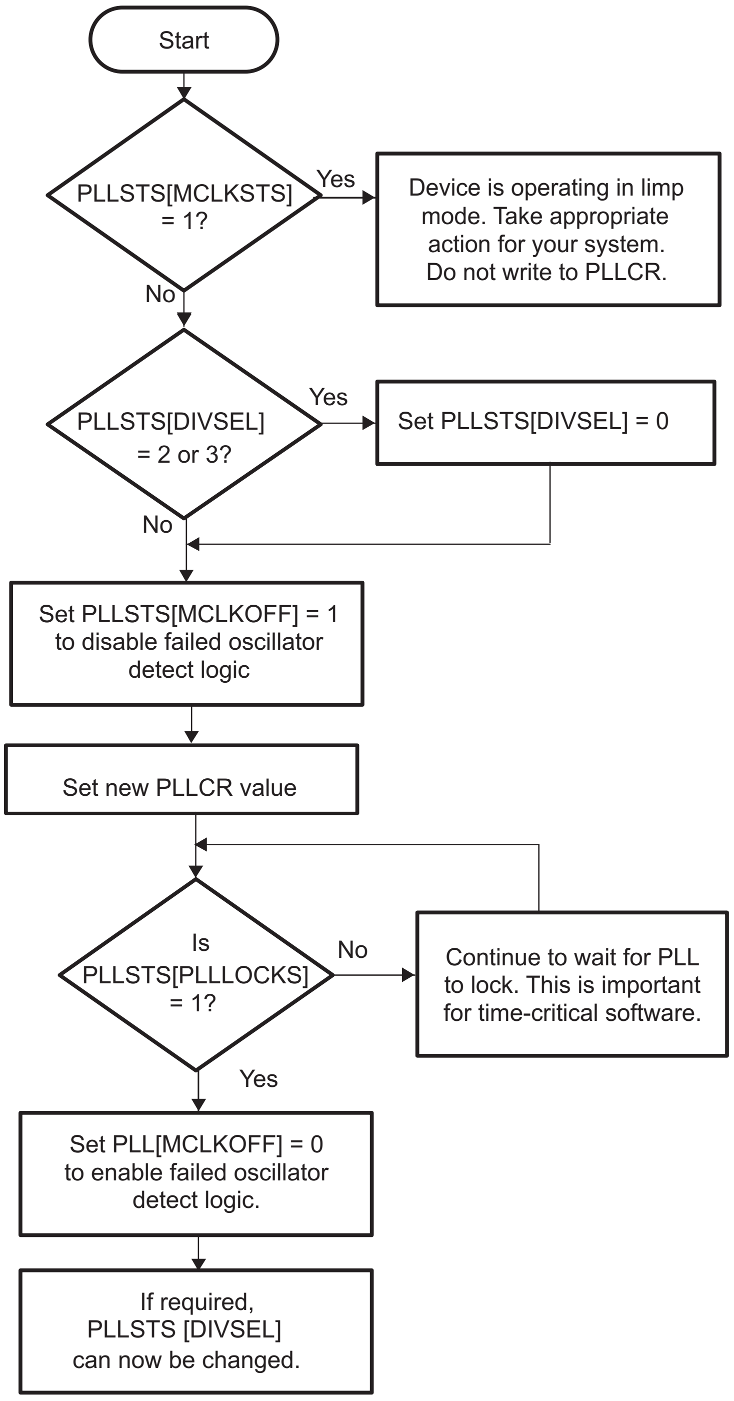
The PLLCR register is used to change the PLL multiplier of the device. Before writing to the PLLCR register, the following requirements must be met:

- 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 1-24. 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 1-24 any time you are writing to the PLLCR register.

Figure 1-24. PLLCR Change Procedure Flow Chart


Start
PLLSTS[MCLKSTS] = 1?
Yes
Device is operating in limp mode. Take appropriate action for your system. Do not write to PLLCR.
No
PLLSTS[DIVSEL] = 2 or 3?
Yes
Set PLLSTS[DIVSEL] = 0
No
Set PLLSTS[MCLKOFF] = 1 to disable failed oscillator detect logic
Set new PLLCR value
Is PLLSTS[PLLLOCKS] = 1?
No
Continue to wait for PLL to lock. This is important for time-critical software.
Yes
Set PLL[MCLKOFF] = 0 to enable failed oscillator detect logic.
If required, PLLSTS [DIVSEL] can now be changed.

1.3.2.6 PLL Control, Status and XCLKOUT Register Descriptions

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 1-24.

Figure 1-25. PLLCR Register Layout

15 — 5	4 — 0
Reserved	DIV
R-0	R/W-0

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

Table 1-24. PLL Settings[(1)]

	SYSCLKOUT (CLKIN)[(2)]		
PLLCR[DIV] Value[(3)]	PLLSTS[DIVSEL] = 0 or 1	PLLSTS[DIVSEL] = 2	PLLSTS[DIVSEL] = 3
0000 (PLL bypass)	OSCCLK/4 (Default)	OSCCLK/2	OSCCLK/1
00001	(OSCCLK * 1)/4	(OSCCLK * 1)/2	(OSCCLK * 1)/1
00010	(OSCCLK * 2)/4	(OSCCLK * 2)/2	(OSCCLK * 2)/1
00011	(OSCCLK * 3)/4	(OSCCLK * 3)/2	(OSCCLK * 3)/1
00100	(OSCCLK * 4)/4	(OSCCLK * 4)/2	(OSCCLK * 4)/1
00101	(OSCCLK * 5)/4	(OSCCLK * 5)/2	(OSCCLK * 5)/1
00110	(OSCCLK * 6)/4	(OSCCLK * 6)/2	(OSCCLK * 6)/1
00111	(OSCCLK * 7)/4	(OSCCLK * 7)/2	(OSCCLK * 7)/1
01000	(OSCCLK * 8)/4	(OSCCLK * 8)/2	(OSCCLK * 8)/1
01001	(OSCCLK * 9)/4	(OSCCLK * 9)/2	(OSCCLK * 9)/1
01010	(OSCCLK * 10)/4	(OSCCLK * 10)/2	(OSCCLK * 10)/1
01011	(OSCCLK * 11)/4	(OSCCLK * 11)/2	(OSCCLK * 11)/1
01100	(OSCCLK * 12)/4	(OSCCLK * 12)/2	(OSCCLK * 12)/1
01101	(OSCCLK * 13)/4	(OSCCLK * 13)/2	(OSCCLK * 13)/1
01110	(OSCCLK * 14)/4	(OSCCLK * 14)/2	(OSCCLK * 14)/1
01111	(OSCCLK * 15)/4	(OSCCLK * 15)/2	(OSCCLK * 15)/1
10000	(OSCCLK * 16)/4	(OSCCLK * 16)/2	(OSCCLK * 16)/1
10001	(OSCCLK * 17)/4	(OSCCLK * 17)/2	(OSCCLK * 17)/1
10010	(OSCCLK * 18)/4	(OSCCLK * 18)/2	(OSCCLK * 18)/1
10011-11111	Reserved	Reserved	Reserved

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

(2) PLLSTS[DIVSEL] must be 0 or 1 before writing to the PLLCR and should be changed only after PLLSTS[PLLLOCKS] = 1. See Figure 1-24.

(3) The PLL control register (PLLCR) and PLL Status Register (PLLSTS) are reset to their default state by the $\overline{\text{XRS}}$ signal or a watchdog reset only. A reset issued by the debugger or the missing clock detect logic have no effect.

Figure 1-26. PLL Status Register (PLLSTS)

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 1-25. PLL Status Register (PLLSTS) Field Descriptions

Bits	Field	Value	Description [(1)] [(2)]
15	NORMRDYE		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:
		0	NORMRDY signal from VREG does not gate PLL (PLL ignores NORMRDY)
		1	NORMRDY signal from VREG will gate PLL (PLL off when NORMRDY low)
			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.
14-9	Reserved		Any writes to these bits(s) must always have a value of 0.
8:7	DIVSEL		Divide Select: This bit selects between /4, /2, and /1 for CLKIN to the CPU. The configuration of the DIVSEL bit is as follows:
		00, 01	Select Divide By 4 for CLKIN
		10	Select Divide By 2 for CLKIN
		11	Select Divide By 1 for CLKIN
6	MCLKOFF		Missing clock-detect off bit
		0	Main oscillator fail-detect logic is enabled. (default)
		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.
5	OSCOFF		Oscillator Clock Off Bit
		0	The OSCCLK signal from X1/X2 or XCLKIN is fed to the PLL block. (default)
		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.
			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.
			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: • X1/X2: The watchdog is not functional. • XCLKIN: The watchdog is functional and should be disabled before setting OSCOFF.
4	MCLKCLR		Missing Clock Clear Bit.
		0	Writing a 0 has no effect. This bit always reads 0.
		1	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		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.
		0	Indicates normal operation. A missing clock condition has not been detected.
		1	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		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.
		0	PLL On (default)
		1	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 bits(s) must always have a value of 0.

(1) This register is reset to its default state only by the $\overline{\text{XRS}}$ 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.

Table 1-25. PLL Status Register (PLLSTS) Field Descriptions (continued)

Bits	Field	Value	Description [(1)] [(2)]
0	PLLLOCKS		PLL Lock Status Bit.
		0	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.
		1	Indicates that the PLL has finished locking and is now stable.

Figure 1-27. PLL Lock Period (PLLLOCKPRD) Register

15 … 0
PLLLOCKPRD
R/W-FFFFh

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

Table 1-26. PLL Lock Period (PLLLOCKPRD) Register Field Descriptions

Bit	Field	Value	Description[(1)] [(2)]
15:0	PLLLOCKPRD		PLL Lock Counter Period Value
			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.
			PLL Lock Period
		FFFFh	65535 OSCLK Cycles (default on reset)
		FFFEh	65534 OSCLK Cycles
		...	...
		0001h	1 OSCCLK Cycles
		0000h	0 OSCCLK Cycles (no PLL lock period)

(1) PLLLOCKPRD is affected by $\overline{\text{XRS}}$n signal only.
(2) This register is EALLOW protected. See Section 1.5.2 for more information.

1.3.2.7 PLL2 Registers

In addition to the main PLL that clocks the CPU, a second PLL (PLL2) exists for clocking the USB and HRCAP modules. The figure below shows the possible input and output configurations for PLL2.

Figure 1-28. PLL2 Input and Output Configurations

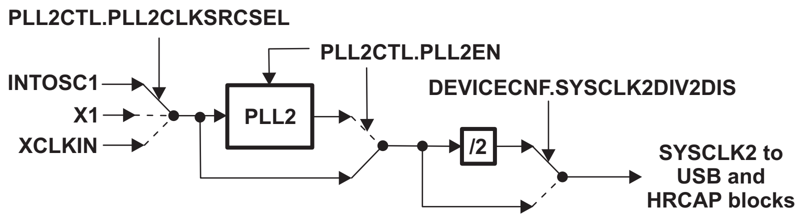


Figure 1-29. PLL2 Configuration (PLL2CTL) Register (EALLOW protected)

15–3	2	1–0
Reserved	PLL2EN	PLL2CLKSRCSEL
R-0	R/W-1	R/W-00

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

Table 1-27. PLL2 Configuration (PLL2CTL) Register Field Descriptions

Bit	Field	Value	Description
15-3	Reserved		Any writes to these bits(s) must always have a value of 0.
2	PLL2EN		PLL enabled or disabled: This bit decides if PLL2 is enabled or not
		0	PLL2 is powered off – clock to SYSCLK2 is a direct feed from input clock source as decided by the PLL2CLKSRCSEL bit
		1	PLL2 is enabled and clock to SYSCLK2 will depend on the DEVICECNF[SYSCLK2DIV2DIS] bit.
1-0	PLL2CLKSRCSEL		PLL2 Clock Source Select Bits: These bit select the source for the PLL2 input clock:
		00	Internal oscillator 1 is selected as clock to PLL2
		01	Internal oscillator 1 is selected as clock to PLL2
		10	X1 clock source is selected as clock to PLL2
		11	GPIO_XCLKIN is selected as clock to PLL2
			On $\overline{\text{XRS}}$ low and after $\overline{\text{XRS}}$ goes high, X1 is selected as clock source to the USB PLL by default. The user would need to select X1 or GPIO_XCLKIN as clock source during their initialization process.
			Whenever the user changes the clock source using these bits, the DEVICECNF[SYSCLK2DIV2DIS] bit will be automatically forced to zero. This prevents potential PLL overshoot. The user will then have to write to the DEVICECNF[SYSCLK2DIV2DIS] bit to configure the appropriate divisor ratio.
Note: PLL2CTL is affected by $\overline{\text{XRS}}$ signal only.			

Figure 1-30. PLL2 Multiplier (PLL2MULT) Register (EALLOW protected)

15–4	3–0
Reserved	PLL2MULT
R-0	R/W-0x0

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

Table 1-28. PLL2 Multiplier (PLL2MULT) Register Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Any writes to these bits(s) must always have a value of 0.
3-0	PLL2MULT		PLL2 Multiplier. This bit field determines the output frequency of PLL2 ($PLL2F_{out}$) for a given input ($PLL2F_{in}$) 0000 $PLL2F_{out} = PLL2F_{in}$ (PLLBYPASS) 0001 $PLL2F_{out} = PLL2F_{in} * 1$ 0010 $PLL2F_{out} = PLL2F_{in} * 2$ 0011 $PLL2F_{out} = PLL2F_{in} * 3$ ... ... 1111 $PLL2F_{out} = PLL2F_{in} * 15$ PLL2 should be enabled (PLL2EN = 1) prior to setting these bits.
Note: PLL2MULT is affected by $\overline{\text{XRS}}$n signal only.			

Figure 1-31. PLL2 Lock Status (PLL2STS) Register

15 ... 1	0
Reserved	PLL2LOCKS
R-0	R-0x0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset **Note:** PLL2STS is affected by XRnS signal only.

Table 1-29. PLL2 Lock Status (PLL2STS) Register Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Any writes to these bits(s) must always have a value of 0.
3-0	PLL2MULT		PLL2 Lock Status Bit: This bit indicates whether PLL2 is locked or not. When the PLL2MULT setting is modified, a counter is loaded with the value from the main PLL PLLLOCKPRD bit field and the PLL2LOCKS bit is cleared. Once set to a non-zero value, the counter begins down-counting. Upon reaching zero, the counter stops and the PLL2LOCKS bit is set.
		0	PLL2 is not yet locked
		1	PLL2 is locked
Note: PLL2STS is affected by $\overline{\text{XRS}}$n signal only.			

Figure 1-32. SYSCLK2 Clock Counter (SYSCLK2CNTR) Register

15 ... 0
COUNT
R/C-0x0

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

Table 1-30. SYSCLK2 Clock Counter (SYSCLK2CNTR) Register Field Descriptions

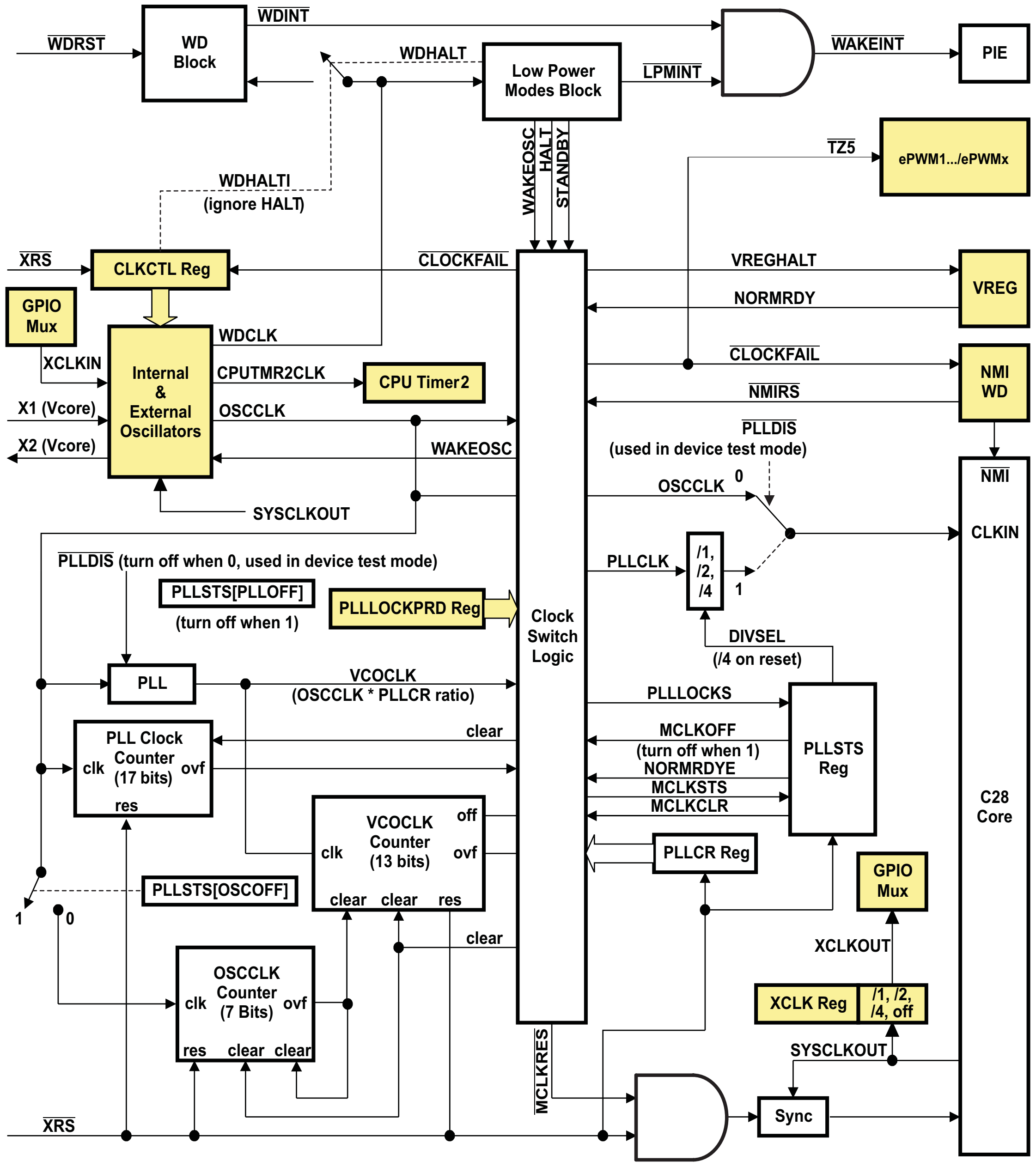
Bit	Field	Value	Description
15-0	COUNT		SYSCLK2 Counter Bit Field: This bit field is a free-running counter based off the SYSCLK2 clock. Software can compare the rate of update of this bit field against the rate of update of a CPU Timer to determine at what approximate frequency the SYSCLK2 clock is running.
Note 1: Since COUNT will begin updating as soon as reset is released, the value of this register after reset will be non-zero before software can read it. **Note 2:** SYSCLK2CNTR will tap off the clock after the /2 from PLL2OUT if it is enabled. If not enabled it will tap off PLL2OUT.			

1.3.2.8 Input Clock Fail Detection

It is possible for the clock source (internal or external) of the DSP to fail. When the PLL is not disabled, the main oscillator fail logic allows the device to detect this condition and default to a known state as described in this section.

Two counters are used to monitor the presence of the OSCCLK signal as shown in Figure 1-33. 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.

Figure 1-33. Oscillator Logic Diagram



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 and, if required, the device switches the CLKIN input to the CPU to the limp mode output frequency of the PLL.

When the VCOCLK counter overflows, the missing clock detection logic resets the CPU, peripherals, and other device logic. The reset generated is known as a missing clock detect logic reset ($\overline{\text{MCLKRS}}$). 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. This is the default behavior at reset.

In addition to resetting the device, the missing oscillator logic sets the PLLSTS[MCLKSTS] register bit. When the MCLKCSTS 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 1-24 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 1-24.
- **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 by-passed. 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, set the missing clock detect status bit, and generate a missing clock reset. The device will now run at the PLL limp mode frequency or one-half of the PLL limp mode frequency.
- **PLL enabled mode**

  When the PLL control register is non-zero (PLLCR = n, where n ≠ 0x0000), the PLL is enabled. In this mode, OSCCLK*n, OSCCLK*n/2, or OSCCLK*n/4 is connected to CLKIN of the CPU. If OSCCLK is detected as missing, the missing clock detect status bit will be set and the device will generate a missing clock reset. The device will now run at one-half of the PLL limp mode frequency.

- **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.

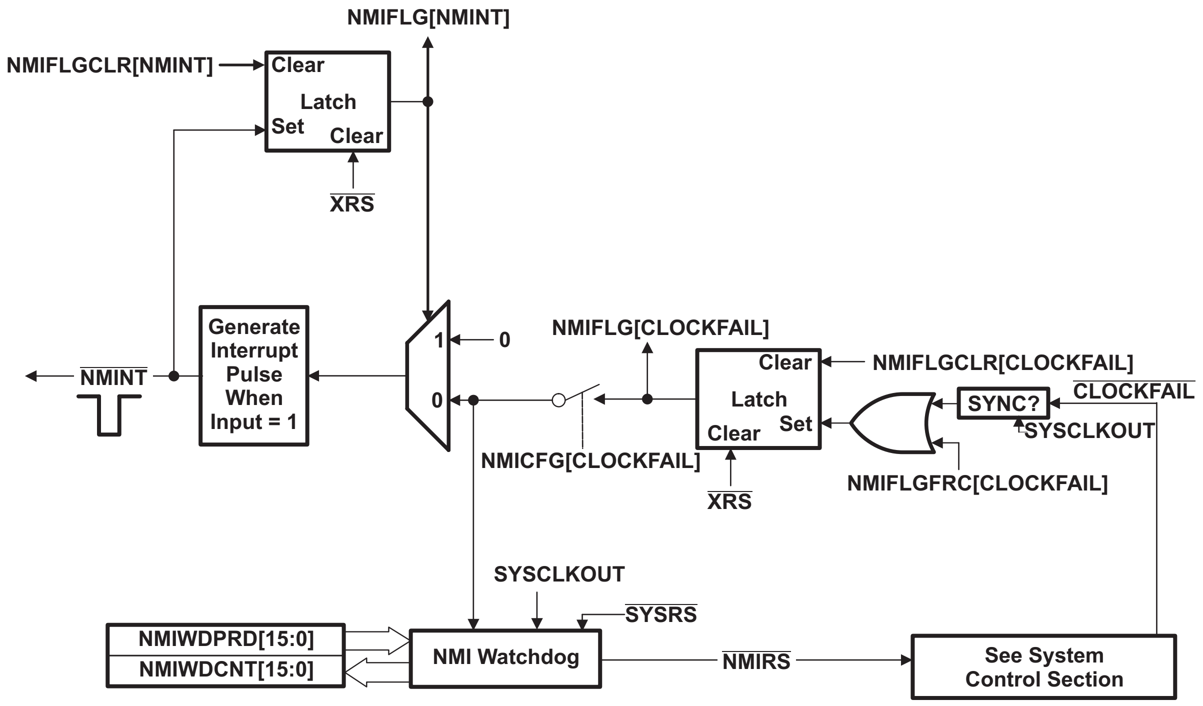
1.3.2.9 NMI Interrupt and Watchdog

The NMI watchdog (NMIWD) is used to detect and aid in recovery from a clock failure condition. The NMI interrupt enables the monitoring of the erroneous CLOCKFAIL condition in the system. In 280x/2833x/2823x devices, when a missing clock is detected, a missing-clock-reset ($\overline{\text{MCLKRS}}$) is generated immediately. In Piccolo devices however, a CLOCKFAIL signal can be generated first, which is then fed to the NMI Watchdog circuit and a reset is generated after a preprogrammed delay. This feature is not enabled upon power-up, however. That is, when Piccolo first powers up, the $\overline{\text{MCLKRS}}$ signal is generated immediately upon clock failure like other 28xx devices. The user must enable the generation of the CLOCKFAIL signal via the CLKCTL[NMIRESETSEL] bit. Note that the NMI watchdog is different from the watchdog described in Section 1.3.4.

When the OSCCLK goes missing, the CLOCKFAIL signal triggers the NMI and gets the NMIWD counter running. In the NMI ISR, the application is expected to take corrective action (such as switch to an alternate clock source) and clear the CLOCKFAIL and NMIINT flags. If this is not done, the NMIWDCTR overflows and generates an NMI reset ($\overline{\text{NMIRS}}$) after a pre-programmed number of SYSCLKOUT cycles. $\overline{\text{NMIRS}}$ is fed to $\overline{\text{MCLKRS}}$ to generate a system reset back into the core. The purpose of this is to allow software to gracefully shut down the system before a reset is generated. Note that NMI reset will not be reflected on the $\overline{\text{XRS}}$ pin and is internal to the device.

The CLOCKFAIL signal could also be used to activate the TZ5 signal to drive the PWM pins into a high impedance state. This allows the PWM outputs to b tripped in case of clock failure. Figure 1-34 shows the CLOCKFAIL interrupt mechanism.

Figure 1-34. Clock Fail Interrupt



A The NMI watchdog module is clocked by SYSCLKOUT. Due to the limp mode function of the PLL, SYSCLKOUT is present even if the source clock for OSCCLK fails.

The NMI Interrupt support registers are listed in Table 1-31.

Table 1-31. NMI Interrupt Registers

Name	Address Range	Size (x16)	EALLOW	Description
NMICFG	0x7060	1	yes	NMI Configuration Register
NMIFLG	0x7061	1	yes	NMI Flag Register
NMIFLGCLR	0x7062	1	yes	NMI Flag Clear Register
NMIFLGFRC	0x7063	1	yes	NMI Flag Force Register
NMIWDCNT	0x7064	1	-	NMI Watchdog Counter Register
NMIWDPRD	0x7065	1	yes	NMI Watchdog Period Register

Table 1-32. NMI Configuration (NMICFG) Register Bit Definitions (EALLOW)

Bits	Name	Type	Description
15:2	Reserved		Any writes to these bits(s) must always have a value of 0.
1	CLOCKFAIL		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 bits(s) must always have a value of 0.

Table 1-33. NMI Flag (NMIFLG) Register Bit Definitions (EALLOW Protected):

Bits	Name	Type	Description
15:2	Reserved		Any writes to these bits(s) must always have a value of 0.
1	CLOCKFAIL		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		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.

Table 1-34. NMI Flag Clear (NMIFLGCLR) Register Bit Definitions (EALLOW Protected)

Bits	Name	Type	Description
15:2	Reserved		Any writes to these bits(s) must always have a value of 0.
1	CLOCKFAIL[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[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.

Table 1-35. NMI Flag Force (NMIFLGFRC) Register Bit Definitions (EALLOW Protected):

Bits	Name	Value	Description
15:02	Reserved		Any writes to these bits(s) must always have a value of 0.
1	CLOCKFAIL		CLOCKFAIL flag force
		0	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 bits(s) must always have a value of 0.

Table 1-36. NMI Watchdog Counter (NMIWDCNT) Register Bit Definitions

Bits	Name	Type	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 $\overline{NMIRS}$ 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 FLAG(s) 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.

Table 1-37. NMI Watchdog Period (NMIWDPRD) Register Bit Definitions (EALLOW Protected)

Bits	Name	Type	Description
15:0	NMIWDPRD	R/W	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. Writing a PERIOD value that is equal to the current counter value automatically forces an $\overline{\text{NMIRS}}$ 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 $\overline{\text{NMIRS}}$ is forced which resets the watchdog counter .

1.3.2.9.1 NMI Watchdog Emulation Considerations

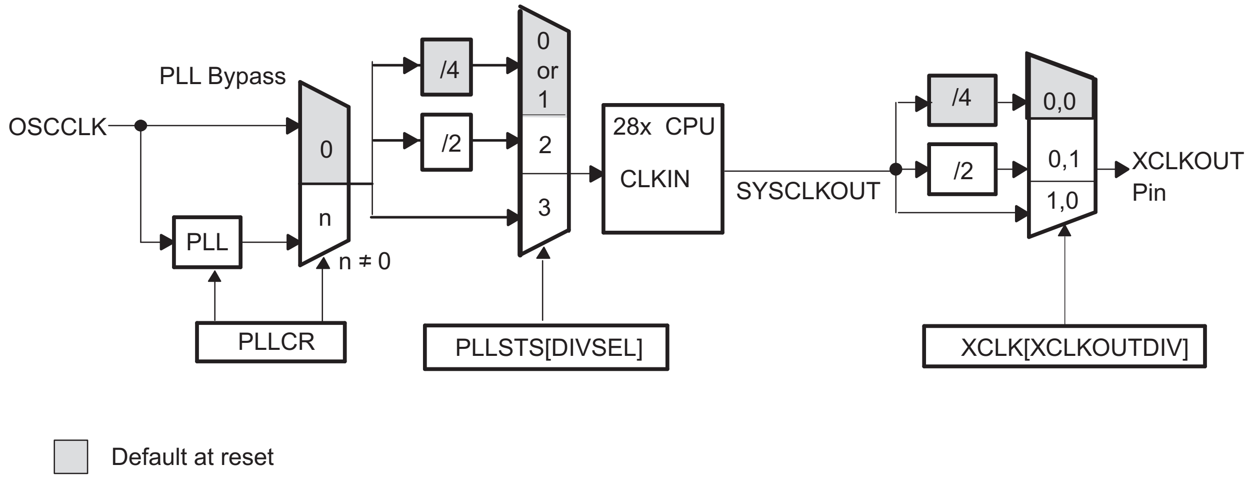
The NMI watchdog module does not operate when trying to debug the target device (emulation suspend such as breakpoint). The NMI watchdog module behaves as follows under various debug conditions:

CPU Suspended:	When the CPU is suspended, the NMI watchdog counter is suspended.
Run-Free Mode:	When the CPU is placed in run-free mode, the NMI watchdog counter resumes operation as normal.
Real-Time Single-Step Mode:	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.
Real-Time Run-Free Mode:	When the CPU is in real-time run-free mode, the NMI watchdog counter operates as normal.

1.3.2.10 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.

Figure 1-35. XCLKOUT Generation



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

1.3.2.11 External Reference Oscillator Clock Option

TI recommends that customers have the resonator/crystal vendor characterize the operation of their device with the DSP 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 1-38 summarizes the various modes.

The various low-power modes operate as shown in Table 1-39.

See the *TMS320F2806x Piccolo Microcontrollers* (literature number SPRS698) for exact timing for entering and exiting the low power modes.

Table 1-38. Low-Power Mode Summary

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

(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 1-39. 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:	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: • Enable the WAKEINT interrupt in the PIE module. This interrupt is connected to both the watchdog and the low power mode module interrupt. • 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 $\overline{\text{XRS}}$ input and the watchdog interrupt, if enabled in the LPMCR0 register, can wake the device from the STANDBY mode. • Select the input qualification in the LPMCR0 register for the signal that will wake the device. 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.

Table 1-39. Low Power Modes (continued)

Mode	Description
HALT Mode:	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: • 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. • 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 $\overline{XRS}$ input can also wake the device from the HALT mode. • 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. 1. For device to exit HALT mode properly, the following conditions must be met: Bit 7 (INT1.8) of PIEIER1 register should be 1. Bit 0 (INT1) of IER register must be 1. 2. If the above conditions are met, (a) WAKE_INT ISR will be executed first, followed by the instruction(s) after IDLE, if INTM = 0. (b) WAKE_INT ISR will not be executed and instruction(s) after IDLE will be executed, if INTM = 1. **Do not enter HALT low power mode when the device is operating in limp mode (PLLSTS[MCLKSTS] = 1).** 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. 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.

The low-power modes are controlled by the LPMCR0 register (Figure 1-36).

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 1-40. Low-Power Mode Control 0 Register (LPMCR0) Field Descriptions

Bits	Field	Value	Description [(1)]
15	WDINTE		Watchdog interrupt enable
		0	The watchdog interrupt is not allowed to wake the device from STANDBY. (default)
		1	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 bits(s) must always have a value of 0.
7-2	QUALSTDBY		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 GPIOLPMSEL register.
		000000	2 OSCCLKs
		000001	3 OSCCLKs
		. . .	. . .
		111111	65 OSCCLKs (default)

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

Table 1-40. Low-Power Mode Control 0 Register (LPMCR0) Field Descriptions (continued)

Bits	Field	Value	Description [1]
1-0	LPM[2]		These bits set the low power mode for the device.
		00	Set the low power mode to IDLE (default)
		01	Set the low power mode to STANDBY
		10	Set the low power mode to HALT
		11	Set the low power mode to HALT

[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.

1.3.3.1 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:

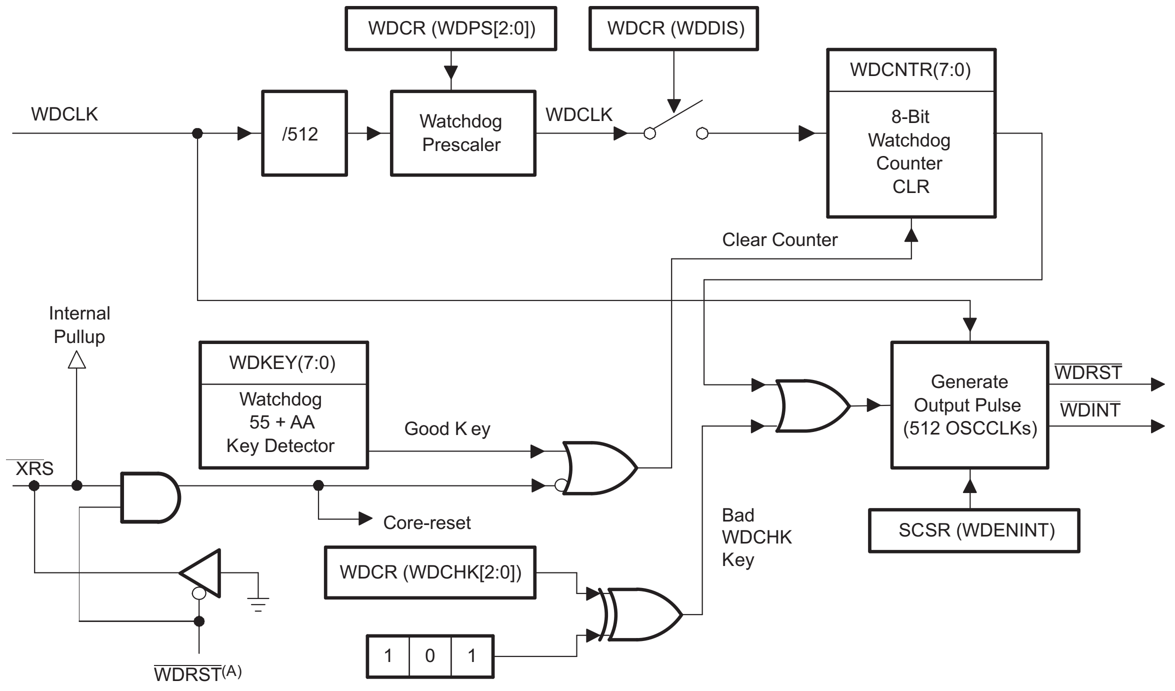
Wakeup from HALT: Set WDHALTI bit in CLKCTL register to 1. When the device wakes up from HALT, it will be through a CPU-watchdog reset. The WDFLAG bit in the WDCR register can be used to differentiate between a CPU-watchdog-reset and a device reset.

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, the user can 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.

Figure 1-37. CPU Watchdog Module



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

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 1-41. 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 1-41 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 ro 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 re-configured 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.

In HALT mode, this feature cannot be used because the oscillator (and PLL) are turned off and, therefore, so is the watchdog.

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

The system control and status register (SCSR) contains the watchdog override bit and the watchdog interrupt enable/disable bit. Figure 1-38 describes the bit functions of the SCSR register.

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

15–8
Reserved
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 1-42. System Control and Status Register (SCSR) Field Descriptions

Bit	Field	Value	Description [(1)]
15-3	Reserved		Any writes to these bits(s) must always have a value of 0.
2	WDINTS		Watchdog interrupt status bit. WDINTS reflects the current state of the $\overline{\text{WDINT}}$ signal from the watchdog block. WDINTS follows the state of $\overline{\text{WDINT}}$ by two SYSCLKOUT cycles. If the watchdog interrupt is used to wake the device from IDLE or STANDBY low power mode, use this bit to make sure $\overline{\text{WDINT}}$ is not active before attempting to go back into IDLE or STANDBY mode.
		0	Watchdog interrupt signal ($\overline{\text{WDINT}}$) is active.
		1	Watchdog interrupt signal ($\overline{\text{WDINT}}$) is not active.
1	WDENINT		Watchdog interrupt enable.
		0	The watchdog reset ($\overline{\text{WDRST}}$) output signal is enabled and the watchdog interrupt ($\overline{\text{WDINT}}$) output signal is disabled. This is the default state on reset ($\overline{\text{XRS}}$). When the watchdog interrupt occurs the $\overline{\text{WDRST}}$ signal will stay low for 512 OSCCLK cycles. If the WDENINT bit is cleared while $\overline{\text{WDINT}}$ is low, a reset will immediately occur. The WDINTS bit can be read to determine the state of the $\overline{\text{WDINT}}$ signal.
		1	The $\overline{\text{WDRST}}$ output signal is disabled and the $\overline{\text{WDINT}}$ output signal is enabled. When the watchdog interrupt occurs, the $\overline{\text{WDINT}}$signal will stay low for 512 OSCCLK cycles. If the watchdog interrupt is used to wake the device from IDLE or STANDBY low power mode, use the WDINTS bit to make sure $\overline{\text{WDINT}}$ is not active before attempting to go back into IDLE or STANDBY mode.
0	WDOVERRIDE		Watchdog override
		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.
		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.

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

Figure 1-39. Watchdog Counter Register (WDCNTR)

15 … 8	7 … 0
Reserved	WDCNTR
R-0	R-0

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

Table 1-43. Watchdog Counter Register (WDCNTR) Field Descriptions

Bits	Field	Description
15-8	Reserved	Any writes to these bits(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.

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

15 … 8	7 … 0
Reserved	WDKEY
R-0	R/W-0

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

Table 1-44. Watchdog Reset Key Register (WDKEY) Field Descriptions

Bits	Field	Value	Description [(1)]
15-8	Reserved		Any writes to these bits(s) must always have a value of 0.
7-0	WDKEY		Refer to Table 1-41 for examples of different WDKEY write sequences.
		0x55 + 0xAA	Writing 0x55 followed by 0xAA to WDKEY causes the WDCNTR bits to be cleared.
		Other value	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.

Figure 1-41. Watchdog Control Register (WDCR)

15 … 8
Reserved
R-0

7	6	5 … 3	2 … 0
WDFLAG	WDDIS	WDCHK	WDPS
R/W1C-0	R/W-0	R/W-0	R/W-0

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

Table 1-45. Watchdog Control Register (WDCR) Field Descriptions

Bits	Field	Value	Description [(1)]
15-8	Reserved		Any writes to these bits(s) must always have a value of 0.
7	WDFLAG		Watchdog reset status flag bit
		0	The reset was caused either by the $\overline{\text{XRS}}$ pin or because of power-up. The bit remains latched until you write a 1 to clear the condition. Writes of 0 are ignored.
		1	Indicates a watchdog reset ($\overline{\text{WDRST}}$) generated the reset condition. .
6	WDDIS		Watchdog disable. On reset, the watchdog module is enabled.
		0	Enables the watchdog module. WDDIS can be modified only if the WDOVERRIDE bit in the SCSR register is set to 1. (default)
		1	Disables the watchdog module.

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

Table 1-45. Watchdog Control Register (WDCR) Field Descriptions (continued)

Bits	Field	Value	Description [1]
5-3	WDCHK		Watchdog check.
		0,0,0	You must ALWAYS write 1,0,1 to these bits whenever a write to this register is performed unless the intent is to reset the device via software.
		other	Writing any other value causes an immediate device reset or watchdog interrupt to be taken. Note that this happens even when watchdog module is disabled. Do not write to WDCHK bits when the watchdog module is disabled. These bits can be used to generate a software reset of the device. These three bits always read back as zero (0, 0, 0).
2-0	WDPS		Watchdog pre-scale. These bits configure the watchdog counter clock (WDCLK) rate relative to OSCCLK/512:
		000	WDCLK = OSCCLK/512/1 (default)
		001	WDCLK = OSCCLK/512/1
		010	WDCLK = OSCCLK/512/2
		011	WDCLK = OSCCLK/512/4
		100	WDCLK = OSCCLK/512/8
		101	WDCLK = OSCCLK/512/16
		110	WDCLK = OSCCLK/512/32
		111	WDCLK = OSCCLK/512/64

When the $\overline{\text{XRS}}$ line is low, the WDFLAG bit is forced low. The WDFLAG bit is only set if a rising edge on $\overline{\text{WDRST}}$ signal is detected (after synch and an 8192 SYSCLKOUT cycle delay) and the $\overline{\text{XRS}}$ signal is high. If the $\overline{\text{XRS}}$ signal is low when $\overline{\text{WDRST}}$ goes high, then the WDFLAG bit remains at 0. In a typical application, the $\overline{\text{WDRST}}$ signal connects to the $\overline{\text{XRS}}$ input. Hence to distinguish between a watchdog reset and an external device reset, an external reset must be longer in duration then 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 DSP/BIOS. If the application is not using DSP/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

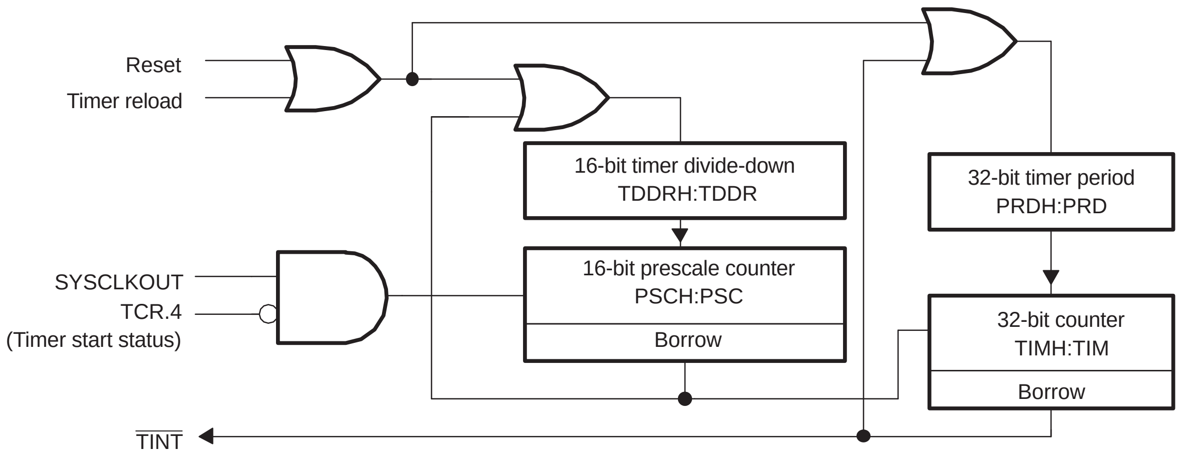
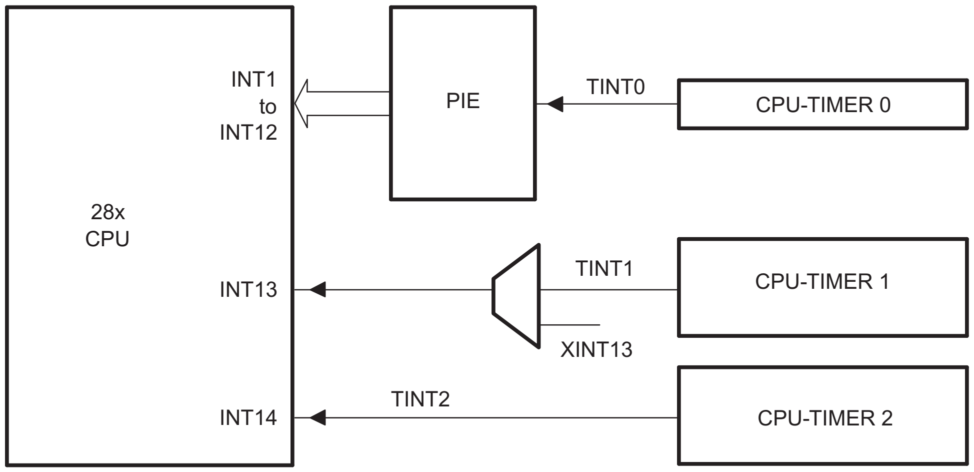


Figure 1-43. CPU-Timer Interrupts Signals and Output Signal



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.

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) SYSCLKOUT 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 1-46 are used to configure the timers.

Table 1-46. CPU-Timers 0, 1, 2 Configuration and Control Registers

Name	Address	Size (x16)	Description	Bit Description
TIMER0TIM	0x0C00	1	CPU-Timer 0, Counter Register	Figure 1-44
TIMER0TIMH	0x0C01	1	CPU-Timer 0, Counter Register High	Figure 1-45
TIMER0PRD	0x0C02	1	CPU-Timer 0, Period Register	Figure 1-46
TIMER0PRDH	0x0C03	1	CPU-Timer 0, Period Register High	Figure 1-47
TIMER0TCR	0x0C04	1	CPU-Timer 0, Control Register	Figure 1-48
TIMER0TPR	0x0C06	1	CPU-Timer 0, Prescale Register	Figure 1-49
TIMER0TPRH	0x0C07	1	CPU-Timer 0, Prescale Register High	Figure 1-50
TIMER1TIM	0x0C08	1	CPU-Timer 1, Counter Register	Figure 1-44
TIMER1TIMH	0x0C09	1	CPU-Timer 1, Counter Register High	Figure 1-45
TIMER1PRD	0x0C0A	1	CPU-Timer 1, Period Register	Figure 1-46
TIMER1PRDH	0x0C0B	1	CPU-Timer 1, Period Register High	Figure 1-47
TIMER1TCR	0x0C0C	1	CPU-Timer 1, Control Register	Figure 1-48
TIMER1TPR	0x0C0E	1	CPU-Timer 1, Prescale Register	Figure 1-49
TIMER1TPRH	0x0C0F	1	CPU-Timer 1, Prescale Register High	Figure 1-50
TIMER2TIM	0x0C10	1	CPU-Timer 2, Counter Register	Figure 1-44
TIMER2TIMH	0x0C11	1	CPU-Timer 2, Counter Register High	Figure 1-45
TIMER2PRD	0x0C12	1	CPU-Timer 2, Period Register	Figure 1-46
TIMER2PRDH	0x0C13	1	CPU-Timer 2, Period Register High	Figure 1-47
TIMER2TCR	0x0C14	1	CPU-Timer 2, Control Register	Figure 1-48
TIMER2TPR	0x0C16	1	CPU-Timer 2, Prescale Register	Figure 1-49
TIMER2TPRH	0x0C17	1	CPU-Timer 2, Prescale Register High	Figure 1-50

Figure 1-44. TIMERxTIM Register (x = 0, 1, 2)

15 — 0
TIM
R/W-0

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

Table 1-47. 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.

Figure 1-45. TIMERxTIMH Register (x = 0, 1, 2)

15 — 0
TIMH
R/W-0

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

Table 1-48. TIMERxTIMH Register Field Descriptions

Bits	Field	Description
15-0	TIMH	See description for TIMERxTIM.

Figure 1-46. TIMERxPRD Register (x = 0, 1, 2)

15 … 0
PRD
R/W-1

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

Table 1-49. 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).

Figure 1-47. TIMERxPRDH Register (x = 0, 1, 2)

15 … 0
PRDH
R/W-0

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

Table 1-50. TIMERxPRDH Register Field Descriptions

Bits	Field	Description
15-0	PRDH	See description for TIMERxPRD

Figure 1-48. 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 1-51. TIMERxTCR Register Field Descriptions

Bits	Field	Value	Description
15	TIF		CPU-Timer Interrupt Flag.
		0	The CPU-Timer has not decremented to zero. Writes of 0 are ignored.
		1	This flag gets set when the CPU-timer decrements to zero. Writing a 1 to this bit clears the flag.
14	TIE		CPU-Timer Interrupt Enable.
		0	The CPU-Timer interrupt is disabled.
		1	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 bits(s) must always have a value of 0.

Table 1-51. TIMERxTCR Register Field Descriptions (continued)

Bits	Field	Value	Description
11-10	FREE SOFT		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 *don't care*. 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.
		FREE SOFT	CPU-Timer Emulation Mode
		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
			In the SOFT STOP mode, the timer generates an interrupt before shutting down (since reaching 0 is the interrupt causing condition).
9-6	Reserved		Any writes to these bits(s) must always have a value of 0.
5	TRB		CPU-Timer Reload bit.
		0	The TRB bit is always read as zero. Writes of 0 are ignored.
		1	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 (TDDRH:TDDR).
4	TSS		CPU-Timer stop status bit. TSS is a 1-bit flag that stops or starts the CPU-timer.
		0	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 bits(s) must always have a value of 0.

Figure 1-49. 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 1-52. 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.

Figure 1-50. TIMERxTPRH Register (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 1-53. TIMERxTPRH Register Field Descriptions

Bits	Field	Description
15-8	PSCH	See description of TIMERxTPR.
7-0	TDDRH	See description of TIMERxTPR.

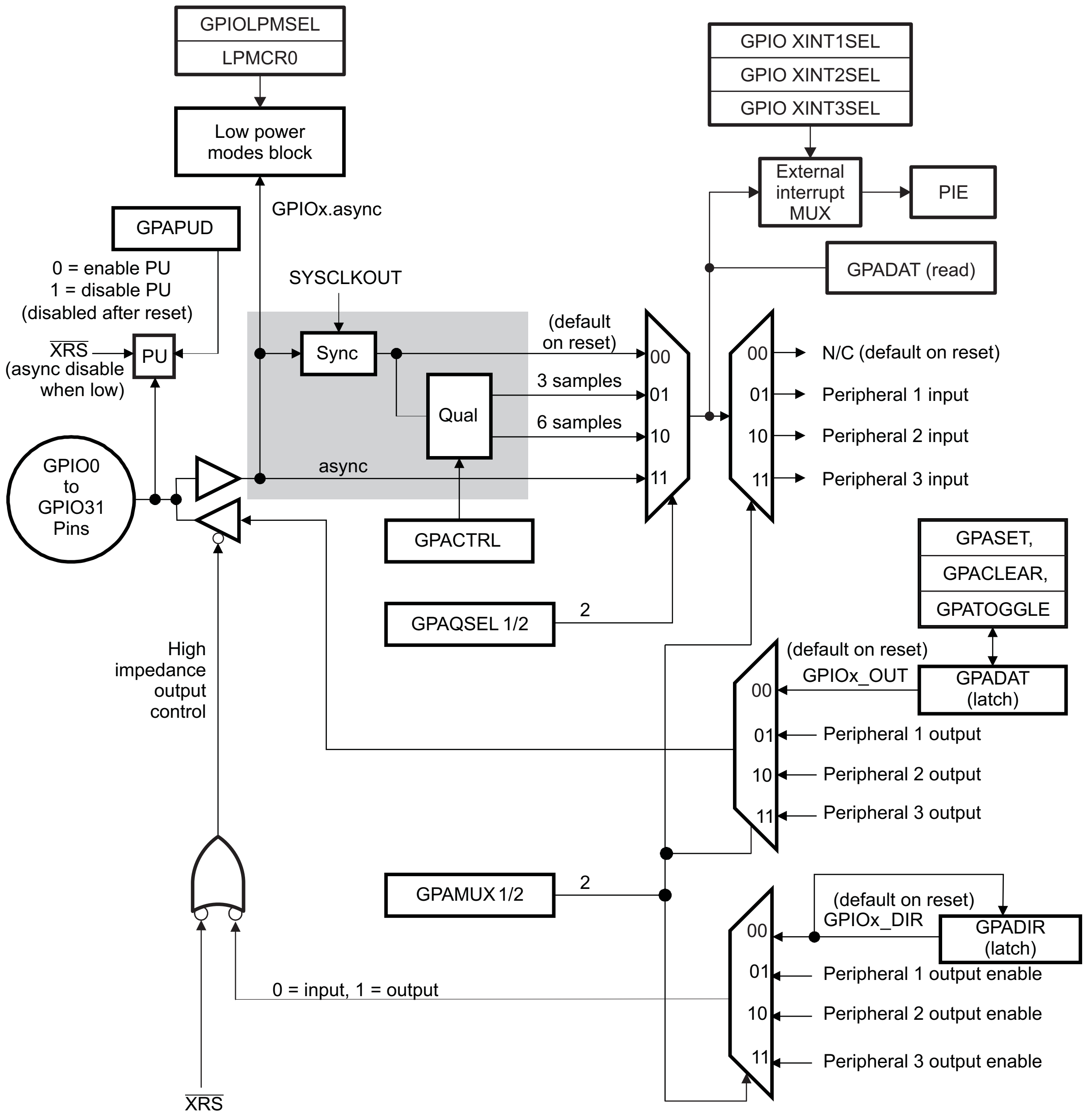
1.4 General-Purpose Input/Output (GPIO)

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 (GPIO0 - GPIO 58). 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 (via the GPxMUXn registers). If selected for digital I/O mode, registers are provided to configure the pin direction (via the GPxDIR registers). You can also qualify the input signals to remove unwanted noise (via the GPxQSELn, GPACTRL, and GPBCTRL registers).

1.4.1 GPIO Module Overview

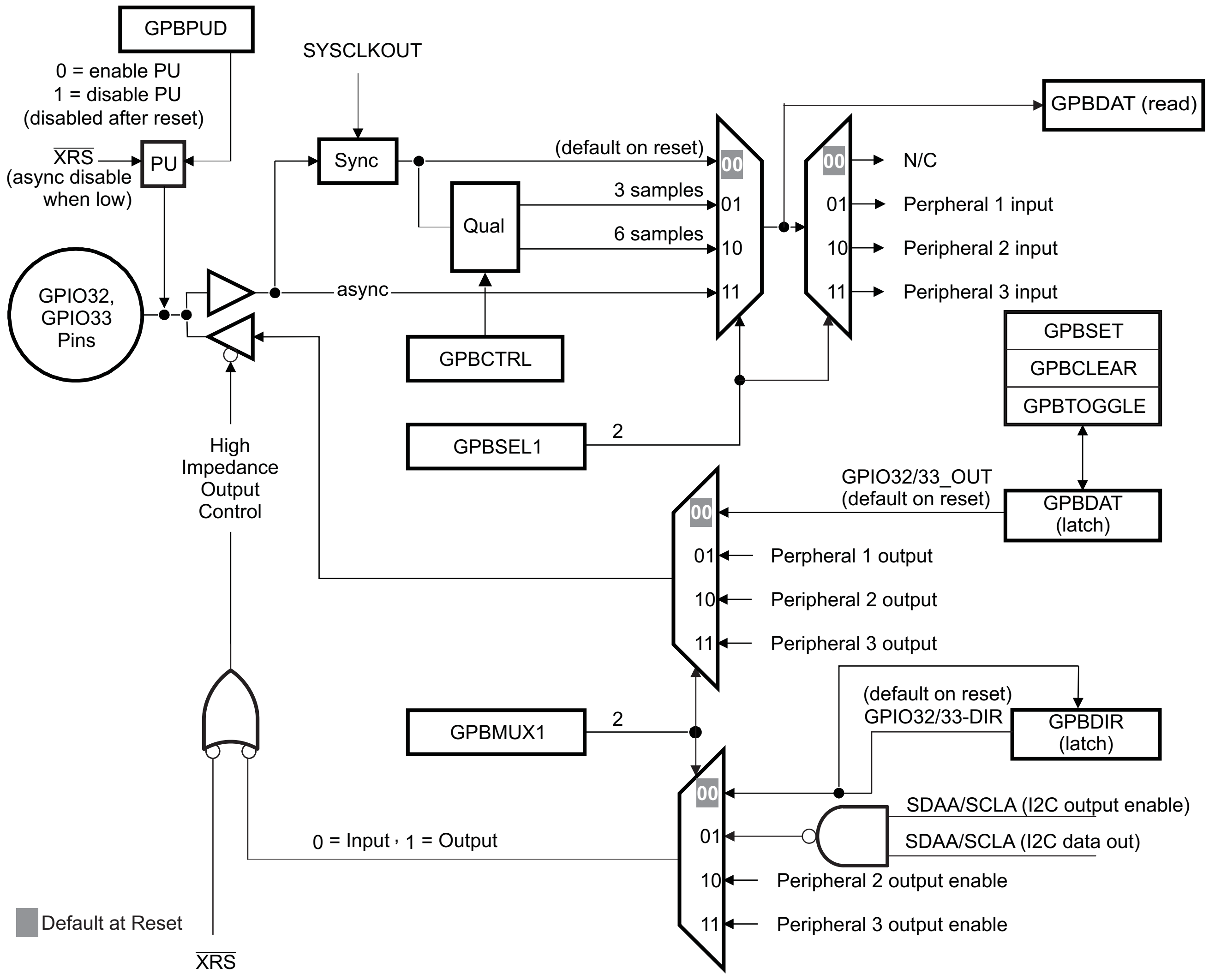
Up to three independent peripheral signals are multiplexed on a single GPIO-enabled pin in addition to individual pin bit-I/O capability. There are three I/O ports. Port A consists of GPIO0-GPIO31, port B consists of GPIO32-GPIO 58. The analog port consists of AIO0-AIO15. Figure 1-51 shows the basic modes of operation for the GPIO module. Note that GPIO functionality is provided on JTAG pins as well.

Figure 1-51. GPIO0 to GPIO31, GPIO34, GPIO40-GPIO58 Multiplexing Diagram



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

Figure 1-52. GPIO32, GPIO33 Multiplexing Diagram



A The GPIOINENCLK bit in the PCLKCR3 register does not affect the above GPIOs (I^2C pins) since the pins are bi-directional.

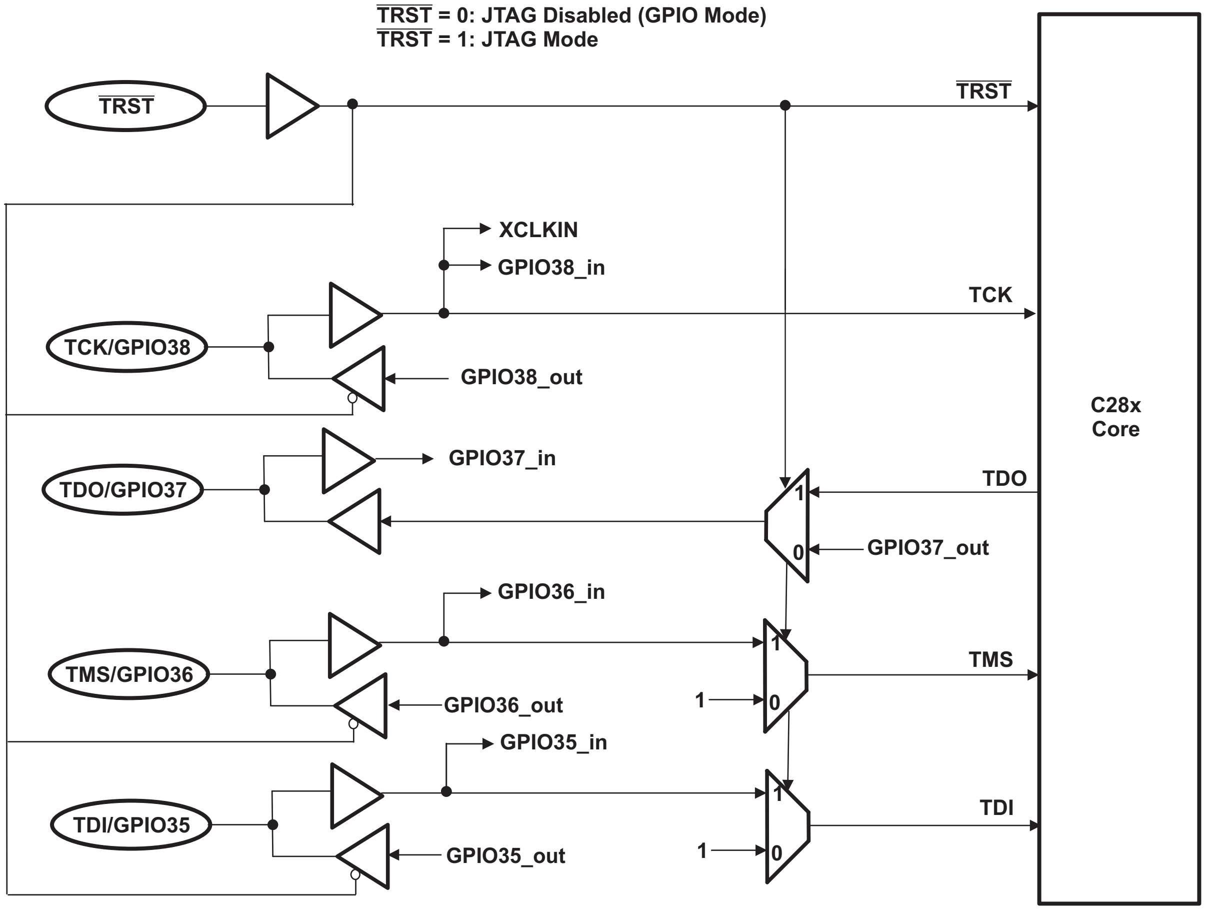
B The input qualification circuit is not reset when modes are changed (such as changing from output to input mode). Any state will get flushed by the circuit eventually.

1.4.1.1 JTAG Port

The JTAG port is reduced to 5 pins ($\overline{\text{TRST}}$, TCK, TDI, TMS, TDO). TCK, TDI, TMS and TDO pins are also GPIO pins. The $\overline{\text{TRST}}$ signal selects either 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 emulator 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 follows.

$\overline{\text{TRST}}$	JTAGDISBbit	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 and described below.

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

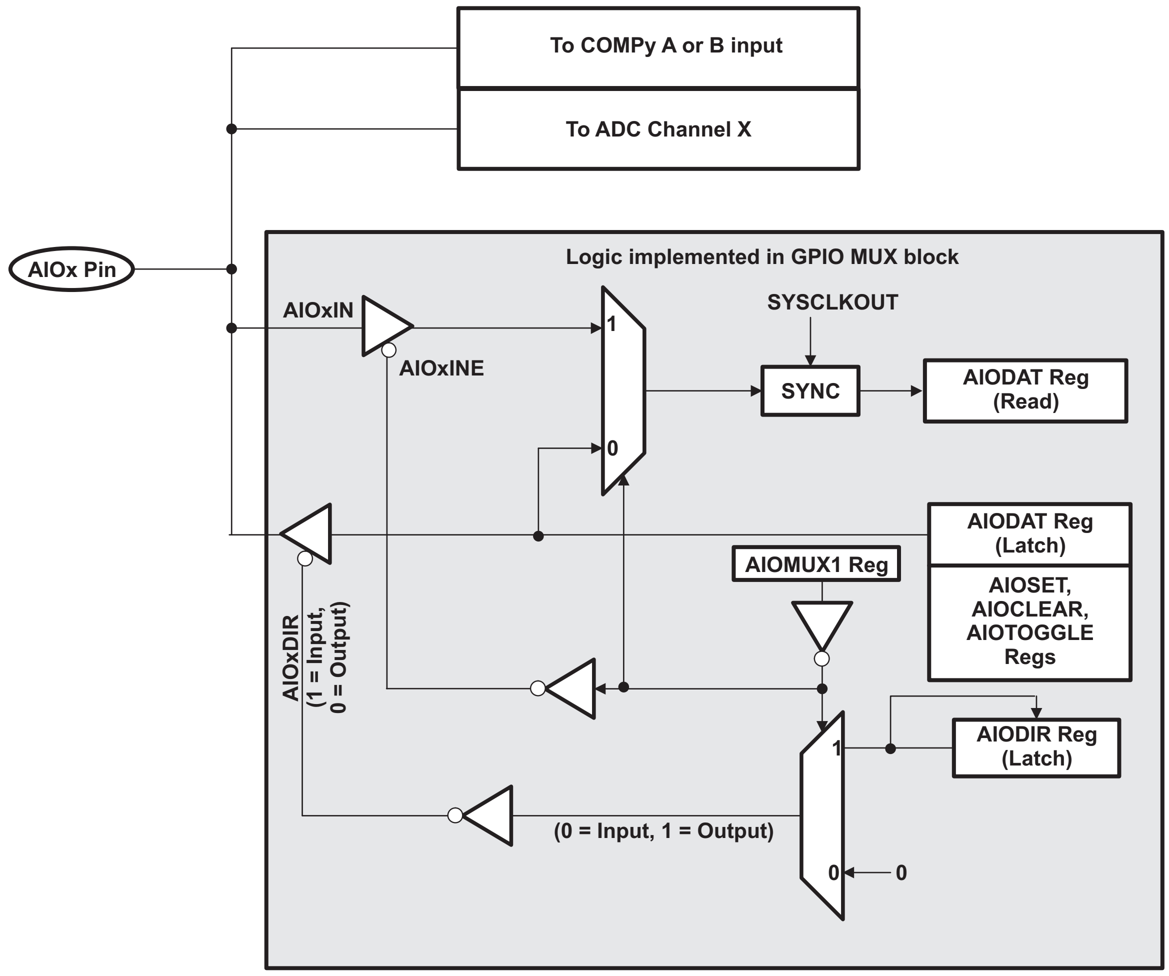
15–1	0
Reserved	JTAGDIS
R-0	R/W-0

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

Table 1-54. JTAGDEBUG Register Field Descriptions

Bits	Field	Value	Description
15-1	Reserved		Any writes to these bits(s) must always have a value of 0.
0	JTAGDIS		JTAG Port Disable Bit: This bit enables/disables the JTAG port. When disabled, the JTAG pins can be used as GPIOs:
		0	JTAG Port Enabled
		1	JTAG Port Disabled (GPIO Mode)
			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.
Note: Ensure no contention with the emulator signals when JTAGDIS=1			

Figure 1-55. Analog/GPIO Multiplexing



A The ADC/Comparator path is always enabled, irrespective of the AIOMUX1 value.

B The AIO section is blocked off when the corresponding AIOMUX1 bit is 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 1-55 and Table 1-56 list the registers that are used to configure the GPIO pins to match the system requirements.

Table 1-55. GPIO Control Registers

Name [(1)]	Address	Size (x16)	Register Description	Bit Description
GPACTRL	0x6F80	2	GPIO A Control Register (GPIO0-GPIO31)	Figure 1-63
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register (GPIO0-GPIO15)	Figure 1-66
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register (GPIO16-GPIO31)	Figure 1-67
GPAMUX1	0x6F86	2	GPIO A MUX 1 Register (GPIO0-GPIO15)	Figure 1-58
GPAMUX2	0x6F88	2	GPIO A MUX 2 Register (GPIO16-GPIO31)	Figure 1-59
GPADIR	0x6F8A	2	GPIO A Direction Register (GPIO0-GPIO31)	Figure 1-70
GPAPUD	0x6F8C	2	GPIO A Pull Up Disable Register (GPIO0-GPIO31)	Figure 1-73
GPACTRL2	0x6F8E	2	USB I/O Control	Figure 1-65
GPBCTRL	0x6F90	2	GPIO B Control Register (GPIO32-GPIO58)	Figure 1-64
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register (GPIO32-GPIO44)	Figure 1-68
GPBQSEL2	0x6F94	2	GPIO B Qualifier Select 2 Register (GPIO50-GPIO58)	Figure 1-69
GPBMUX1	0x6F96	2	GPIO B MUX 1 Register (GPIO32-GPIO44)	Figure 1-60
GPBMUX2	0x6F98	2	GPIO B MUX2 Register (GPIO50-GPIO58)	Figure 1-61
GPBDIR	0x6F9A	2	GPIO B Direction Register (GPIO32-GPIO58)	Figure 1-71
GPBPUD	0x6F9C	2	GPIO B Pull Up Disable Register (GPIO32-GPIO58)	Figure 1-74
AIOMUX1	0x6FB6	2	Analog, I/O MUX 1 register (AIO0 - AIO15)	Figure 1-62
AIODIR	0x6FBA	2	Analog, I/O Direction Register (AIO0 - AIO15)	Figure 1-72

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

Table 1-56. GPIO Interrupt and Low Power Mode Select Registers

Name [(1)]	Address	Size (x16)	Register Description	Bit Description
GPIOXINT1SEL	0x6FE0	1	XINT1 Source Select Register (GPIO0-GPIO31)	Figure 1-81
GPIOXINT2SEL	0x6FE1	1	XINT2 Source Select Register (GPIO0-GPIO31)	Figure 1-81
GPIOXINT3SEL	0x6FE2	1	XINT3 Source Select Register (GPIO0 - GPIO31)	Figure 1-81
GPIOLPMSEL	0x6FE8	1	LPM wakeup Source Select Register (GPIO0-GPIO31)	Figure 10-3

(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:

Step 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.

Step 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.

Step 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.

Step 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.

Step 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.

Step 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.

Step 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 1.6.6.

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 you can change the values on the pins by using the registers in Table 1-57.

Table 1-57. GPIO Data Registers

Name	Address	Size (x16)	Register Description	Bit Description
GPADAT	0x6FC0	2	GPIO A Data Register (GPIO0-GPIO31)	Figure 1-75
GPASET	0x6FC2	2	GPIO A Set Register (GPIO0-GPIO31)	Figure 1-78
GPACLEAR	0x6FC4	2	GPIO A Clear Register (GPIO0-GPIO31)	Figure 1-78
GPATOGGLE	0x6FC6	2	GPIO A Toggle Register (GPIO0-GPIO31)	Figure 1-78
GPBDAT	0x6FC8	2	GPIO B Data Register (GPIO32-GPIO58)	Figure 1-76
GPBSET	0x6FCA	2	GPIO B Set Register (GPIO32-GPIO58)	Figure 1-79
GPBCLEAR	0x6FCC	2	GPIO B Clear Register (GPIO32-GPIO58)	Figure 1-79
GPBTOGGLE	0x6FCE	2	GPIO B Toggle Register (GPIO32-GPIO58)	Figure 1-79
AIODAT	0x6FD8	2	Analog I/O Data Register (AIO0 - AIO15)	Figure 1-77
AIOSET	0x6FDA	2	Analog I/O Data Set Register (AIO0 - AIO15)	Figure 1-80
AIOCLEAR	0x6FDC	2	Analog I/O Clear Register (AIO0 - AIO15)	Figure 1-80
AIOTOGGLE	0x6FDE	2	Analog I/O Toggle Register (AIO0 - AIO15)	Figure 1-80

- **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 bit(s) 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^{2}C. In addition, it may be desirable to have the ePWM trip zone ($\overline{\text{TZn}}$) 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.

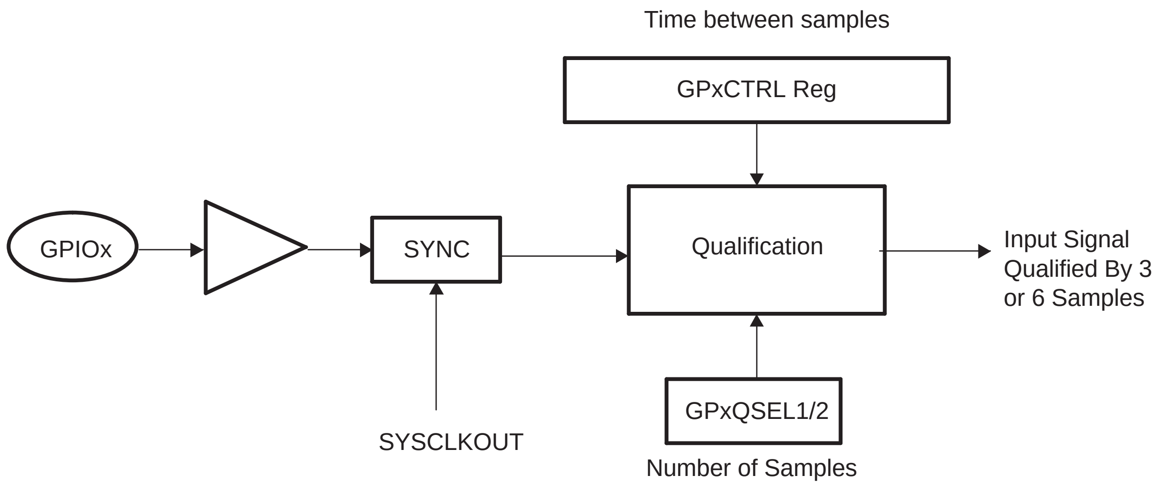
1.4.4.2 Synchronization to SYSCLKOUT Only

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 DSP 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 GPACTRL[QUALPRD0] setting and GPIO8 to GPIO15 use GPACTRL[QUALPRD1]. Table 1-58 and Table 1-59 show the relationship between the sampling period or sampling frequency and the GPxCTRL[QUALPRDn] setting.

Table 1-58. Sampling Period

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

Table 1-59. Sampling Frequency

	Sampling Frequency
If GPxCTRL[QUALPRDn] = 0	$f_{SYSCLKOUT}$
If GPxCTRL[QUALPRDn] ≠ 0	$f_{SYSCLKOUT}$ × 1 ÷ (2 × GPxCTRL[QUALPRDn])
	Where $f_{SYSCLKOUT}$ 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 $f_{SYSCLKOUT}$

If, for example, $f_{SYSCLKOUT}$ = 60 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 (255)

then the sampling frequency is $f_{SYSCLKOUT}$ × 1 ÷ (2 × GPxCTRL[QUALPRDn])

If, for example, $f_{SYSCLKOUT}$ = 60 MHz

then the signal will be sampled at 60 MHz × 1 ÷ (2 × 255) or one sample every 8.5 μs.

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 DSP.

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 then the number of samples taken. For a thee-sample window, the sampling window width is 2 sampling periods wide where the sampling period is defined in Table 1-58. Likewise, for a six-sample window, the sampling window width is 5 sampling periods wide. Table 1-60 and Table 1-61 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 1-60. Case 1: Three-Sample Sampling Window Width

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

Table 1-61. Case 2: Six-Sample Sampling Window Width

	Total Sampling Window Width
If GPxCTRL[QUALPRDn] = 0	$5 \times T_{SYSCLKOUT}$
If GPxCTRL[QUALPRDn] ≠ 0	$5 \times 2 \times$ GPxCTRL[QUALPRDn] $\times T_{SYSCLKOUT}$
	Where $T_{SYSCLKOUT}$ 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 + $T_{SYSCLKOUT}$.

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:

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

This configuration results in the following:

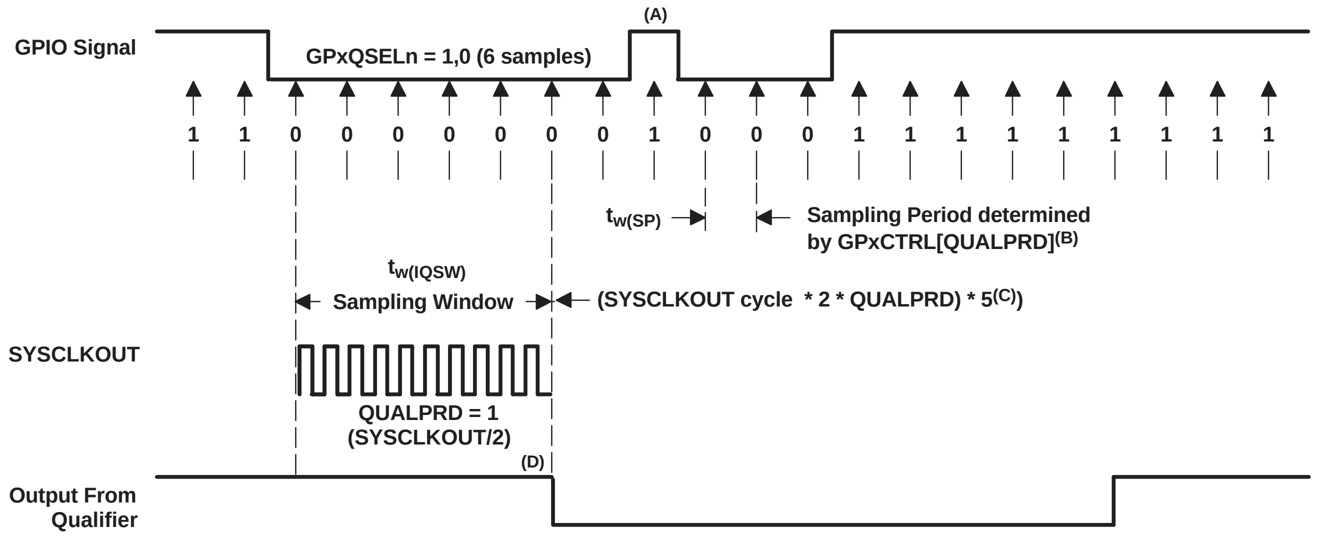
- The width of the sampling window is: .

  $t_w(IQSW) = 5 \times t_w(SP) = 5 \times 2 \times GPxCTRL[QUALPRDn] \times T_{SYSCLKOUT}$ or $5 \times 2 \times T_{SYSCLKOUT}$
- If, for example, $T_{SYSCLKOUT}$ = 16.67 ns, then the duration of the sampling window is:

  $t_w(IQSW) = 5 \times 2 \times 16.67$ ns =166.7 ns.
- To account for the asynchronous nature of the input relative to the sampling period and SYSCLKOUT, up to an additional sampling period, $t_w(SP)$, + $T_{SYSCLKOUT}$ may be required to detect a change in the input signal. For this example:

  $t_w(SP) + T_{SYSCLKOUT}$ = 333.4 ns + 166.67 ns = 500.1 ns
- In Figure 1-57, the glitch (A) is shorter then the qualification window and will be ignored by the input qualifier.

Figure 1-57. Input Qualifier Clock Cycles


(A)
GPIO Signal
GPxQSELn = 1,0 (6 samples)
1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1
tw(SP)
Sampling Period determined by GPxCTRL[QUALPRD](B)
tw(IQSW)
Sampling Window
(SYSCLKOUT cycle * 2 * QUALPRD) * 5(C))
SYSCLKOUT
QUALPRD = 1 (SYSCLKOUT/2)
(D)
Output From Qualifier

A. 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 in 2n SYSCLKOUT cycles (i.e., at every 2n SYSCLKOUT cycles, the GPIO pin will be sampled).

B. The qualification period selected via the GPxCTRL register applies to groups of 8 GPIO pins.

C. The qualification block can take either three or six samples. The GPxQSELn Register selects which sample mode is used.

D. 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 x QUALPRD x 2) SYSCLKOUT cycles. That would ensure 5 sampling periods for detection to occur. Since external signals are driven asynchronously, an 13-SYSCLKOUT-wide pulse ensures reliable recognition.

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 1-63 and Table 1-64 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 only refer 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 follows:

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 EPWMSYNCI (I)
If GPAMUX1[13:12] = 1,1	Pin configured as EPWMSYNCO (O)

The devices have different multiplexing schemes. 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 1-63 and Table 1-64) these options are indicated as Reserved .

Some peripherals can be assigned to more than one pin via the MUX registers. For example, the SPISIMOB can be assigned to either the GPIO12 or GPIO24 pin, depending on individual system requirements as shown below:

Pin Assigned to SPISIMOB		MUX Configuration
Choice 1	GPIO12	GPAMUX[25:24] = 1,1
or Choice 2	GPIO24	GPAMUX2[17:16] = 1,1

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 1-62. For example, if SPISIMOB were assigned to both GPIO12 and GPIO24, the input to the SPI peripheral would default to a high state as shown in Table 1-62 and the input would not be connected to GPIO12 or GPIO24.

Table 1-62. Default State of Peripheral Input

Peripheral Input	Description	Default Input [(1)]
$\overline{\text{TZ1}}$-$\overline{\text{TZ3}}$	Trip zone 1-3	1
EPWMSYNCI	ePWM Synch Input	0
ECAP1	eCAP1 input	1
EQEP1A	eQEP input	1
EQEP1I	eQEP index	1
EQEP1S	eQEP strobe	1
SPICLKA/SPICLKB	SPI-A clock	1
$\overline{\text{SPISTEA}}$ /$\overline{\text{SPISTEB}}$	SPI-A transmit enable	0
SPISIMOA/SPISIMOB	SPI-A Slave-in, master-out	1
SPISOMIA/SPISOMIB	SPI-A Slave-out, master-in	1
SCIRXDA - SCIRXDB	SCI-A - SCI-B receive	1
CANRXA	eCAN-A receive	1
SDAA	I^{2}C data	1
SCLA1	I^{2}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 1-63. GPIOA MUX[1] [2]

	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	Reserved
3-2	GPIO1	EPWM1B (O)	Reserved	COMP1OUT (O)
5-4	GPIO2	EPWM2A (O)	Reserved	Reserved
7-6	GPIO3	EPWM2B (O)	SPISOMIA (I/O)	COMP2OUT (O)
9-8	GPIO4	EPWM3A (O)	Reserved	Reserved
11-10	GPIO5	EPWM3B (O)	SPISIMOA (I/O)	ECAP1 (I/O)
13-12	GPIO6	EPWM4A (O)	EPWMSYNCI (I)	EPWMSYNCO (O)
15-14	GPIO7	EPWM4B (O)	SCIRXDA (I)	ECAP2 (I/O)
17-16	GPIO8	EPWM5A (O)	Reserved	$\overline{\text{ADCSOCAO}}$ (O)
19-18	GPIO9	EPWM5B (O)	SCITXDB[3] (O)	ECAP3 (I/O)
21-20	GPIO10	EPWM6A (O)	Reserved	$\overline{\text{ADCSOCBO}}$ (O)
23-22	GPIO11	EPWM6B (O)	SCIRXDB[3] (I)	ECAP1 (I/O)
25-24	GPIO12	$\overline{\text{TZ1}}$ (I)	SCITXDA (O)	SPISIMOB (I/O)
27-26	GPIO13	$\overline{\text{TZ2}}$ (I)	Reserved	SPISOMIB (I/O)
29-28	GPIO14	$\overline{\text{TZ3}}$ (I)	SCITXDB[3] (O)	SPICLKB (I/O)
31-30	GPIO15	ECAP2 (I/O)	SCIRXDB[3] (I)	$\overline{\text{SPISTEB}}$ (I/O)
GPAMUX2 REGISTER BITS	**(GPAMUX2 BITS = 00)**	**(GPAMUX2 BITS = 01)**	**(GPAMUX2 BITS = 10)**	**(GPAMUX2 BITS = 11)**
1-0	GPIO16	SPISIMOA (I/O)	Reserved	$\overline{\text{TZ2}}$ (I)
3-2	GPIO17	SPISOMIA (I/O)	Reserved	$\overline{\text{TZ3}}$ (I)
5-4	GPIO18	SPICLKA (I/O)	SCITXDB[3] (O)	XCLKOUT (O)
7-6	GPIO19/XCLKIN	$\overline{\text{SPISTEA}}$ (I/O)	SCIRXDB[3] (I)	ECAP1 (I/O)
9-8	GPIO20	EQEP1A (I)	MDXA (O)	COMP1OUT (O)
11-10	GPIO21	EQEP1B (I)	MDRA (I)	COMP2OUT (O)
13-12	GPIO22	EQEP1S (I/O)	MCLKXA (I/O)	SCITXDB[3] (O)
15-14	GPIO23	EQEP1I (I/O)	MFSXA (I/O)	SCIRXDB[3] (I)
17-16	GPIO24	ECAP1 (I/O)	EQEP2A[3] (I)	SPISIMOB (I/O)
19-18	GPIO25	ECAP2 (I/O)	EQEP2B[3] (I)	SPISOMIB (I/O)
21-20	GPIO26	ECAP3 (I/O)	EQEP2I[3] (I/O)	SPICLKB (I/O)
23-22	GPIO27	HRCAP2 (I)	EQEP2S[3] (I/O)	$\overline{\text{SPISTEB}}$ (I/O)
25-24	GPIO28	SCIRXDA (I)	SDAA (I/OD)	$\overline{\text{TZ2}}$ (I)
27-26	GPIO29	SCITXDA (O)	SCLA (I/OD)	$\overline{\text{TZ3}}$ (I)
29-28	GPIO30	CANRXA (I)	EQEP2I[3] (I/O)	EPWM7A (O)
31-30	GPIO31	CANTXA (O)	EQEP2S[3] (I/O)	EPWM8A (O)

(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.
(2) I = Input, O = Output, OD = Open Drain
(3) eQEP2 is not available on the 80-pin PN/PFP package.

Table 1-64. GPIOB MUX(1)(2)

	DEFAULT AT RESET PRIMARY I/O FUNCTION	**PERIPHERAL SELECTION 1**	**PERIPHERAL SELECTION 2**	**PERIPHERAL SELECTION 3**
GPBMUX1 REGISTER BITS	**(GPBMUX1 BITS = 00)**	**(GPBMUX1 BITS = 01)**	**(GPBMUX1 BITS = 10)**	**(GPBMUX1 BITS = 11)**
1-0	GPIO32	SDAA (I/OD)	EPWMSYNCI (I)	$\overline{\text{ADCSOCAO}}$ (O)
3-2	GPIO33	SCLA (I/OD)	EPWMSYNCO (O)	$\overline{\text{ADCSOCBO}}$ (O)
5-4	GPIO34	COMP2OUT (O)	Reserved	COMP3OUT (O)
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	GPIO39	Reserved	Reserved	Reserved
17-16	GPIO40(3)	EPWM7A (O)	SCITXDB (O)	Reserved
19-18	GPIO41(3)	EPWM7B (O)	SCIRXDB (I)	Reserved
21-20	GPIO42(3)	EPWM8A (O)	$\overline{\text{TZ1}}$ (I)	COMP1OUT (O)
23-22	GPIO43(3)	EPWM8B (O)	$\overline{\text{TZ2}}$ (I)	COMP2OUT (O)
25-24	GPIO44(3)	MFSRA (I/O)	SCIRXDB (I)	EPWM7B (O)
27-26	Reserved	Reserved	Reserved	Reserved
29-28	Reserved	Reserved	Reserved	Reserved
31-30	Reserved	Reserved	Reserved	Reserved
GPBMUX2 REGISTER BITS	**(GPBMUX2 BITS = 00)**	**(GPBMUX2 BITS = 01)**	**(GPBMUX2 BITS = 10)**	**(GPBMUX2 BITS = 11)**
1-0	Reserved	Reserved	Reserved	Reserved
3-2	Reserved	Reserved	Reserved	Reserved
5-4	GPIO50(3)	EQEP1A (I)	MDXA (O)	$\overline{\text{TZ1}}$ (I)
7-6	GPIO51(3)	EQEP1B (I)	MDRA (I)	$\overline{\text{TZ2}}$ (I)
9-8	GPIO52(3)	EQEP1S (I/O)	MCLKXA (I/O)	$\overline{\text{TZ3}}$ (I)
11-10	GPIO53(3)	EQEP1I (I/O)	MFSXA (I/O)	Reserved
13-12	GPIO54(3)	SPISIMOA (I/O)	EQEP2A (I)	HRCAP1 (I)
15-14	GPIO55(3)	SPISOMIA (I/O)	EQEP2B (I)	HRCAP2 (I)
17-16	GPIO56(3)	SPICLKA (I/O)	EQEP2I (I/O)	HRCAP3 (I)
19-18	GPIO57(3)	$\overline{\text{SPISTEA}}$ (I/O)	EQEP2S (I/O)	HRCAP4 (I)
21-20	GPIO58(3)	MCLKRA (I/O)	SCITXDB (O)	EPWM7A (O)
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

(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.
(2) I = Input, O = Output, OD = Open Drain
(3) This pin is not available in the 80-pin PN/PFP package.

Table 1-65. Analog MUX

	AIOx and Peripheral Selection1	**Default at Reset Peripheral Selection 2 and Peripheral Selection 3**
AIOMUX1 Register bits	**AIOMUX1 bits = 0,x**	**AIOMUX1 bits = 1,x**
1-0	ADCINA0 (I)	ADCINA0 (I)

Table 1-65. Analog MUX (continued)

	AIOx and Peripheral Selection1	Default at Reset Peripheral Selection 2 and Peripheral Selection 3
AIOMUX1 Register bits	AIOMUX1 bits = 0,x	AIOMUX1 bits = 1,x
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 Register Bit Definitions

Figure 1-58. GPIO Port A MUX 1 (GPAMUX1) 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 1-66. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-30	GPIO15		Configure the GPIO15 pin as:
		00	GPIO15 - General purpose input/output 15 (default) (I/O)
		01	ECAP2 (I/O)
		10	SCIRXDB (I)
		11	$\overline{\text{SPISTEB}}$ (I/O)
29-28	GPIO14		Configure the GPIO14 pin as:
		00	GPIO14 - General purpose I/O 14 (default) (I/O)
		01	$\overline{\text{TZ3}}$ - Trip zone 3 (I)
		10	SCITXDB (O)
		11	SPICLKB (IO) - SPI-B clock This option is reserved on devices that do not have an SPI-B port.

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

Table 1-66. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions (continued)

Bits	Field	Value	Description [(1)]
27-26	GPIO13		Configure the GPIO13 pin as:
		00	GPIO13 - General purpose I/O 13 (default) (I/O)
		01	$\overline{\text{TZ2}}$ - Trip zone 2 (I)
		10	Reserved
		11	SPISOMIB (I/O) - SPI-B Slave Output/Master input This option is reserved on devices that do not have an SPI-B port.
25-24	GPIO12		Configure the GPIO12 pin as:
		00	GPIO12 - General purpose I/O 12 (default) (I/O)
		01	$\overline{\text{TZ1}}$ - Trip zone 1 (I)
		10	SCITXDA - SCI-A Transmit (O)
		11	SPISIMOB (I/O) - SPI-B Slave input/Master output This option is reserved on devices that do not have an SPI-B port.
23-22	GPIO11		Configure the GPIO11 pin as:
		00	GPIO11 - General purpose I/O 11 (default) (I/O)
		01	EPWM6B - ePWM 6 output B (O)
		10	SCIRXDB (I)
		11	ECAP1 (I/O)
21-20	GPIO10		Configure the GPIO10 pin as:
		00	GPIO10 - General purpose I/O 10 (default) (I/O)
		01	EPWM6A - ePWM6 output A (O)
		10	Reserved
		11	$\overline{\text{ADCSOCBO}}$ - ADC Start of conversion B (O)
19-18	GPIO9		Configure the GPIO9 pin as:
		00	GPIO9 - General purpose I/O 9 (default) (I/O)
		01	EPWM5B - ePWM5 output B
		10	SCITXDB (O)
		11	ECAP3 (I/O)
17-16	GPIO8		Configure the GPIO8 pin as:
		00	GPIO8 - General purpose I/O 8 (default) (I/O)
		01	EPWM5A - ePWM5 output A (O)
		10	Reserved
		11	$\overline{\text{ADCSOCAO}}$ - ADC Start of conversion A
15-14	GPIO7		Configure the GPIO7 pin as:
		00	GPIO7 - General purpose I/O 7 (default) (I/O)
		01	EPWM4B - ePWM4 output B (O)
		10	SCIRXDA (I) - SCI-A receive (I)
		11	ECAP2 (I/O)
13-12	GPIO6		Configure the GPIO6 pin as:
		00	GPIO6 - General purpose I/O 6 (default)
		01	EPWM4A - ePWM4 output A (O)
		10	EPWMSYNCI - ePWM Synch-in (I)
		11	EPWMSYNCO - ePWM Synch-out (O)
11-10	GPIO5		Configure the GPIO5 pin as:
		00	GPIO5 - General purpose I/O 5 (default) (I/O)
		01	EPWM3B - ePWM3 output B
		10	SPISIMOA (I/O) - SPI-A Slave input/Master output
		11	ECAP1 - eCAP1 (I/O)

Table 1-66. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions (continued)

Bits	Field	Value	Description [(1)]
9-8	GPIO4		Configure the GPIO4 pin as:
		00	GPIO4 - General purpose I/O 4 (default) (I/O)
		01	EPWM3A - ePWM3 output A (O)
		10	Reserved
		11	Reserved
7-6	GPIO3		Configure the GPIO3 pin as:
		00	GPIO3 - General purpose I/O 3 (default) (I/O)
		01	EPWM2B - ePWM2 output B (O)
		10	SPISOMIA (I/O) - SPI-A Slave output/Master input
		11	COMP2OUT (O) - Comparator 2 output
5-4	GPIO2		Configure the GPIO2 pin as:
		00	GPIO2 (I/O) General purpose I/O 2 (default) (I/O)
		01	EPWM2A - ePWM2 output A (O)
		10	Reserved
		11	Reserved
3-2	GPIO1		Configure the GPIO1 pin as:
		00	GPIO1 - General purpose I/O 1 (default) (I/O)
		01	EPWM1B - ePWM1 output B (O)
		10	Reserved
		11	COMP1OUT (O) - Comparator 1 output
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
		11	Reserved

Figure 1-59. GPIO Port A MUX 2 (GPAMUX2) 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/XCLKIN	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 1-67. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-30	GPIO31		Configure the GPIO31 pin as:
		00	GPIO31 - General purpose I/O 31 (default) (I/O)
		01	CANTXA - eCAN-A transmit (O)
		10	EQEP2S (I/O)
		11	EPWM8A (O)
29-28	GPIO30		Configure the GPIO30 pin as:
		00	GPIO30 (I/O) General purpose I/O 30 (default) (I/O)

[(1)] 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.

Table 1-67. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions (continued)

Bits	Field	Value	Description [1]
		01	CANRXA - eCAN-A receive (I)
		10	EQEP2I (I/O)
		11	EPWM7A (O)
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^{2}C clock open drain bidirectional port (I/O)
		11	$\overline{\text{TZ3}}$ - 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^{2}C data open drain bidirectional port (I/O)
		11	$\overline{\text{TZ2}}$ - Trip zone 2 (I)
23-22	GPIO27		Configure the GPIO27 pin as:
		00	GPIO27 - General purpose I/O 27 (default) (I/O)
		01	HRCAP2 (I)
		10	EQEP2S (I/O)
		11	$\overline{\text{SPISTEB}}$ (I/O) - SPI-B Slave transmit enable
21-20	GPIO26		Configure the GPIO26 pin as:
		00	GPIO26 - General purpose I/O 26 (default) (I/O)
		01	ECAP3 (I/O)
		10	EQEP2I (I/O)
		11	SPICLKB (I/O) - SPI-B clock
19-18	GPIO25		Configure the GPIO25 pin as:
		00	GPIO25 - General purpose I/O 25 (default) (I/O)
		01	ECAP2 (I/O)
		10	EQEP2B (I)
		11	SPISOMIB (I/O) - SPI-B Slave Output/Master input
17-16	GPIO24		Configure the GPIO24 pin as:
		00	GPIO24 - General purpose I/O 24 (default) (I/O)
		01	ECAP1 - eCAP1 (I/O)
		10	EQEP2A (I)
		11	SPISIMOB (I/O) - SPI-B Slave input/Master output
15-14	GPIO23		Configure the GPIO23 pin as:
		00	GPIO23 - General purpose I/O 23 (default) (I/O)
		01	EQEP1I - eQEP1 index (I/O)
		10	MFSXA (I/O)
		11	SCIRXDB (I)
13-12	GPIO22		Configure the GPIO22 pin as:
		00	GPIO22 - General purpose I/O 22 (default) (I/O)
		01	EQEP1S - eQEP1 strobe (I/O)
		10	MCLKXA (I/O)
		11	SCITXDB (O)

Table 1-67. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions (continued)

Bits	Field	Value	Description [(1)]
11-10	GPIO21		Configure the GPIO21 pin as:
		00	GPIO21 - General purpose I/O 21 (default) (I/O)
		01	EQEP1B - eQEP1 input B (I)
		10	MDRA (I)
		11	COMP2OUT (O) - Comparator 2 output
9-8	GPIO20		Configure the GPIO20 pin as:
		00	GPIO20 - General purpose I/O 20 (default) (I/O)
		01	EQEP1A - eQEP1 input A (I)
		10	MDXA (O)
		11	COMP1OUT (O) - Comparator 1 output
7-6	GPIO19/XCLKIN		Configure the GPIO19 pin as:
		00	GPIO19 - General purpose I/O 19 (default) (I/O) or XCLKIN
		01	$\overline{\text{SPISTEA}}$ - SPI-A slave transmit enable (I/O)
		10	SCIRXDB (I)
		11	ECAP1 - eCAP1 (I/O)
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	SCITXDB (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 output/Master input (I/O)
		10	Reserved
		11	$\overline{\text{TZ3}}$ - 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	$\overline{\text{TZ2}}$ - Trip zone 2 (I)

Figure 1-60. GPIO Port B MUX 1 (GPBMUX1) Register

31-26	25-24	23-22	21-20	19-18	17-16
Reserved	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
R-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
GPIO39	GPIO38/XCLKIN (TCK)	GPIO37(TDO)	GPIO36(TMS)	GPIO35(TDI)	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 1-68. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions

Bit	Field	Value	Description
31:26	Reserved	Reserved	Reserved
25:24	GPIO44		Configure this pin as:
		00	GPIO44 - general purpose I/O 44 (default)
		01	MFSRA (I/O)
		10	SCIRXDB (I)
		11	EPWM7B (O)
23:22	GPIO43		Configure this pin as:
		00	GPIO43 - general purpose I/O 43 (default)
		01	EPWM8B (O)
		10	$\overline{\text{TZ2}}$ (I)
		11	COMP2OUT (O) - Comparator 2 output
21:20	GPIO42		Configure this pin as:
		00	GPIO42 - general purpose I/O 42 (default)
		01	EPWM8A (O)
		10	$\overline{\text{TZ1}}$ (I)
		11	COMP1OUT (O) - Comparator 1 output
19:18	GPIO41		Configure this pin as:
		00	GPIO41 - general purpose I/O 41 (default)
		01	EPWM7B (O) ePWM7 output B (O)
		10	SCIRXDB (I)
		11	Reserved
17:16	GPIO40		Configure this pin as:
		00	GPIO40 - general purpose I/O 40 (default)
		01	EPWM7A (O) - ePWM7 output A (O)
		10	SCITXDB (O)
		11	Reserved
15:14	GPIO39		Configure this pin as:
		00	GPIO39 - general purpose I/O 39 (default)
		01	Reserved
		10 or 11	Reserved
13:12	GPIO38/XCLKIN (TCK)		Configure this pin as:
		00	GPIO38 - general purpose I/O 38 (default). 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.
		01	Reserved
		10 or 11	Reserved
11:10	GPIO37(TDO)		Configure this pin as:
		00	GPIO37 - general purpose I/O 37 (default). If $\overline{\text{TRST}}$ = 1, JTAG TDO function is chosen for this pin.
		01	Reserved
		10 or 11	Reserved
9:8	GPIO36(TMS)		Configure this pin as:
		00	GPIO36 - general purpose I/O 36 (default). If $\overline{\text{TRST}}$ = 1, JTAG TMS function is chosen for this pin.
		01	Reserved
		10 or 11	Reserved

Table 1-68. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions (continued)

Bit	Field	Value	Description
7:6	GPIO35(TDI)		Configure this pin as:
		00	GPIO35 - general purpose I/O 35 (default). If $\overline{\text{TRST}}$ = 1, JTAG TDI function is chosen for this pin.
		01	Reserved
		10 or 11	Reserved
5:4	GPIO34		Configure this pin as:
		00	GPIO34 - general purpose I/O 34 (default)
		01	COMP2OUT (O) - Comparator 2 output
		10	Reserved
		11	COMP3OUT (O) - Comparator 3 output
3:2	GPIO33		Configure this pin as:
		00	GPIO33 - general purpose I/O 33 (default)
		01	SCLA - I^{2}C clock open drain bidirectional port (I/O)
		10	EPWMSYNCO - External ePWM sync pulse output (O)
		11	$\overline{\text{ADCSOCBO}}$ - ADC start-of-conversion B (O)
1:0	GPIO32		Configure this pin as:
		00	GPIO32 - general purpose I/O 32 (default)
		01	SDAA - I^{2}C data open drain bidirectional port (I/O)
		10	EPWMSYNCI - External ePWM sync pulse input (I)
		11	$\overline{\text{ADCSOCAO}}$ - ADC start-of-conversion A (O)

Figure 1-61. GPIO Port B MUX 2 (GPBMUX2) Register

31–22	21–20	19–18	17–16
Reserved	GPIO58	GPIO57	GPIO56
R-0	R/W-0	R/W-0	R/W-0

15–14	13–12	11–10	9–8	7–6	5–4	3–0
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 1-69. GPIO Port B MUX 2 (GPBMUX2) Register Field Descriptions

Bit	Field	Value	Description
31:22	Reserved		Any writes to these bit(s) must always have a value of 0.
21:20	GPIO58		Configure this pin as:
		00	GPIO58 - general purpose I/O 42 (default)
		01	MCLKRA (I/O)
		10	SCITXDB (O)
		11	EPWM7A (O)
19:18	GPIO57		Configure this pin as:
		00	GPIO57 - general purpose I/O 57 (default)
		01	$\overline{\text{SPISTEA}}$ (I/O)
		10	EQEP2S (I/O)
		11	HRCAP4 (I)

Table 1-69. GPIO Port B MUX 2 (GPBMUX2) Register Field Descriptions (continued)

Bit	Field	Value	Description
17:16	GPIO56		Configure this pin as:
		00	GPIO56 - general purpose I/O 56(default)
		01	SPICLKA (I/O)
		10	EQEP2I (I/O)
		11	HRCAP3 (I)
15:14	GPIO55		Configure this pin as:
		00	GPIO55 - general purpose I/O 55 (default)
		01	SPISOMIA (I/O)
		10	EQEP2B (I)
		11	HRCAP2 (I)
13:12	GPIO54		Configure this pin as:
		00	GPIO54 - general purpose I/O 54 (default).
		01	SPISIMOA (I/O)
		10	EQEP2A (I)
		11	HRCAP1 (I)
11:10	GPIO53		Configure this pin as:
		00	GPIO53 - general purpose I/O 53 (default).
		01	EQEP1I (I/O)
		10	MFSXA (I/O)
		11	Reserved
9:8	GPIO52		Configure this pin as:
		00	GPIO52 - general purpose I/O 52 (default).
		01	EQEP1S (I/O)
		10	MCLKXA (I/O)
		11	$\overline{\text{TZ3}}$ (I)
7:6	GPIO51		Configure this pin as:
		00	GPIO51 - general purpose I/O 51 (default).
		01	EQEP1B (I)
		10	MDRA (I)
		11	$\overline{\text{TZ2}}$ (I)
5:4	GPIO50		Configure this pin as:
		00	GPIO50 - general purpose I/O 50 (default)
		01	EQEP1A (I)
		10	MDXA (O)
		11	$\overline{\text{TZ1}}$ (I)
3:0	Reserved		Any writes to these bit(s) must always have a value of 0.

Figure 1-62. 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 1-70. Analog I/O MUX (AIOMUX1) Register Field Descriptions

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

Figure 1-63. GPIO Port A Qualification Control (GPACTRL) Register

31 – 24	23 – 16
QUALPRD3	QUALPRD2
R/W-0	R/W-0

15 – 8	7 – 0
QUALPRD1	QUALPRD0
R/W-0	R/W-0

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

The GPxCTRL registers specify the sampling period for input pins when configured for input qualification using a window of three or six 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 GPxQSELn registers.

Table 1-71. GPIO Port A Qualification Control (GPACTRL) Register Field Descriptions

Bits	Field	Value	Description [(1)](#)
31-24	QUALPRD3		Specifies the sampling period for pins GPIO24 to GPIO31.
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [(2)]
		0x01	Sampling Period = $2 \times T_{SYSCLKOUT}$
		0x02	Sampling Period = $4 \times T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	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.

Table 1-71. GPIO Port A Qualification Control (GPACTRL) Register Field Descriptions (continued)

Bits	Field	Value	Description [(1)]
23-16	QUALPRD2		Specifies the sampling period for pins GPIO16 to GPIO23.
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [(2)]
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
15-8	QUALPRD1		Specifies the sampling period for pins GPIO8 to GPIO15.
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [(2)]
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
7-0	QUALPRD0		Specifies the sampling period for pins GPIO0 to GPIO7.
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [(2)]
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$

Figure 1-64. GPIO Port B Qualification Control (GPBCTRL) Register

31–24	23–16
QUALPRD3	QUALPRD2
R/W-0	R/W-0

15–8	7–0
QUALPRD1	QUALPRD0
R/W-0	R/W-0

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

Table 1-72. GPIO Port B Qualification Control (GPBCTRL) Register Field Descriptions

Bits	Field	Value	Description [1]
31-24	QUALPRD3		Specifies the sampling period for pins GPIO56 to GPIO58
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [2]
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
23-16	QUALPRD2		Specifies the sampling period for pins GPIO50 to GPIO55
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [2]
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
15-8	QUALPRD1		Specifies the sampling period for pins GPIO40 to GPIO44
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [2]
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
7-0	QUALPRD0		Specifies the sampling period for pins GPIO32 to GPIO39
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
		0x00	Sampling Period = $T_{SYSCLKOUT}$ [2]
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$

[1] This register is EALLOW protected. See Section 1.5.2 for more information.
[2] $T_{SYSCLKOUT}$ indicates the period of SYSCLKOUT.

Figure 1-65. GPIO A Control Register 2 Register (GPACTRL2) Register

31–16
Reserved
R-0

15–1	0
Reserved	USBIOEN
R-0	R/W-0

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

Table 1-73. (GPACTRL2) Register Field Descriptions

Bits	Field	Value	Description
31-1	Reserved		Any writes to these bit(s) must always have a value of 0.
0	USBIOEN		USB I/O Enable Bit
		0	USB0DP and USB0DM pins are controlled by GPIO Mux register settings. USBPHY is powered down.
		1	USB0DP and USB0DM pins configured as USB function. GPIO function is disabled.

Figure 1-66. 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 1-74. GPIO Port A Qualification Select 1 (GPAQSEL1) Register Field Descriptions

Bits	Field	Value	Description [(1)]
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 Figure 1-66.
		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.

Figure 1-67. 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 1-75. GPIO Port A Qualification Select 2 (GPAQSEL2) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-0	GPIO31-GPIO16		Select input qualification type for GPIO16 to GPIO31. The input qualification of each GPIO input is controlled by two bits as shown in Figure 1-67.
		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.

Figure 1-68. GPIO Port B Qualification Select 1 (GPBQSEL1) Register

31–26	25–24	23–22	21–20	19–18	17–16
Reserved	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
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
GPIO39	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 1-76. GPIO Port B Qualification Select 1 (GPBQSEL1) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31- 26	Reserved		Any writes to these bit(s) must always have a value of 0.
25-0	GPIO 44-GPIO32		Select input qualification type for GPIO32 to GPIO44. The input qualification of each GPIO input is controlled by two bits as shown in Figure 1-68 .
		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.

Figure 1-69. GPIO Port B Qualification Select 2 (GPBQSEL2) Register

31-22	21-20	19-18	17-16
Reserved	GPIO58	GPIO57	GPIO56
R/W-0	R/W-0	R/W-0	R/W-0

15-14	13-12	11-10	9-8	7-6	5-4	3-0
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	Reserved
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 1-77. GPIO Port B Qualification Select 2 (GPBQSEL2) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31- 22	Reserved		Any writes to these bit(s) must always have a value of 0.
21-0	GPIO58-GPIO50		Select input qualification type for GPIO58 to GPIO50. The input qualification of each GPIO input is controlled by two bits as shown in Figure 1-69 .
		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.

The GPADIR and GPBDIR registers control 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 1-70. 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; R = Read only; -*n* = value after reset

Table 1-78. GPIO Port A Direction (GPADIR) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-0	GPIO31-GPIO0		Controls direction of GPIO Port A pins when the specified pin is configured as a GPIO in the appropriate GPAMUX1 or GPAMUX2 register.
		0	Configures the GPIO pin as an input. (default)
		1	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.

Figure 1-71. GPIO Port B Direction (GPBDIR) Register

31–27	26	25	24
Reserved	GPIO58	GPIO57	GPIO56
R-0	R/W-0	R/W-0	R/W-0

23	22	21	20	19	18	17–16
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

15–13	12	11	10	9	8
Reserved	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
GPIO39	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 1-79. GPIO Port B Direction (GPBDIR) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-27	Reserved		Any writes to these bit(s) must always have a value of 0.
26-18	GPIO58-GPIO50		Controls direction of GPIO pin when GPIO mode is selected. Reading the register returns the current value of the register setting.
		0	Configures the GPIO pin as an input. (default)
		1	Configures the GPIO pin as an output
17-13	Reserved		Any writes to these bit(s) must always have a value of 0.
12-0	GPIO44-GPIO32		Controls direction of GPIO pin when GPIO mode is selected. Reading the register returns the current value of the register setting.
		0	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.

Figure 1-72. Analog I/O DIR (AIODIR) Register

31	16
Reserved	
R-0	

15	14	13	12	11	10	9	8
Reserved	AIO14	Reserved	AIO12	Reserved	AIO10	Reserved	
R-0	R/W-x	R-0	R/W-x	R-0	R/W-x	R-0	

7	6	5	4	3	2	1	0
Reserved	AIO6	Reserved	AIO4	Reserved	AIO2	Reserved	
R-0	R/W-x	R-0	R/W-x	R-0	R/W-x	R-0	

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

Table 1-80. Analog I/O DIR (AIODIR) Register Field Descriptions

Bit	Field	Value	Description
31:15	Reserved		Any writes to these bit(s) must always have a value of 0.
14:0	AIOn		Controls direction of the available AIO pin when AIO mode is selected. Reading the register returns the current value of the register setting
		0	Configures the AIO pin as an input. (default)
		1	Configures the AIO pin as an output

The pullup disable (GPxPUD) registers allow you to specify which pins should have an internal pullup resister 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 1-73. GPIO Port A Pullup Disable (GPAPUD) 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-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 1-81. GPIO Port A Internal Pullup Disable (GPAPUD) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-0	GPIO31-GPIO0		Configure the internal pullup resister 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.

Figure 1-74. GPIO Port B Pullup Disable (GPBPUD) Registers

31				27	26	25	24
Reserved					GPIO58	GPIO57	GPIO56
R-0					R/W-0	R/W-0	R/W-0

23	22	21	20	19	18	17	16
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	Reserved	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	

15		13	12	11	10	9	8
Reserved			GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
R-0			R/W-0	R/W-0	R/W-0	R/W-1	R/W-1

7	6	5	4	3	2	1	0
GPIO39	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 1-82. GPIO Port B Internal Pullup Disable (GPBPUD) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31- 27	Reserved		Any writes to these bit(s) must always have a value of 0.
26-18	GPIO58-GPIO50		Configure the internal pullup resister on the selected GPIO Port B pin. Each GPIO pin corresponds to one bit in this register.
		0	Enable the internal pullup on the specified pin. (default)
		1	Disable the internal pullup on the specified pin.
17-13	Reserved		Any writes to these bit(s) must always have a value of 0.
12-0	GPIO44-GPIO32		Configure the internal pullup resister on the selected GPIO Port B pin. Each GPIO pin corresponds to one bit in this register.
		0	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.

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 1-75. 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[(1)]

(1) x = The state of the GPADAT register is unknown after reset. It depends on the level of the pin after reset.

Table 1-83. GPIO Port A Data (GPADAT) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO31-GPIO0		Each bit corresponds to one GPIO port A pin (GPIO0-GPIO31) as shown in Figure 1-75.
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured for.
			Writing a 0 will force 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 for.
			Writing a 1will force an output of 1if 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.

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

31-27	26	25	24
Reserved	GPIO58	GPIO57	GPIO56
R-0	R/W-0	R/W-0	R/W-0

23	22	21	20	19	18	17-16
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

15-13	12	11	10	9	8
Reserved	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
GPIO39	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 1-84. GPIO Port B Data (GPBDAT) Register Field Descriptions

Bit	Field	Value	Description
31-27	Reserved		Any writes to these bit(s) must always have a value of 0.
26-18	GPIO 58-GPIO50		Each bit corresponds to one GPIO port B pin (GPIO58-GPIO50) as shown in Figure 1-76.
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured for.
			Writing a 0 will force 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 for.
			Writing a 1 will force 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.
17-13	Reserved		Any writes to these bit(s) must always have a value of 0.
12-0	GPIO44-GPIO32		Each bit corresponds to one GPIO port B pin (GPIO44-GPIO32) as shown in Figure 1-76.
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured for.
			Writing a 0 will force 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 for.
			Writing a 1 will force 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.

Figure 1-77. Analog I/O DAT (AIODAT) Register

31-16
Reserved
R-0

15	14	13	12	11	10	9-8
Reserved	AIO14	Reserved	AIO12	Reserved	AIO10	Reserved
R-0	R/W-x	R-0	R/W-x	R-0	R/W-x	R-0

7	6	5	4	3	2	1-0
Reserved	AIO6	Reserved	AIO4	Reserved	AIO2	Reserved
R-0	R/W-x	R-0	R/W-x	R-0	R/W-x	R-0

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

Table 1-85. Analog I/O DAT (AIODAT) Register Field Descriptions

Bit	Field	Value	Description
31:15	Reserved		Any writes to these bit(s) must always have a value of 0.
14-0	AIOn		Each bit corresponds to one AIO port pin
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured for.
			Writing a 0 will force 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 for.
			Writing a 1will force an output of 1if the pin is configured as a AIO output in the appropriate registers; otherwise, the value is latched but not used to drive the pin.

Figure 1-78. 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 1-86. GPIO Port A Set (GPASET) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO31-GPIO0		Each GPIO port A pin (GPIO0-GPIO31) corresponds to one bit in this register as shown in Figure 1-78.
		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 1-87. GPIO Port A Clear (GPACLEAR) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO31 - GPIO0		Each GPIO port A pin (GPIO0-GPIO31) corresponds to one bit in this register as shown in Figure 1-78.
		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 1-88. GPIO Port A Toggle (GPATOGGLE) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO31-GPIO0		Each GPIO port A pin (GPIO0-GPIO31) corresponds to one bit in this register as shown in Figure 1-78.
		0	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.

Figure 1-79. GPIO Port B Set, Clear and Toggle (GPBSET, GPBCLEAR, GPBTOGGLE) Registers

31				27	26	25	24
Reserved					GPIO58	GPIO57	GPIO56
R-0					R/W-0	R/W-0	R/W-0

23	22	21	20	19	18	17	16
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	Reserved	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	

15		13	12	11	10	9	8
Reserved			GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
R-0			R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
GPIO39	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 1-89. GPIO Port B Set (GPBSET) Register Field Descriptions

Bits	Field	Value	Description
31- 27	Reserved		Any writes to these bit(s) must always have a value of 0.
26-18	GPIO58 -GPIO50		Each GPIO port B pin (GPIO58-GPIO50) corresponds to one bit in this register as shown in Figure 1-79.
		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 but the pin is not driven.
17-13	Reserved		Any writes to these bit(s) must always have a value of 0.
12-0	GPIO44-GPIO32		Each GPIO port B pin (GPIO44-GPIO32) corresponds to one bit in this register as shown in Figure 1-79.
		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 but the pin is not driven.

Table 1-90. GPIO Port B Clear (GPBCLEAR) Register Field Descriptions

Bits	Field	Value	Description
31- 27	Reserved		Any writes to these bit(s) must always have a value of 0.
26-18	GPIO58-GPIO50		Each GPIO port B pin (GPIO58-GPIO50) corresponds to one bit in this register as shown in Figure 1-79.
		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.
17-13	Reserved		Any writes to these bit(s) must always have a value of 0.
12-0	GPIO44-GPIO32		Each GPIO port B pin (GPIO32-GPIO 44) corresponds to one bit in this register as shown in Figure 1-79.
		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 1-91. GPIO Port B Toggle (GPBTOGGLE) Register Field Descriptions

Bits	Field	Value	Description
31- 27	Reserved		Any writes to these bit(s) must always have a value of 0.
26-18	GPIO58 -GPIO50		Each GPIO port B pin (GPIO58-GPIO50) corresponds to one bit in this register as shown in Figure 1-79.
		0	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 cleared but the pin is not driven.
17-13	Reserved		Any writes to these bit(s) must always have a value of 0.
12-0	GPIO44-GPIO32		Each GPIO port B pin (GPIO44-GPIO32) corresponds to one bit in this register as shown in Figure 1-79.
		0	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 cleared but the pin is not driven.

Figure 1-80. Analog I/O Toggle (AIOSET, AIOCLEAR, AIOTOGGLE) Register

31–16
Reserved
R-0

15	14	13	12	11	10	9–8
Reserved	AIO14	Reserved	AIO12	Reserved	AIO10	Reserved
R-0	R/W-x	R-0	R/W-x	R-0	R/W-x	R-0

7	6	5	4	3	2	1–0
Reserved	AIO6	Reserved	AIO4	Reserved	AIO2	Reserved
R-0	R/W-x	R-0	R/W-x	R-0	R/W-x	R-0

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

Table 1-92. Analog I/O Set (AIOSET) Register Field Descriptions

Bits	Field	Value	Description
31-15	Reserved		Any writes to these bit(s) must always have a value of 0.
14-0	AIOn		Each AIO pin corresponds to one bit in this register.
		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 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 1-93. Analog I/O Clear (AIOCLEAR) Register Field Descriptions

Bits	Field	Value	Description
31-15	Reserved		Any writes to these bit(s) must always have a value of 0.
14-0	AIOn		Each AIO pin corresponds to one bit in this register.
		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 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 1-94. Analog I/O Toggle (AIOTOGGLE) Register Field Descriptions

Bits	Field	Value	Description
31-15	Reserved		Any writes to these bit(s) must always have a value of 0.
14-0	AIOn		Each AIO pin corresponds to one bit in this register.
		0	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 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.

Figure 1-81. GPIO XINTn Interrupt Select (GPIOXINTnSEL) Registers

15 - 5	4 - 0
Reserved	GPIOXINTnSEL
R-0	R/W-0

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

Table 1-95. GPIO XINTn Interrupt Select (GPIOXINTnSEL)[1] Register Field Descriptions

Bits	Field	Value	Description [2]
15-5	Reserved		Any writes to these bit(s) must always have a value of 0.
4-0	GPIOXINTnSEL		Select the port A GPIO signal (GPIO0 - GPIO31) that will be 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 Section 1.6.6. To use XINT2 as ADC start of conversion, enable it in the desired ADCSOCxCTL register. The $\overline{\text{ADCSOC}}$ 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] n = 1 or 2
[2] This register is EALLOW protected. See Section 1.5.2 for more information.

Table 1-96. 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

Figure 1-82. 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; R = Read only; -*n* = value after reset

Table 1-97. GPIO Low Power Mode Wakeup Select (GPIOLPMSEL) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-0	GPIO31 - GPIO0		Low Power Mode Wakeup Selection. Each bit in this register corresponds to one GPIO port A pin (GPIO0 - GPIO31) as shown in Figure 10-3.
		0	If the bit is cleared, the signal on the corresponding pin will have 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.

1.5 Peripheral Frames

This chapter describes the peripheral frames and device emulation registers.

1.5.1 Peripheral Frame Registers

This device contains four peripheral register spaces. The spaces are categorized as follows:

- Peripheral Frame 0: These are peripherals that are mapped directly to the CPU memory bus. See Table 1-98.
- Peripheral Frame 1: These are peripherals that are mapped to the 32-bit peripheral bus. See Table 1-99.
- Peripheral Frame 2: These are peripherals that are mapped to the 16-bit peripheral bus. See Table 1-100.
- Peripheral Frame 3: These are peripherals that are mapped to the 16-bit peripheral bus. See Table 1-101.

Table 1-98. Peripheral Frame 0 Registers(1)

NAME	ADDRESS RANGE	SIZE (×16)	EALLOW PROTECTED(2)
Device Emulation Registers	0x00 0880 – 0x00 0984	261	Yes
System Power Control Registers	0x00 0985 – 0x00 0987	3	Yes
FLASH Registers(3)	0x00 0A80 – 0x00 0ADF	96	Yes
Code Security Module Registers	0x00 0AE0 – 0x00 0AEF	16	Yes
ADC registers (0 wait read only)	0x00 0B00 – 0x00 0B0F	16	No
CPU–TIMER0/1/2 Registers	0x00 0C00 – 0x00 0C3F	64	No
PIE Registers	0x00 0CE0 – 0x00 0CFF	32	No
PIE Vector Table	0x00 0D00 – 0x00 0DFF	256	No
DMA Registers	0x00 1000 – 0x00 11FF	512	Yes
CLA Registers	0x00 1400 – 0x00 147F	128	Yes
CLA to CPU Message RAM (CPU writes ignored)	0x00 1480 – 0x00 14FF	128	NA
CPU to CLA Message RAM (CLA writes ignored)	0x00 1500 – 0x00 157F	128	NA

(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 1-99. Peripheral Frame 1 Registers[1]

NAME	ADDRESS RANGE	SIZE (×16)	EALLOW PROTECTED[2]
eCAN-A registers	0x00 6000 – 0x00 61FF	512	[3]
Comparator 1 registers	0x00 6400 – 0x00 641F	32	[3]
Comparator 2 registers	0x00 6420 – 0x00 643F	32	[3]
Comparator 3 registers	0x00 6440 – 0x00 645F	32	[3]
ePWM1 + HRPWM1 registers	0x00 6800 – 0x00 683F	64	[3]
ePWM2 + HRPWM2 registers	0x00 6840 – 0x00 687F	64	[3]
ePWM3 + HRPWM3 registers	0x00 6880 – 0x00 68BF	64	[3]
ePWM4 + HRPWM4 registers	0x00 68C0 – 0x00 68FF	64	[3]
ePWM5 + HRPWM5 registers	0x00 6900 – 0x00 693F	64	[3]
ePWM6 + HRPWM6 registers	0x00 6940 – 0x00 697F	64	[3]
ePWM7 + HRPWM7 registers	0x00 6980 – 0x00 69BF	64	[3]
ePWM8 + HRPWM8 registers	0x00 69C0 – 0x00 69FF	64	[3]
eCAP1 registers	0x00 6A00 – 0x00 6A1F	32	No
eCAP2 registers	0x00 6A20 – 0x00 6A3F	32	No
eCAP3 registers	0x00 6A40 – 0x00 6A57	32	No
HRCAP1 registers	0x00 6AC0 – 0x00 6ADF	32	[3]
HRCAP2 registers	0x00 6AE0 – 0x00 6AFF	32	[3]
eQEP1 registers	0x00 6B00 – 0x00 6B3F	64	[3]
eQEP2 registers	0x00 6B40 – 0x00 6B7F	64	[3]
HRCAP3 registers	0x00 6C80 – 0x00 6C9F	32	[3]
HRCAP4 registers	0x00 6CA0 – 0x00 6CBF	32	[3]
GPIO registers	0x00 6F80 – 0x00 6FFF	128	[3]

[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 are EALLOW protected. See the module reference guide for more information.

Table 1-100. Peripheral Frame 2 Registers

NAME	ADDRESS RANGE	SIZE (×16)	EALLOW PROTECTED
System Control Registers	0x00 7010 – 0x00 702F	32	Yes
SPI-A Registers	0x00 7040 – 0x00 704F	16	No
SCI-A Registers	0x00 7050 – 0x00 705F	16	No
NMI Watchdog Interrupt Registers	0x00 7060 – 0x00 706F	16	Yes
External Interrupt Registers	0x00 7070 – 0x00 707F	16	Yes
ADC Registers	0x00 7100 – 0x00 717F	128	[1]
SPI-B Registers	0x00 7740 – 0x00 774F	16	No
SCI-B Registers	0x00 7750 – 0x00 775F	16	No
I2C-A Registers	0x00 7900 – 0x00 793F	64	[1]

[1] Some registers are EALLOW protected. See the module reference guide for more information.

Table 1-101. Peripheral Frame 3 Registers

NAME	ADDRESS RANGE	SIZE (×16)	EALLOW PROTECTED
USB0 Registers	0x00 4000 – 0x4FFF	4096	No
McBSP-A Registers	0x00 5000 – 0x00 503F	64	No

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 1-102.

Table 1-102. Access to EALLOW-Protected Registers

EALLOW Bit	CPU Writes	CPU Reads	JTAG Writes	JTAG Reads
0	Ignored	Allowed	Allowed(1)	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 interface.

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
- Flash Registers
- CSM Registers
- PIE Vector Table
- System Control Registers
- GPIO MUX Registers

Table 1-103. EALLOW-Protected Device Emulation Registers

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

Table 1-104. 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 1-105. EALLOW-Protected Code Security Module (CSM) Registers

Register Name	Address	Size (x16)	Register 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 1-106. EALLOW-Protected PLL, Clocking, Watchdog, and Low-Power Mode Registers

NAME	ADDRESS	SIZE (x16)	DESCRIPTION
BORCFG	0x00 0985	1	BOR Configuration Register
XCLK	0x00 7010	1	XCLKOUT Control
PLLSTS	0x00 7011	1	PLL Status Register
CLKCTL	0x00 7012	1	Clock Control Register
PLLLOCKPRD	0x00 7013	1	PLL Lock Period
INTOSC1TRIM	0x00 7014	1	Internal Oscillator 1 Trim Register
INTOSC2TRIM	0x00 7016	1	Internal Oscillator 2 Trim Register
PCLKCR2	0x00 7019	1	Peripheral Clock Control Register 2
LOSPCP	0x00 701B	1	Low-Speed Peripheral Clock Prescaler Register
PCLKCR0	0x00 701C	1	Peripheral Clock Control Register 0
PCLKCR1	0x00 701D	1	Peripheral Clock Control Register 1
LPMCR0	0x00 701E	1	Low Power Mode Control Register 0
PCLKCR3	0x00 7020	1	Peripheral Clock Control Register 3
PLLCR	0x00 7021	1	PLL Control Register
SCSR	0x00 7022	1	System Control and Status Register
WDCNTR	0x00 7023	1	Watchdog Counter Register
WDKEY	0x00 7025	1	Watchdog Reset Key Register
WDCR	0x00 7029	1	Watchdog Control Register
PLL2CTL	0x00 7030	1	PLL2 Configuration Register
PLL2MULT	0x00 7032	1	PLL2 Multiplier Register
PLL2STS	0x00 7034	1	PLL2 Lock Status Register
SYSCLK2CNTR	0x00 7036	1	SYSCLK2 Clock Counter Register
EPWMCFG	0x00 703A	1	ePWM DMA/CLA Configuration Register

Table 1-107. EALLOW-Protected GPIO Registers

Name [1]	Address	Size (x16)	Register Description
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
GPBQSEL2	0x6F94	2	GPIO B Qualifier Select 2 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 1-109 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 1-108. 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 1-109. EALLOW-Protected ePWM1 - ePWM 7 Registers

	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
ePWM5	0x6912	0x6914	0x6915	0x6917	0x6918	0x6920	1
ePWM6	0x6952	0x6954	0x6955	0x6957	0x6958	0x6960	1
ePWM7	0x6992	0x6994	0x6995	0x6997	0x6998	0x69A0	1

1.5.3 Device Emulation Registers

These registers are used to control the protection mode of the C28x CPU and to monitor some critical device signals. The registers are defined in Table 1-110.

Table 1-110. Device Emulation Registers

Name	Address	Size (x16)	Description
DEVICECNF	0x0880 0x0881	2	Device Configuration Register
PARTID	0x3D 7E80	1	Part ID Register
CLASSID	0x0882	1	Class ID Register
REVID	0x0883	1	Revision ID Register

Figure 1-83. Device Configuration (DEVICECNF) Register

31	30	29-28	27	26-20	19	18-16
Rsvd	SYSCLK2DIV2DIS	Reserved	TRST	Reserved	ENPROT	Reserved
R-0	R/W-1	R-0	R-0	R-0	R/W-1	R-111

15-6	5	4	3	2-0
Reserved	$\overline{\text{XRS}}$	Res	VMAPS	Reserved
R-0	R-P	R-0	R-1	R-011

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

Table 1-111. DEVICECNF Register Field Descriptions

Bits	Field	Value	Description
31	Reserved		Any writes to these bit(s) must always have a value of 0.
30	SYSCLK2DIV2DIS		SYSCLK2 Clock Divide by 2 Disable Bit
		0	PLL2 Output/2
		1	PLL2 Output/1
29-28	Reserved		
27	$\overline{\text{TRST}}$		Read status of $\overline{\text{TRST}}$ signal. Reading this bit gives the current status of the $\overline{\text{TRST}}$ signal.
		0	No emulator is connected.
		1	An emulator is connected.
26:20	Reserved		Any writes to these bit(s) must always have a value of 0.
19	ENPROT		Enable Write-Read Protection Mode Bit.
		0	Disables write-read protection mode
		1	Enables write-read protection for the address range 0x4000-0x7FFF
18-6	Reserved		Any writes to these bit(s) must always have a value of 0.
5	$\overline{\text{XRS}}$		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		Any writes to these bit(s) must always have a value of 0.

Figure 1-84. Part ID Register

15 … 8	7 … 0
PARTTYPE	PARTID
R	R

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

Table 1-112. PARTID Register Field Descriptions

Bit	Field	Value[(1)]	Description	
15:8	PARTTYPE		Part type ID register. Specifies the type of device such as flash-based.	
		0x00	Flash-based device	
			All other values are reserved.	
7:0	PARTID[(2)]	0x3D 7E80	Part ID Register. These 8 bits specify the feature set of this device:	
			TMS320F28069PZP/PZ	0x9E
			TMS320F28069**U**PZP/ PZ	0x9F
			TMS320F28069PFP/PN	0x9C
			TMS320F28069**U**PFP/PN	0x9D
			TMS320F28068PZP/PZ	0x8E
			TMS320F28068**U**PZP/PZ	0x8F
			TMS320F28068PFP/PN	0x8C
			TMS320F28068**U**PFP/PN	0x8D
			TMS320F28067PZP/PZ	0x8A
			TMS320F28067**U**PZP/PZ	0x8B
			TMS320F28067PFP/PN	0x88
			TMS320F28067**U**PFP/PN	0x89
			TMS320F28066PZP/PZ	0x86
			TMS320F28066**U**PZP/PZ	0x87
			TMS320F28066PFP/PN	0x84
			TMS320F28066**U**PFP/PN	0x85
			TMS320F28065PZP/PZ	0x7E
			TMS320F28065**U**PZP/PZ	0x7F
			TMS320F28065PFP/PN	0x7C
			TMS320F28065**U**PFP/PN	0x7D
			TMS320F28064PZP/PZ	0x6E
			TMS320F28064**U**PZP/PZ	0x6F
			TMS320F28064PFP/PN	0x6C
			TMS320F28064**U**PFP/PN	0x6D
			TMS320F28063PZP/PZ	0x6A
			TMS320F28063**U**PZP/PZ	0x6B
			TMS320F28063PFP/PN	0x68
			TMS320F28063**U**PFP/PN	0x69
			TMS320F28062PZP/PZ	0x66
			TMS320F28062**U**PZP/PZ	0x67
			TMS320F28062PFP/PN	0x65
			TMS320F28062**U**PFP/PN	0x64

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

(2) For TMS320F28069**U** devices, the PARTID/CLASSID numbers are also used for TMX devices. In the case of TMX320F28069UPFPA and TMX320F28069UPZPA devices, the temperature rating is "A" instead of "T".

Table 1-113. CLASSID Register Field Descriptions

Bit	Field	Value[1]	Description	
7:0	CLASSID	0x0882	Class ID register. These 8 bits specify the feature set of this device:	
			TMS320F28069	0x009F
			TMS320F28068	0x008F
			TMS320F28067	0x008F
			TMS320F28066	0x008F
			TMS320F28065	0x007F
			TMS320F28064	0x006F
			TMS320F28063	0x006F
			TMS320F28062	0x006F

[1] The reset value depends on the device as indicated in the register description.

Figure 1-85. REVID Register

15 — 0
REVID
R-[1]

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

[1] The reset value depends on the silicon revision as described in the register field description.

Table 1-114. REVID Register Field Descriptions

Bits	Field	Value	Description
15-0	REVID		[1]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.
		0x0000	Silicon Revision 0 - TMX
		0x0001	Silicon Revision A - 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 then 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 then 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

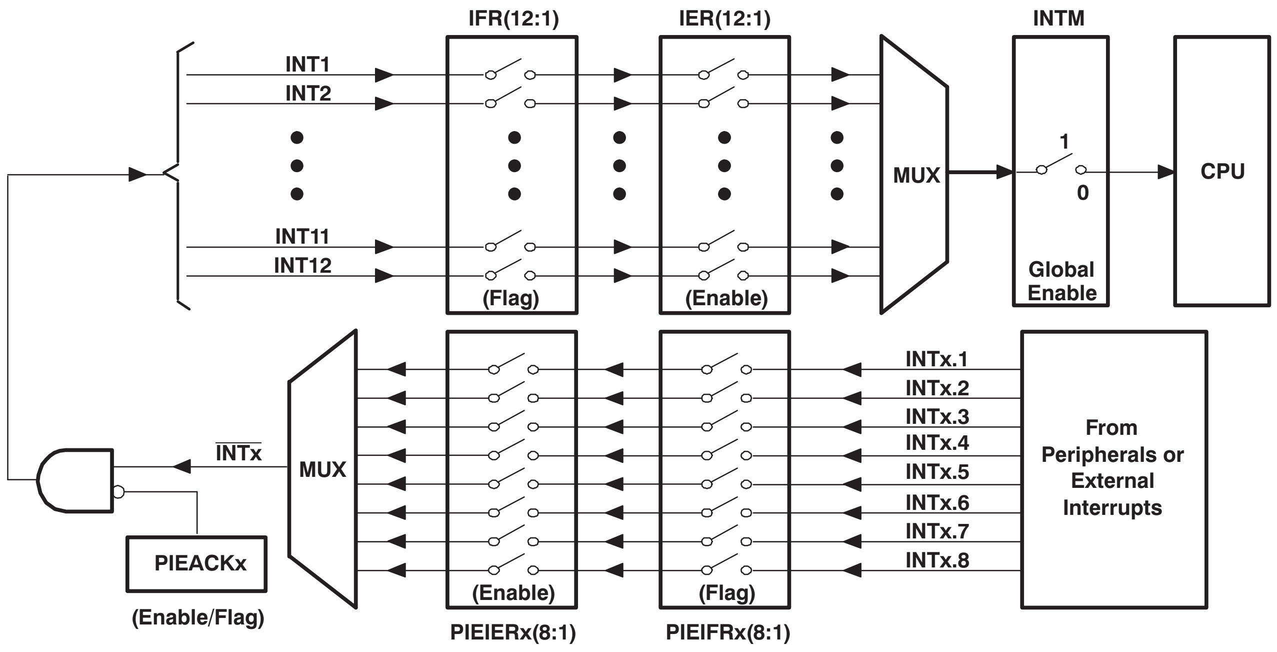
The 28x CPU supports one nonmaskable interrupt (NMI) and 16 maskable prioritized interrupt requests (INT1-INT14, RTOSINT, and DLOGINT) at the CPU level. The 28x devices have many peripherals and each peripheral is capable of generating one or more interrupts in response to many events at the peripheral 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 1-86 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 1-86. 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, then 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.

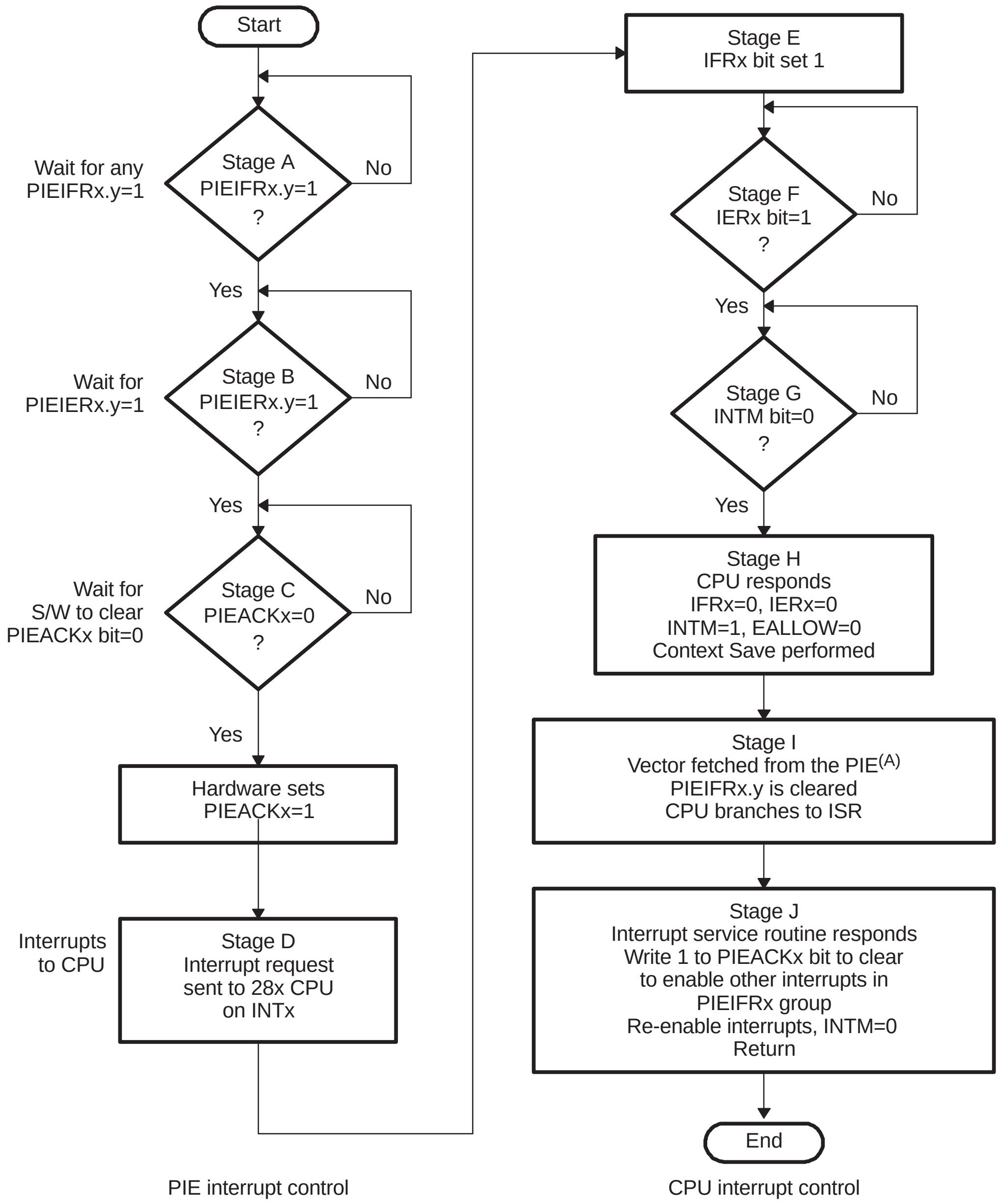
  For multiplexed interrupt sources, each interrupt group in the PIE block has an associated flag register (PIEIFRx) and enable (PIEIERx) register (x = PIE group 1 - PIE group 12). Each bit, referred to as y, corresponds to one of the 8 MUXed interrupts within the group. Thus PIEIFRx.y and PIEIERx.y correspond 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 1-87 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 then 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, then the PIE sends the interrupt request to the CPU. If PIEACKx is set, then the PIE waits until it is cleared to send the request for INTx. See Section 6.3 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.

Figure 1-87. Typical PIE/CPU Interrupt Response - INTx.y


Start
Wait for any PIEIFRx.y=1
Stage A PIEIFRx.y=1 ?
No
Yes
Wait for PIEIERx.y=1
Stage B PIEIERx.y=1 ?
No
Yes
Wait for S/W to clear PIEACKx bit=0
Stage C PIEACKx=0 ?
No
Yes
Hardware sets PIEACKx=1
Interrupts to CPU
Stage D Interrupt request sent to 28x CPU on INTx
Stage E IFRx bit set 1
Stage F IERx bit=1 ?
No
Yes
Stage G INTM bit=0 ?
No
Yes
Stage H CPU responds IFRx=0, IERx=0 INTM=1, EALLOW=0 Context Save performed
Stage I Vector fetched from the PIE(A) PIEIFRx.y is cleared CPU branches to ISR
Stage J Interrupt service routine responds Write 1 to PIEACKx bit to clear to enable other interrupts in PIEIFRx group Re-enable interrupts, INTM=0 Return
End
PIE interrupt control
CPU interrupt control

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 Section 1.6.3.3 for details.

As shown in Table 1-115, 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 DSP is in real-time mode and the CPU is running, the standard interrupt-handling process applies.

Table 1-115. Enabling Interrupt

Interrupt Handling Process	Interrupt Enabled If…
Standard	INTM = 0 and bit in IER is 1
DSP 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 detail in *TMS320x28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430). 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 detail in Section 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 1-116.

Table 1-116. Interrupt Vector Table Mapping

Vector MAPS	Vectors Fetched From	Address Range	VMAP	M0M1MAP	ENPIE
M1 Vector[1]	M1 SARAM Block	0x000000 - 0x00003F	0	0	X
M0 Vector[1]	M0 SARAM Block	0x000000 - 0x00003F	0	1	X
BROM Vector	Boot ROM Block	0x3FFFC0 - 0x3FFFFF	1	X	0
PIE Vector	PIE Block	0x000D00 - 0x000DFF	1	X	1

(1) 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 1-117.

Table 1-117. Vector Table Mapping After Reset Operation

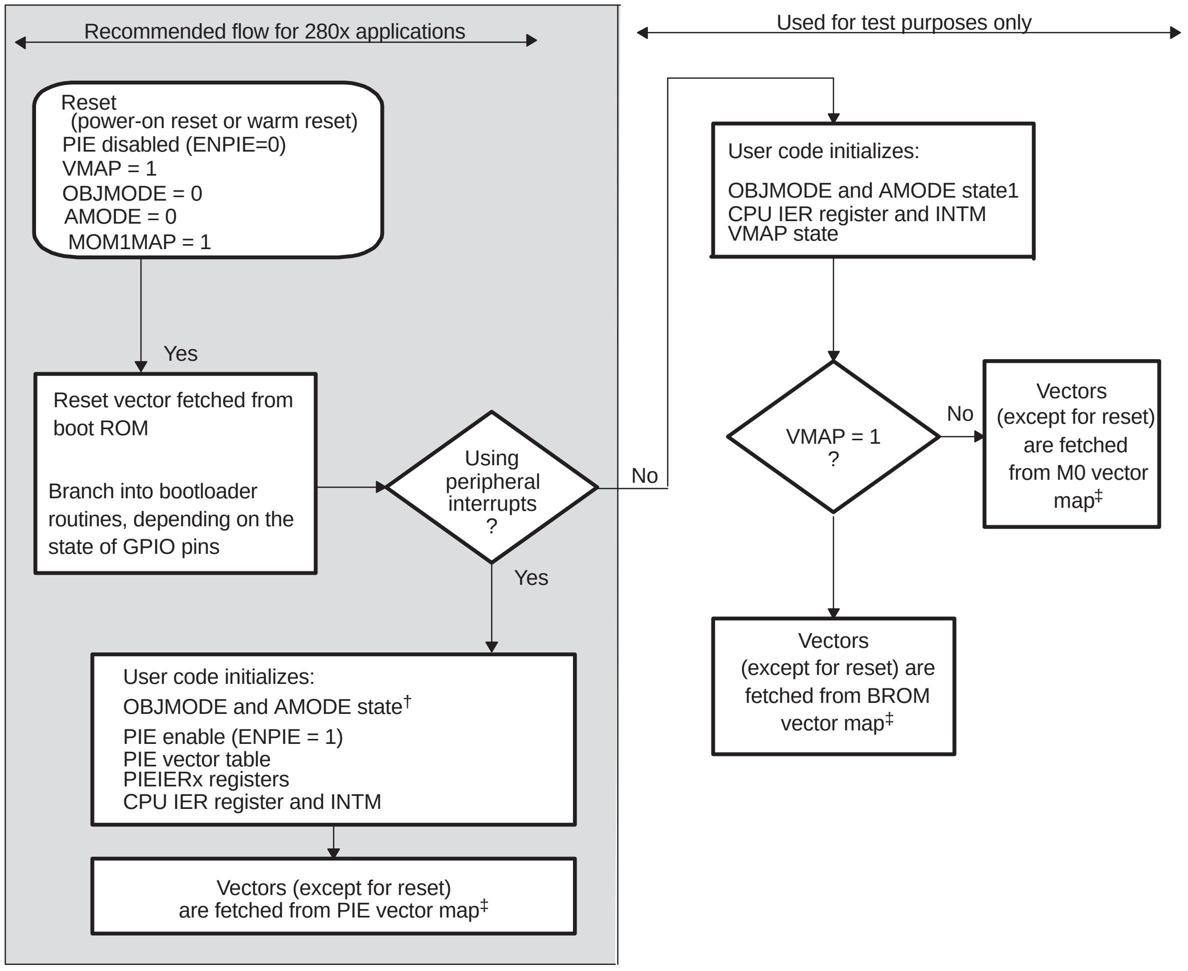
Vector MAPS	Reset Fetched From	Address Range	VMAP [1]	M0M1MAP [1]	ENPIE [1]
BROM Vector [2]	Boot ROM Block	0x3FFFC0 - 0x3FFFFF	1	1	0

(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 1-117. After a reset the PIE vector table is always disabled.

Figure 1-88 illustrates the process by which the vector table mapping is selected.

Figure 1-88. Reset Flow Diagram


Recommended flow for 280x applications
Used for test purposes only
Reset
(power-on reset or warm reset)
PIE disabled (ENPIE=0)
VMAP = 1
OBJMODE = 0
AMODE = 0
MOM1MAP = 1
Yes
Reset vector fetched from boot ROM
Branch into bootloader routines, depending on the state of GPIO pins
Using peripheral interrupts ?
No
Yes
User code initializes:
OBJMODE and AMODE state†
PIE enable (ENPIE = 1)
PIE vector table
PIEIERx registers
CPU IER register and INTM
Vectors (except for reset) are fetched from PIE vector map‡
User code initializes:
OBJMODE and AMODE state1
CPU IER register and INTM
VMAP state
VMAP = 1 ?
No
Vectors (except for reset) are fetched from M0 vector map‡
Vectors (except for reset) are fetched from BROM vector map‡

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):

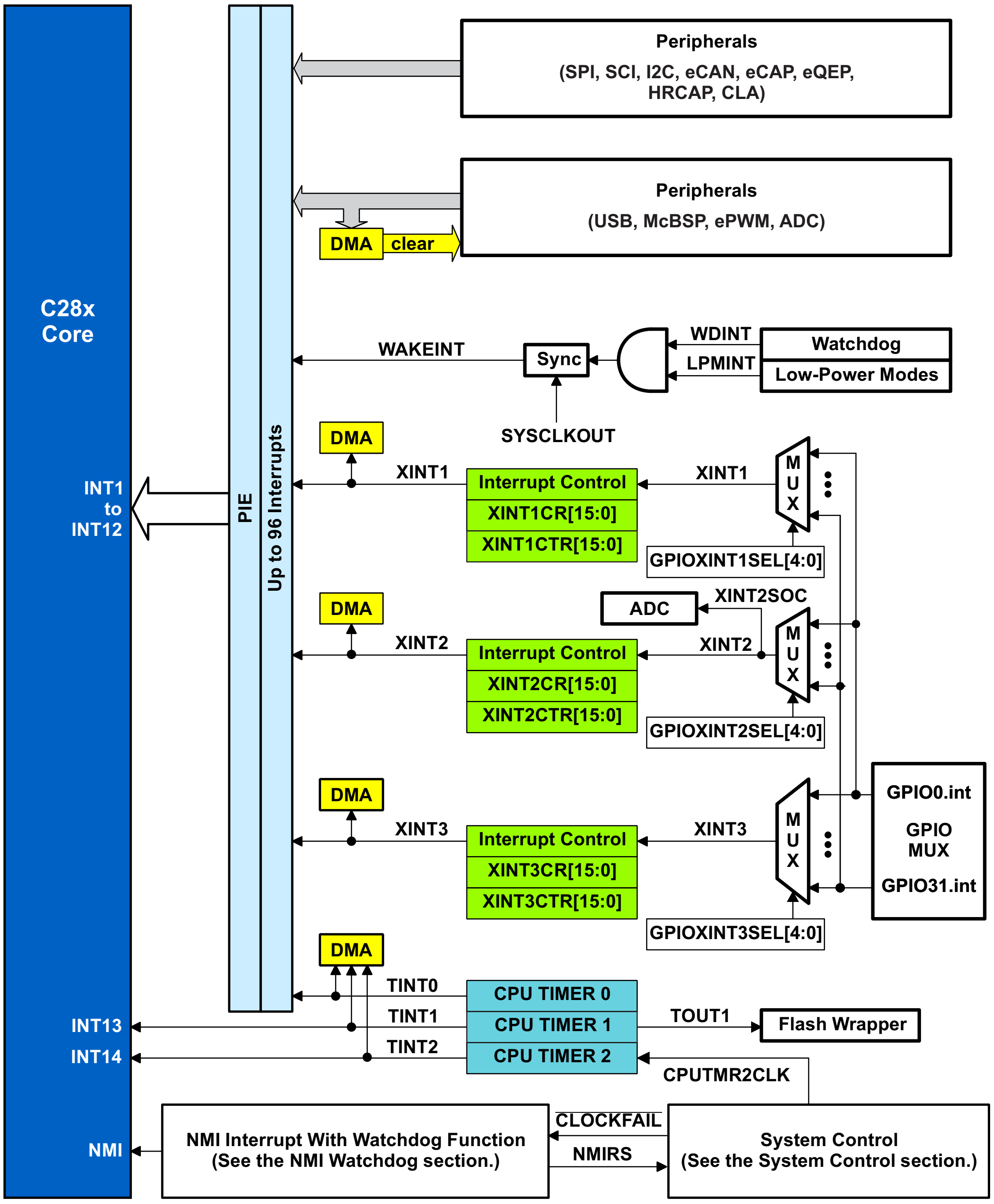
Operating Mode	OBJMODE	AMODE	
C28x Mode	1	0	
24x/240xA Source-Compatible	1	1	
C27x Object-Compatible	0	0	(Default at reset)

B The reset vector is always fetched from the boot ROM.

1.6.3 Interrupt Sources

Figure 1-89 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 1-89. 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, then 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 *C2833x C/C++ Header Files and Peripheral Examples in C* (literature number SPRC530).

(a) 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.

(b) 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 then 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. The software package *C280x C/C++ Header Files and Peripheral Examples in C* (literature number SPRC191) includes an example that illustrates this method of software prioritizing interrupts.

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:

Step a. Disable global interrupts (INTM = 1).

Step b. 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.

Step c. 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.

Step d. Clear the CPU IFRx bit for the peripheral group. This is a safe operation on the CPU IFR register.

Step e. Clear the PIEACKx bit for the peripheral group.

Step f. 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:

Step 1. Disable global interrupts (INTM = 1).

Step 2. Set the EALLOW bit.

Step 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.

Step 4. Disable the peripheral interrupt at the peripheral register.

Step 5. Enable global interrupts (INTM = 0).

Step 6. Wait for any pending interrupt from the peripheral to be serviced by the empty ISR routine.

Step 7. Disable global interrupts (INTM = 1).

Step 8. Modify the PIE vector table to map the peripheral vector back to its original ISR.

Step 9. Clear the EALLOW bit.

Step 10. Disable the PIEIER bit for given peripheral.

Step 11. Clear the IFR bit for given peripheral group (this is safe operation on CPU IFR register).

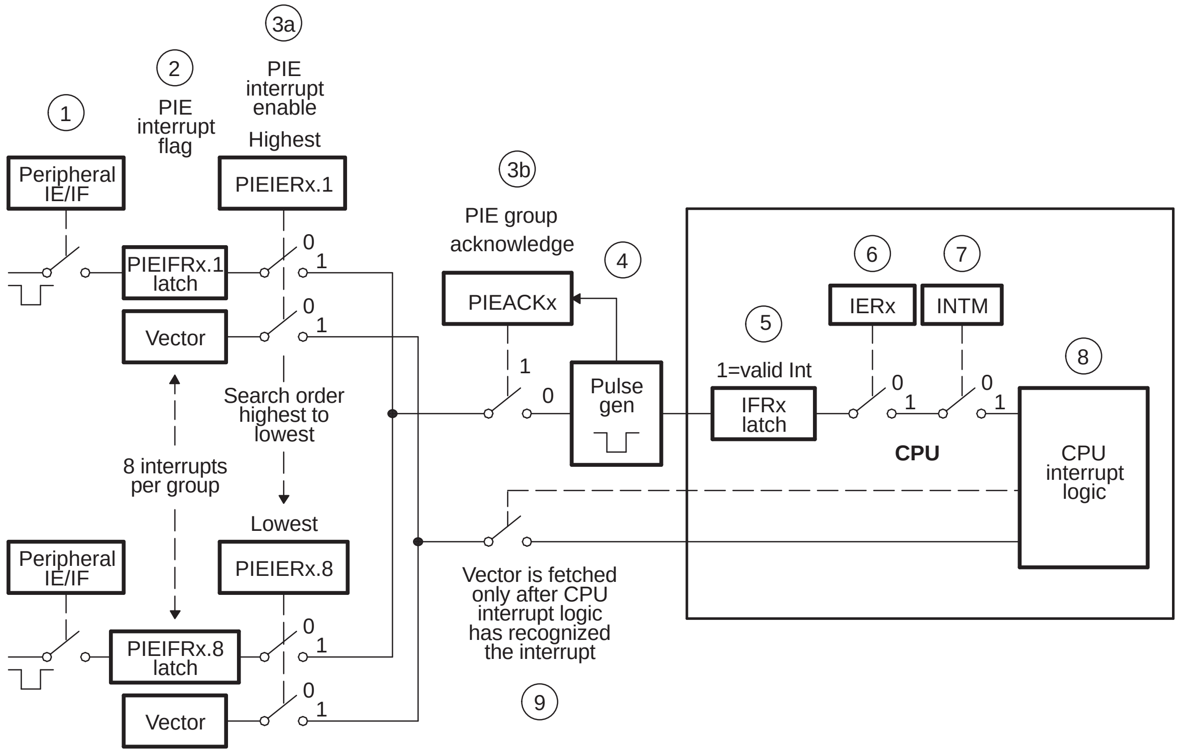
Step 12. Clear the PIEACK bit for the PIE group.

Step 13. Enable global interrupts.

1.6.3.3 Flow of a Multiplexed Interrupt Request From a Peripheral to the CPU

Figure 1-90 shows the flow with the steps shown in circled numbers. Following the diagram, the steps are described.

Figure 1-90. Multiplexed Interrupt Request Flow Diagram



Step 1. Any peripheral or external interrupt within the PIE group generates an interrupt. If interrupts are enabled within the peripheral module then the interrupt request is sent to the PIE module.

Step 2. The PIE module recognizes that interrupt y within PIE group x (INTx.y) has asserted an interrupt and the appropriate PIE interrupt flag bit is latched: PIEIFRx.y = 1.

Step 3. For the interrupt request to be sent from the PIE to the CPU, both of the following conditions must be true:

(a) The proper enable bit must be set (PIEIERx.y = 1) and

(b) The PIEACKx bit for the group must be clear.

Step 4. If both conditions in 3a and 3b are true, then an interrupt request is sent to the CPU and the acknowledge bit is again set (PIEACKx = 1). The PIEACKx bit will remain set until you clear it to indicate that additional interrupts from the group can be sent from the PIE to the CPU.

Step 5. The CPU interrupt flag bit is set (CPU IFRx = 1) to indicate a pending interrupt x at the CPU level.

Step 6. If the CPU interrupt is enabled (CPU IER bit x = 1, or DBGIER bit x = 1) AND the global interrupt mask is clear (INTM = 0) then the CPU will service the INTx.

Step 7. The CPU recognizes the interrupt and performs the automatic context save, clears the IER bit, sets INTM, and clears EALLOW. All of the steps that the CPU takes in order to prepare to service the interrupt are documented in the *TM S320C28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430).

Step 8. The CPU will then request the appropriate vector from the PIE.

Step 9. For multiplexed interrupts, the PIE module uses the current value in the PIEIERx and PIEIFRx registers to decode which vector address should be used. There are two possible cases:

(a) The vector for the highest priority interrupt within the group that is both enabled in the

PIEIERx register, and flagged as pending in the PIEIFRx is fetched and used as the branch address. In this manner if an even higher priority enabled interrupt was flagged after Step 7, it will be serviced first.

(b) If no flagged interrupts within the group are enabled, then 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 6-5. 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 The PIE Vector Table

The PIE vector table (see Table 1-119) 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 INT8.1, 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 1-118, 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 1-118. 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 1-119.

Table 1-118. PIE MUXed Peripheral Interrupt Vector Table

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1.y	WAKEINT	TINT0	ADCINT9	XINT2	XINT1	Reserved	ADCINT2	ADCINT1
	(LPM/WD)	(TIMER 0)	(ADC)	Ext. int. 2	Ext. int. 1	–	(ADC)	(ADC)
	0xD4E	0xD4C	0xD4A	0xD48	0xD46	0xD44	0xD42	0xD40
INT2.y	EPWM8_TZINT	EPWM7_TZINT	EPWM6_TZINT	EPWM5_TZINT	EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
	(ePWM8)	(ePWM7)	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD5E	0xD5C	0xD5A	0xD58	0xD56	0xD54	0xD52	0xD50
INT3.y	EPWM8_INT	EPWM7_INT	EPWM6_INT	EPWM5_INT	EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
	(ePWM8)	(ePWM7)	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD6E	0xD6C	0xD6A	0xD68	0xD66	0xD64	0xD62	0xD60

Table 1-118. PIE MUXed Peripheral Interrupt Vector Table (continued)

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT4.y	HRCAP2_INT (HRCAP2) 0xD7E	HRCAP1_INT (HRCAP1) 0xD7C	Reserved – 0xD7A	Reserved – 0xD78	Reserved – 0xD76	ECAP3_INT (eCAP3) 0xD74	ECAP2_INT (eCAP2) 0xD72	ECAP1_INT (eCAP1) 0xD70
INT5.y	Reserved USB0_INT (USB0) 0xD8E	Reserved – 0xD8C	Reserved – 0xD8A	HRCAP4_INT (HRCAP4) 0xD88	HRCAP3_INT (HRCAP3) 0xD86	Reserved – 0xD84	EQEP2_INT (eQEP2) 0xD82	EQEP1_INT (eQEP1) 0xD80
INT6.y	Reserved – 0xD9E	Reserved – 0xD9C	MXINTA (McBSP-A) 0xD9A	MRINTA (McBSP-A) 0xD98	SPITXINTB (SPI-B) 0xD96	SPIRXINTB (SPI-B) 0xD94	SPITXINTA (SPI-A) 0xD92	SPIRXINTA (SPI-A) 0xD90
INT7.y	Reserved – 0xDAE	Reserved – 0xDAC	DINTCH6 (DMA) 0xDAA	DINTCH5 (DMA) 0xDA8	DINTCH4 (DMA) 0xDA6	DINTCH3 (DMA) 0xDA4	DINTCH2 (DMA) 0xDA2	DINTCH1 (DMA) 0xDA0
INT8.y	Reserved – 0xDBE	Reserved – 0xDBC	Reserved – 0xDBA	Reserved – 0xDB8	Reserved – 0xDB6	Reserved – 0xDB4	I2CINT2A (I2C-A) 0xDB2	I2CINT1A (I2C-A) 0xDB0
INT9.y	Reserved – 0xDCE	Reserved – 0xDCC	ECAN1_INT1 (CAN-A) 0xDCA	ECAN0_INT0 (CAN-A) 0xDC8	SCITXINTB (SCI-B) 0xDC6	SCIRXINTB (SCI-B) 0xDC4	SCITXINTA (SCI-A) 0xDC2	SCIRXINTA (SCI-A) 0xDC0
INT10.y	ADCINT8 (ADC) 0xDDE	ADCINT7 (ADC) 0xDDC	ADCINT6 (ADC) 0xDDA	ADCINT5 (ADC) 0xDD8	ADCINT4 (ADC) 0xDD6	ADCINT3 (ADC) 0xDD4	ADCINT2 (ADC) 0xDD2	ADCINT1 (ADC) 0xDD0
INT11.y	CLA1_INT8 (CLA) 0xDEE	CLA1_INT7 (CLA) 0xDEC	CLA1_INT6 (CLA) 0xDEA	CLA1_INT5 (CLA) 0xDE8	CLA1_INT4 (CLA) 0xDE6	CLA1_INT3 (CLA) 0xDE4	CLA1_INT2 (CLA) 0xDE2	CLA1_INT1 (CLA) 0xDE0
INT12.y	LUF (CLA) 0xDFE	LVF (CLA) 0xDFC	Reserved – 0xDFA	Reserved – 0xDF8	Reserved – 0xDF6	Reserved – 0xDF4	Reserved – 0xDF2	XINT3 Ext. Int. 3 0xDF0

Table 1-119. PIE Vector Table

Name	VECTOR ID	Address[1]	Size (x16)	Description[2]	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	External Interrupt 13 (XINT13) or CPU-Timer1	17	-
INT14	14	0x0000 0D1C	2	CPU-Timer2 (for TI/RTOS use)	18	-
DATALOG	15	0x0000 0D1E	2	CPU Data Logging Interrupt	19 (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.

Table 1-119. PIE Vector Table (continued)

Name	VECTOR ID	Address[(1)]	Size (x16)	Description[(2)]		CPU Priority	PIE Group Priority
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		-	-
PIE Group 1 Vectors - MUXed into CPU INT1							
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)
PIE Group 2 Vectors - MUXed into CPU INT2							
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	EPWM5_TZINT	(EPWM5)	6	5
INT2.6	45	0x0000 0D5A	2	EPWM6_TZINT	(EPWM6)	6	6
INT2.7	46	0x0000 0D5C	2	EPWM7_TZINT	(EPWM7)	6	7
INT2.8	47	0x0000 0D5E	2	Reserved		6	8 (lowest)
PIE Group 3 Vectors - MUXed into CPU INT3							
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	EPWM5_INT	(EPWM5)	7	5
INT3.6	53	0x0000 0D6A	2	EPWM6_INT	(EPWM6)	7	6
INT3.7	54	0x0000 0D6C	2	EPWM7_INT	(EPWM7)	7	7
INT3.8	55	0x0000 0D6E	2	EPWM8_INT	(EPWM8)	7	8 (lowest)
PIE Group 4 Vectors - MUXed into CPU INT4							
INT4.1	56	0x0000 0D70	2	ECAP1_INT	(ECAP1)	8	1 (highest)
INT4.2	57	0x0000 0D72	2	ECAP2_INT	(ECAP2)	8	2

Table 1-119. PIE Vector Table (continued)

Name	VECTOR ID	Address[(1)]	Size (x16)	Description[(2)]		CPU Priority	PIE Group Priority
INT4.3	58	0x0000 0D74	2	ECAP3_INT	(ECAP3)	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)
PIE Group 5 Vectors - MUXed into CPU INT5							
INT5.1	64	0x0000 0D80	2	EQEP1_INT	(EQEP1)	9	1 (highest)
INT5.2	65	0x0000 0D82	2	EQEP2_INT	(EQEP2)	9	2
INT5.3	66	0x0000 0D84	2	Reserved		9	3
INT5.4	67	0x0000 0D86	2	HRCAP3INT	HRCAP3	9	4
INT5.5	68	0x0000 0D88	2	HRCAP4INT	HRCAP4	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)
PIE Group 6 Vectors - MUXed into CPU INT6							
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	SPIRXINTB	(SPI-B)	10	3
INT6.4	75	0x0000 0D96	2	SPITXINTB	(SPI-B)	10	4
INT6.5	76	0x0000 0D98	2	MRINTA	(McBSP-A)	10	5
INT6.6	77	0x0000 0D9A	2	MXINTA	(McBSP-A)	10	6
INT6.7	78	0x0000 0D9C	2	Reserved	-	10	7
INT6.8	79	0x0000 0D9E	2	Reserved	-	10	8 (lowest)
PIE Group 7 Vectors - MUXed into CPU INT7							
INT7.1	80	0x0000 0DA0	2	DINTCH1	(DMA)	11	1 (highest)
INT7.2	81	0x0000 0DA2	2	DINTCH2	(DMA)	11	2
INT7.3	82	0x0000 0DA4	2	DINTCH3	(DMA)	11	3
INT7.4	83	0x0000 0DA6	2	DINTCH4	(DMA)	11	4
INT7.5	84	0x0000 0DA8	2	DINTCH5	(DMA)	11	5
INT7.6	85	0x0000 0DAA	2	DINTCH6	(DMA)	11	6
INT7.7	86	0x0000 0DAC	2	Reserved	-	11	7
INT7.8	87	0x0000 0DAE	2	Reserved	-	11	8 (lowest)
PIE Group 8 Vectors - MUXed into CPU INT8							
INT8.1	88	0x0000 0DB0	2	I2CINT1A	(I^2C-A)	12	1 (highest)
INT8.2	89	0x0000 0DB2	2	I2CINT2A	(I^2C-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)
PIE Group 9 Vectors - MUXed into CPU INT9							
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	SCIRXINTB	(SCI-B)	13	3
INT9.4	99	0x0000 0DC6	2	SCIRXINTAB	(SCI-B)	13	4

Table 1-119. PIE Vector Table (continued)

Name	VECTOR ID	Address[(1)]	Size (x16)	Description[(2)]		CPU Priority	PIE Group Priority
INT9.5	100	0x0000 0DC8	2	ECANAINT0	(CAN-A)	13	5
INT9.6	101	0x0000 0DCA	2	ECANAINT1	(CAN-A)	13	6
INT9.7	102	0x0000 0DCC	2	Reserved	-	13	7
INT9.8	103	0x0000 0DCE	2	Reserved	-	13	8 (lowest)
PIE Group 10 Vectors - MUXed into CPU INT10							
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)
PIE Group 11 Vectors - MUXed into CPU INT11							
INT11.1	112	0x0000 0DE0	2	CLA1_INT1	(CLA)	15	1 (highest)
INT11.2	113	0x0000 0DE2	2	CLA1_INT2	(CLA)	15	2
INT11.3	114	0x0000 0DE4	2	CLA1_INT3	(CLA)	15	3
INT11.4	115	0x0000 0DE6	2	CLA1_INT4	(CLA)	15	4
INT11.5	116	0x0000 0DE8	2	CLA1_INT5	(CLA)	15	5
INT11.6	117	0x0000 0DEA	2	CLA1_INT6	(CLA)	15	6
INT11.7	118	0x0000 0DEC	2	CLA1_INT7	(CLA)	15	7
INT11.8	119	0x0000 0DEE	2	CLA1_INT8	(CLA)	15	8 (lowest)
PIE Group 12 Vectors - Muxed into CPU INT12							
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	LVF	(CLA)	16	7
INT12.8	127	0x0000 0DFE	2	LUF	(CLA)	16	8 (lowest)

1.6.4 PIE Configuration Registers

The registers controlling the functionality of the PIE block are shown in Table 1-120.

Table 1-120. PIE Configuration and Control Registers

Name	Address	Size (x16)	Description
PIECTRL	0x0000 - 0CE0	1	PIE, Control Register
PIEACK	0x0000 - 0CE1	1	PIE, Acknowledge Register
PIEIER1	0x0000 - 0CE2	1	PIE, INT1 Group Enable Register
PIEIFR1	0x0000 - 0CE3	1	PIE, INT1 Group Flag Register
PIEIER2	0x0000 - 0CE4	1	PIE, INT2 Group Enable Register
PIEIFR2	0x0000 - 0CE5	1	PIE, INT2 Group Flag Register
PIEIER3	0x0000 - 0CE6	1	PIE, INT3 Group Enable Register
PIEIFR3	0x0000 - 0CE7	1	PIE, INT3 Group Flag Register
PIEIER4	0x0000 - 0CE8	1	PIE, INT4 Group Enable Register
PIEIFR4	0x0000 - 0CE9	1	PIE, INT4 Group Flag Register
PIEIER5	0x0000 - 0CEA	1	PIE, INT5 Group Enable Register
PIEIFR5	0x0000 - 0CEB	1	PIE, INT5 Group Flag Register
PIEIER6	0x0000 - 0CEC	1	PIE, INT6 Group Enable Register
PIEIFR6	0x0000 - 0CED	1	PIE, INT6 Group Flag Register
PIEIER7	0x0000 - 0CEE	1	PIE, INT7 Group Enable Register
PIEIFR7	0x0000 - 0CEF	1	PIE, INT7 Group Flag Register
PIEIER8	0x0000 - 0CF0	1	PIE, INT8 Group Enable Register
PIEIFR8	0x0000 - 0CF1	1	PIE, INT8 Group Flag Register
PIEIER9	0x0000 - 0CF2	1	PIE, INT9 Group Enable Register
PIEIFR9	0x0000 - 0CF3	1	PIE, INT9 Group Flag Register
PIEIER10	0x0000 - 0CF4	1	PIE, INT10 Group Enable Register
PIEIFR10	0x0000 - 0CF5	1	PIE, INT10 Group Flag Register
PIEIER11	0x0000 - 0CF6	1	PIE, INT11 Group Enable Register
PIEIFR11	0x0000 - 0CF7	1	PIE, INT11 Group Flag Register
PIEIER12	0x0000 - 0CF8	1	PIE, INT12 Group Enable Register
PIEIFR12	0x0000 - 0CF9	1	PIE, INT12 Group Flag Register

1.6.5 PIE Interrupt Registers

Figure 1-91. PIECTRL Register (Address 0xCE0)

15 - 1	0
PIEVECT	ENPIE
R-0	R/W-0

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

Table 1-121. PIECTRL Register Address 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. **For Example:** If PIECTRL = 0x0D27 then the vector from address 0x0D26 (illegal operation) was fetched.
0	ENPIE		Enable vector fetching from PIE vector table. **Note:** 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.

Figure 1-92. PIE Interrupt Acknowledge Register (PIEACK) Register (Address 0xCE1)

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 1-122. PIE Interrupt Acknowledge Register (PIEACK) 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 $\overline{\text{INT1}}$ up to Bit 11, which refers to PIE group 12 which is MUXed into CPU $\overline{\text{IN T12}}$
		bit x = 0 [(1)]	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 - PIEACK bit 11. Bit 0 refers to CPU $\overline{\text{INT1}}$ up to Bit 11, which refers to CPU $\overline{\text{INT12}}$

1.6.5.1 PIE Interrupt Flag Registers

There are twelve PIEIFR registers, one for each CPU interrupt used by the PIE module (INT1-INT12).

Figure 1-93. PIEIFRx Register (x = 1 to 12)

15-8
Reserved
R-0

7	6	5	4	3	2	1	0
INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
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 1-123. PIEIFRx Register 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 The PIEIFR register bit is cleared during the interrupt vector fetch portion of the interrupt processing. Hardware has priority over CPU accesses to the PIEIFR registers.
6	INTx.7	
5	INTx.6	
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

NOTE: Never clear a PIEIFR bit. An interrupt may be lost during the read-modify-write operation. See Section Section 1.6.3.1 for a method to clear flagged interrupts.

1.6.5.2 PIE Interrupt Enable Registers

There are twelve PIEIER registers, one for each CPU interrupt used by the PIE module (INT1-INT12).

Figure 1-94. PIEIERx Register (x = 1 to 12)

15-8
Reserved
R-0

7	6	5	4	3	2	1	0
INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
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 1-124. PIEIERx Register (x = 1 to 12) Field Descriptions

Bits	Field	Description
15-8	Reserved	Reserved
7	INTx.8	These register bits individually enable an interrupt within a group and behave very much like the core interrupt enable register. Setting a bit to 1 enables the servicing of the respective interrupt. Setting a bit to 0 disables the servicing of the interrupt. x = 1 to 12. INTx means CPU INT1 to INT12
6	INTx.7	
5	INTx.6	
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

NOTE: Care must be taken when clearing PIEIER bits during normal operation. See Section Section 1.6.3.2 for the proper procedure for handling these bits.

1.6.5.3 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 1-95. 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; R = Read only; -*n* = value after reset

Table 1-125. Interrupt Flag Register (IFR) — CPU Register Field Descriptions

Bits	Field	Value	Description
15	RTOSINT		Real-time operating system flag. RTOSINT is the flag for RTOS interrupts.
		0	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
14	DLOGINT		Data logging interrupt fag. DLOGINT is the flag for data logging interrupts.
		0	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		Interrupt 14 flag. INT14 is the flag for interrupts connected to CPU interrupt level INT14.
		0	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		Interrupt 13 flag. INT13 is the flag for interrupts connected to CPU interrupt level INT13I.
		0	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		Interrupt 12 flag. INT12 is the flag for interrupts connected to CPU interrupt level INT12.
		0	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		Interrupt 11 flag. INT11 is the flag for interrupts connected to CPU interrupt level INT11.
		0	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		Interrupt 10 flag. INT10 is the flag for interrupts connected to CPU interrupt level INT10.
		0	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		Interrupt 9 flag. INT9 is the flag for interrupts connected to CPU interrupt level INT6.
		0	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		Interrupt 8 flag. INT8 is the flag for interrupts connected to CPU interrupt level INT6.
		0	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		Interrupt 7 flag. INT7 is the flag for interrupts connected to CPU interrupt level INT7.
		0	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

Table 1-125. Interrupt Flag Register (IFR) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
5	INT6		Interrupt 6 flag. INT6 is the flag for interrupts connected to CPU interrupt level INT6.
		0	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		Interrupt 5 flag. INT5 is the flag for interrupts connected to CPU interrupt level INT5.
		0	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		Interrupt 4 flag. INT4 is the flag for interrupts connected to CPU interrupt level INT4.
		0	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
2	INT3		Interrupt 3 flag. INT3 is the flag for interrupts connected to CPU interrupt level INT3.
		0	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		Interrupt 2 flag. INT2 is the flag for interrupts connected to CPU interrupt level INT2.
		0	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		Interrupt 1 flag. INT1 is the flag for interrupts connected to CPU interrupt level INT1.
		0	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

1.6.5.4 Interrupt Enable Register (IER) and Debug Interrupt Enable Register (DBGIER)

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 $\overline{\text{XRS}}$ 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.

The IER register is shown in Figure 1-96, and descriptions of the bits follow the figure.

Figure 1-96. 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; R = Read only; -*n* = value after reset

Table 1-126. Interrupt Enable Register (IER) — CPU Register Field Descriptions

Bits	Field	Value	Description
15	RTOSINT		Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
14	DLOGINT		Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
13	INT14		Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14.
		0	Level INT14 is disabled
		1	Level INT14 is enabled
12	INT13		Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13.
		0	Level INT13 is disabled
		1	Level INT13 is enabled
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		Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11.
		0	Level INT11 is disabled
		1	Level INT11 is enabled
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
		1	Level INT10 is enabled
8	INT9		Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0	Level INT9 is disabled
		1	Level INT9 is enabled
7	INT8		Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0	Level INT8 is disabled
		1	Level INT8 is enabled
6	INT7		Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT7.
		0	Level INT7 is disabled
		1	Level INT7 is enabled
5	INT6		Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
4	INT5		Interrupt 5 enable.INT5 enables or disables CPU interrupt level INT5.
		0	Level INT5 is disabled
		1	Level INT5 is enabled

Table 1-126. Interrupt Enable Register (IER) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
3	INT4		Interrupt 4 enable.INT4 enables or disables CPU interrupt level INT4.
		0	Level INT4 is disabled
		1	Level INT4 is enabled
2	INT3		Interrupt 3 enable.INT3 enables or disables CPU interrupt level INT3.
		0	Level INT3 is disabled
		1	Level INT3 is enabled
1	INT2		Interrupt 2 enable.INT2 enables or disables CPU interrupt level INT2.
		0	Level INT2 is disabled
		1	Level INT2 is enabled
0	INT1		Interrupt 1 enable.INT1 enables or disables CPU interrupt level INT1.
		0	Level INT1 is disabled
		1	Level INT1 is enabled

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 1-97. 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; R = Read only; -*n* = value after reset

Table 1-127. Debug Interrupt Enable Register (DBGIER) — CPU Register Field Descriptions

Bits	Field	Value	Description
15	RTOSINT		Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
14	DLOGINT	.	Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt
		0	Level INT6 is disabled
		1	Level INT6 is enabled
13	INT14	.	Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14
		0	Level INT14 is disabled
		1	Level INT14 is enabled
12	INT13		Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13.
		0	Level INT13 is disabled
		1	Level INT13 is enabled

Table 1-127. Debug Interrupt Enable Register (DBGIER) — CPU Register 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		Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11.
		0	Level INT11 is disabled
		1	Level INT11 is enabled
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
		1	Level INT10 is enabled
8	INT9		Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0	Level INT9 is disabled
		1	Level INT9 is enabled
7	INT8		Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0	Level INT8 is disabled
		1	Level INT8 is enabled
6	INT7		Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT77.
		0	Level INT7 is disabled
		1	Level INT7 is enabled
5	INT6		Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
4	INT5		Interrupt 5 enable.INT5 enables or disables CPU interrupt level INT5.
		0	Level INT5 is disabled
		1	Level INT5 is enabled
3	INT4		Interrupt 4 enable.INT4 enables or disables CPU interrupt level INT4.
		0	Level INT4 is disabled
		1	Level INT4 is enabled
2	INT3		Interrupt 3 enable.INT3 enables or disables CPU interrupt level INT3.
		0	Level INT3 is disabled
		1	Level INT3 is enabled
1	INT2		Interrupt 2 enable.INT2 enables or disables CPU interrupt level INT2.
		0	Level INT2 is disabled
		1	Level INT2 is enabled
0	INT1		Interrupt 1 enable.INT1 enables or disables CPU interrupt level INT1.
		0	Level INT1 is disabled
		1	Level INT1 is enabled

1.6.6 External Interrupt Control 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 1-128. Interrupt Control and Counter Registers (not EALLOW Protected)

Name	Address Range	Size (x16)	Description
XINT1CR	0x0000 7070	1	XINT1 configuration register
XINT2CR	0x0000 7071	1	XINT2 configuration register
XINT3CR	0x0000 7072	1	XINT3 configuration register
reserved	0x0000 7073 - 0x0000 7077	5	
XINT1CTR	0x0000 7078	1	XINT1 counter register
XINT2CTR	0x0000 7079	1	XINT2 counter register
XINT3CTR	0x0000 707A	1	XINT3 counter register
reserved	0x0000 707B - 0x0000 707E	5	

XINT1CR through XINT3CR are identical except for the interrupt number; therefore, Figure 1-98 and Table 1-129 represent registers for external interrupts 1 through 3 as XINT*n*CR where *n* = the interrupt number.

Figure 1-98. External Interrupt *n* Control Register (XINT*n*CR)

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 1-129. External Interrupt *n* Control Register (XINT*n*CR) Field Descriptions

Bits	Field	Value	Description
15-4	Reserved		Reads return zero; writes have no effect.
3-2	Polarity		This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin.
		00	Interrupt generated on a falling edge (high-to-low transition)
		01	Interrupt generated on a rising edge (low-to-high transition)
		10	Interrupt generated on a falling edge (high-to-low transition)
		11	Interrupt generated on both a falling edge and a rising edge (high-to-low and low-to-high transition)
1	Reserved		Reads return zero; writes have no effect
0	Enable		This read/write bit enables or disables external interrupt XINT*n.*
		0	Disable interrupt
		1	Enable interrupt

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 1-99 and Table 1-130 represent registers for the external interrupts as XINTnCTR, where n = the interrupt number.

Figure 1-99. External Interrupt n Counter (XINTnCTR) (Address 7078h)

15 … 0
INTCTR[15-8]
R-0

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

Table 1-130. External Interrupt n Counter (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 max 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 μF capacitance for proper regulation of the VREG. These capacitors should be located as close as possible to the device pins. See the *TMS320F2806x Piccolo Microcontrollers* (literature number SPRS698) 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 a more efficient external regulator. To enable this option, the VREGENZ pin must be pulled high. See the *TMS320F2806x Piccolo Microcontrollers* (literature number SPRS698) 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 overvoltage trip point. See the device datasheet for the various trip points as well as the delay time from the removal of the fault condition to enable the BOR function, 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. See Table 1-131 for a description of the BORCFG register. The BORCFG register state can only be modified by software and the $\overline{XRS}$ pin signal. A CPU reset from the debugger will not modify this register.

Figure 1-100. 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 1-131. BOR Configuration (BORCFG) Field Descriptions

Bits	Field	Value	Description
15-1	Reserved		Reserved
0	BORENZ		BOR enable active low bit.
		0	BOR functions are enabled.
		1	BOR functions are disabled.

Boot ROM

This chapter is applicable for the code and data stored in the on-chip boot ROM on the TMS320F2806x Piccolo™ processors, which includes all devices within this family. The boot ROM is factory-programmed with bootloading software. Bootmode signals (TRST and general purpose I/Os) are used to tell the bootloader software which mode to use on power-up. The boot ROM also contains standard math tables, such as SIN/COS waveforms, for use in IQ math related algorithms found in the C28x IQMath Library - A Virtual Floating Point Engine (literature number SPRC087). Described here are the purpose and features of the bootloader, as well as other contents of the device on-chip boot ROM, and identifies where all of the information is located within that memory. This chapter also refers to associated code that can be downloaded via the latest version of controlSuite from the TI website.

Topic **Page**

2.1 Boot ROM Memory Map

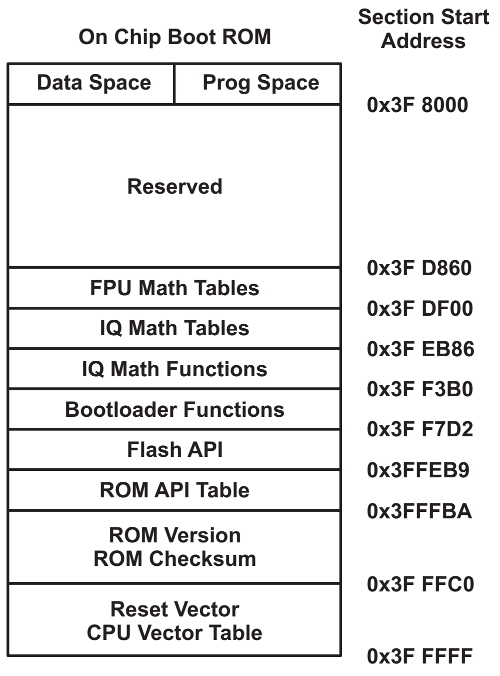
The boot ROM is an 32K x 16 block of read-only memory located at addresses 0x3F 8000 - 0x3F FFFF.

The on-chip boot ROM is factory programmed with boot-load routines and both fixed-point and floating-point math tables. These are for use with the *C28x IQMath Library - A Virtual Floating Point Engine* (SPRC087) and the C28x FPU Fast RTS Library (SPRC664). 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
- Floating Point unit (FPU) math tables
- Flash API library

Figure 2-1 shows the memory map of the on-chip boot ROM. The memory block is 32Kx16 in size and is located at 0x3F 8000 - 0x3F FFFF in both program and data space.

Figure 2-1. Memory Map of On-Chip ROM



2.1.1 On-Chip Boot ROM Math Tables

Approximately 4K of the boot ROM is reserved for floating-point and IQ math tables. These tables are provided to help improve performance and save SARAM space.

The floating-point math tables included in the boot ROM are used by the Texas Instruments™ C28x FPU Fast RTS Library (SPRC664). The C28x Fast RTS Library is a collection of optimized floating-point math functions for C programmers of the C28x with floating-point unit. Designers of computationally intensive real-time applications can achieve execution speeds considerably faster than what are currently available without having to rewrite existing code. The functions listed in the features section are specifically optimized for the C28x + FPU controllers. The Fast RTS library accesses the floating-point tables through the FPUmathTables memory section. If you do not wish to load a copy of these tables into the device, use the boot ROM memory addresses and label the section as “NOLOAD” as shown in Example 2-1. This facilitates referencing the look-up tables without actually loading the section to the target.

The following math tables are included in the Boot ROM:

- **Sine/Cosine Table, Single-precision Floating point**
  - Table size: 1282 words
  - Contents: 32-bit floating-point samples for one and a quarter period sine wave
- **Normalized Arctan Table, Single-precision Floating point**
  - Table size: 388 words
  - Contents 32-bit second order coefficients for line of best fit
- **Exp Coefficient Table, Single-precision Floating point**
  - Table size: 20 words
  - Contents: 32-bit coefficients for calculating exp (X) using a Taylor series

Example 2-1. Linker Command File to Access FPU Tables

```
MEMORY
{
   PAGE 0 :
   ...
   FPUTABLES   : origin = 0x3FD860, length = 0x0006A0
   ...
}
SECTIONS
{
   ...
   FPUmathTables :  > FPUTABLES, PAGE = 0, TYPE = NOLOAD,
   ...
}
```

The fixed-point math tables included in the boot ROM are used by the Texas Instruments™ *C28x IQMath Library - A Virtual Floating Point Engine* (SPRC087). 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 TMS320C28x devices.

These routines are typically used in computational-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 shorten significantly your DSP application development time.

The IQmath library accesses the tables through the IQmathTables and the IQmathTablesRam linker sections. The IQmathTables section is completely included in the boot ROM. From the IQmathTablesRam section, only the IQexp table is included and the remainder must be loaded into the device if used. 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 2-2. This facilitates referencing the look-up tables without actually loading the section to the target. Refer to the IQMath Library documentation for more information.

Example 2-2. Linker Command File to Access IQ Tables

```
MEMORY
{
   PAGE 0 :
   ...
   IQTABLES  (R)  : origin = 0x3FDF00, length = 0x000B50
   IQTABLES2 (R)  : origin = 0x3FEA50, length = 0x00008C
   IQTABLES3 (R)  : origin = 0x3FEADC, 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)
   }

       IQmathTablesRam : load = DRAML1, PAGE = 1
       ...
}
```

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.

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
- IQNsqrt 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.

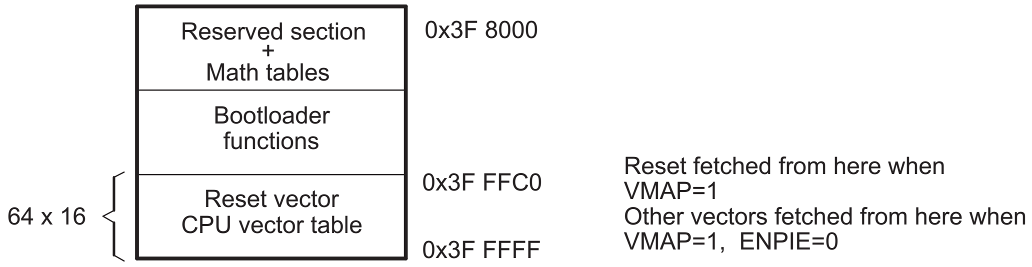
2.1.3 On-Chip Flash API

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 2806x Flash API Library documentation for information on how to use the symbol library.

2.1.4 CPU Vector Table

A CPU vector table resides in boot ROM memory from address 0x3F 8000 - 0x3F FFFF. This vector table is active after reset when VMAP = 1, ENPIE = 0 (PIE vector table disabled).

Figure 2-2. Vector Table Map



A The VMAP bit is located in Status Register 1 (ST1). VMAP is alwatys 1 on reset. It can be changed after reset by software, however, the normal operating mode will be to leave VMAP=1.

B 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).

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 boot load process. A series of checking operations is performed on $\overline{\text{TRST}}$ and General-Purpose I/O (GPIO I/O) 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 described 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 (i.e., points to)	Vector	Location in Boot ROM	Contents (i.e., 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

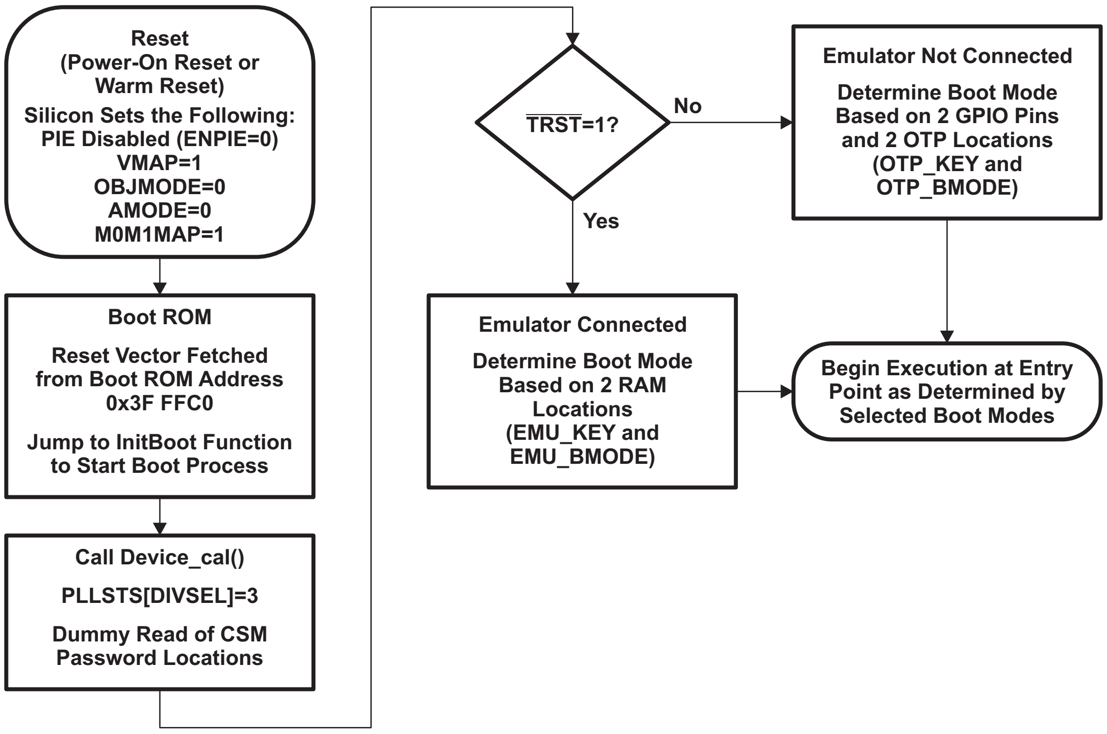
2.2 Bootloader Features

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 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:

Figure 2-3. Bootloader Flow Diagram


Reset
(Power-On Reset or
Warm Reset)
Silicon Sets the Following:
PIE Disabled (ENPIE=0)
VMAP=1
OBJMODE=0
AMODE=0
M0M1MAP=1
Boot ROM
Reset Vector Fetched from Boot ROM Address 0x3F FFC0
Jump to InitBoot Function to Start Boot Process
Call Device_cal()
PLLSTS[DIVSEL]=3
Dummy Read of CSM Password Locations
TRST=1?
No
Yes
Emulator Not Connected
Determine Boot Mode Based on 2 GPIO Pins and 2 OTP Locations (OTP_KEY and OTP_BMODE)
Emulator Connected
Determine Boot Mode Based on 2 RAM Locations (EMU_KEY and EMU_BMODE)
Begin Execution at Entry Point as Determined by Selected Boot Modes

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.

- **Emulation Boot (Emulation Pod is connected and $\overline{\text{TRST}}$ = 1)**

In emulation 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 emulation boot.

Stand-alone Boot ($\overline{\text{TRST}}$ = 0)

If the device is in stand-alone boot mode, then the state of two GPIO pins are used to determine which boot mode execute. Options include: GetMode, wait, SCI, and parallel I/O. Each of the modes is described in detail in Table 2-4. The GetMode option by default boots to flash but can be customized by programming two values into OTP to select another boot loader.

These boot modes mentioned here are discussed in detail in Section 2.2.9.

After the selection process and if the required boot loading 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.

2.2.2 Bootloader Device Configuration

At reset, any 28x CPU-based device is in 27x object-compatible mode. It is up to the application to place the device in the proper operating mode before execution proceeds.

On the 28x devices, when booting from the internal boot ROM, the device is configured for 28x operating mode by the boot ROM software. You are responsible for any additional configuration required.

For example, if your application includes C2xLP™ 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)	28x Mode	C2xLP Source Compatible Mode
OBJMODE	0	1	1
AMODE	0	0	1
PAGE0	0	0	0
M0M1MAP[1]	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 28x mode.

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 (PLLSTS) and divider (PLLSTS[DIVSEL]) are not affected by a reset from the debugger. Therefore, a boot that is initialized from a reset from Code Composer Studio™ may be at a different speed than booting by pulling the external reset line ($\overline{XRS}$) low.

NOTE: 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.

NOTE: The boot ROM leaves PLLSTS[DIVSEL] in the CLKIN/1 state when the boot ROM exits. This is not a valid configuration if the PLL is used. Thus the application must 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 28x 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 Resisters

Each GPIO pin has an internal pullup resistor 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 that you configure each of the boot mode selection pins externally.

The peripheral bootloaders all enable the pullup resistors for the pins that are used for control and data transfer. The bootloader leaves the resistors enabled for these pins when it exits. For example, the SCI-A bootloader enables the pullup resistors on the SCITXA and SCIRXA pins. It is your 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.

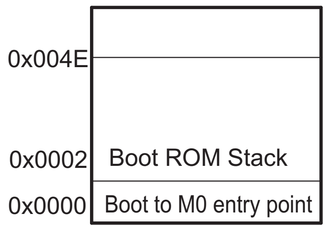
The boot ROM does, however, use the first six locations within the PIE vector table for emulation boot mode information and Flash API variables. These locations are not used by the PIE itself and not used by typical applications.

NOTE: If you are porting code from another 28x processor, check to see if the code initializes the first six locations in the PIE vector table to some default value. If it does, then consider modifying the code to not write to these locations so the EMU boot mode will not be over written during debug. Refer to the 2806x C/C++ Header Files and Peripheral Examples.

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.

Figure 2-4. Boot ROM Stack



Boot ROM loaders on older C28x devices had the stack in M1 memory.

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 6 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 effect 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:

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_CPUScaleFactor	Used by the flash API
0x0D04 x 32	Flash_CallbackPtr	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 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.

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 !=0x005A and/or OTP_BMODE is not a valid value.

NOTE: 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 (i.e., secured). When the device is powered up, the CPU will start running and may execute an instruction that performs an access to a protected emulation code security logic (ECSL) area. If this happens, the ECSL will trip and cause the emulator connection to be cut. The "wait" mode keeps this from happening by looping within the boot ROM until an emulator 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

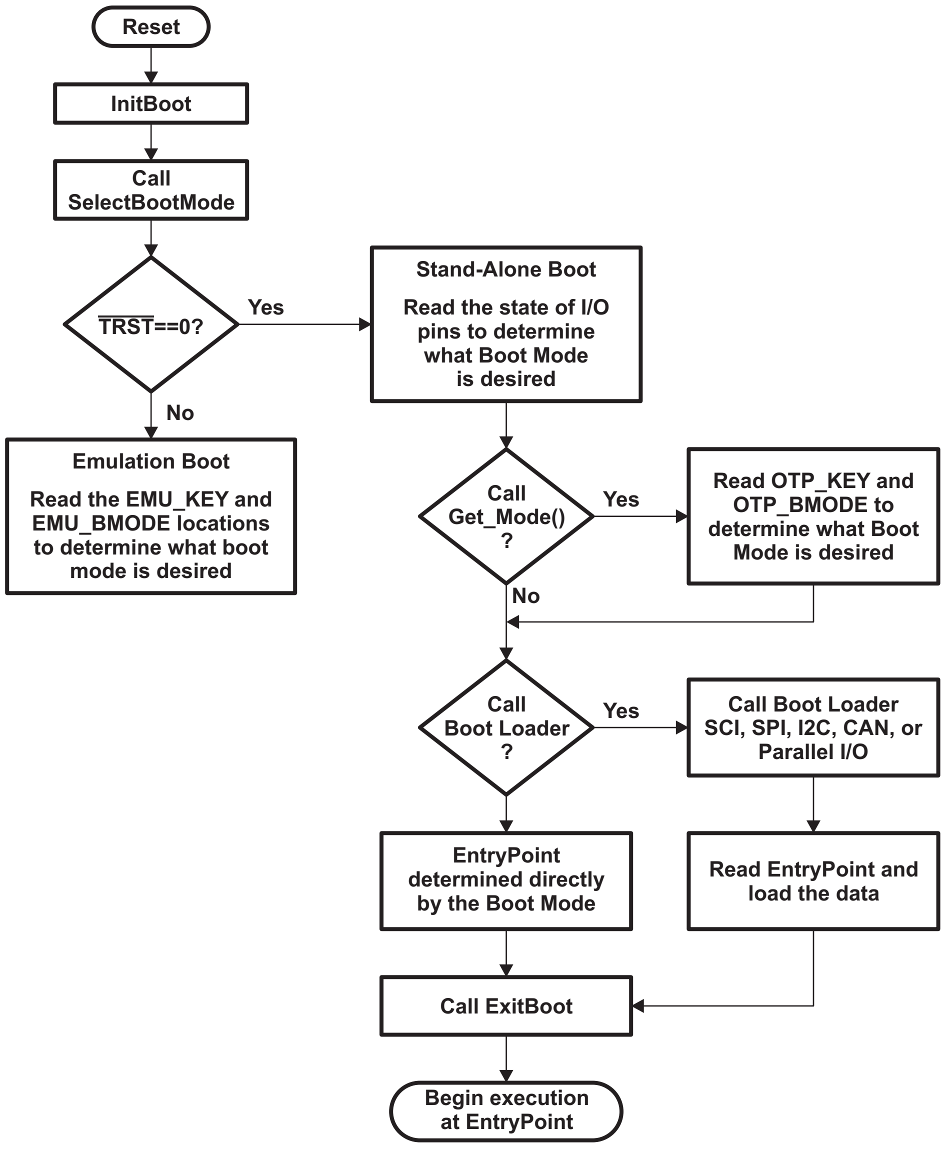

Reset
InitBoot
Call SelectBootMode
TRST==0?
Yes
No
Stand-Alone Boot
Read the state of I/O pins to determine what Boot Mode is desired
Emulation Boot
Read the EMU_KEY and EMU_BMODE locations to determine what boot mode is desired
Call Get_Mode() ?
Yes
No
Read OTP_KEY and OTP_BMODE to determine what Boot Mode is desired
Call Boot Loader ?
Yes
Call Boot Loader SCI, SPI, I2C, CAN, or Parallel I/O
EntryPoint determined directly by the Boot Mode
Read EntryPoint and load the data
Call ExitBoot
Begin execution at EntryPoint

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
		0x0006	Boot mode = OTP_BOOT
		0x0007	Boot mode = CAN_BOOT
		0x000A	Boot mode = RAM_BOOT
		0x000B	Boot mode = FLASH_BOOT
		Other	Boot mode = WAIT_BOOT

Table 2-7 shows the expanded emulation boot mode table.

Here are two examples of an emulation boot:

Example 2-3. 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 sits in the SCI loader waiting 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.

Example 2-4. You want to connect your emulator, but do not want application code to start executing before the emulator connects.

To connect your emulator, but keep application code from executing before the emulator 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 sits 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 emulators with regards to $\overline{\text{TRST}}$ differs.** Some emulators pull $\overline{\text{TRST}}$ high only when Code Composer Studio is in a connected state. For these emulators, 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 emulators, this is true even if the emulator pod is physically connected.

Some emulators 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 emulators, 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 emulator is not connected:

- **Wait**

  This devices 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 (i.e., 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 emulator connection to be cut. The "wait" mode keeps this from happening by looping within the boot ROM until an emulator is connected

  This mode writes WAIT_BOOT to EMU_BMODE. Once the emulator 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 USER OTP to determine the boot mode. On an un-programmed 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 last six words of user OTP region (0x3D7BFA to 0x3D7BFF) are reserved for the GetMode function usage.

  The values used by the Get_Mode() function are shown in Table 2-6.

Table 2-6. OTP Values for GetMode

Address	Name	Value	
0x3D7BFB	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 == 0x005A, then check OTP_BMODE for the boot mode else { Invalid key: Boot mode = FLASH_BOOT }	
0x3D7BFE	OTP_BMODE	0x0001	Boot mode = SCI_BOOT
		0x0004	Boot mode = SPI_BOOT
		0x0005	Boot mode = I2C_BOOT
		0x0006	Boot mode = OTP_BOOT
		0x0007	Boot mode = CAN_BOOT
		Other	Boot mode = FLASH_BOOT

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.

- **Jump to OTP Memory**

  Jump to OTP is available only as an option programmed into OTP_BMODE via the get mode function. With the emulator connected, jump to OTP can also be achieved by manually writing the OTP_BOOT value to EMU_BMODE. The entry point location is 0x3D 7800.

- **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.

  SCI 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.

- **eCAN-A boot mode (eCAN-A)**

  Jump to eCAN is available in stand-alone mode as a programmed Get mode option. That is, to configure a device for eCAN boot in stand-alone mode, the OTP_KEY and OTP_BMODE locations must be programmed for CAN_BOOT and the boot mode pins configured for the Get Mode boot option. eCAN boot is also available as an emulation boot option. In this mode, the eCAN-A peripheral is used to transfer data and code into the on-chip memory using eCAN-A mailbox 1. The transfer is an 8-bit data stream with two 8-bit values being transferred during each communication.

Table 2-7. Emulation Boot modes (TRST = 1)

TRST	GPIO37 TDO	GPIO34	EMU KEY Read from 0x0D00	EMU BMODE Read from 0x0D01	OTP KEY Read from 0x3D7BFB	OTP BMODE Read from 0x3D7BFE	Boot Mode Selected [1]	EMU KEY Written to 0x0D00	EMU BMODE Written to 0x0D01
1	x [2]	x	!=0x55AA	x	x	x	Wait	-	-
			0x55AA	0x0000	x	x	Parallel I/O	-	-
				0x0001	x	x	SCI	-	-
				0x0002	x	x	Wait	-	-
				0x0003	!= 0x005A	x	GetMode: Flash	-	-
					0x005A	0x0001	GetMode: SCI	-	-
						0x000B	GetMode: Flash	-	-
						0x0004	GetMode: SPI	-	-
						0x0005	GetMode: I2C	-	-
						0x0006	GetMode: OTP	-	-
						0x0007	GetMode: CAN	-	-
						Other	GetMode: Flash	-	-
				0x0004	x	x	SPI	-	-
				0x0005	x	x	I2C	-	-
				0x0006	x	x	OTP	-	-
				0x0007	x	x	CAN	-	-
				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.

Table 2-8. Stand-Alone Boot Modes with (TRST = 0)

TRST	GPIO37 TDO	GPIO34	EMU KEY Read from 0x0D00	EMU BMODE Read from 0x0D01	OTP KEY Read from 0x3D7BFB	OTP BMODE Read from 0x3D7BFE	Boot Mode Selected[1]	[2] EMU KEY Written to 0x0D00	[2] EMU BMODE Written to 0x0D01
0	0	0	x [3]	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	!=0x005A	x	GetMode: Flash	0x55AA	0x0003
					0x005A	0x0001	GetMode: SCI		
						0x000B	GetMode: Flash		
						0x0004	GetMode: SPI		
						0x0005	GetMode: I2C		
						0x0006	GetMode: OTP		
						0x0007	GetMode: CAN		
						Other	GetMode: Flash		

[1] Get Mode indicates 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 during the development process, then the calibration must be initialized by application. For working examples, see the system initialization in the *C280 3x C/C++ Header Files and Peripheral Examples*.

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 2-5. 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 2-5. The ADC clocks must be enabled before making this call.

_Example 2-5. 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 following two tables and associated examples show the structure of the data stream incoming to the bootloader. 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. The key value is used to tell the bootloader the width of the incoming stream: 8 or 16 bits. Note that not all bootloaders will accept both 8 and 16-bit streams. Please refer to the detailed information on each loader for the valid data stream width. For an 8-bit data stream, the key value is 0x08AA and for a 16-bit stream it is 0x10AA. 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. Currently only the SPI and I2C and parallel 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 both 8-bit and 16-bit data stream formats as the number of 16-bit words in the block. For example, to transfer a block of 20 8-bit data values from an 8-bit data stream, the block size would be 0x000A to indicate 10 16-bit words.

The next two words tell the loader the destination address of the block of data. Following the size and address will be the 16-bit words that makeup 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.

Table 2-9. General Structure Of Source Program Data Stream In 16-Bit Mode

Word	Contents
1	10AA (KeyValue for memory width = 16bits)
2	Register initialization value or reserved for future use
3	Register initialization value or reserved for future use
4	Register initialization value or reserved for future use
5	Register initialization value or reserved for future use
6	Register initialization value or reserved for future use
7	Register initialization value or reserved for future use
8	Register initialization value or reserved for future use
9	Register initialization value or reserved for future use
10	Entry point PC[22:16]
11	Entry point PC[15:0]
12	Block size (number of words) of the first block of data to load. If the block size is 0, this indicates the end of the source program. Otherwise another section follows.
13	Destination address of first block Addr[31:16]
14	Destination address of first block Addr[15:0]
15	First word of the first block in the source being loaded
... ...	... ...
.	Last word of the first block of the source being loaded
.	Block size of the 2nd block to load.
.	Destination address of second block Addr[31:16]
.	Destination address of second block Addr[15:0]
.	First word of the second block in the source being loaded
.	…
.	Last word of the second block of the source being loaded
.	Block size of the last block to load
.	Destination address of last block Addr[31:16]
.	Destination address of last block Addr[15:0]
.	First word of the last block in the source being loaded
... ...	... ...
n	Last word of the last block of the source being loaded
n+1	Block size of 0000h - indicates end of the source program

Example 2-6. Data Stream Structure 16-bit

```
10AA                ; 0x10AA 16-bit key value
0000 0000 0000 0000 ; 8 reserved words
0000 0000 0000 0000
003F 8000           ; 0x003F8000 EntryAddr, starting point after boot load completes
0005                ; 0x0005 - First block consists of 5 16-bit words
003F 9010           ; 0x003F9010 - First block will be loaded starting at 0x3F9010
0001 0002 0003 0004 ; Data loaded = 0x0001 0x0002 0x0003 0x0004 0x0005
0005
0002                ; 0x0002 - 2nd block consists of 2 16-bit words
003F 8000           ; 0x003F8000 - 2nd block will be loaded starting at 0x3F8000
7700 7625           ; Data loaded = 0x7700 0x7625
0000                ; 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
```

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 2-10. LSB/MSB Loading Sequence in 8-Bit Data Stream

Byte		Contents	
		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 2-7. Data Stream Structure 8-bit

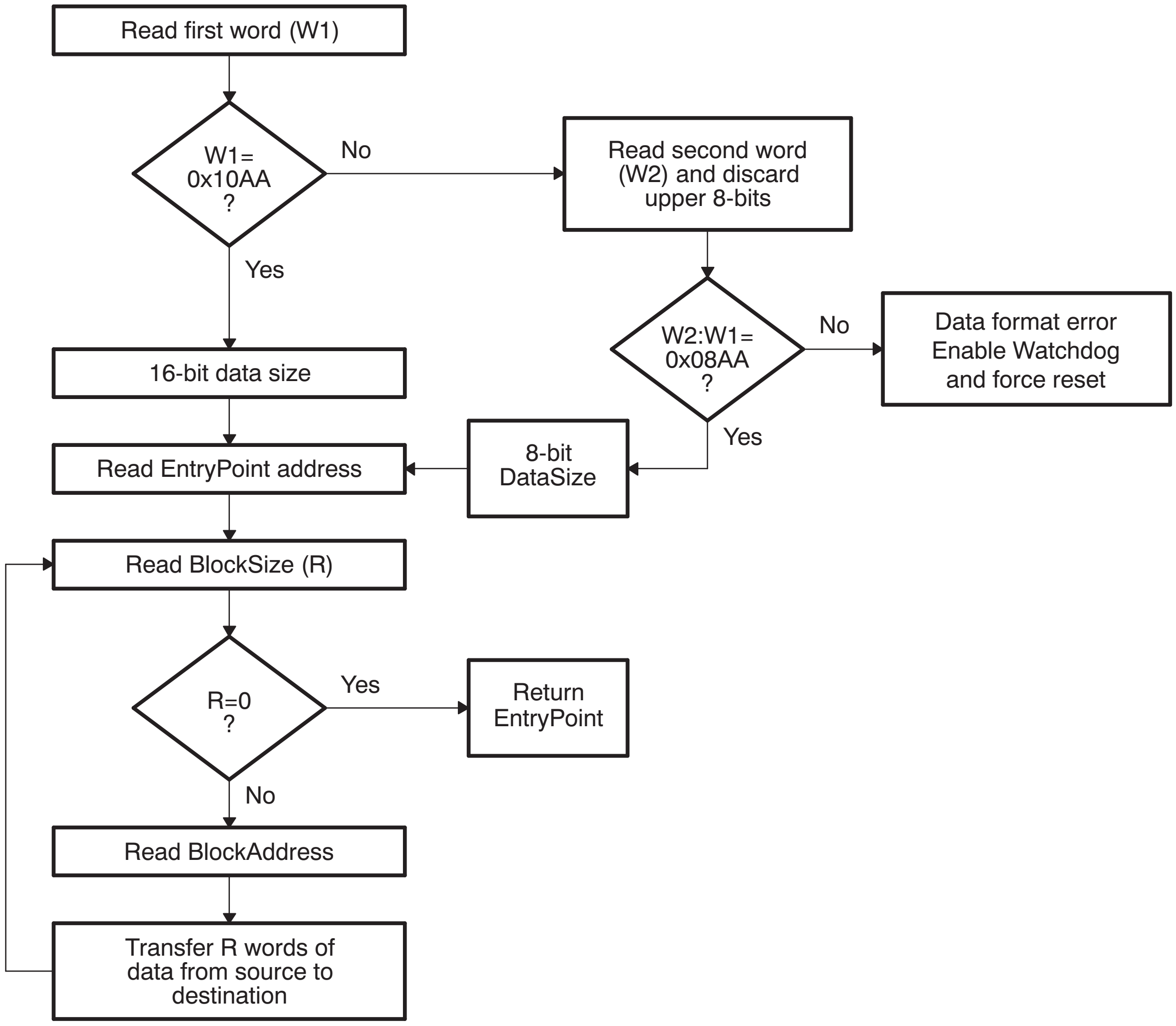
```
AA 08          ; 0x08AA 8-bit key value
00 00 00 00    ; 8 reserved words
00 00 00 00
00 00 00 00
00 00 00 00
3F 00 00 80    ; 0x003F8000 EntryAddr, starting point after boot load completes
05 00          ; 0x0005 - First block consists of 5 16-bit words
3F 00 10 90    ; 0x003F9010 - First block 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 determine whether 8-bit or 16-bit data stream has been selected, transfer that data, and start program execution. This process occurs after the bootloader finds the valid boot mode selected by the state of $\overline{\text{TRST}}$ and GPIO pins.

The loader first compares the first value sent by the host against the 16-bit key value of 0x10AA. If the value fetched does not match then the loader will read a second value. This value will be combined with the first value to form a word. This will then be checked against the 8-bit key value of 0x08AA. If the loader finds that the header does not match either the 8-bit or 16-bit key value, or if the value is not valid for the given boot mode then the load will abort.

Figure 2-6. Bootloader Basic Transfer Procedure



8-bit and 16-bit transfers are not valid for all boot modes. If only one mode is valid, then this decision tree is skipped and the key value is only checked for correctness. See the info specific to a particular bootloader for any limitations.

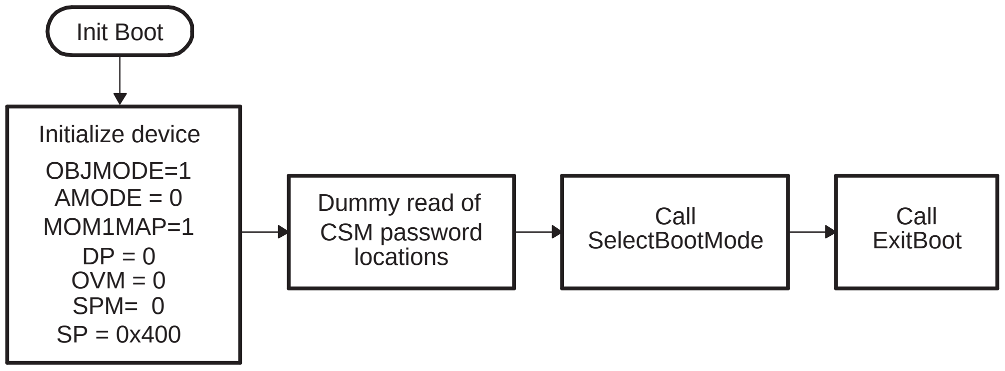
In 8-bit mode, the LSB of the 16-bit word is read first followed by the MSB.

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 CSM. Otherwise the CSM 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 $\overline{\text{TRST}}$ andcertain 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. 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 high or low 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

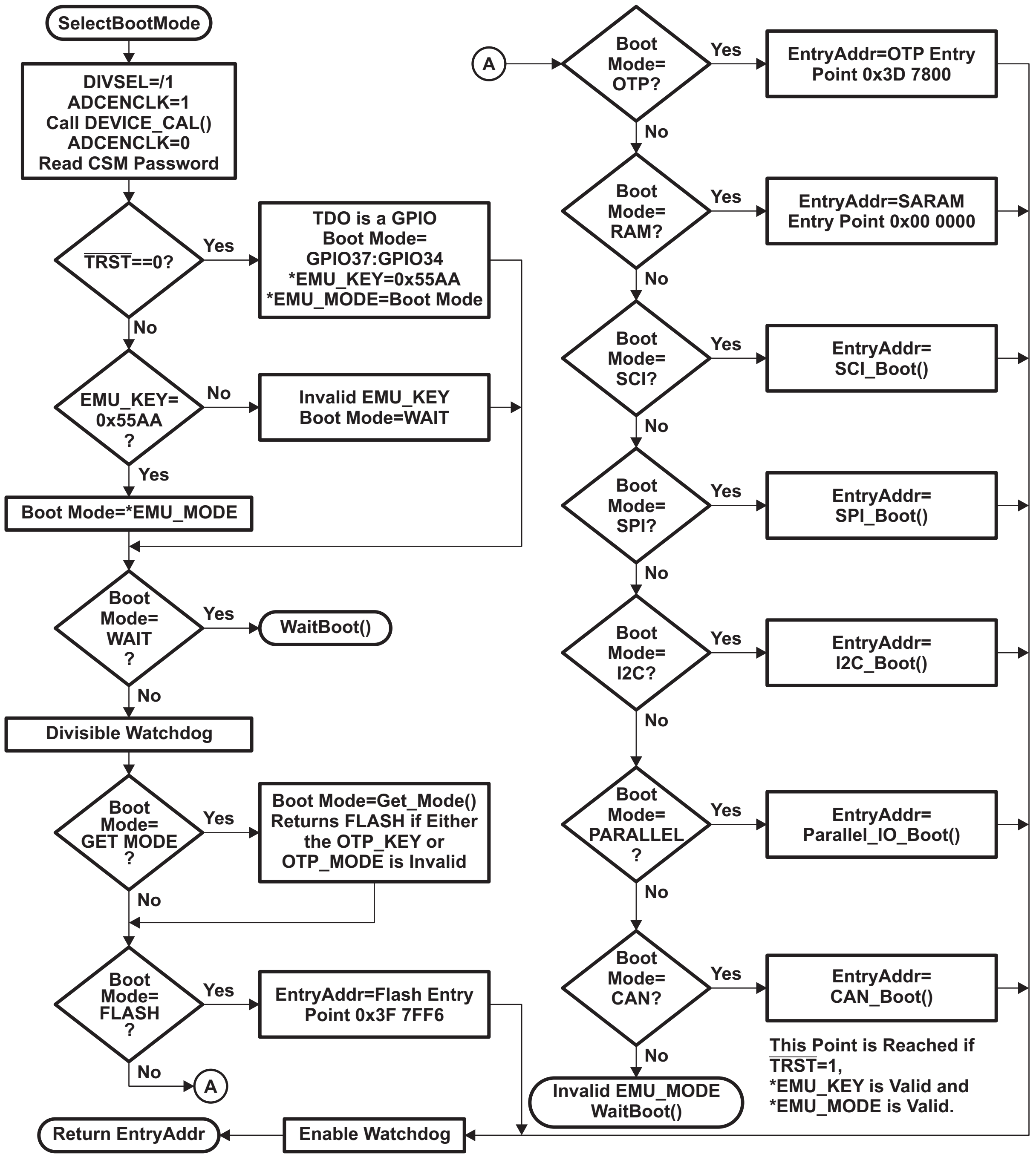
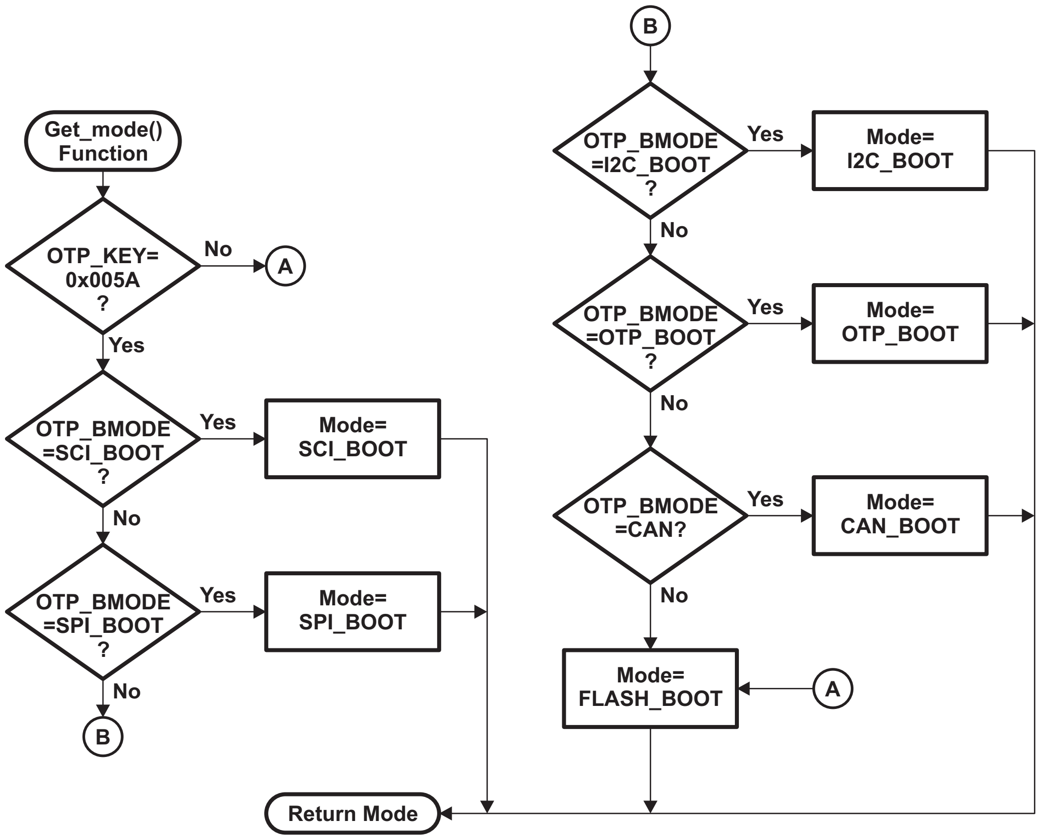


Figure 2-9. Overview of Get_mode() Function

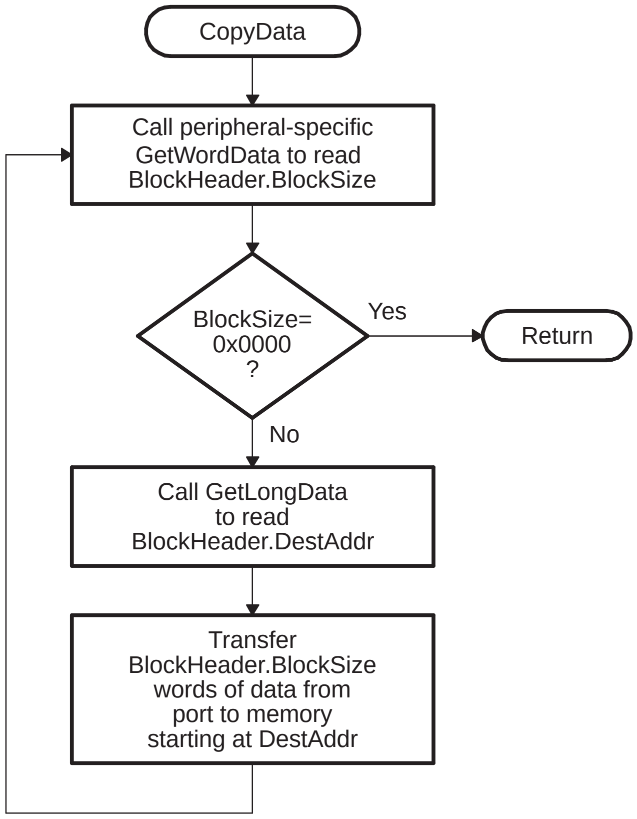


2.2.15 CopyData Function

Each of the bootloaders uses 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.

Note: BlockSize must be less than 0xFFFF for correct operation of the CopyData function. This means the max possible value of BlockSize is 0xFFFE, not 0xFFFF.

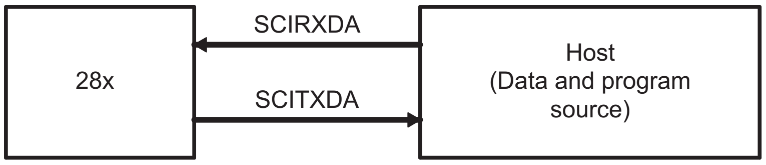
Figure 2-10. Overview of CopyData Function


CopyData
Call peripheral-specific GetWordData to read BlockHeader.BlockSize
BlockSize= 0x0000 ?
Yes
Return
No
Call GetLongData to read BlockHeader.DestAddr
Transfer BlockHeader.BlockSize words of data from port to memory starting at DestAddr

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 2-7.

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 by communication 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 effected 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

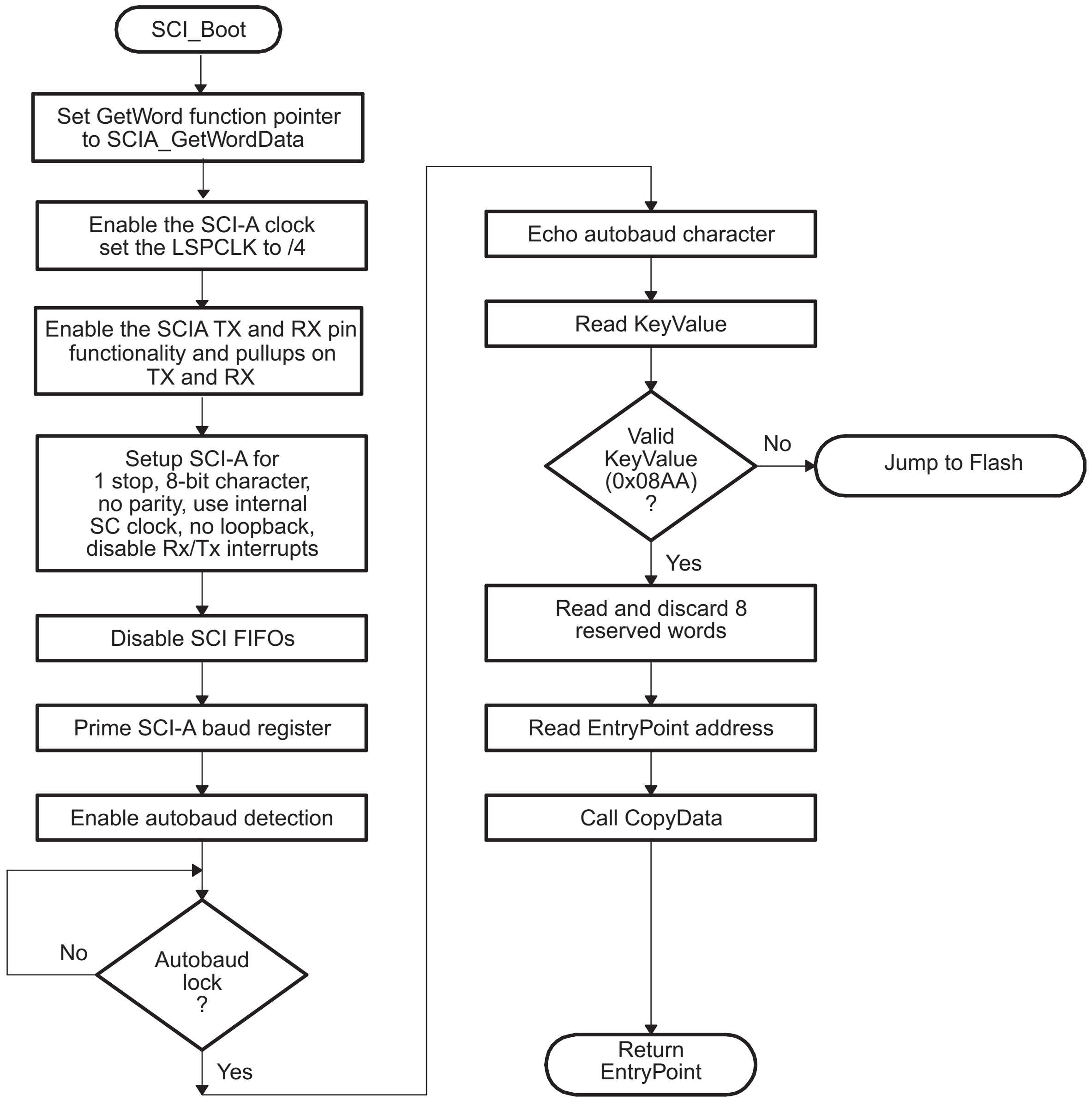
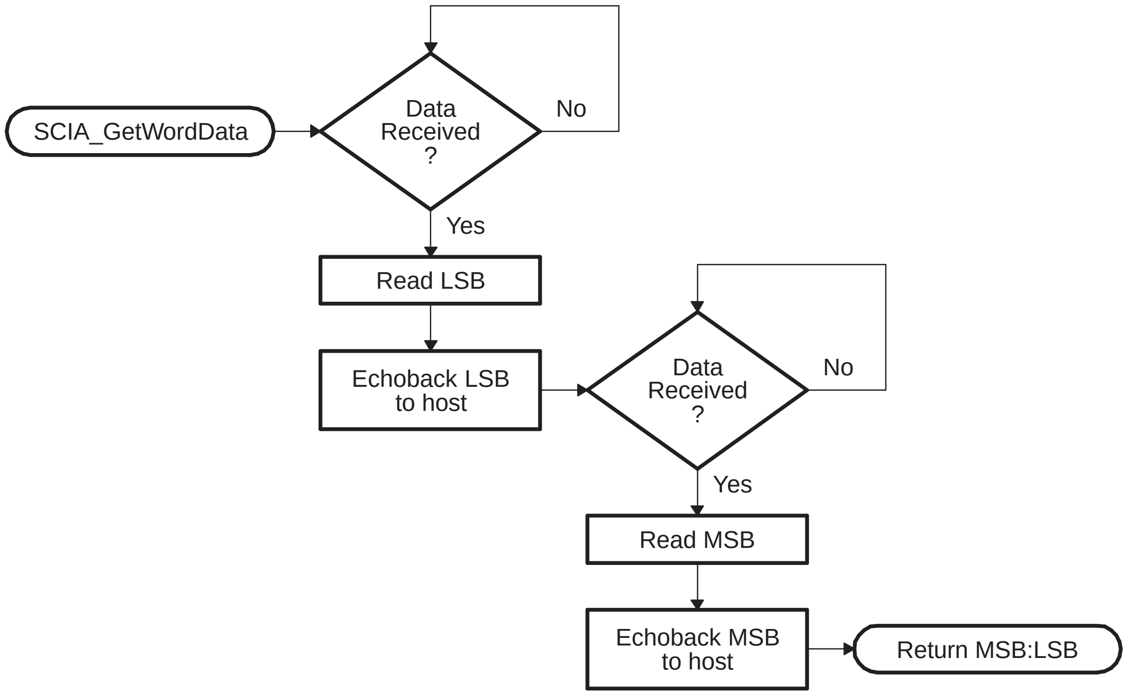

SCI_Boot
Set GetWord function pointer to SCIA_GetWordData
Enable the SCI-A clock set the LSPCLK to /4
Enable the SCIA TX and RX pin functionality and pullups on TX and RX
Setup SCI-A for 1 stop, 8-bit character, no parity, use internal SC clock, no loopback, disable Rx/Tx interrupts
Disable SCI FIFOs
Prime SCI-A baud register
Enable autobaud detection
Autobaud lock ?
No
Yes
Echo autobaud character
Read KeyValue
Valid KeyValue (0x08AA) ?
No
Jump to Flash
Yes
Read and discard 8 reserved words
Read EntryPoint address
Call CopyData
Return EntryPoint

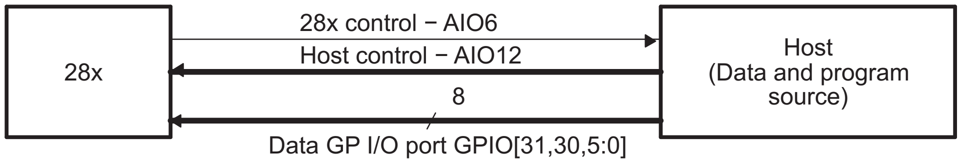
Figure 2-13. Overview of SCI_GetWordData Function



2.2.17 *Parallel_Boot Function (GPIO)*

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from GPIO0 -GPIO5, GPIO30-GPIO31 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[31,30,5:0]
- 28x Control on AIO6 (external pull-up resistor may be required)
- Host Control on AIO12 (external pull-up resistor required)

The 28x communicates with the external host device by polling/driving the AIO12 and AIO6 lines. An external pull-up resistor is required for AIO12 because AIO pins lack internal pull-up circuitry required to prevent the 28x from reading data prematurely. Depending on your system an external pull-up may also be required on AIO6. The handshake protocol shown in Figure 2-15 must be used to successfully transfer each word via . 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 GPIO[31,30,5:0].

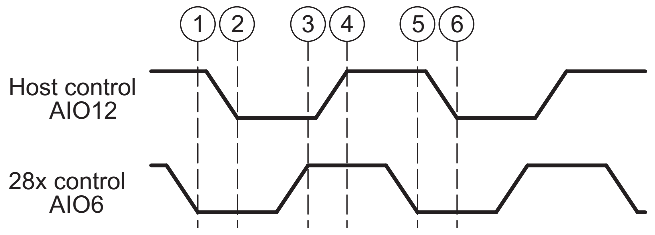
The 8-bit data stream is shown in Table 2-11.

Table 2-11. Parallel GPIO Boot 8-Bit Data Stream

Bytes		GPIO[31,30,5:0] (Byte 1 of 2)	GPIO[31,30,5:0] (Byte 2 of 2)	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 16bits)
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 AIO6 pin low. The host load then initiates the data transfer by pulling the AIO12 pin low. The complete protocol is shown in the diagram below:

Figure 2-15. Parallel GPIO Boot Loader Handshake Protocol



1. The 28x device indicates it is ready to start receiving data by pulling the AIO6pin low.
2. The bootloader waits until the host puts data on GPIO [31,30,5:0].The host signals to the 28x device that data is ready by pulling the AIO12 pin low.
3. The 28x device reads the data and signals the host that the read is complete by pulling AIO6 high.
4. The bootloader waits until the host acknowledges the 28x by pulling AIO12 high.
5. The 28x device again indicates it is ready for more data by pulling the AIO6 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

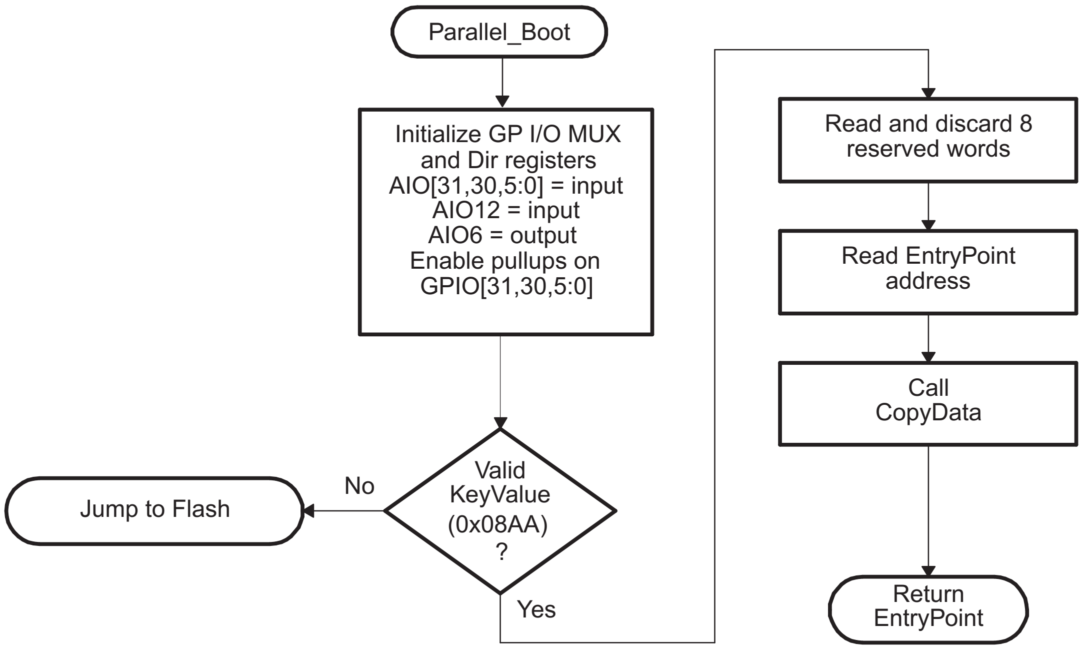

Parallel_Boot
Initialize GP I/O MUX and Dir registers AIO[31,30,5:0] = input AIO12 = input AIO6 = output Enable pullups on GPIO[31,30,5:0]
Valid KeyValue (0x08AA) ?
No
Yes
Jump to Flash
Read and discard 8 reserved words
Read EntryPoint address
Call CopyData
Return EntryPoint

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 and the 28x 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.

Figure 2-17. Parallel GPIO Mode - Host Transfer Flow

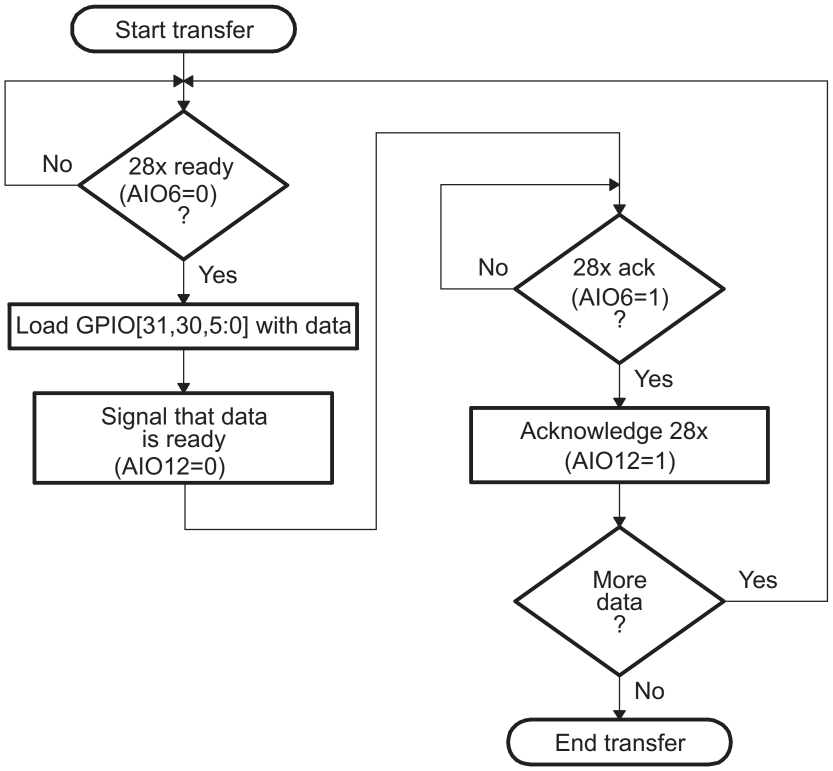
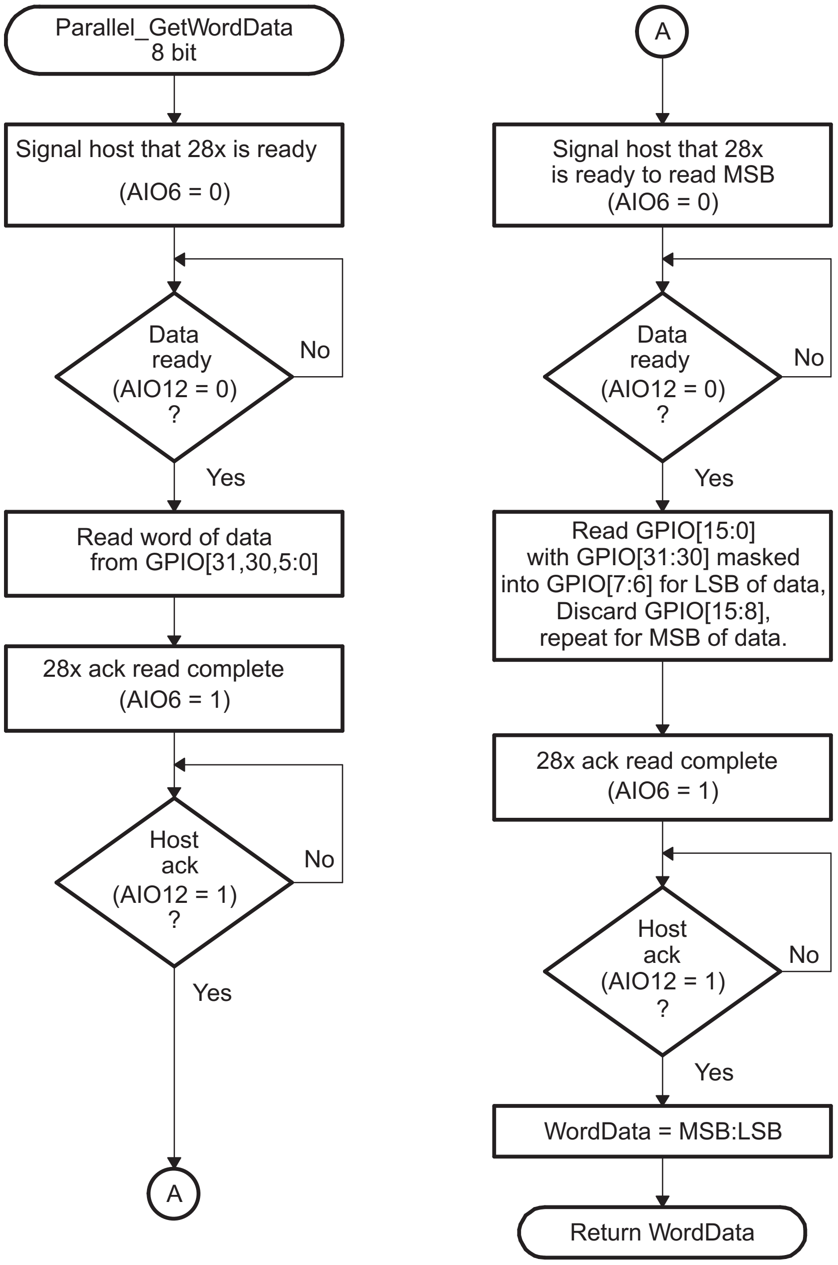

Start transfer
No
28x ready (AIO6=0) ?
Yes
Load GPIO[31,30,5:0] with data
Signal that data is ready (AIO12=0)
No
28x ack (AIO6=1) ?
Yes
Acknowledge 28x (AIO12=1)
More data ?
Yes
No
End transfer

Figure 2-18 shows 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 masked with GPIO31 in bit position 7 and GPIO30 in bit position 6 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.

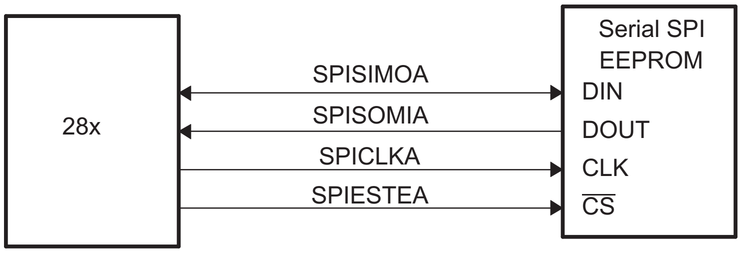
Figure 2-18. 8-Bit Parallel GetWord Function


Parallel_GetWordData 8 bit
Signal host that 28x is ready (AIO6 = 0)
Data ready (AIO12 = 0) ?
No
Yes
Read word of data from GPIO[31,30,5:0]
28x ack read complete (AIO6 = 1)
Host ack (AIO12 = 1) ?
No
Yes
A
A
Signal host that 28x is ready to read MSB (AIO6 = 0)
Data ready (AIO12 = 0) ?
No
Yes
Read GPIO[15:0] with GPIO[31:30] masked into GPIO[7:6] for LSB of data, Discard GPIO[15:8], repeat for MSB of data.
28x ack read complete (AIO6 = 1)
Host ack (AIO12 = 1) ?
No
Yes
WordData = MSB:LSB
Return WordData

2.2.18 SPI_Boot Function

The SPI loader expects an SPI-compatible 16-bit or 24-bit addressable serial EEPROM or serial flash 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 module to interface to a serial SPI EEPROM or flash. Devices of this type include, but are not limited to, the Xicor X25320 (4Kx8) and Xicor X25256 (32Kx8) SPI serial SPI EEPROMs and the Atmel AT25F1024A serial flash.

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 2-12. 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 2-12. 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:

Step 1. The SPI-A port is initialized

Step 2. The GPIO19 (SPISTE) pin is used as a chip-select for the serial SPI EEPROM or flash

Step 3. The SPI-A outputs a read command for the serial SPI EEPROM or flash

Step 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.

Step 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.

Step 6. The next 2 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.

Step 7. The next 2 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.

Step 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

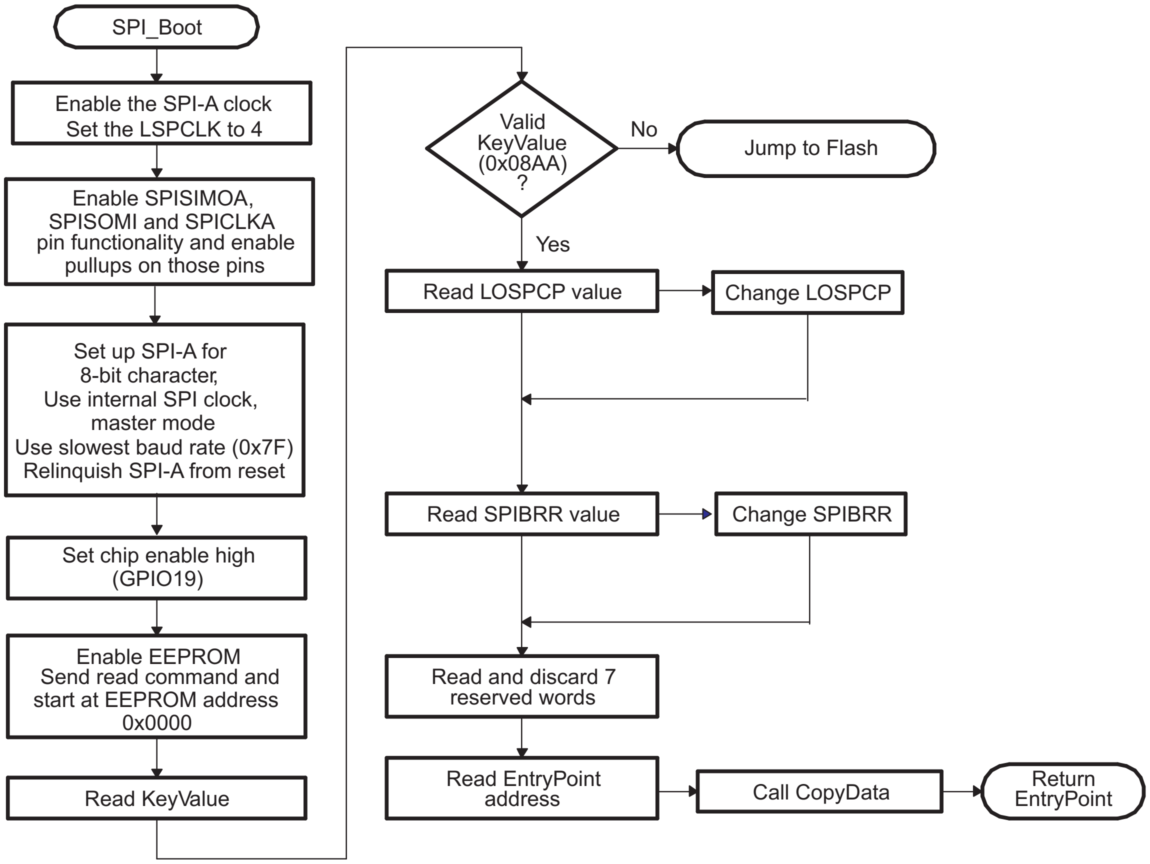
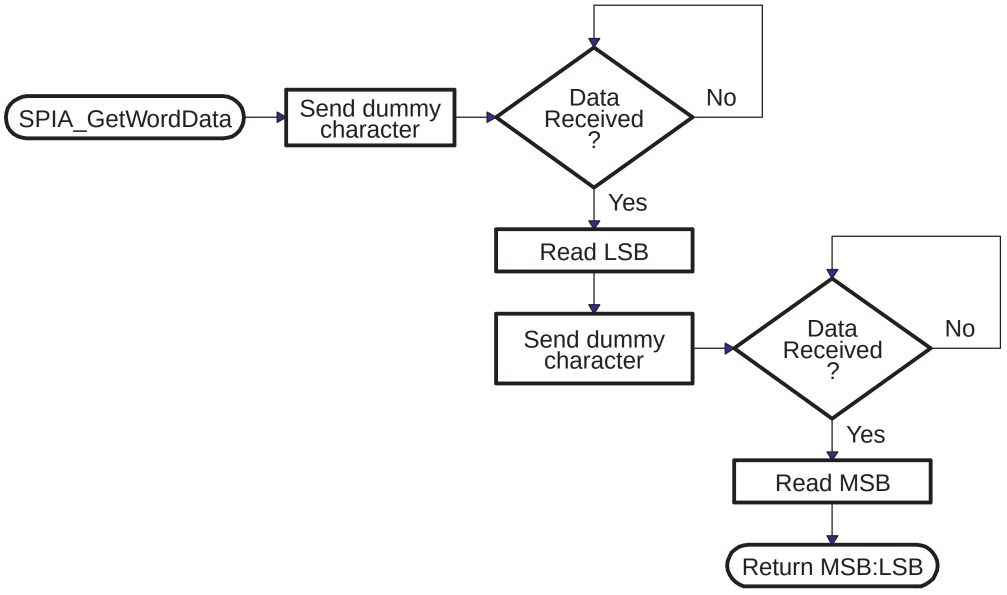

SPI_Boot
Enable the SPI-A clock Set the LSPCLK to 4
Enable SPISIMOA, SPISOMI and SPICLKA pin functionality and enable pullups on those pins
Set up SPI-A for 8-bit character, Use internal SPI clock, master mode Use slowest baud rate (0x7F) Relinquish SPI-A from reset
Set chip enable high (GPIO19)
Enable EEPROM Send read command and start at EEPROM address 0x0000
Read KeyValue
Valid KeyValue (0x08AA) ?
No
Jump to Flash
Yes
Read LOSPCP value
Change LOSPCP
Read SPIBRR value
Change SPIBRR
Read and discard 7 reserved words
Read EntryPoint address
Call CopyData
Return EntryPoint

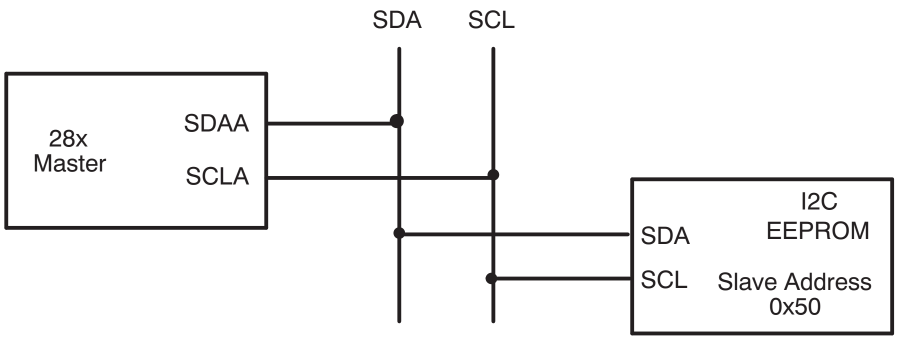
Figure 2-21. Overview of SPIA_GetWordData Function



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



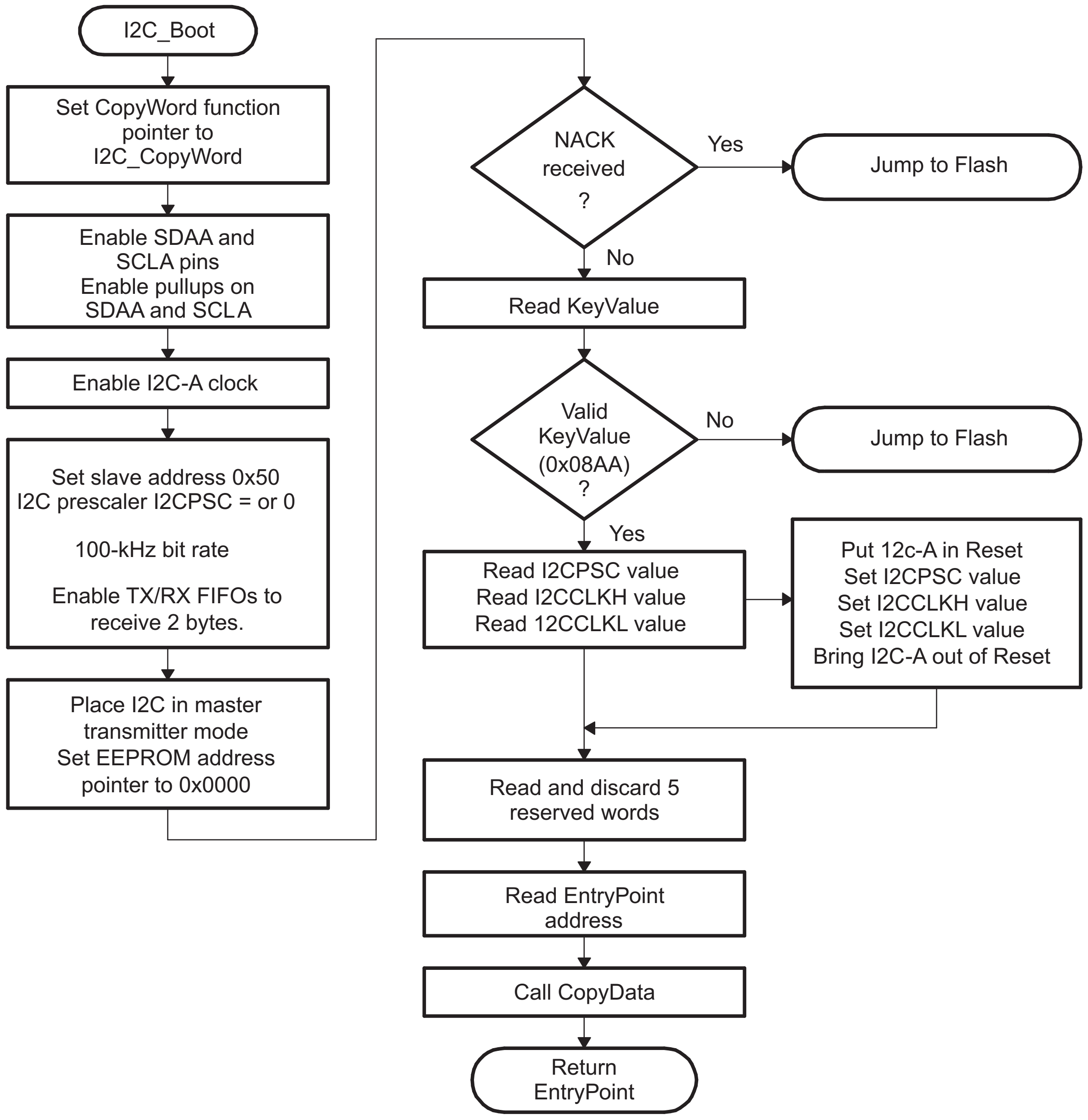
The I2C loader uses following pins:

- SDAA on GPIO 28
- SCLA on GPIO 29

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.

Figure 2-23. Overview of I2C_Boot Function


I2C_Boot
Set CopyWord function pointer to I2C_CopyWord
Enable SDAA and SCLA pins Enable pullups on SDAA and SCLA
Enable I2C-A clock
Set slave address 0x50
I2C prescaler I2CPSC = or 0
100-kHz bit rate
Enable TX/RX FIFOs to receive 2 bytes.
Place I2C in master transmitter mode Set EEPROM address pointer to 0x0000
NACK received ?
Yes
Jump to Flash
No
Read KeyValue
Valid KeyValue (0x08AA) ?
No
Jump to Flash
Yes
Read I2CPSC value Read I2CCLKH value Read 12CCLKL value
Put 12c-A in Reset Set I2CPSC value Set I2CCLKH value Set I2CCLKL value Bring I2C-A out of Reset
Read and discard 5 reserved words
Read EntryPoint address
Call CopyData
Return EntryPoint

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.

The non-acknowledgment 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 non-acknowledgment 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 3-2 shows the 8-bit data stream used by the I2C.

Table 2-13. 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. ...	
n		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

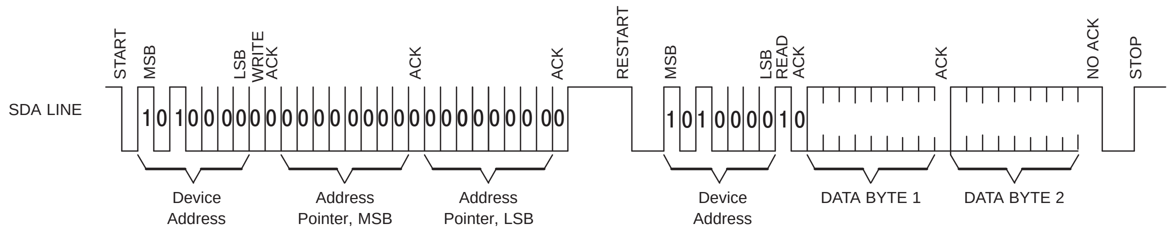
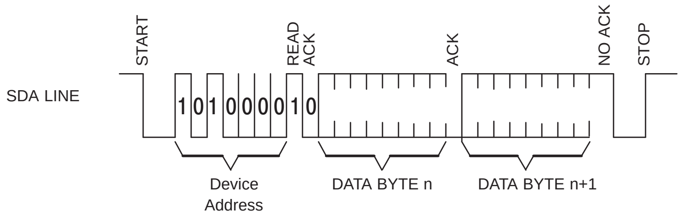


Figure 2-25. Sequential Read



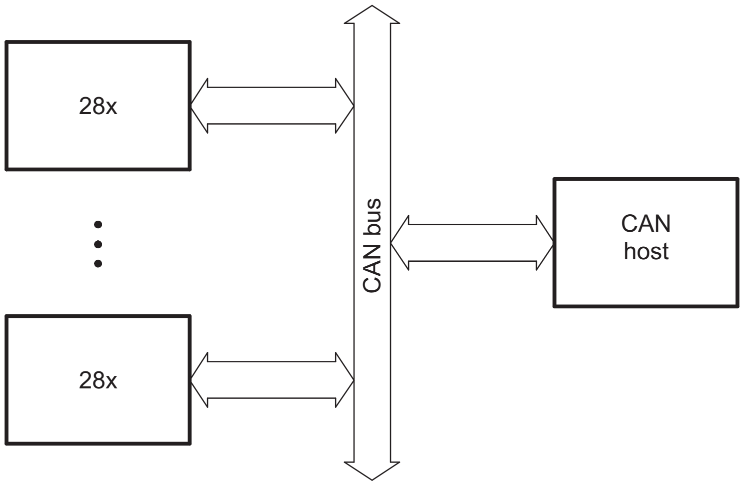
2.2.20 eCAN Boot Function

The eCAN bootloader asynchronously transfers code from eCAN-A to internal memory. The host can be any CAN node. The communication is first done with 11-bit standard identifiers (with a MSGID of 0x1) using two bytes per data frame. The host can download a kernel to reconfigure the eCAN if higher data throughput is desired.

The eCAN-A loader uses following pins:

- CANRXA on GPIO30
- CANTXA on GPIO31

Figure 2-26. Overview of eCAN-A bootloader Operation



The bit-timing registers are programmed in such a way that a valid bit-rate is achieved for a 10 MHz internal oscillator frequency as shown in Table 2-14.

Table 2-14. Bit-Rate Value for Internal Oscillators

OSCCLK	SYSCLKOUT	Bit Rate
10 MHz	10 MHz	100 kbps

The SYSCLKOUT values shown are the reset values with the default PLL setting. The BRP_{reg} and bit-time values are hard-coded to 1 and 25, respectively.

Mailbox 1 is programmed with a standard MSGID of 0x1 for boot-loader communication. The CAN host should transmit only 2 bytes at a time, LSB first and MSB next. For example, to transmit the word 0x08AA to the device, transmit AA first, followed by 08. The program flow of the CAN bootloader is identical to the SCI bootloader. The data sequence for the CAN bootloader is shown in Table 2-15:

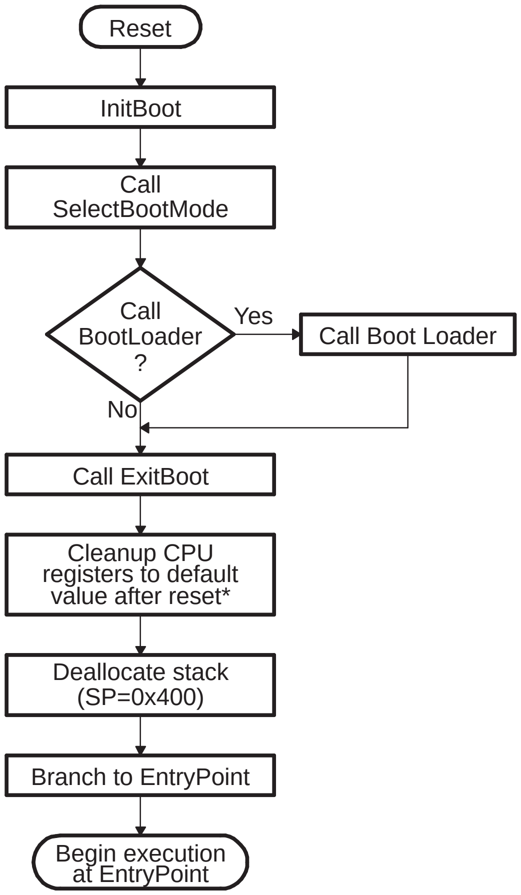
Table 2-15. eCAN 8-Bit Data Stream

Bytes		Byte 1 of 2	Byte 2 of 2	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 16bits)
3	4	00	00	reserved
5	6	00	00	reserved
7	8	00	00	reserved
9	10	00	00	reserved
11	12	00	00	reserved
13	14	00	00	reserved
15	16	00	00	reserved
17	18	00	00	reserved
19	20	BB	00	Entry point PC[22:16]
21	22	DD	CC	Entry point PC[15:0] (PC = 0xAABBCCDD)
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

2.2.21 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 the following diagram:

Figure 2-27. ExitBoot Procedure Flow


Reset
InitBoot
Call SelectBootMode
Call BootLoader ?
Yes
Call Boot Loader
No
Call ExitBoot
Cleanup CPU registers to default value after reset*
Deallocate stack (SP=0x400)
Branch to EntryPoint
Begin execution at EntryPoint

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:

Table 2-16. 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	15:10	OVC = 0	ST1	0x0A0B	15:13	ARP = 0
		9:7	PM = 0			12	XF = 0
		6	V = 0			11	M0M1MAP = 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, eCAN, 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, hex, etc.) 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.

See the *TMS320C28x Assembly Language Tools User's Guide* (SPRU513) and the *TMS320C28x Optimizing C/C++ Compiler User's Guide* (SPRU514) for more information on the compiling and linking process.

Table 2-17 summarizes the hex conversion utility options available for the bootloader. See the *TMS320C28x Assembly Language Tools User's Guide* (SPRU513) 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 2-17. Boot Loader 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
-gpio16	Specify the source of the bootloader table as the GPIO port, 16-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 _c_int00 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.
-i2cclkh 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.
-i2cclkl 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 eCAN Bootloading*

This section shows how to convert a COFF file into a format suitable for CAN 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 2-8 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 User's Guide* (SPRU513) for detailed information on generating a linker command file and a memory map.

All sections shown in Example 2-8 that are initialized need to be loaded into the DSP 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 2-8. GPIO34TOG Map File

```
output                                           attributes/
section        page    origin      length       input sections
--------       ----   ----------  ----------   ----------------
codestart
                 0     00000000    00000002
                       00000000    00000002    DSP280x_CodeStartBranch.obj (codestart)
.pinit           0     00000002    00000000

.switch          0     00000002    00000000    UNINITIALIZED

ramfuncs         0     00000002    00000016
                       00000002    00000016    DSP280x_SysCtrl.obj (ramfuncs)

.cinit           0     00000018    00000019
                       00000018    0000000e    rts2800_ml.lib : exit.obj (.cinit)
                       00000026    0000000a                   : _lock.obj (.cinit)
                       00000030    00000001    --HOLE-- [fill = 0]

myreset          0     00000032    00000002
                       00000032    00000002    DSP280x_CodeStartBranch.obj (myreset)

IQmath           0     003fa000    00000000    UNINITIALIZED

.text            0     003fa000    00000155
                       003fa000    00000046    rts2800_ml.lib : boot.obj (.text)
```

To load the code using the CAN 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 2-9. 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 eCAN
         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 2-9 will generate an ASCII-Hex output file called GPIO34TOG.a00, whose contents are explained in Example 2-10. 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 2-8. After the data stream is loaded, the boot ROM will jump to the Entrypoint address that was read as part of the data stream. In this case, execution will begin at 0x3FA0000.

Example 2-10. 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. Taking a 64-bit summation of all addresses within the ROM, except for the checksum locations, generates this checksum.

Table 2-18. Bootloader Revision and Checksum Information

Address	Contents
0x3F FFB9	
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

2.5 Revision History

The following technical changes were made to this document since the last release.

Table 2-19. Additions, Deletes, Changes

Location	Changes
Table 2-12	Added the following to Step 5 following the table: "the device will branch to the flash entry point address"

Enhanced Pulse Width Modulator (ePWM) Module

The enhanced pulse width modulator (ePWM) peripheral 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 peripheral 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 guide is applicable for ePWM type 1. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566) 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 its sub-modules:

- 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 features:

- **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 the device-specific HRPWM Reference Guide.
- **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.

Topic **Page**

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 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 the device-specific *High-Resolution Pulse Width Modulator (HRPWM) Reference Guide*. See the device-specific data manual 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 3rd instance in the system and ePWMx indicates any instance.

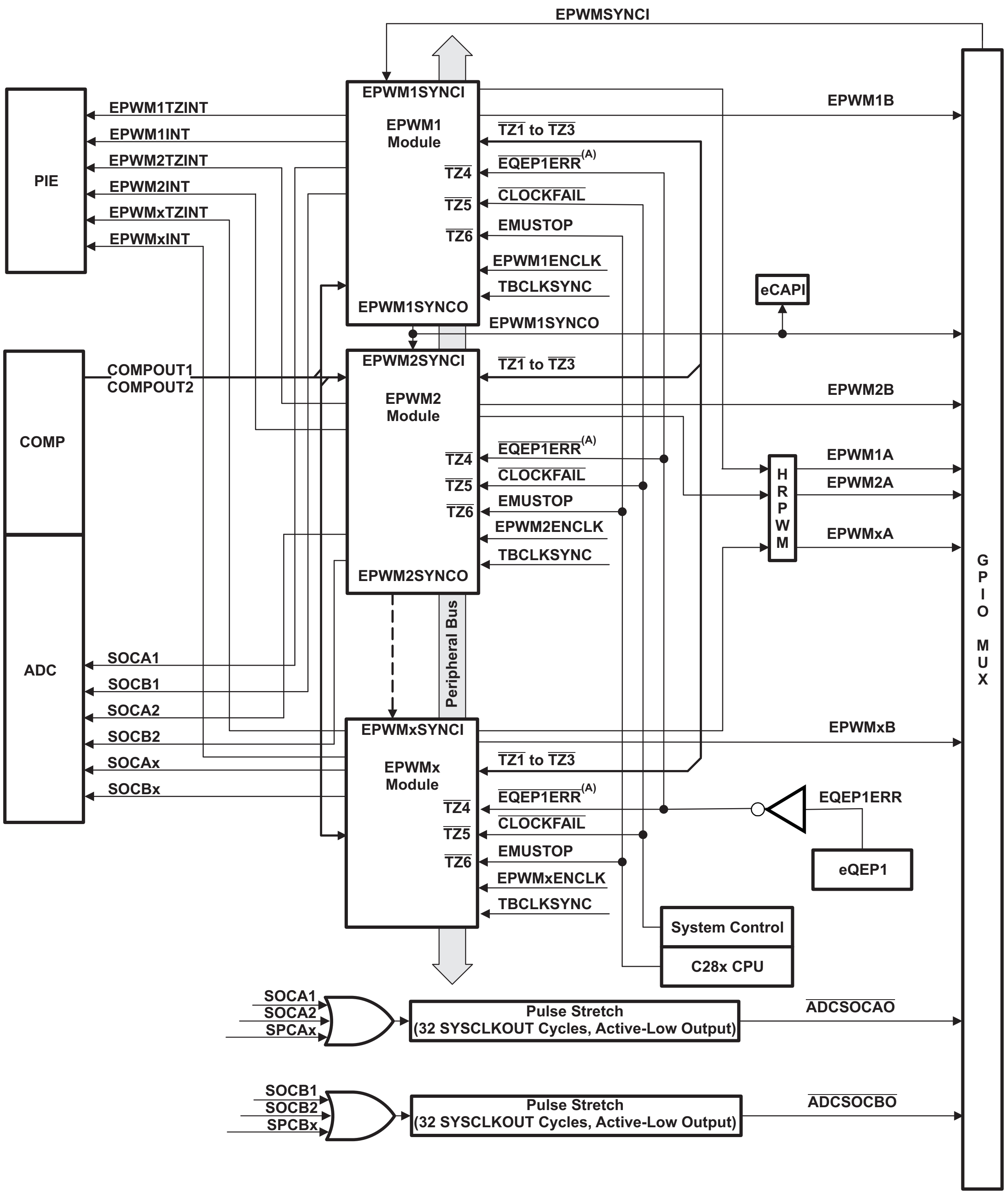
The ePWM modules are chained together via a clock synchronization scheme that allows them to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral modules (eCAP). The number of modules is device-dependent and based on target application 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.
- 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.

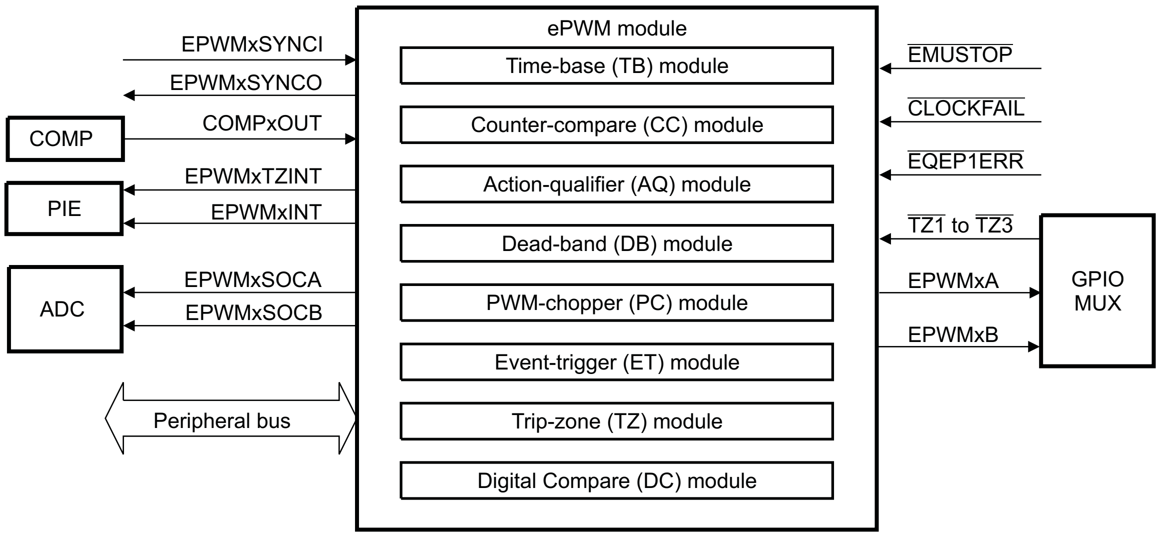
Figure 3-1. Multiple ePWM Modules



A This signal exists only on devices with an eQEP1 module.

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 via the signals shown in Figure 3-2.

Figure 3-2. Submodules and Signal Connections for an ePWM Module


ePWM module
Time-base (TB) module
Counter-compare (CC) module
Action-qualifier (AQ) module
Dead-band (DB) module
PWM-chopper (PC) module
Event-trigger (ET) module
Trip-zone (TZ) module
Digital Compare (DC) module
EPWMxSYNCI
EPWMxSYNCO
COMP
COMPxOUT
PIE
EPWMxTZINT
EPWMxINT
ADC
EPWMxSOCA
EPWMxSOCB
Peripheral bus
$\overline{\text{EMUSTOP}}$
$\overline{\text{CLOCKFAIL}}$
$\overline{\text{EQEP1ERR}}$
$\overline{\text{TZ1}}$ to $\overline{\text{TZ3}}$
EPWMxA
EPWMxB
GPIO MUX

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 ($\overline{\text{TZ1}}$ to $\overline{\text{TZ6}}$).**

  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 $\overline{\text{TZ1}}$ to $\overline{\text{TZ3}}$ trip-zone signals can be configured as asynchronous inputs through the GPIO peripheral. $\overline{\text{TZ4}}$ is connected to an inverted EQEP1 error signal (EQEP1ERR) from the EQEP1 module (for those devices with an EQEP1 module). $\overline{\text{TZ5}}$ is connected to the system clock fail logic, and $\overline{\text{TZ6}}$ is connected to the $\overline{\text{EMUSTOP}}$ output from the CPU. This allows you to configure a trip action when the clock fails or the CPU halts.

- **Time-base synchronization input (EPWMxSYNCI) 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 SYNCI 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.

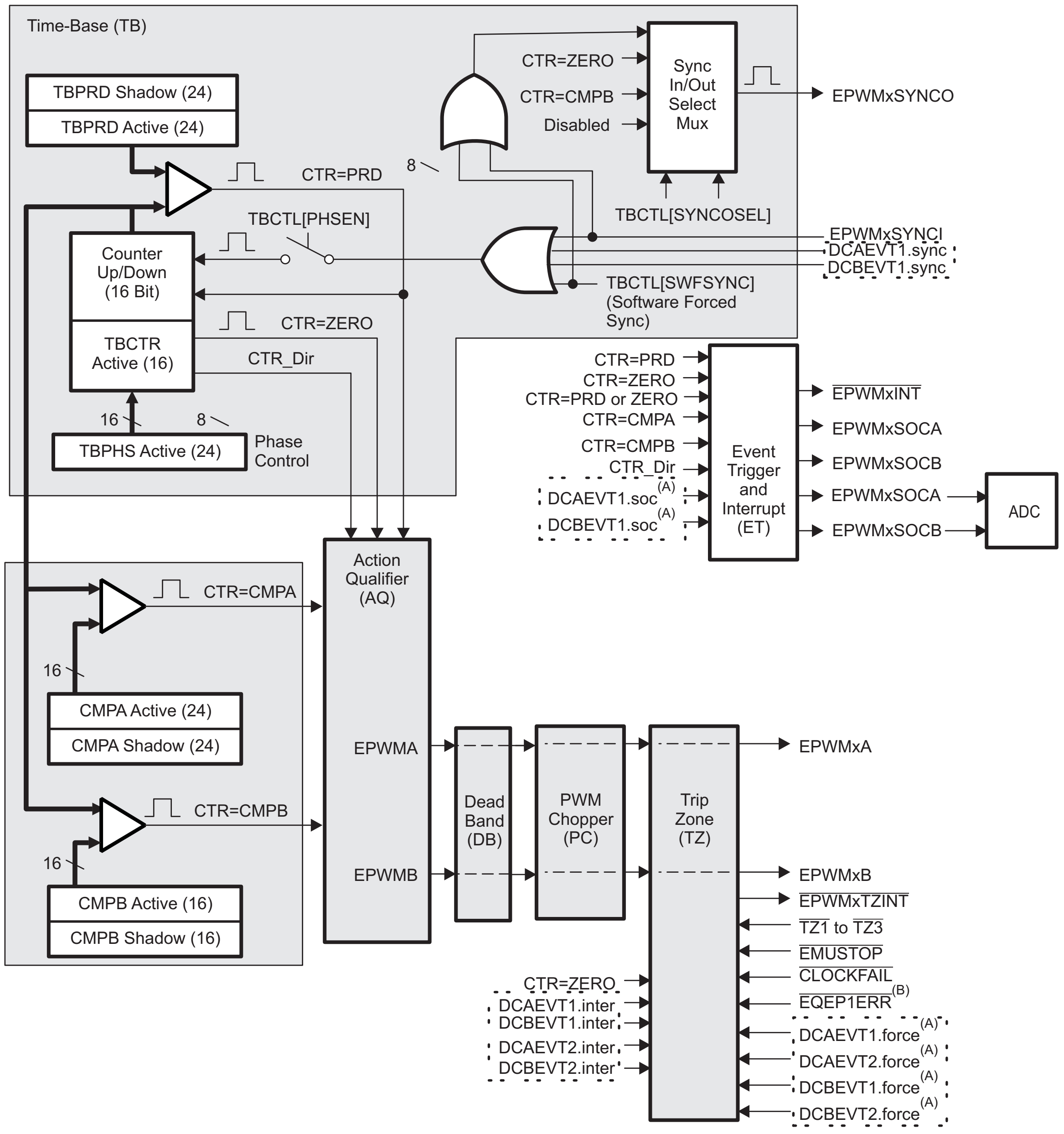
- **Comparator output signals (COMPxOUT).**

  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. ePWM Submodules and Critical Internal Signal Interconnects



A These events are generated by the type 1 ePWM digital compare (DC) submodule based on the levels of the COMPxOUT and $\overline{\text{TZ}}$ signals.

B This signal exists only on devices with in eQEP1 module

also shows the key internal submodule interconnect signals. Each submodule is described in detail in its respective section.

3.1.2 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 [1]	Size (x16)	Shadow	EALLOW	Description
					Time-Base Submodule Registers
TBCTL	0x0000	1	No		Time-Base Control Register
TBSTS	0x0001	1	No		Time-Base Status Register
TBPHSHR	0x0002	1	No		Extension for HRPWM Phase Register [2]
TBPHS	0x0003	1	No		Time-Base Phase Register
TBCTR	0x0004	1	No		Time-Base Counter Register
TBPRD	0x0005	1	Yes		Time-Base Period Register
TBPRDHR	0x0006	1	Yes		Time Base Period High Resolution Register [3]
					Counter-Compare Submodule Registers
CMPCTL	0x0007	1	No		Counter-Compare Control Register
CMPAHR	0x0008	1	Yes		Extension for HRPWM Counter-Compare A Register [2]
CMPA	0x0009	1	Yes		Counter-Compare A Register
CMPB	0x000A	1	Yes		Counter-Compare B Register
					Action-Qualifier Submodule Registers
AQCTLA	0x000B	1	No		Action-Qualifier Control Register for Output A (EPWMxA)
AQCTLB	0x000C	1	No		Action-Qualifier Control Register for Output B (EPWMxB)
AQSFRC	0x000D	1	No		Action-Qualifier Software Force Register
AQCSFRC	0x000E	1	Yes		Action-Qualifier Continuous S/W Force Register Set
					Dead-Band Generator Submodule Registers
DBCTL	0x000F	1	No		Dead-Band Generator Control Register
DBRED	0x0010	1	No		Dead-Band Generator Rising Edge Delay Count Register
DBFED	0x0011	1	No		Dead-Band Generator Falling Edge Delay Count Register
					Trip-Zone Submodule Registers
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 [3]
TZEINT	0x0015	1		Yes	Trip-Zone Enable Interrupt Register [3]
TZFLG	0x0016	1			Trip-Zone Flag Register [3]
TZCLR	0x0017	1		Yes	Trip-Zone Clear Register [3]
TZFRC	0x0018	1		Yes	Trip-Zone Force Register [3]
					Event-Trigger Submodule Registers
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
					PWM-Chopper Submodule Registers
PCCTL	0x001E	1			PWM-Chopper Control Register
					High-Resolution Pulse Width Modulator (HRPWM) Extension Registers
HRCNFG	0x0020	1		Yes	HRPWM Configuration Register [2] [3]
HRPWR	0x0021	1		Yes	HRPWM Power Register [3] [4]

[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 described in the *High-Resolution Pulse Width Modulator (HRPWM)* section of this manual. See the device specific data manual to determine which instances include the HRPWM.

[3] EALLOW protected registers as described in the specific device version of the *System Control and Interrupts Reference Guide*.

[4] These registers only exist in the ePWM1 register space. They cannot be accessed from any other ePWM module's register space.

Table 3-1. ePWM Module Control and Status Register Set Grouped by Submodule (continued)

Name	Offset [(1)]	Size (x16)	Shadow	EALLOW	Description
HRMSTEP	0x0026	1		Yes	HRPWM MEP Step Register[(3)] [(4)]
HRPCTL	0x0028	1		Yes	High Resolution Period Control Register[(3)]
TBPRDHRM	0x002A	1	Writes		Time Base Period High Resolution Register Mirror[(3)]
TBPRDM	0x002B	1	Writes		Time Base Period Register Mirror
CMPAHRM	0x002C	1	Writes		Compare A High Resolution Register Mirror[(3)]
CMPAM	0x002D	1	Writes		Compare A Register Mirror
					Digital Compare Event Registers
DCTRIPSEL	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

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:

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)	• Scale the time-base clock (TBCLK) relative to the system clock (SYSCLKOUT). • Configure the PWM time-base counter (TBCTR) frequency or period. • Set the mode for the time-base counter: – count-up mode: used for asymmetric PWM – count-down mode: used for asymmetric PWM – count-up-and-down mode: used for symmetric PWM • Configure the time-base phase relative to another ePWM module. • Synchronize the time-base counter between modules through hardware or software. • Configure the direction (up or down) of the time-base counter after a synchronization event. • Configure how the time-base counter will behave when the device is halted by an emulator. • Specify the source for the synchronization output of the ePWM module: – Synchronization input signal – Time-base counter equal to zero – Time-base counter equal to counter-compare B (CMPB) – No output synchronization signal generated.
Counter-compare (CC)	• Specify the PWM duty cycle for output EPWMxA and/or output EPWMxB • Specify the time at which switching events occur on the EPWMxA or EPWMxB output
Action-qualifier (AQ)	• Specify the type of action taken when a time-base or counter-compare submodule event occurs: – No action taken – Output EPWMxA and/or EPWMxB switched high – Output EPWMxA and/or EPWMxB switched low – Output EPWMxA and/or EPWMxB toggled • Force the PWM output state through software control • Configure and control the PWM dead-band through software
Dead-band (DB)	• Control of traditional complementary dead-band relationship between upper and lower switches • Specify the output rising-edge-delay value • Specify the output falling-edge delay value • Bypass the dead-band module entirely. In this case the PWM waveform is passed through without modification. • Option to enable half-cycle clocking for double resolution.
PWM-chopper (PC)	• Create a chopping (carrier) frequency. • Pulse width of the first pulse in the chopped pulse train. • Duty cycle of the second and subsequent pulses. • Bypass the PWM-chopper module entirely. In this case the PWM waveform is passed through without modification.

Table 3-2. Submodule Configuration Parameters (continued)

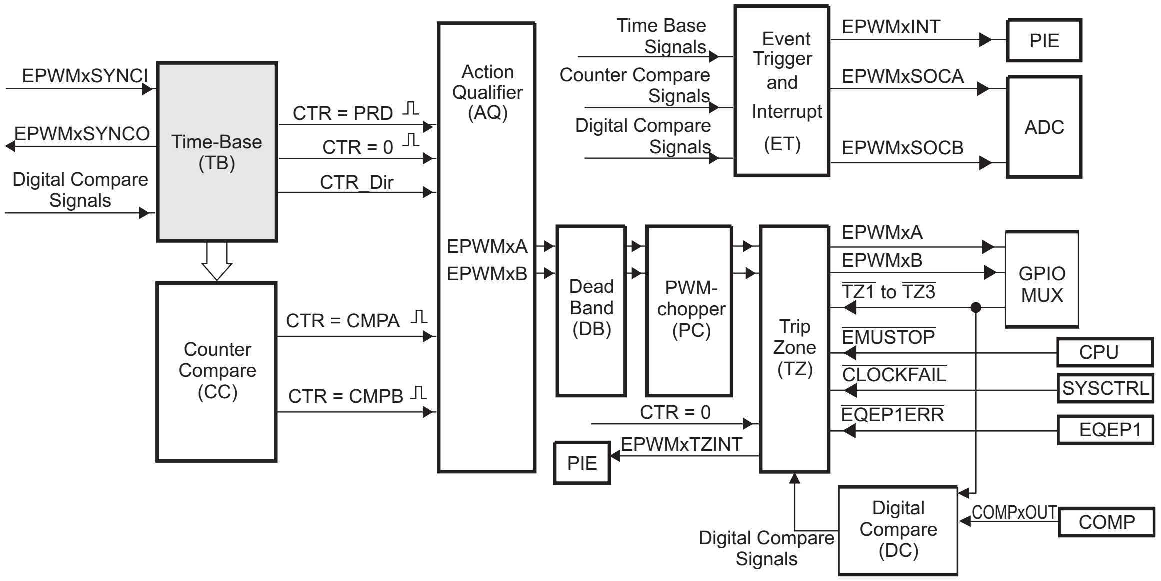
Submodule	Configuration Parameter or Option
Trip-zone (TZ)	• Configure the ePWM module to react to one, all, or none of the trip-zone signals or digital compare events. • Specify the tripping action taken when a fault occurs: – Force EPWMxA and/or EPWMxB high – Force EPWMxA and/or EPWMxB low – Force EPWMxA and/or EPWMxB to a high-impedance state – Configure EPWMxA and/or EPWMxB to ignore any trip condition. • Configure how often the ePWM will react to each trip-zone signal: – One-shot – Cycle-by-cycle • Enable the trip-zone to initiate an interrupt. • Bypass the trip-zone module entirely.
Event-trigger (ET)	• Enable the ePWM events that will trigger an interrupt. • Enable ePWM events that will trigger an ADC start-of-conversion event. • Specify the rate at which events cause triggers (every occurrence or every second or third occurrence) • Poll, set, or clear event flags
Digital-compare (DC)	• Enables comparator (COMP) module outputs and trip zone signals to create events and filtered events • Specify event-filtering options to capture TBCTR counter or generate blanking window

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.

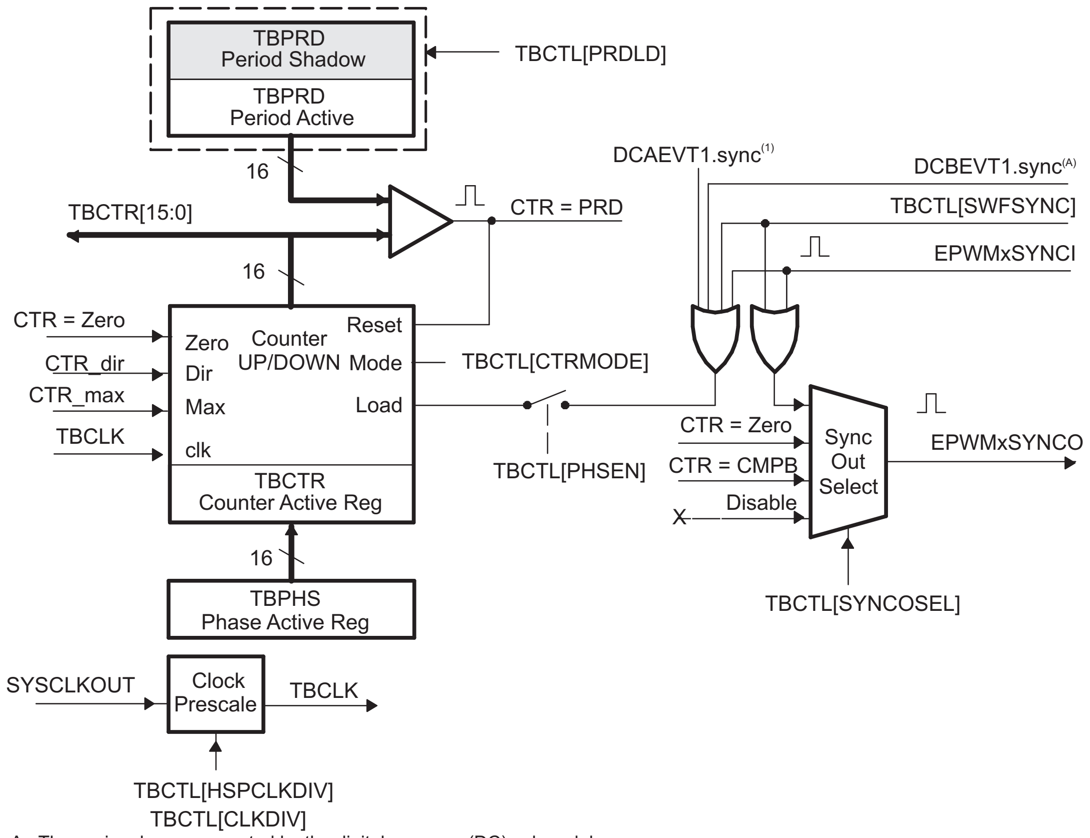
Table 3-3. Time-Base Submodule Registers

Register	Address offset	Shadowed	Description
TBCTL	0x0000	No	Time-Base Control Register
TBSTS	0x0001	No	Time-Base Status Register
TBPHSHR	0x0002	No	HRPWM Extension Phase Register [(1)]
TBPHS	0x0003	No	Time-Base Phase Register
TBCTR	0x0004	No	Time-Base Counter Register
TBPRD	0x0005	Yes	Time-Base Period Register
TBPRDHR	0x0006	Yes	HRPWM Extension Period Register[(1)]
TBPRDHRM	0x002A	Yes	HRPWM Time-Base Period Extension Mirror Register[(1)]
TBPRDM	0x002B	Yes	HRPWM Extension Period Mirror Register[(1)]

(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 described in the device-specific High-Resolution Pulse Width Modulator (HRPWM) Reference Guide. See the device specific data manual 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



A. These signals are generated by the digital compare (DC) submodule.

Table 3-4. Key Time-Base Signals

Signal	Description
EPWMxSYNCI	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, EPWM2SYNCI is generated by the ePWM1 peripheral, EPWM3SYNCI is generated by ePWM2 and so forth. See Section 3.2.2.3.3 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: 1. EPWMxSYNCI (Synchronization input pulse) 2. CTR = Zero: The time-base counter equal to zero (TBCTR = 0x0000). 3. CTR = CMPB: The time-base counter equal to the counter-compare B (TBCTR = CMPB) register.
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 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):

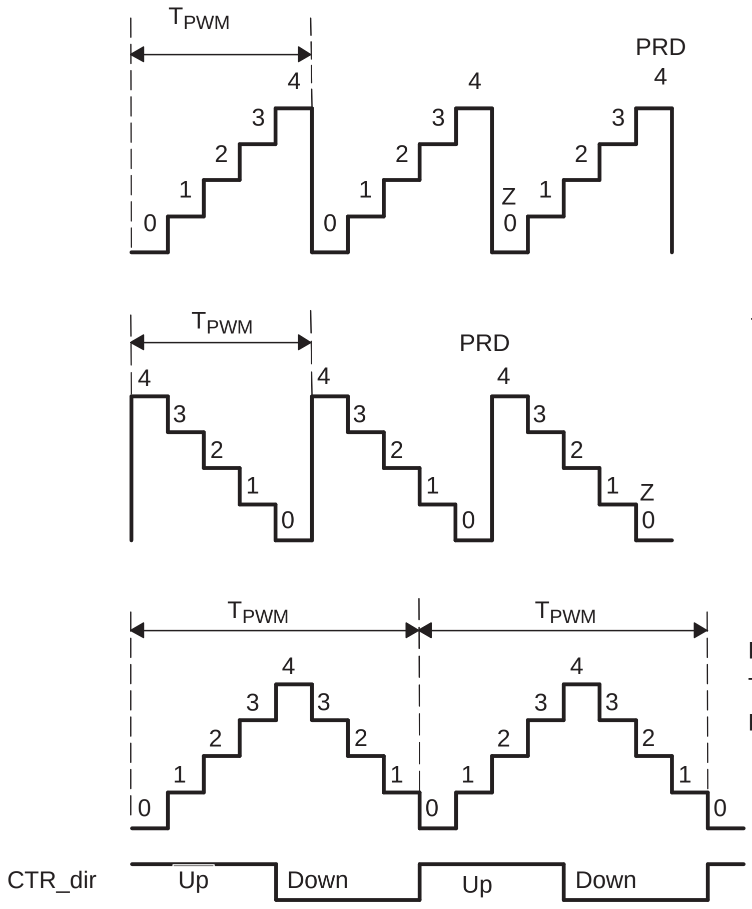
- **Up-Down-Count Mode:**

  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


T_{PWM}
PRD
4
3
2
1
0
Z
For Up Count and Down Count
$T_{PWM} = (TBPRD + 1) \times T_{TBCLK}$
$F_{PWM} = 1/ (T_{PWM})$
For Up and Down Count
$T_{PWM} = 2 \times TBPRD \times T_{TBCLK}$
$F_{PWM} = 1 / (T_{PWM})$
CTR_dir
Up
Down

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[PRDLD] 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[PRDLD] = 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[PRDLD] = 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:

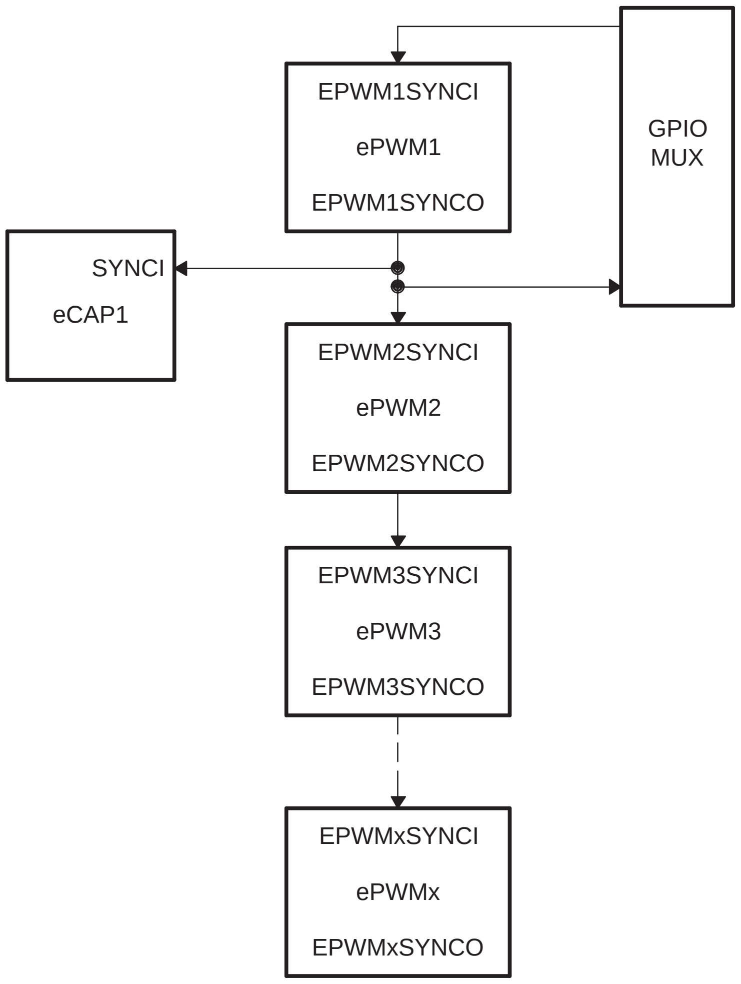
1. Enable ePWM module clocks in the PCLKCRx register
2. Set TBCLKSYNC= 0
3. Configure ePWM modules
4. Set TBCLKSYNC=1

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, Figure 3-8, and Figure 3-9.

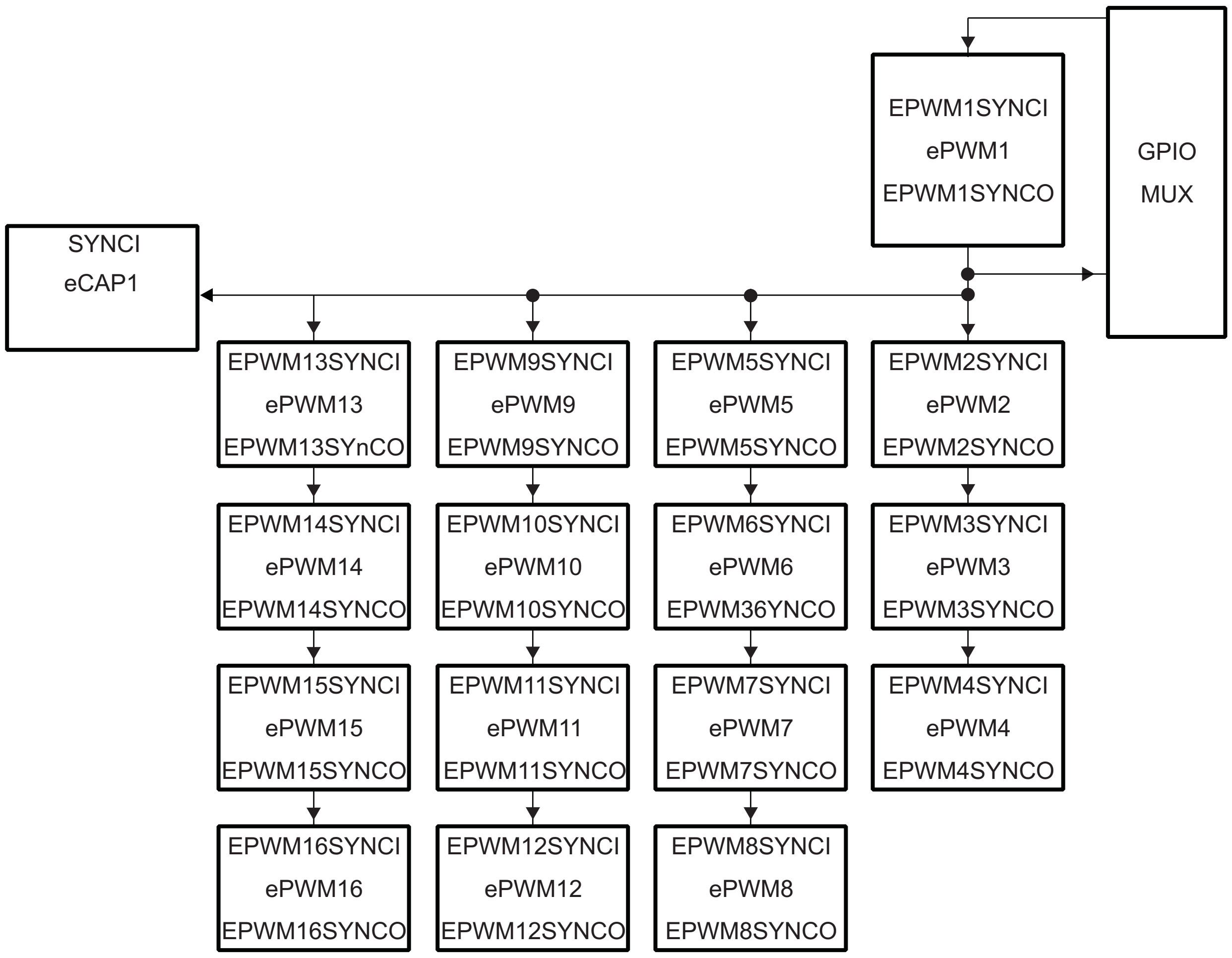
Scheme 1 shown in Figure 3-7 applies to the 280x, 2801x, 2802x, 2803x, and 2806x devices. Scheme 1 also applies to the 2804x devices when the ePWM pinout is configured for 280x compatible mode (GPAMCFG[EPWMMODE] = 0).

Figure 3-7. Time-Base Counter Synchronization Scheme 1



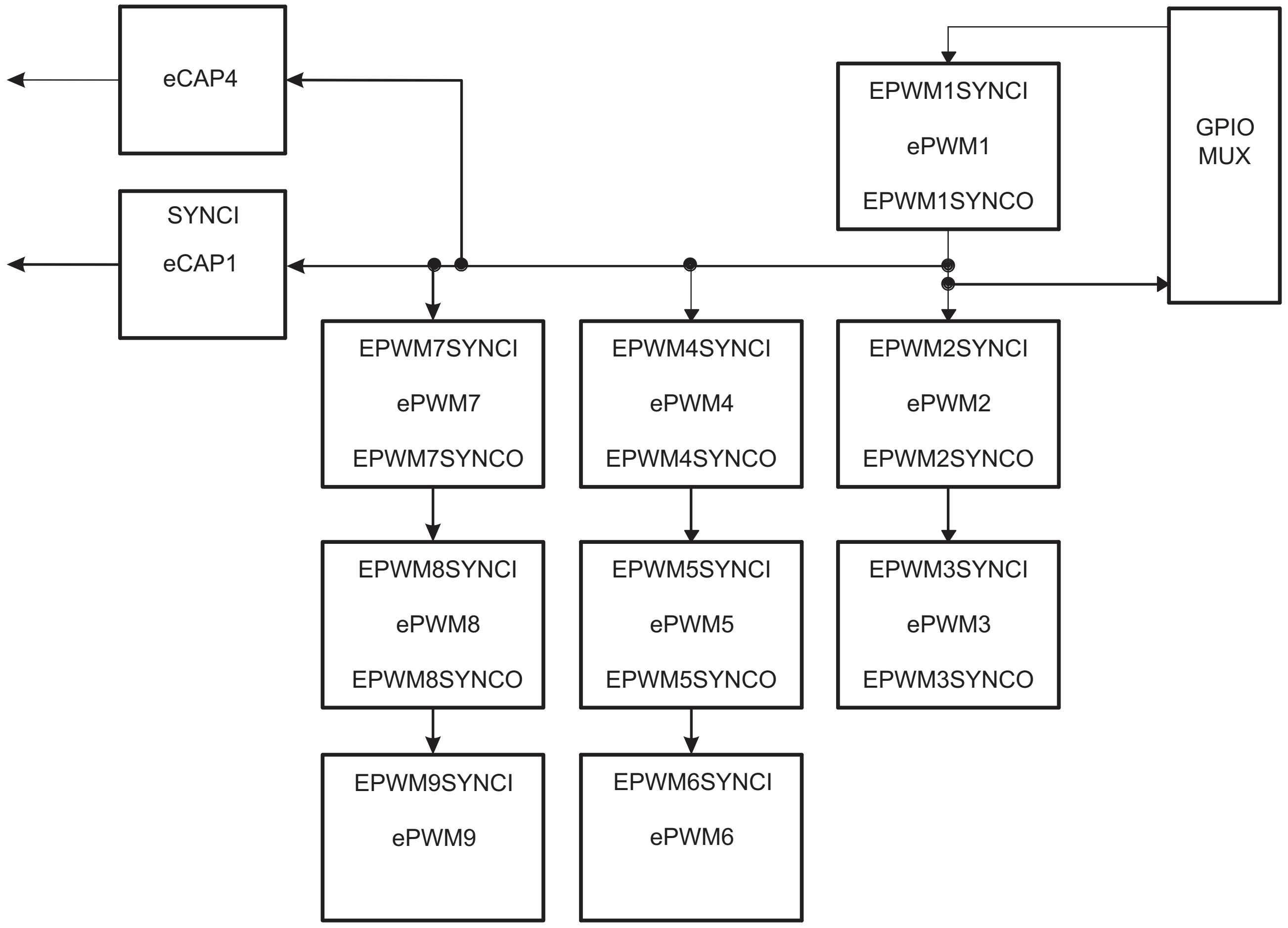
Scheme 2 shown in Figure 3-8 is used by the 2804x devices when the ePWM pinout is configured for A-channel only mode (GPAMCFG[EPWMMODE] = 3). If the 2804x ePWM pinout is configured for 280x compatible mode (GPAMCFG[EPWMMODE] = 0), then Scheme 1 is used.

Figure 3-8. Time-Base Counter Synchronization Scheme 2



Scheme 3, shown in Figure 3-9, is used by all other devices.

Figure 3-9. Time-Base Counter Synchronization Scheme 3



NOTE: All modules shown in the synchronization schemes may not be available on all devices. Please refer to the device specific data manual to determine which modules are available on a particular device.

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 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 PHSDIR bit is ignored in count-up or count-down modes. See Figure 3-10 through Figure 3-13 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 the Application to Power Topologies 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 Reference Guide*.
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 EPWMxSYNCI signal.

Figure 3-10. Time-Base Up-Count Mode Waveforms

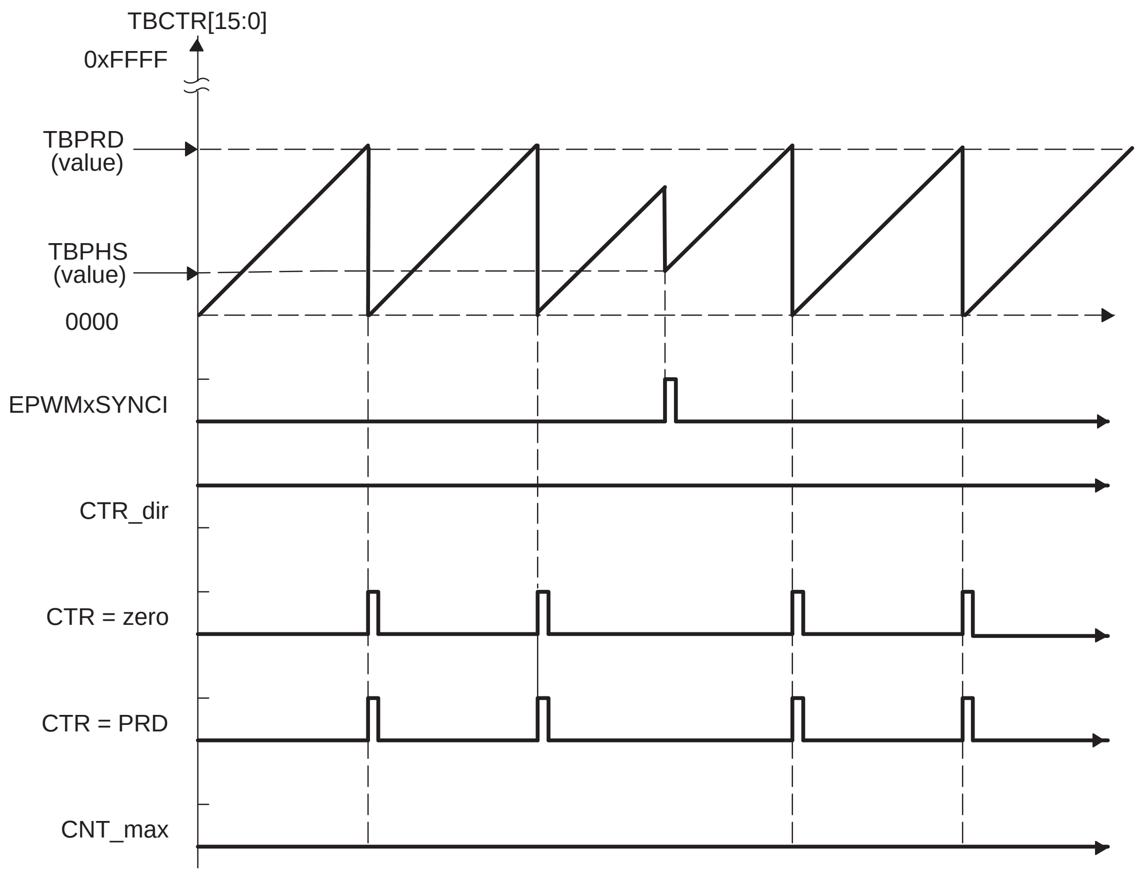


Figure 3-11. Time-Base Down-Count Mode Waveforms

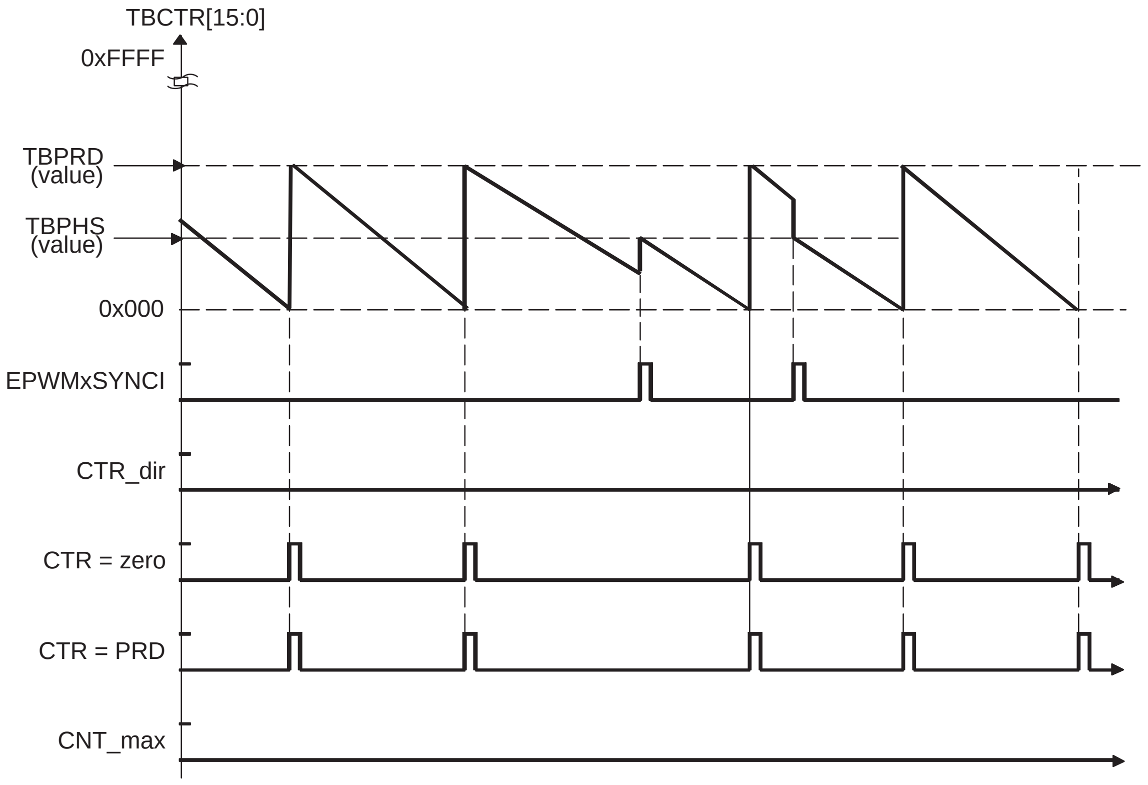


Figure 3-12. Time-Base Up-Down-Count Waveforms, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

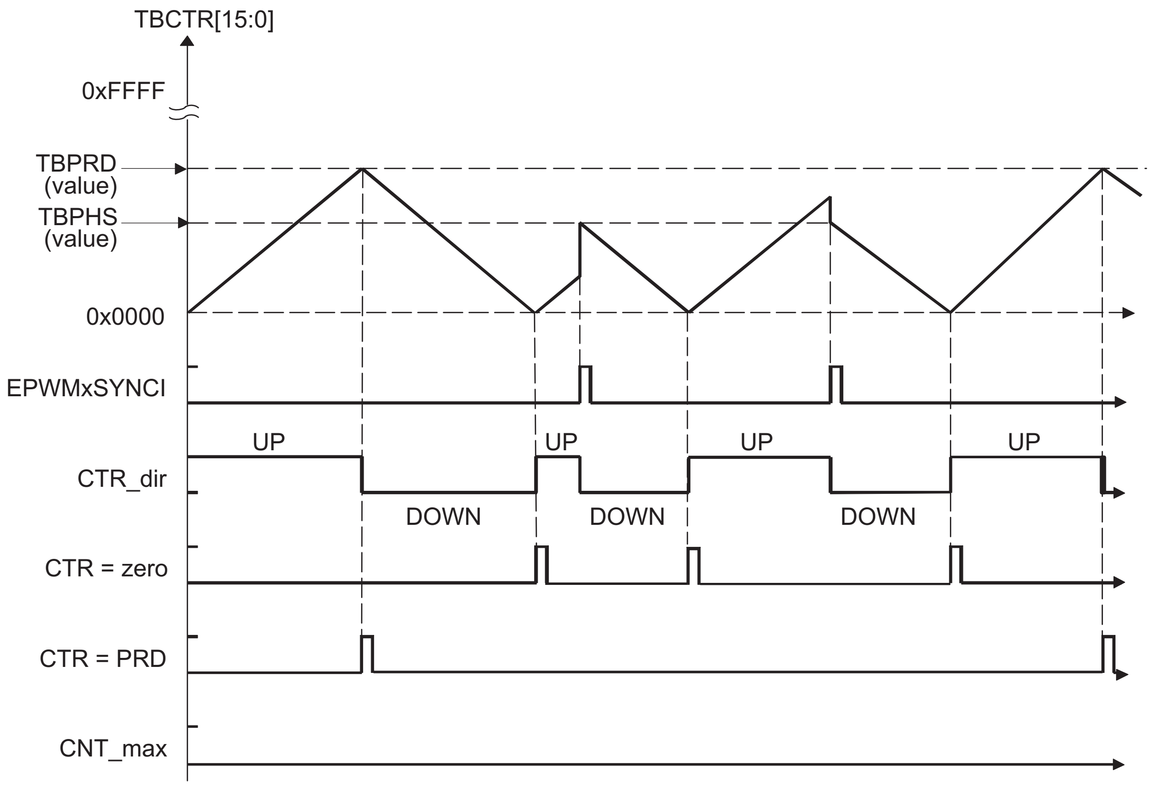
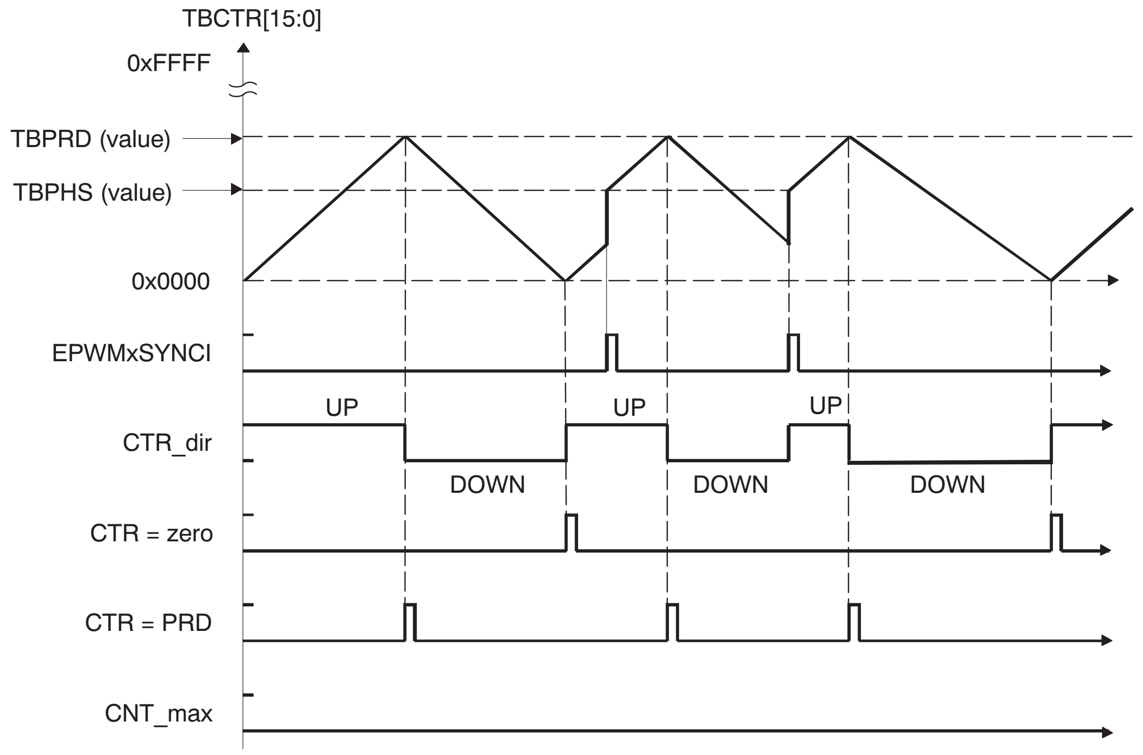


Figure 3-13. Time-Base Up-Down Count Waveforms, TBCTL[PHSDIR = 1] Count Up On Synchronization Event



3.2.3 Counter-Compare (CC) Submodule

Figure 3-14 illustrates the counter-compare submodule within the ePWM.

Figure 3-14. Counter-Compare Submodule

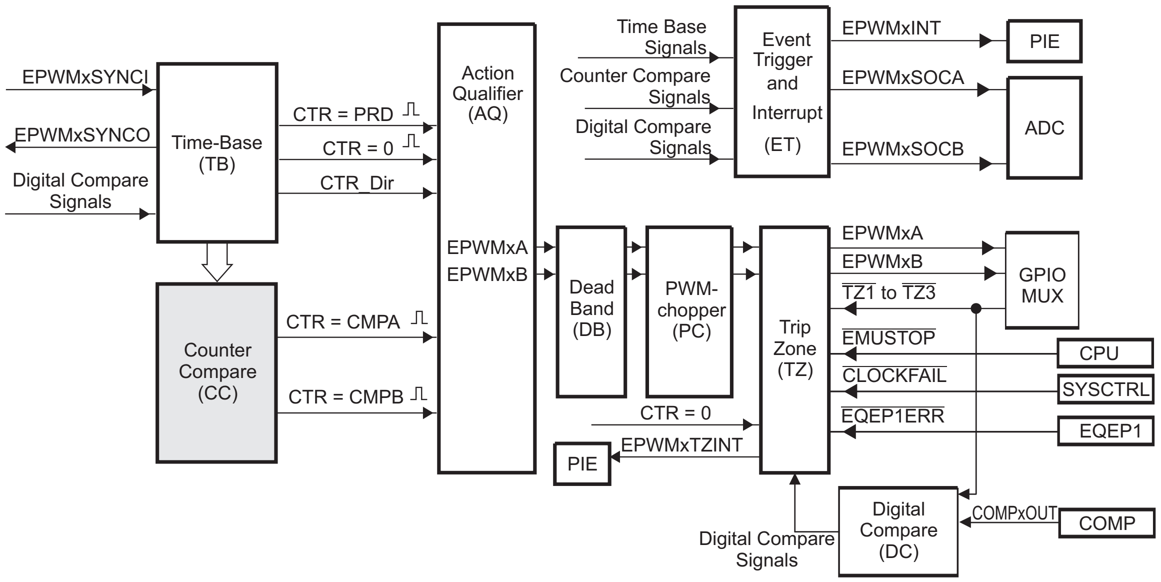


Figure 3-15 shows the basic structure of the counter-compare submodule.

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

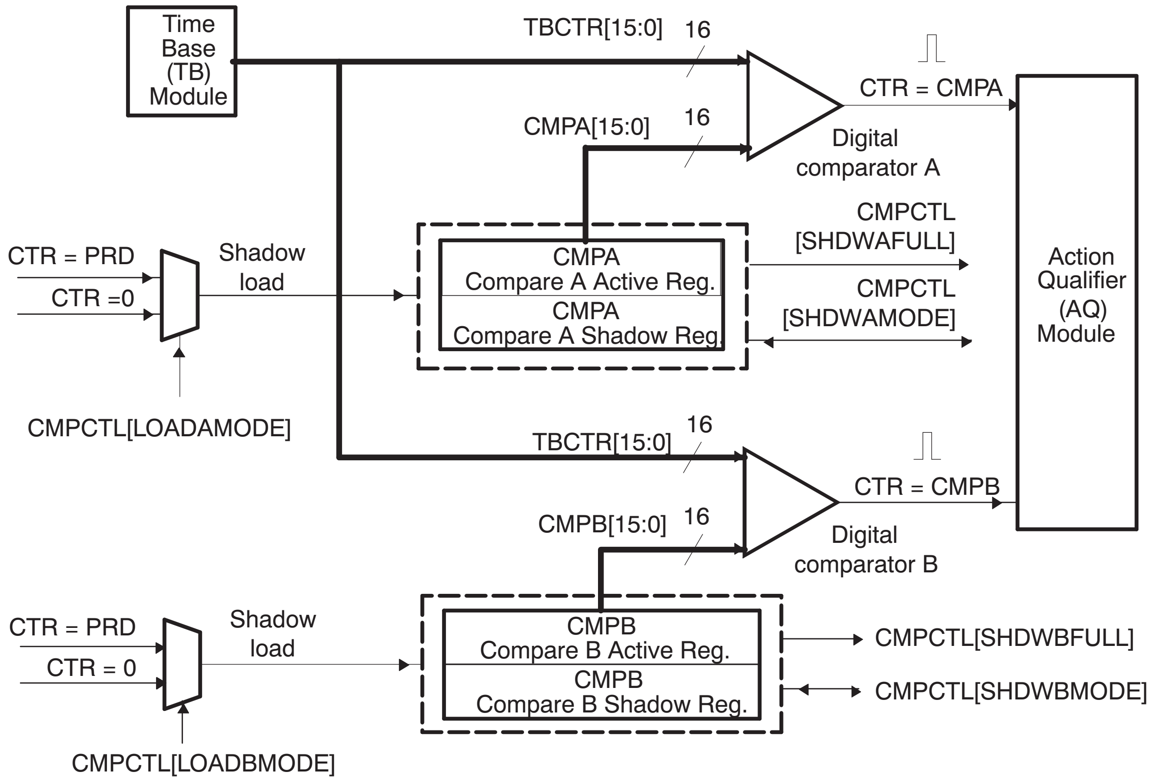
The counter-compare submodule operation is controlled and monitored by the registers shown in Table 3-5:

Table 3-5. Counter-Compare Submodule Registers

Register Name	Address Offset	Shadowed	Description
CMPCTL	0x0007	No	Counter-Compare Control Register.
CMPAHR	0x0008	Yes	HRPWM Counter-Compare A Extension Register [(1)]
CMPA	0x0009	Yes	Counter-Compare A Register
CMPB	0x000A	Yes	Counter-Compare B Register
CMPAHRM	0x002C	Writes	HRPWM counter-compare A Extension Mirror Register[(1)]
CMPAM	0x002D	Writes	Counter-compare A mirror Register

(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 described in the device-specific High-Resolution Pulse Width Modulator (HRPWM) section of this manual. Refer to the device specific data manual to determine which ePWM instances include this feature.

Figure 3-15. Detailed View of the Counter-Compare Submodule



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 described below:

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 (i.e., TBCTL[SHADWAMODE] = 1 or TBCTL[SHADWBMODE] = 1), then a read from or a write to the register will go directly to the active register.

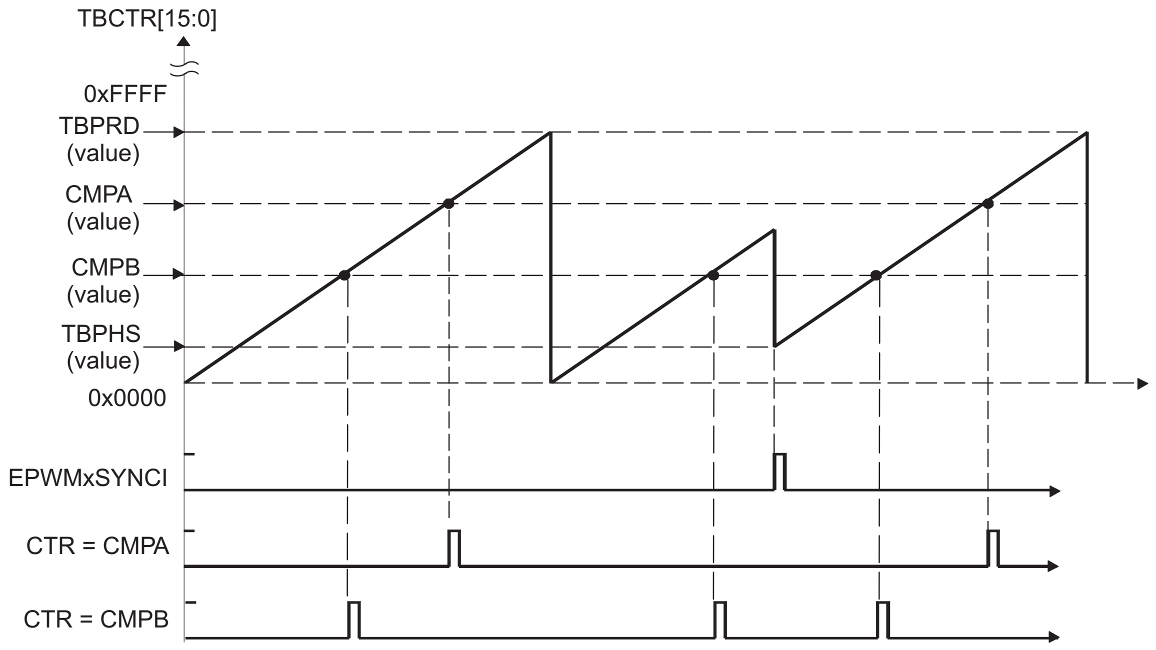
3.2.3.4 Count Mode Timing Waveforms

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 3-16 through Figure 3-19 show when events are generated and how the EPWMxSYNCI signal interacts.

Figure 3-16. Counter-Compare Event Waveforms in Up-Count Mode



NOTE: An EPWMxSYNCI 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 3-17. Counter-Compare Events in Down-Count Mode

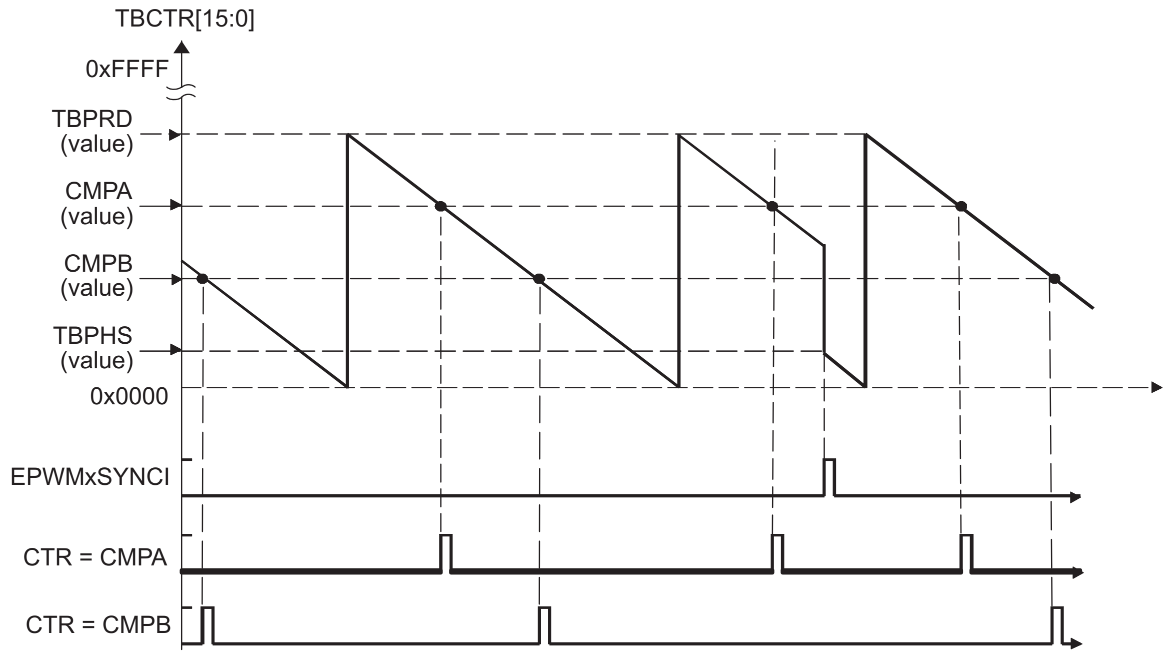


Figure 3-18. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

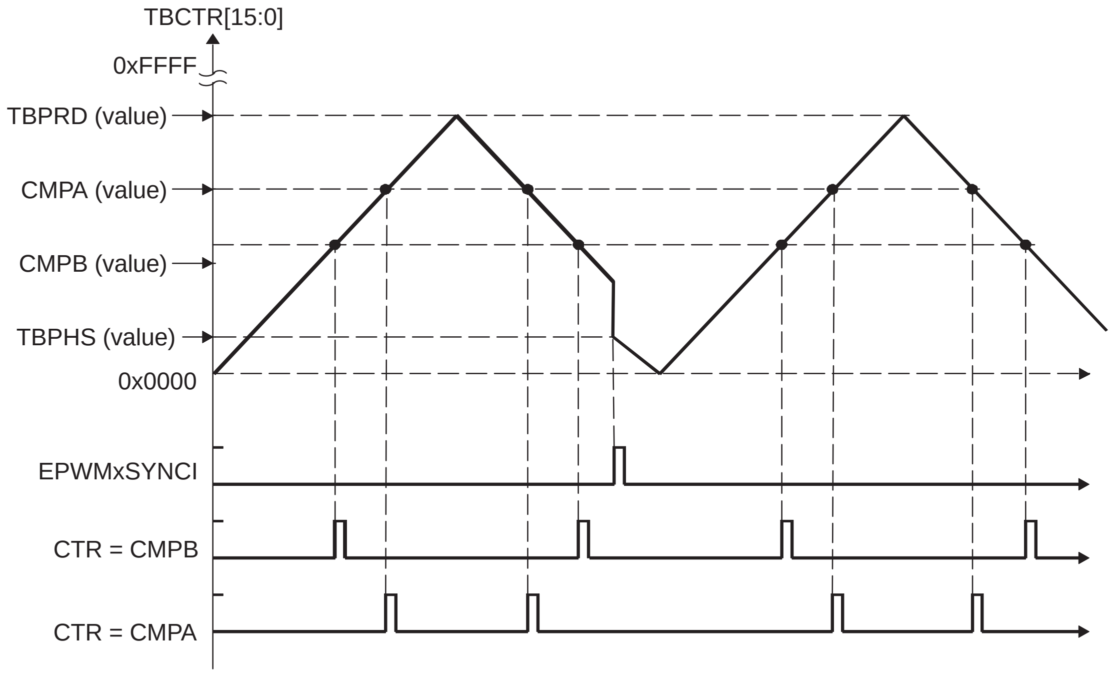
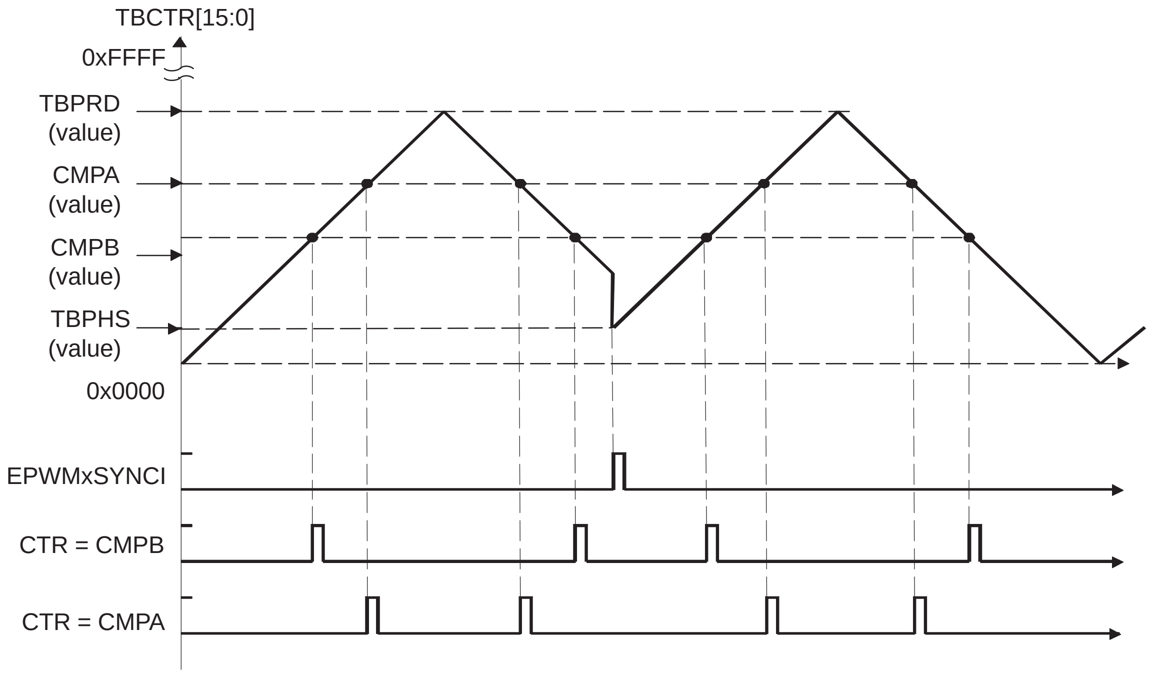


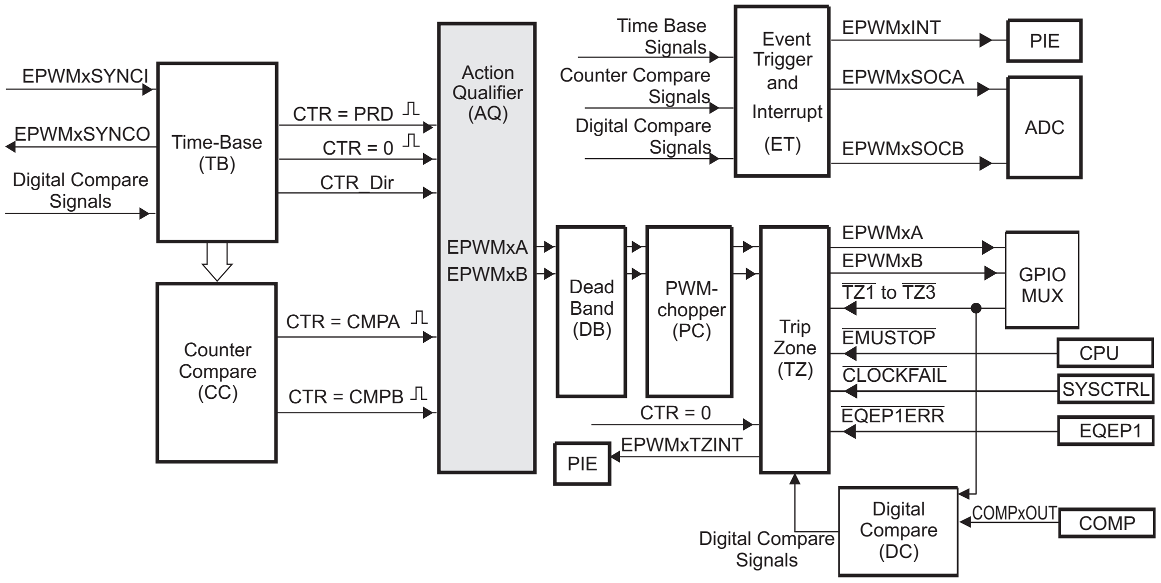
Figure 3-19. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 1] Count Up On Synchronization Event



3.2.4 Action-Qualifier (AQ) Submodule

Figure 3-20 shows the action-qualifier (AQ) submodule (see shaded block) in the ePWM system.

Figure 3-20. Action-Qualifier Submodule



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.

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

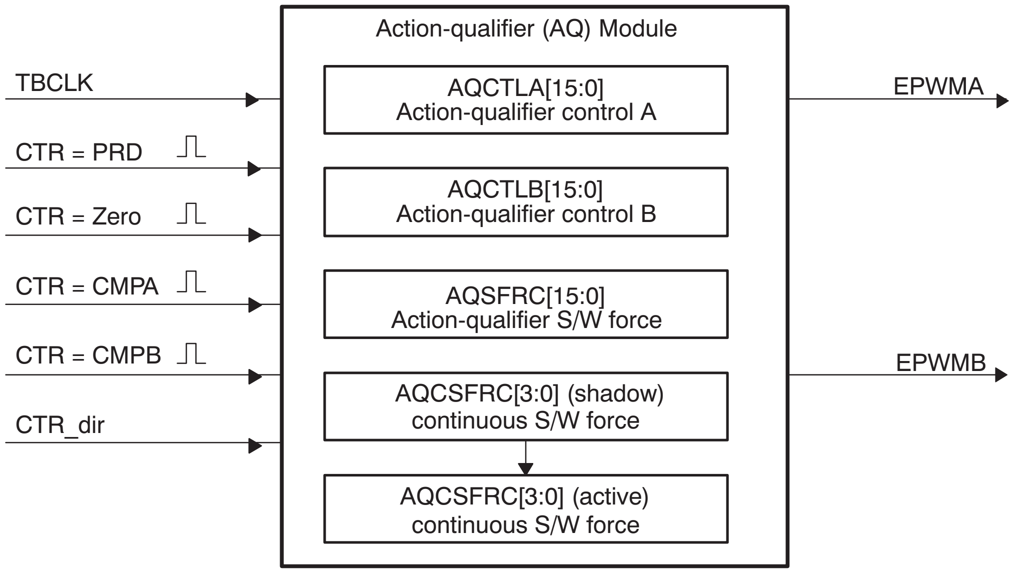
The action-qualifier submodule operation is controlled and monitored via the registers in Table 3-7.

Table 3-7. Action-Qualifier Submodule Registers

Register Name	Address offset	Shadowed	Description
AQCTLA	0x000B	No	Action-Qualifier Control Register For Output A (EPWMxA)
AQCTLB	0x000C	No	Action-Qualifier Control Register For Output B (EPWMxB)
AQSFRC	0x000D	No	Action-Qualifier Software Force Register
AQCSFRC	0x000E	Yes	Action-Qualifier Continuous Software Force

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 3-21. Action-Qualifier Submodule Inputs and Outputs


Action-qualifier (AQ) Module
TBCLK
CTR = PRD
CTR = Zero
CTR = CMPA
CTR = CMPB
CTR_dir
AQCTLA[15:0] Action-qualifier control A
AQCTLB[15:0] Action-qualifier control B
AQSFRC[15:0] Action-qualifier S/W force
AQCSFRC[3:0] (shadow) continuous S/W force
AQCSFRC[3:0] (active) continuous S/W force
EPWMA
EPWMB

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.

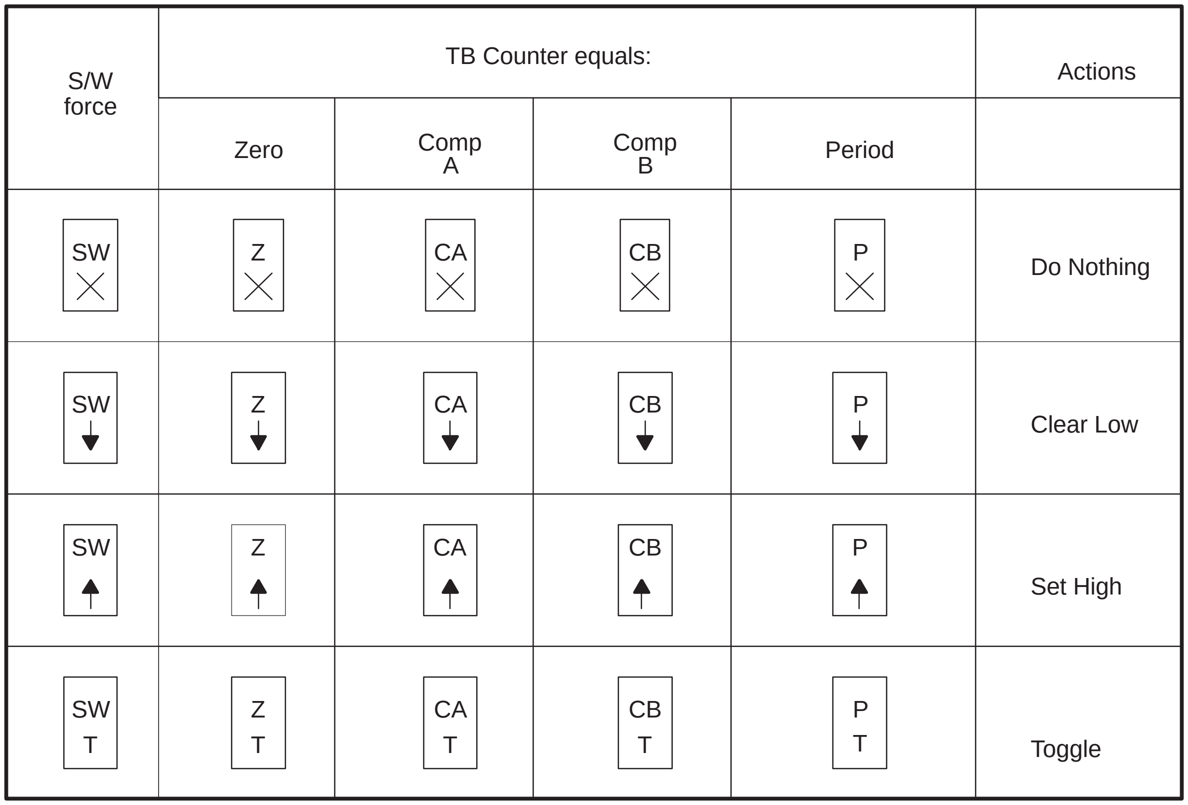
The possible actions imposed on outputs EPWMxA and EPWMxB are:

- **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 3-22. 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.

Figure 3-22. Possible Action-Qualifier Actions for EPWMxA and EPWMxB Outputs



S/W force	TB Counter equals:				Actions
	Zero	Comp A	Comp B	Period	
SW ✕	Z ✕	CA ✕	CB ✕	P ✕	Do Nothing
SW ↓	Z ↓	CA ↓	CB ↓	P ↓	Clear Low
SW ↑	Z ↑	CA ↑	CB ↑	P ↑	Set High
SW T	Z T	CA T	CB T	P T	Toggle

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 CAD/CBD	Compare on Down-Count Event CAD/CBD
Up-Count Mode	If CMPA/CMPB ≤ 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 CAD/CBD	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 < TBPRD and the counter is decrementing, 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, 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:

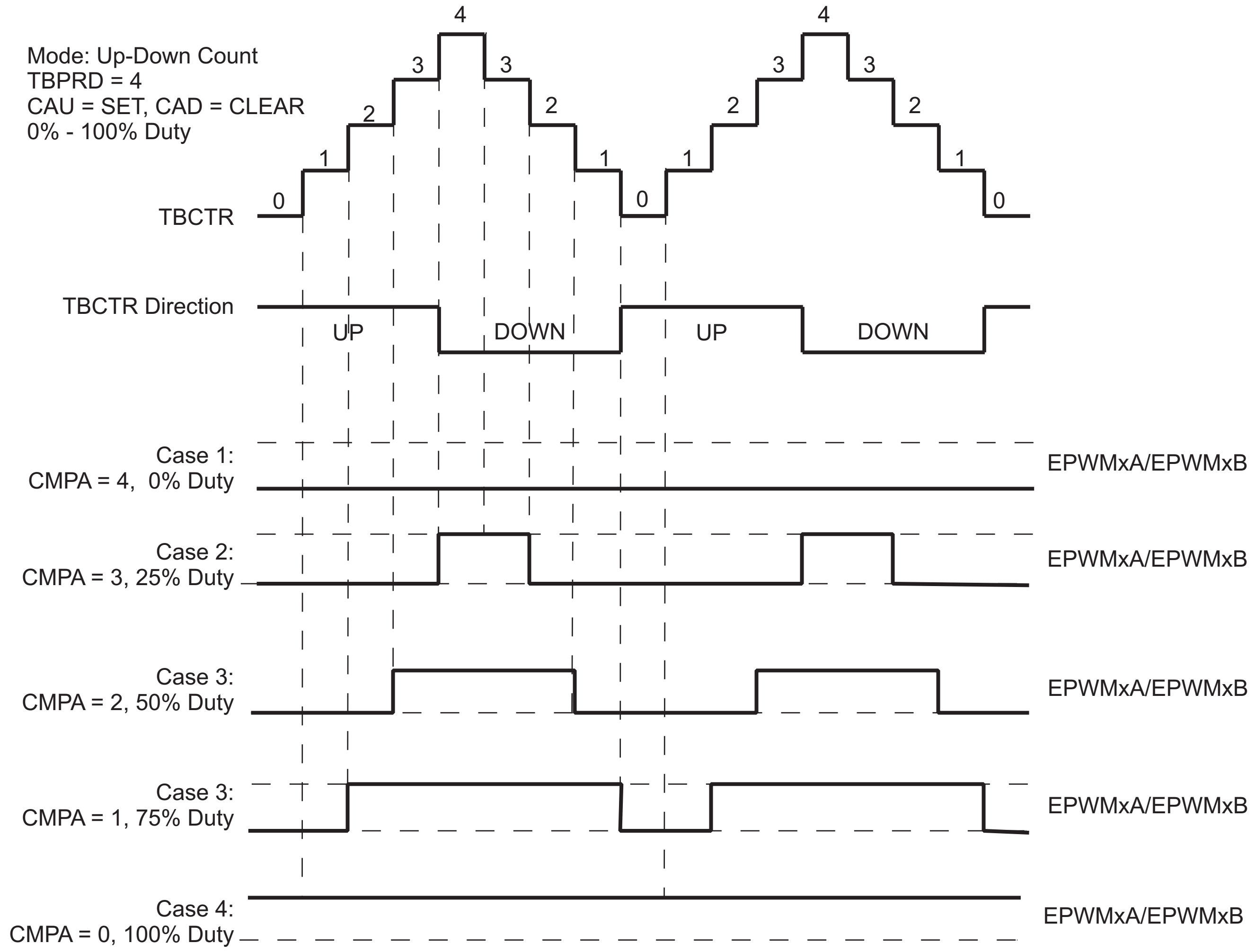
- To achieve 0-100% asymmetric PWM use the following configuration: Load CMPA/CMPB on TBPRD. Use the Zero action to set the PWM and a compare-up action to clear the PWM. Modulate the compare value from 0 to TBPRD+1 to achieve 0-100% PWM duty.

See the *Using Enhanced Pulse Width Modulator (ePWM) Module for 0-100% Duty Cycle Control* Application Report (literature number SPRAAI1)

Figure 3-23 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 = 0, the PWM signal is low for the entire period giving the 0% duty waveform. When CMPA = TBPRD, 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.

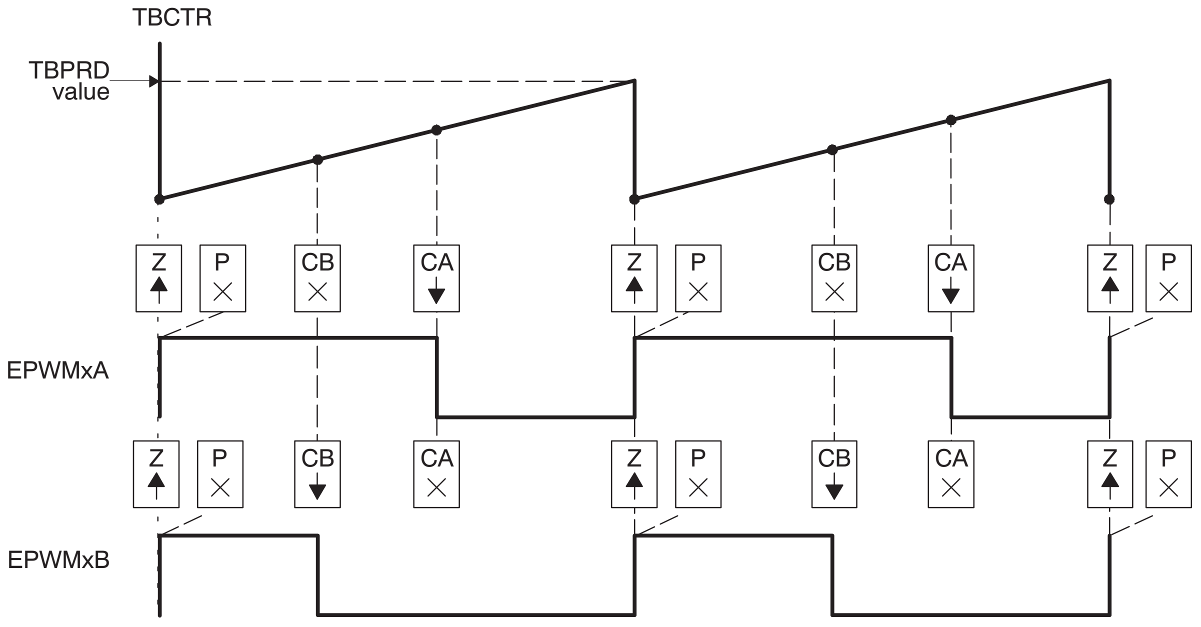
Figure 3-23. Up-Down-Count Mode Symmetrical Waveform



The PWM waveforms in Figure 3-24 through Figure 3-29 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

Figure 3-24. Up, Single Edge Asymmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB—Active High



A PWM period = (TBPRD + 1) × T_{TBCLK}

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.

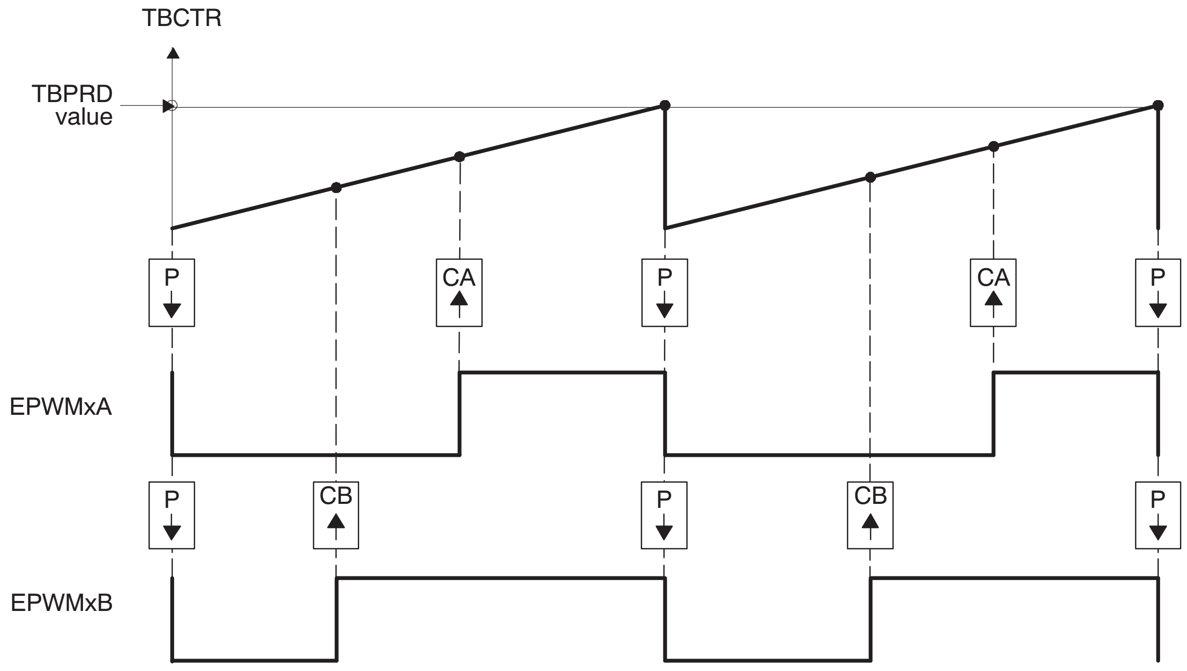
E Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

Example 3-1 contains a code sample showing initialization and run time for the waveforms in Figure 3-24.

Example 3-1. Code Sample for Figure 3-24

```
// 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.PRDLD = 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
```

Figure 3-25. Up, Single Edge Asymmetric Waveform With Independent Modulation on EPWMxA and EPWMxB—Active Low



A PWM period = (TBPRD + 1) × T_{TBCLK}

B Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).

C Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).

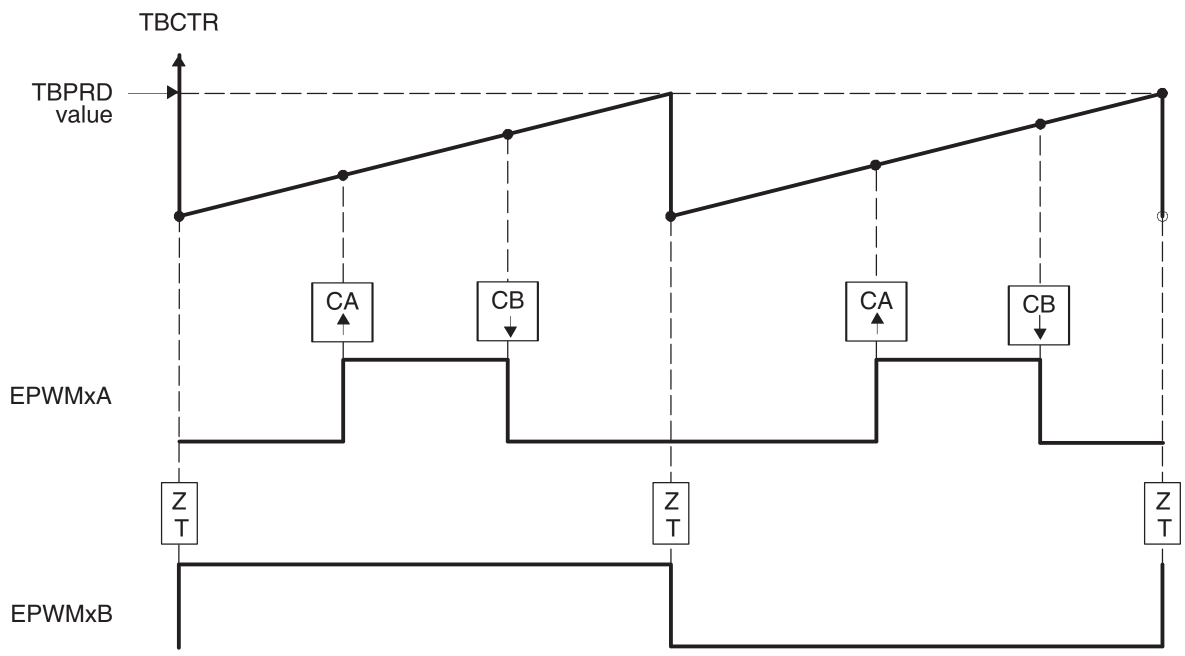
D Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

Example 3-2 contains a code sample showing initialization and run time for the waveforms in Figure 3-25.

Example 3-2. Code Sample for Figure 3-25

```
// 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.PRDLD = 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 TBCTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on TBCTR = Zero
EPwm1Regs.AQCTLA.bit.PRD = AQ_CLEAR;
EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
EPwm1Regs.AQCTLB.bit.PRD = 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
```

Figure 3-26. Up-Count, Pulse Placement Asymmetric Waveform With Independent Modulation on EPWMxA



A PWM frequency = 1/((TBPRD + 1) × T_{TBCLK})

B Pulse can be placed anywhere within the PWM cycle (0000 - TBPRD)

C High time duty proportional to (CMPB - CMPA)

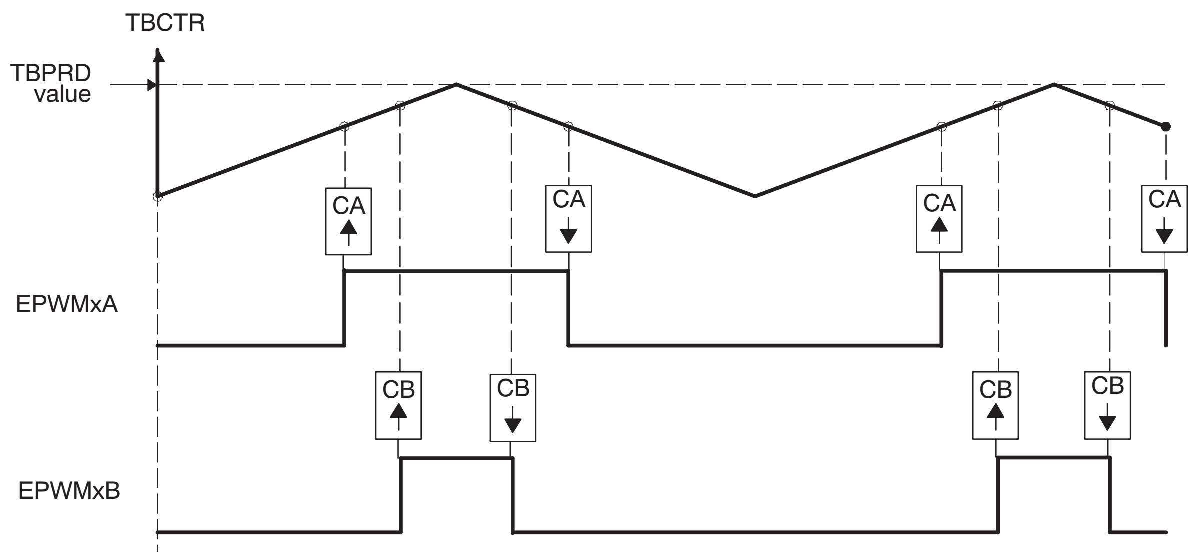
D EPWMxB can be used to generate a 50% duty square wave with frequency = � × ((TBPRD + 1) × TBCLK)

Example 3-3 contains a code sample showing initialization and run time for the waveforms Figure 3-26. Use the code in Example 3-5 to define the headers.

Example 3-3. Code Sample for Figure 3-26

```
// 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.PRDLD = 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 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;
```

Figure 3-27. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Active Low



A PWM period = 2 x TBPRD × T_{TBCLK}

B Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).

C Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).

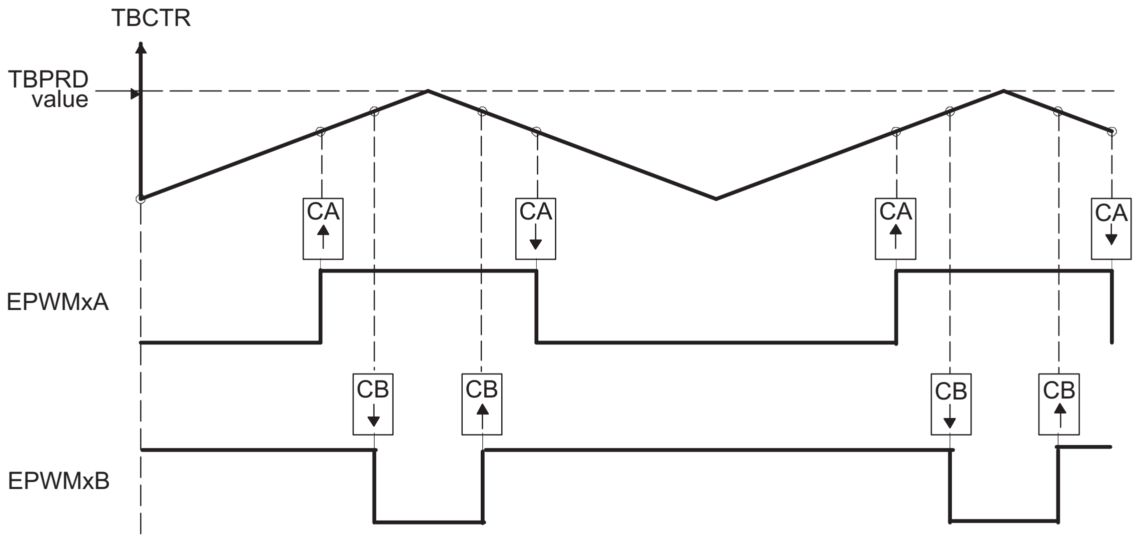
D Outputs EPWMxA and EPWMxB can drive independent power switches

Example 3-4 contains a code sample showing initialization and run time for the waveforms in Figure 3-27. Use the code in Example 3-5 to define the headers.

Example 3-4. Code Sample for Figure 3-27

```
// 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.PRDLD = 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
```

Figure 3-28. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Complementary



A PWM period = 2 × TBPRD × T_{TBCLK}

B Duty modulation for EPWMxA is set by CMPA, and is active low, i.e., low time duty proportional to CMPA

C Duty modulation for EPWMxB is set by CMPB and is active high, i.e., high time duty proportional to CMPB

D Outputs EPWMx can drive upper/lower (complementary) power switches

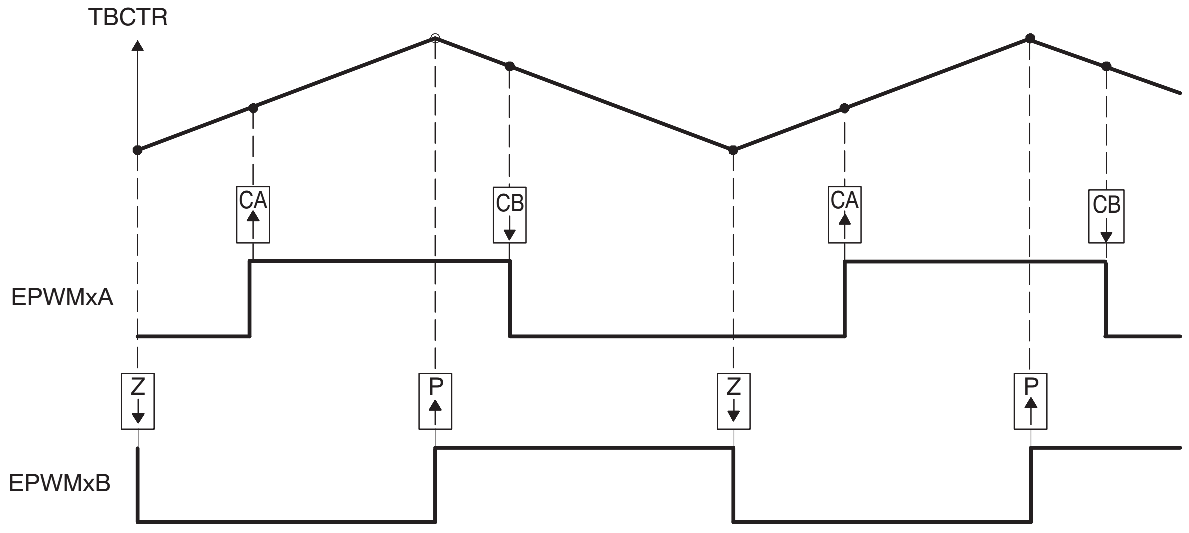
E Dead-band = CMPB - CMPA (fully programmable edge placement by software). Note the dead-band module is also available if the more classical edge delay method is required.

Example 3-5 contains a code sample showing initialization and run time for the waveforms in Figure 3-28. Use the code in Example 3-5 to define the headers.

Example 3-5. Code Sample for Figure 3-28

```
// 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.PRDLD = 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_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
```

Figure 3-29. Up-Down-Count, Dual Edge Asymmetric Waveform, With Independent Modulation on EPWMxA—Active Low



A PWM period = 2 × TBPRD × 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 (CMPA + CMPB).

E To change this example to active high, CMPA and CMPB actions need to be inverted (i.e., Set ! Clear and Clear Set).

F Duty modulation for EPWMxB is fixed at 50% (utilizes spare action resources for EPWMxB)

Example 3-6 contains a code sample showing initialization and run time for the waveforms in Figure 3-29. Use the code in Example 3-5 to define the headers.

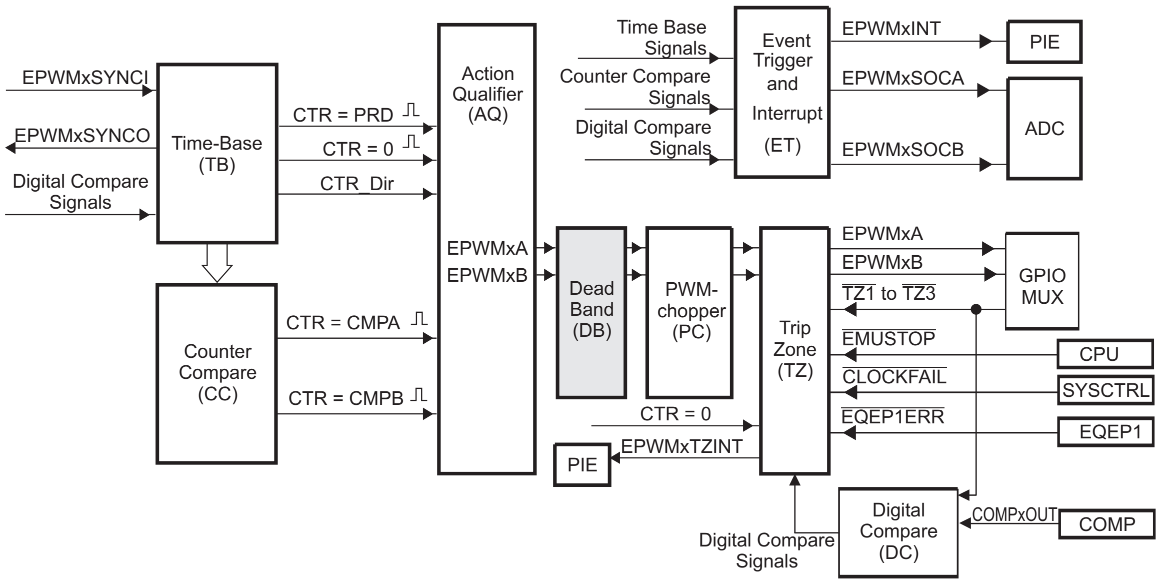
Example 3-6. Code Sample for Figure 3-29

```
// 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.PRDLD = 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 3-30 illustrates the dead-band submodule within the ePWM module.

Figure 3-30. 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 Name	Address offset	Shadowed	Description
DBCTL	0x000F	No	Dead-Band Control Register
DBRED	0x0010	No	Dead-Band Rising Edge Delay Count Register
DBFED	0x0011	No	Dead-Band Falling Edge Delay Count Register

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 3-31.

- **Input Source Selection:**

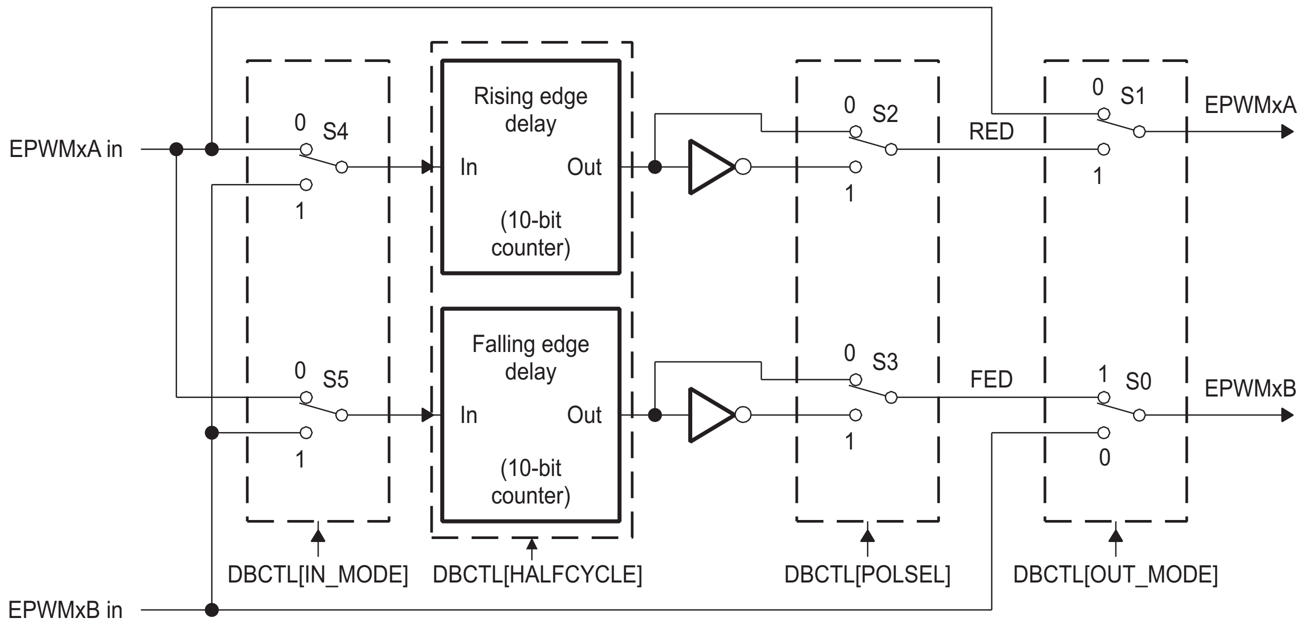
  The input signals to the dead-band module are the EPWMxA and EPWMxB output signals from the action-qualifier. In this section they will be referred to as EPWMxA In and EPWMxB In. Using the DBCTL[IN_MODE) control bits, the signal source for each delay, falling-edge or rising-edge, can be selected:
  - 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× 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 3-31. 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 3-32. Note that to generate equivalent waveforms to Figure 3-32, 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**

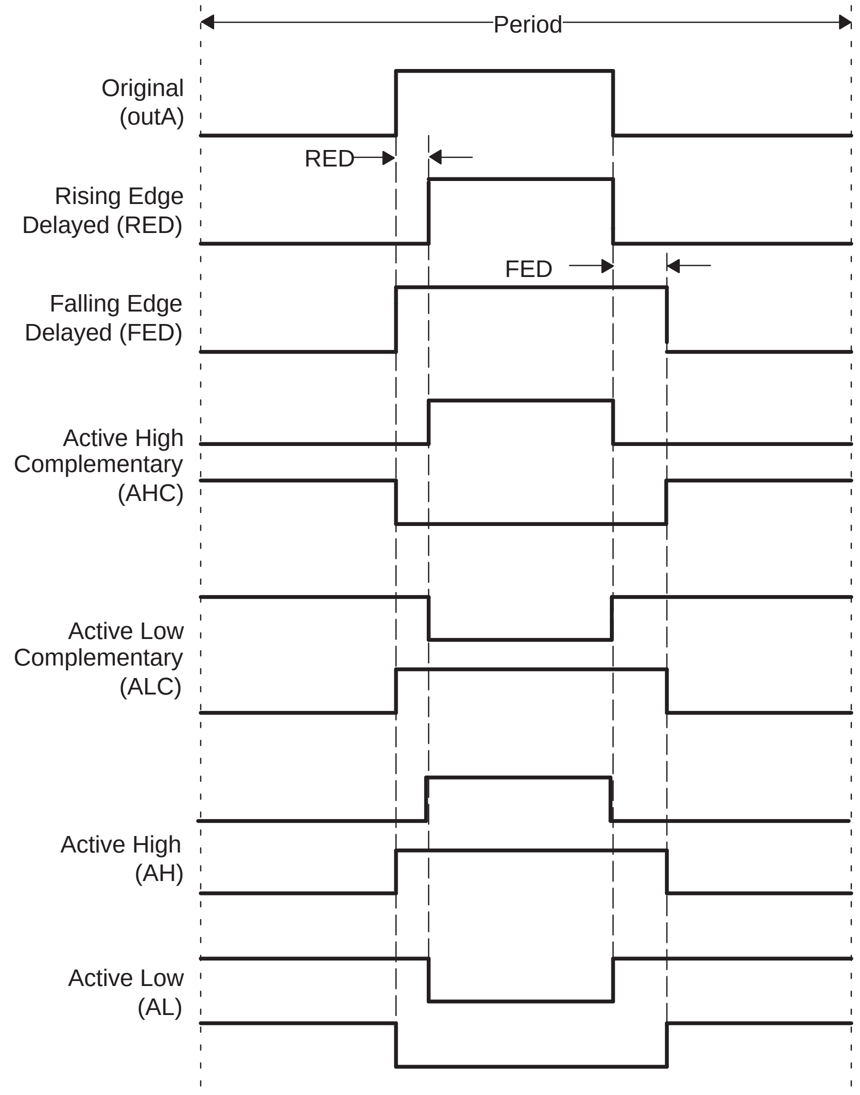
  Finally the last two entries in Table 3-14 show combinations where either the falling-edge-delay (FED) or rising-edge-delay (RED) blocks are bypassed.

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) EPWMxB Out = EPWMxA In with Falling Edge Delay	0 or 1	0 or 1	0	1
7	EPWMxA Out = EPWMxA In with Rising Edge Delay EPWMxB Out = EPWMxB In with No Delay	0 or 1	0 or 1	1	0

Figure 3-32 shows waveforms for typical cases where 0% < duty < 100%.

Figure 3-32. Dead-Band Waveforms for Typical Cases (0% < Duty < 100%)


Period
Original (outA)
RED
Rising Edge Delayed (RED)
FED
Falling Edge Delayed (FED)
Active High Complementary (AHC)
Active Low Complementary (ALC)
Active High (AH)
Active Low (AL)

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 formula to calculate falling-edge-delay and rising-edge-delay are:

$FED = DBFED \times T_{TBCLK}$

$RED = DBRED \times T_{TBCLK}$

Where T_{TBCLK} is the period of TBCLK, the prescaled version of SYSCLKOUT.

For convenience, delay values for various TBCLK options are shown in Table 3-15.

Table 3-15. Dead-Band Delay Values in μS as a Function of DBFED and DBRED

Dead-Band Value	Dead-Band Delay in μS		
DBFED, DBRED	TBCLK = SYSCLKOUT/1	TBCLK = SYSCLKOUT /2	TBCLK = SYSCLKOUT/4
1	0.01 μS	0.03 μS	0.05 μS
5	0.06 μS	0.13μS	0.25 μS
10	0.13 μS	0.25 μS	0.50 μS
100	1.25 μS	2.50 μS	5.00 μS
200	2.50 μS	5.00 μS	10.00 μS
400	5.00 μS	10.00 μS	20.00 μS
500	6.25 μS	12.50 μS	25.00 μS
600	7.50 μS	15.00 μS	30.00 μS
700	8.75 μS	17.50 μS	35.00 μS
800	10.00 μS	20.00 μS	40.00 μS
900	11.25 μS	22.50 μS	45.00 μS
1000	12.50 μS	25.00 μS	50.00 μS

When half-cycle clocking is enabled, the formula to calculate the falling-edge-delay and rising-edge-delay becomes:

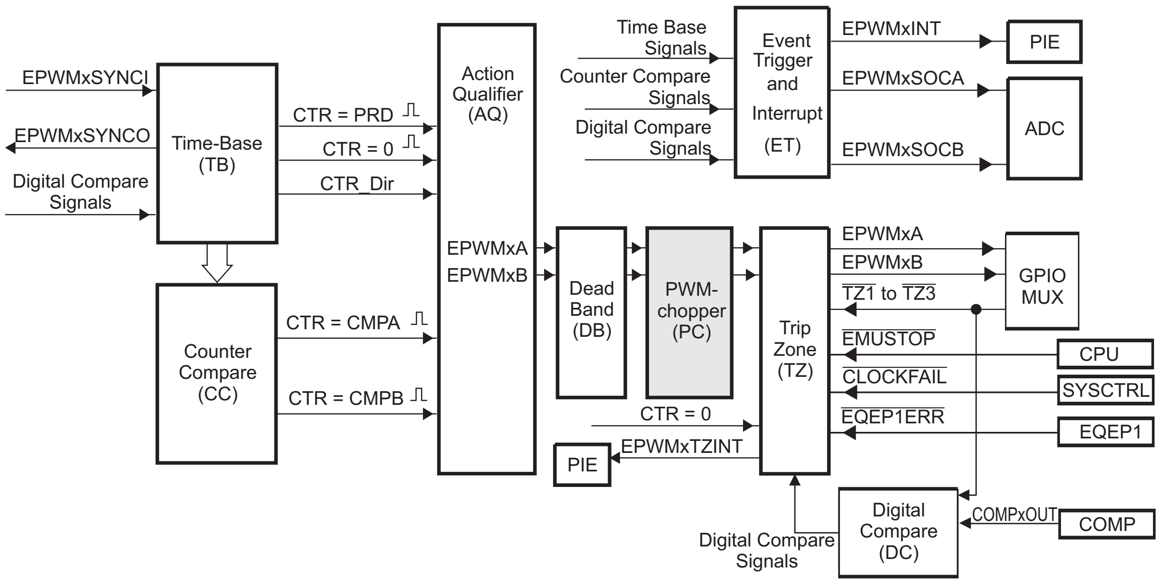
$FED = DBFED \times T_{TBCLK}/2$

$RED = DBRED \times T_{TBCLK}/2$

3.2.6 PWM-Chopper (PC) Submodule

Figure 3-33 illustrates the PWM-chopper (PC) submodule within the ePWM module.

Figure 3-33. PWM-Chopper Submodule



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.

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 3-16.

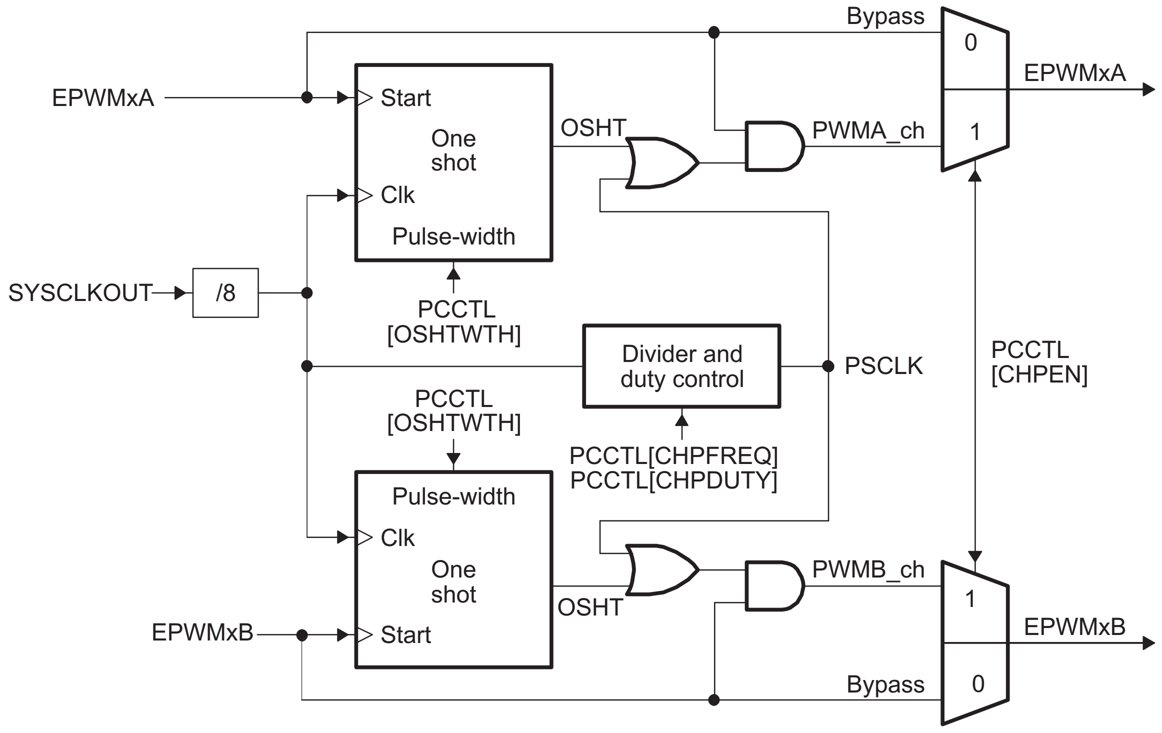
Table 3-16. PWM-Chopper Submodule Registers

mnemonic	Address offset	Shadowed	Description
PCCTL	0x001E	No	PWM-chopper Control Register

3.2.6.3 Operational Highlights for the PWM-Chopper Submodule

Figure 3-34 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.

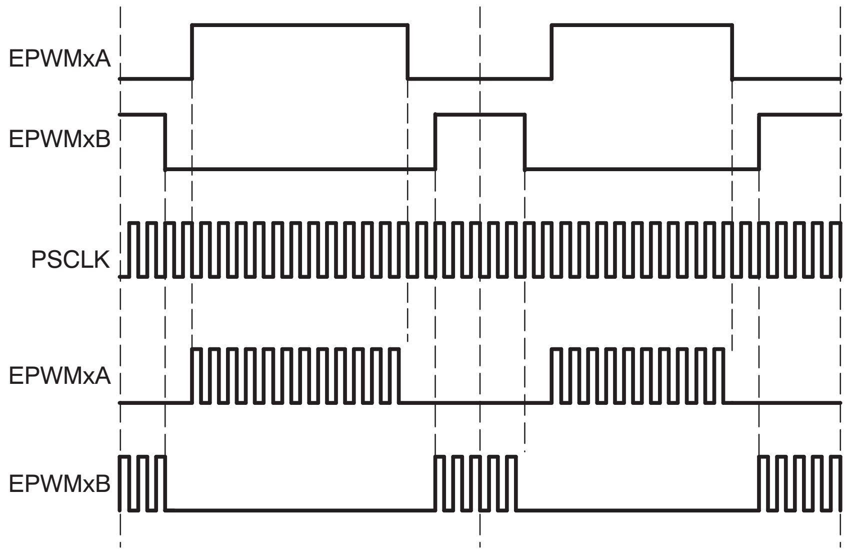
Figure 3-34. PWM-Chopper Submodule Operational Details



3.2.6.4 Waveforms

Figure 3-35 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 3-35. 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:

$T_{1stpulse} = T_{SYSCLKOUT} \times 8 \times OSHTWTH$

Where $T_{SYSCLKOUT}$ is the period of the system clock (SYSCLKOUT) and OSHTWTH is the four control bits (value from 1 to 16)

Figure 3-36 shows the first and subsequent sustaining pulses and Table 7.3 gives the possible pulse width values for a SYSCLKOUT = 80 MHz.

Figure 3-36. PWM-Chopper Submodule Waveforms Showing the First Pulse and Subsequent Sustaining Pulses

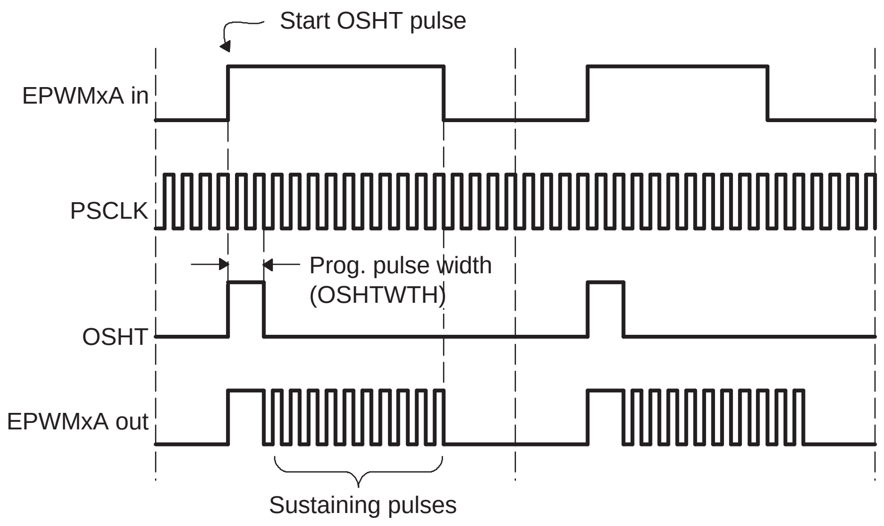


Table 3-17. Possible Pulse Width Values for SYSCLKOUT = 90 MHz

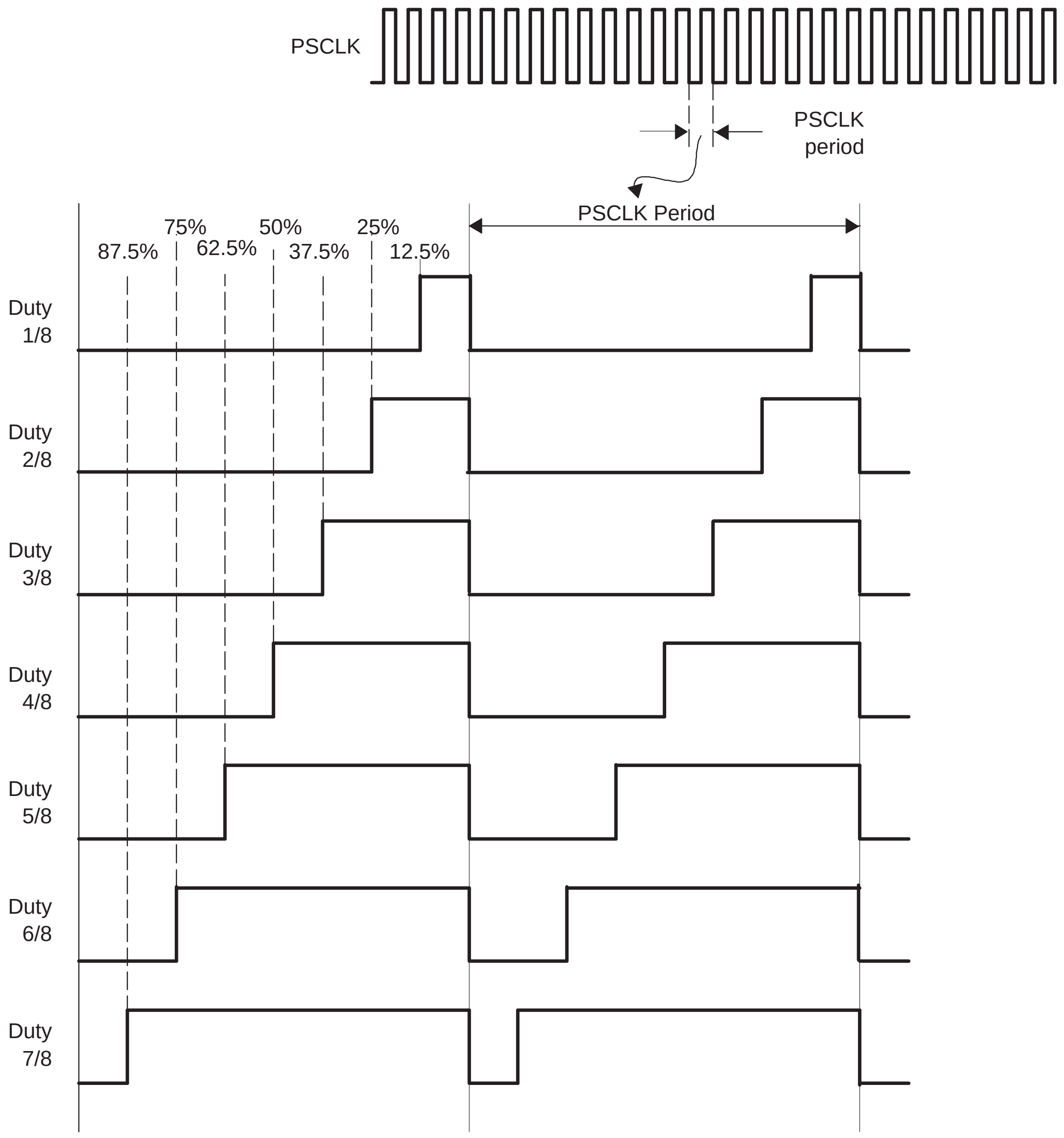
OSHTWTHz (hex)	Pulse Width (nS)
0	89
1	178
2	267
3	356
4	445
5	533
6	622
7	711
8	800
9	889
A	978
B	1067
C	1156
D	1245
E	1334
F	1422

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 3-37 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%.

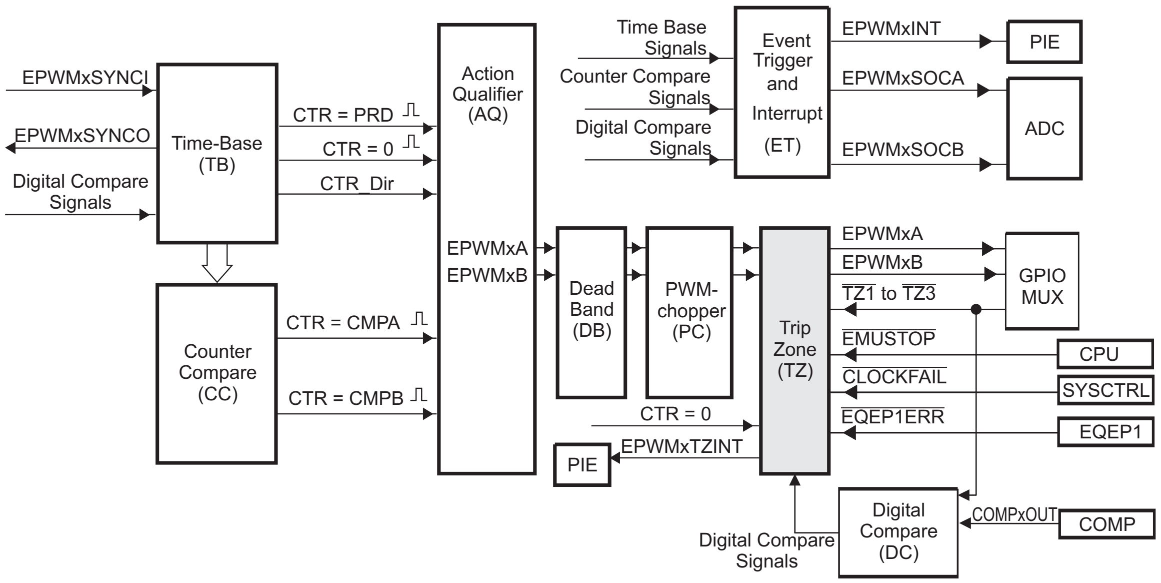
Figure 3-37. PWM-Chopper Submodule Waveforms Showing the Pulse Width (Duty Cycle) Control of Sustaining Pulses



3.2.7 Trip-Zone (TZ) Submodule

Figure 3-38 shows how the trip-zone (TZ) submodule fits within the ePWM module.

Figure 3-38. Trip-Zone Submodule



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.

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 3-18. Trip-Zone Submodule Registers

Register Name	Address offset	Shadowed	Description [(1)]
TZSEL	0x0012	No	Trip-Zone Select Register
TZDCSEL	0x0013	No	Trip-zone Digital Compare Select Register [(2)]
TZCTL	0x0014	No	Trip-Zone Control Register
TZEINT	0x0015	No	Trip-Zone Enable Interrupt Register
TZFLG	0x0016	No	Trip-Zone Flag Register
TZCLR	0x0017	No	Trip-Zone Clear Register
TZFRC	0x0018	No	Trip-Zone Force Register

(1) All trip-zone registers are EALLOW protected and can be modified only after executing the EALLOW instruction. For more information, see the device-specific version of the System Control and Interrupts Reference Guide listed in Section 1.

(2) This register is discussed in more detail in Section 3.2.9 Digital Compare submodule.

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 $\overline{\text{TZ1}}$ to $\overline{\text{TZ6}}$ (also collectively referred to as $\overline{\text{TZn}}$) 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*TBCLK low pulse width on $\overline{\text{TZn}}$ 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 $\overline{\text{TZn}}$ inputs . The GPIOs or peripherals must be appropriately configured. For more information, see the device-specific version of the *System Control and Interrupts Reference Guide*.

Each $\overline{\text{TZn}}$ 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, DCAVET2 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 3-19 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 3-19 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 DCTRIPSEL 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 3-19 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 3-19, can be taken on a trip event.

Table 3-19. Possible Actions On a Trip Event

TZCTL Register bit-field 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 $\overline{\text{TZ1}}$ pulls both EPWM1A, EPWM1B low and also forces EPWM2A and EPWM2B high.

- Configure the ePWM1 registers as follows:
  - TZSEL[OSHT1] = 1: enables $\overline{\text{TZ1}}$ 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 $\overline{\text{TZ1}}$ 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 $\overline{\text{TZ5}}$ pulls both EPWM1A, EPWM1B low.
A one-shot event on $\overline{\text{TZ1}}$ or $\overline{\text{TZ6}}$ puts EPWM2A into a high impedance state.

- Configure the ePWM1 registers as follows:
  - TZSEL[CBC5] = 1: enables $\overline{\text{TZ5}}$ 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 $\overline{\text{TZ1}}$ as a one-shot event source for ePWM2
  - TZSEL[OSHT6] = 1: enables $\overline{\text{TZ6}}$ 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 3-39 and Figure 3-40 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 3-39. Trip-Zone Submodule Mode Control Logic

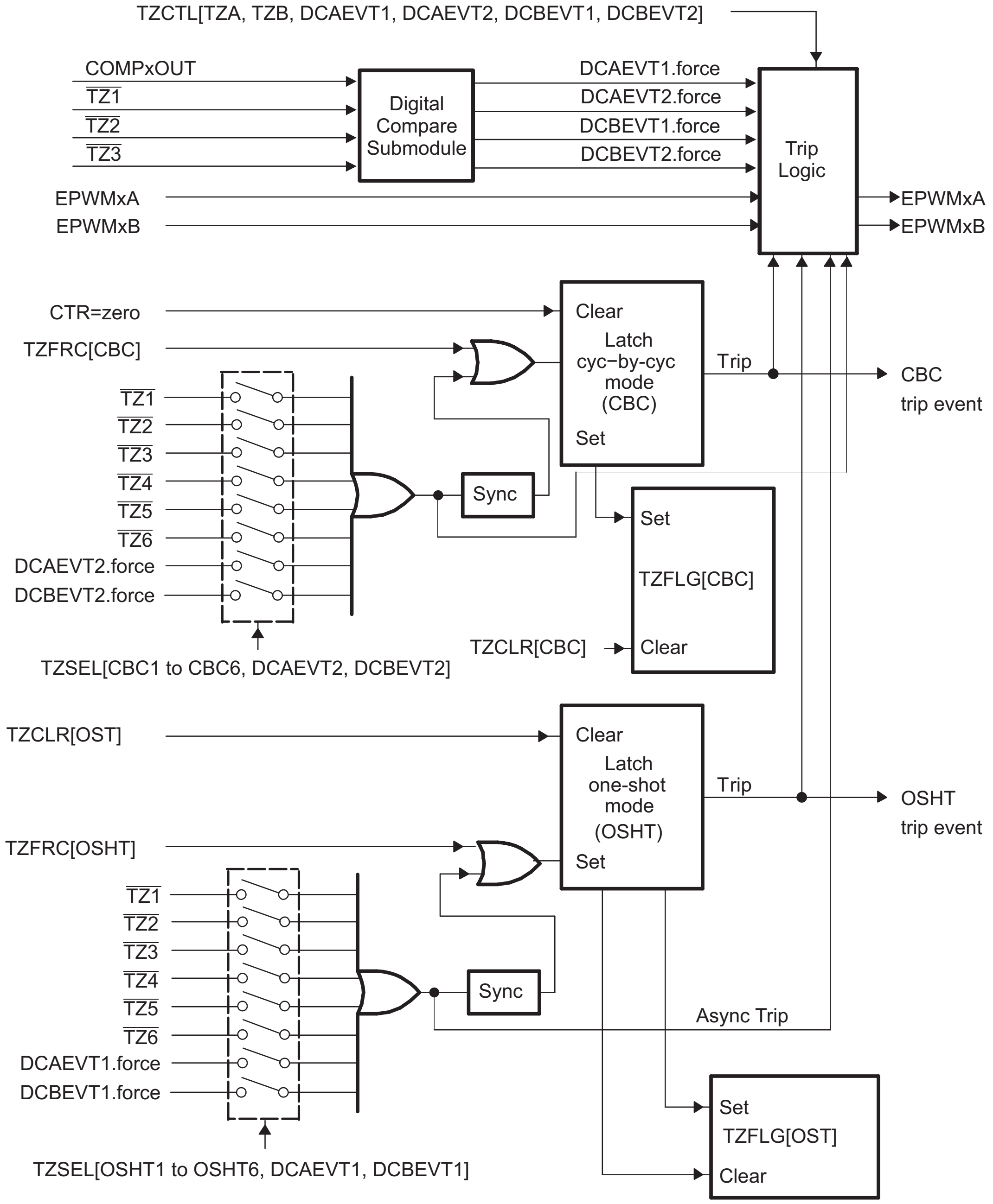
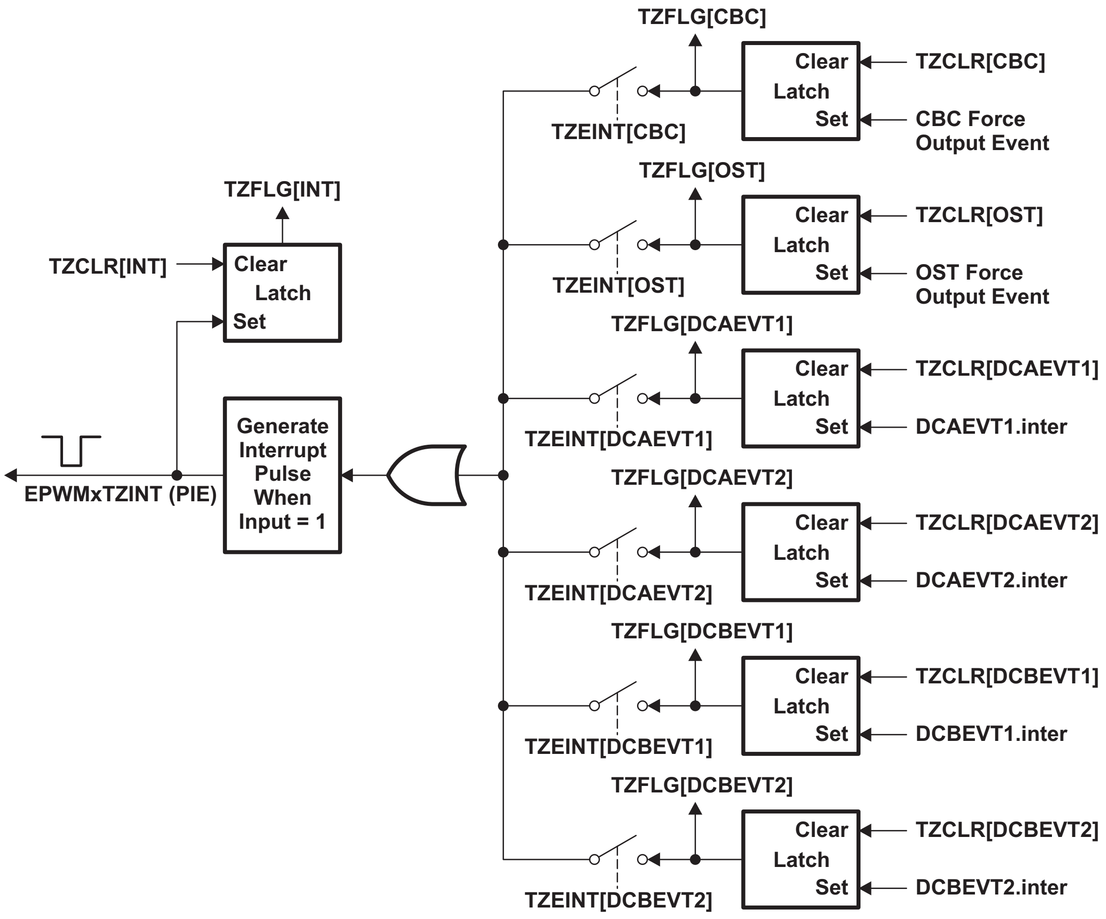


Figure 3-40. Trip-Zone Submodule Interrupt Logic



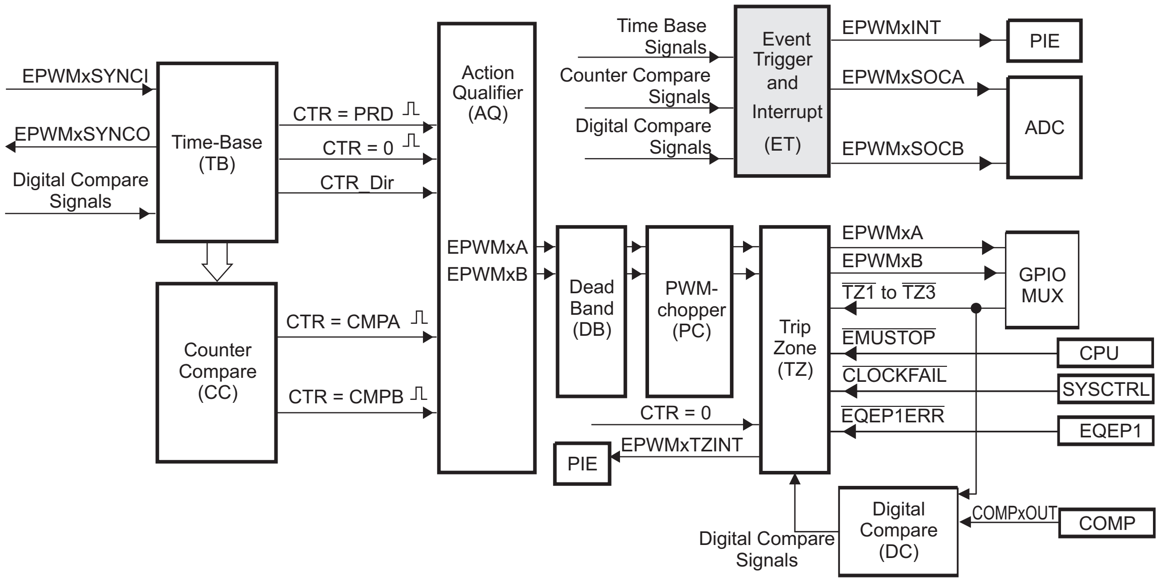
3.2.8 Event-Trigger (ET) Submodule

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 3-41 illustrates where the event-trigger submodule fits within the ePWM system.

Figure 3-41. Event-Trigger Submodule

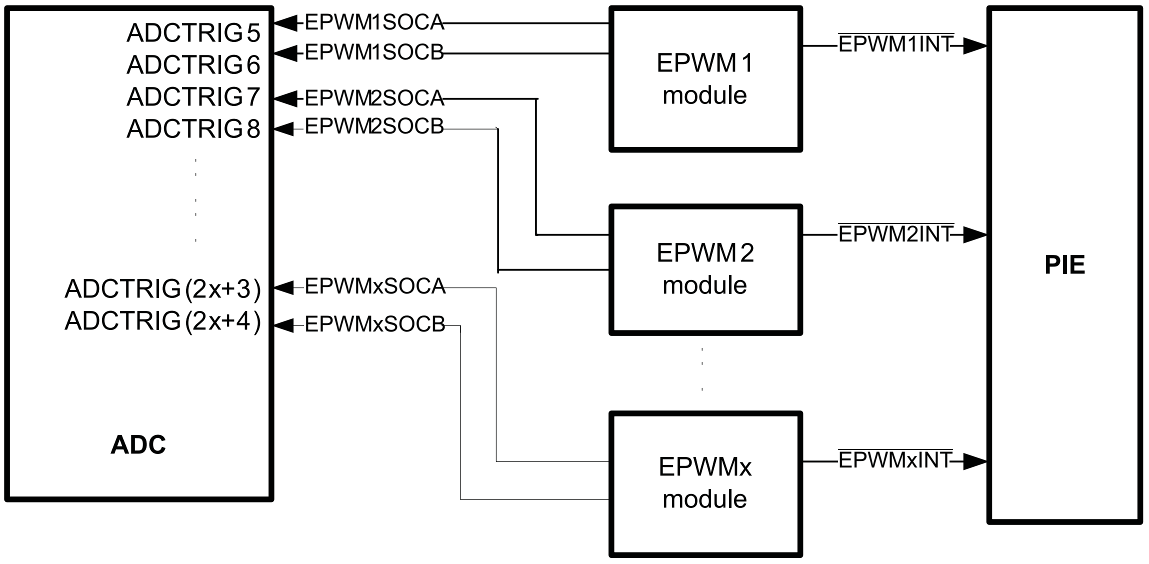


3.2.8.1 Operational Overview of the Event-Trigger Submodule

The following sections describe the event-trigger submodule's operational highlights.

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 3-42, 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 3-42. Event-Trigger Submodule Inter-Connectivity of ADC Start of Conversion

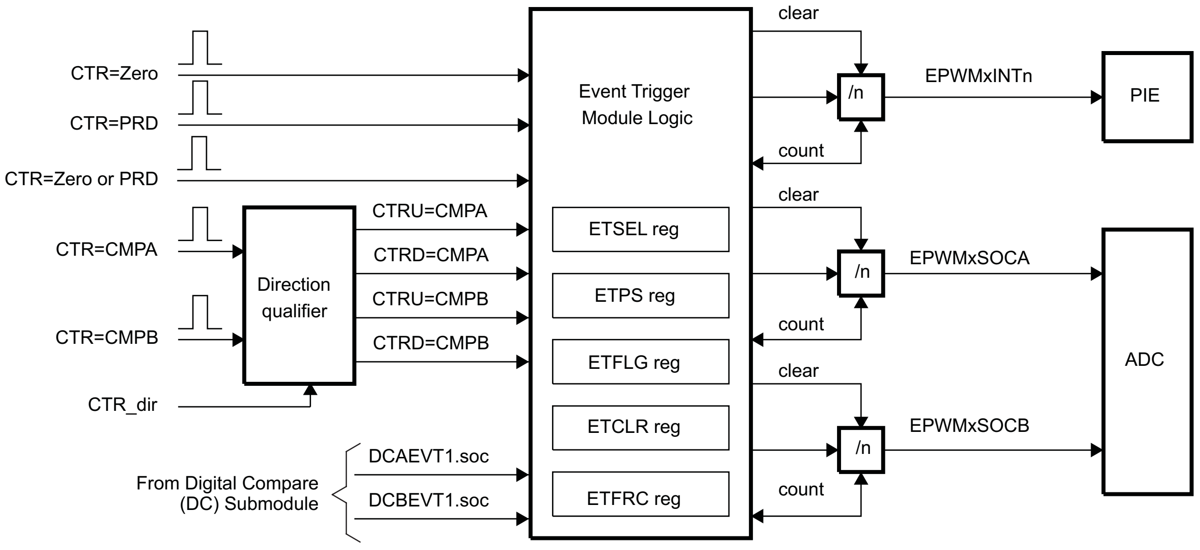


The event-trigger submodule monitors various event conditions (the left side inputs to event-trigger submodule shown in Figure 3-43) and can be configured to prescale these events before issuing an Interrupt request or an ADC start of conversion. The event-trigger prescaling logic can issue Interrupt requests and ADC start of conversion at:

- Every event
- Every second event

- Every third event

Figure 3-43. Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs



The key registers used to configure the event-trigger submodule are shown in Table 3-20:

Table 3-20. Event-Trigger Submodule Registers

Register Name	Address offset	Shadowed	Description
ETSEL	0x0019	No	Event-trigger Selection Register
ETPS	0x001A	No	Event-trigger Prescale Register
ETFLG	0x001B	No	Event-trigger Flag Register
ETCLR	0x001C	No	Event-trigger Clear Register
ETFRC	0x001D	No	Event-trigger Force Register

- 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.

A more detailed look at how the various register bits interact with the Interrupt and ADC start of conversion logic are shown in Figure 3-44, Figure 3-45, and Figure 3-46.

Figure 3-44 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 ENTFLG[INT] flag is cleared. This allows for one interrupt to be pending while one is serviced.

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 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 3-44. Event-Trigger Interrupt Generator

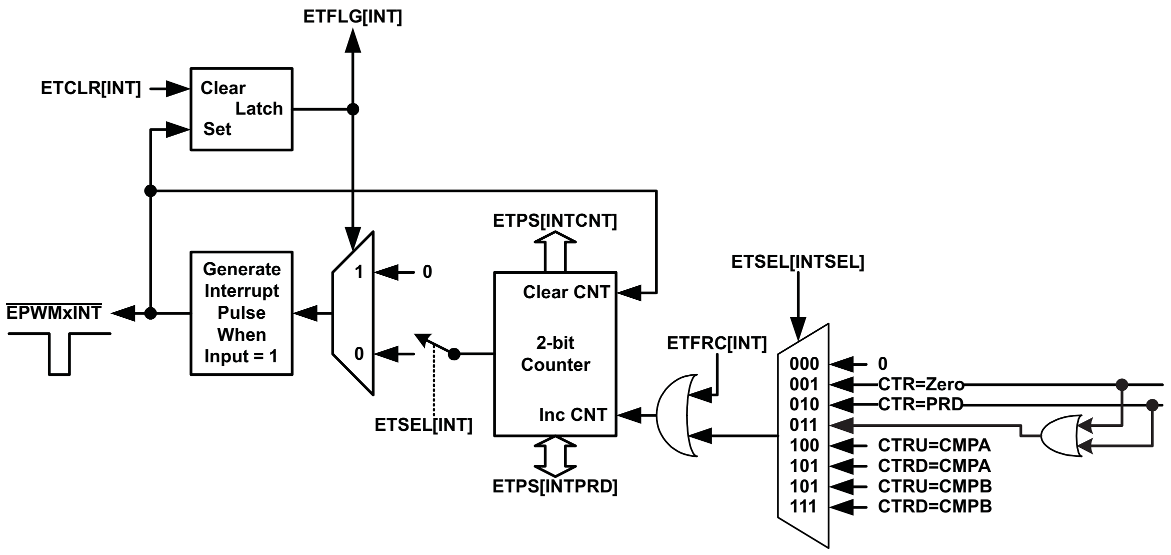
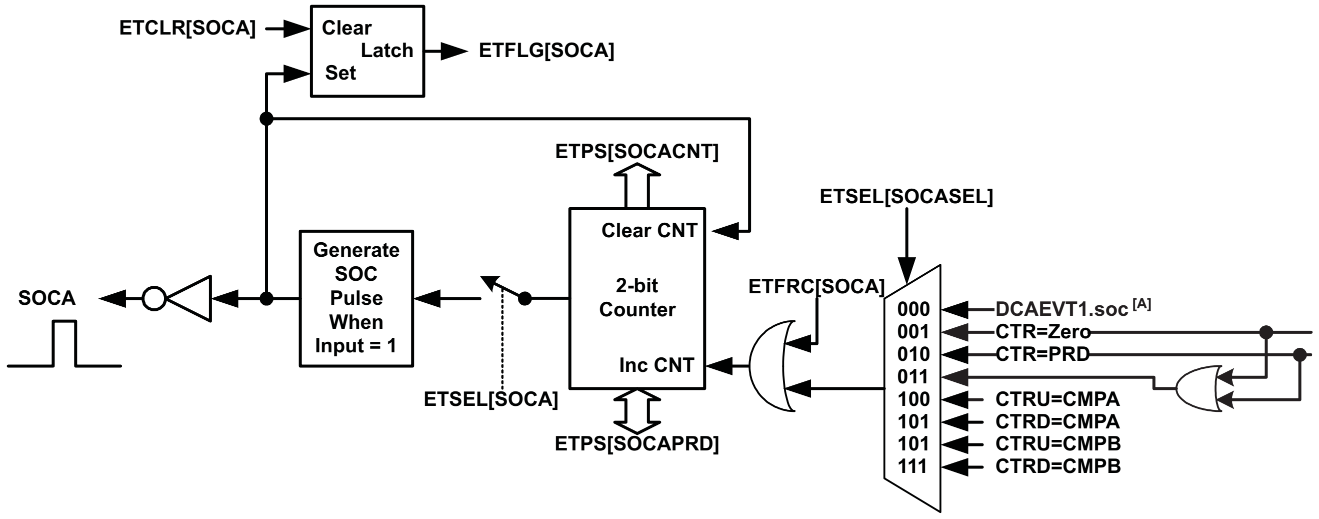


Figure 3-45 shows the operation of the event-trigger's start-of-conversion-A (SOCA) pulse generator. The ETPS[SOCACNT] counter and ETPS[SOCAPRD] period values behave similarly to the interrupt generator except that the pulses are continuously generated. That is, the pulse flag ETFLG[SOCA] is latched 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.

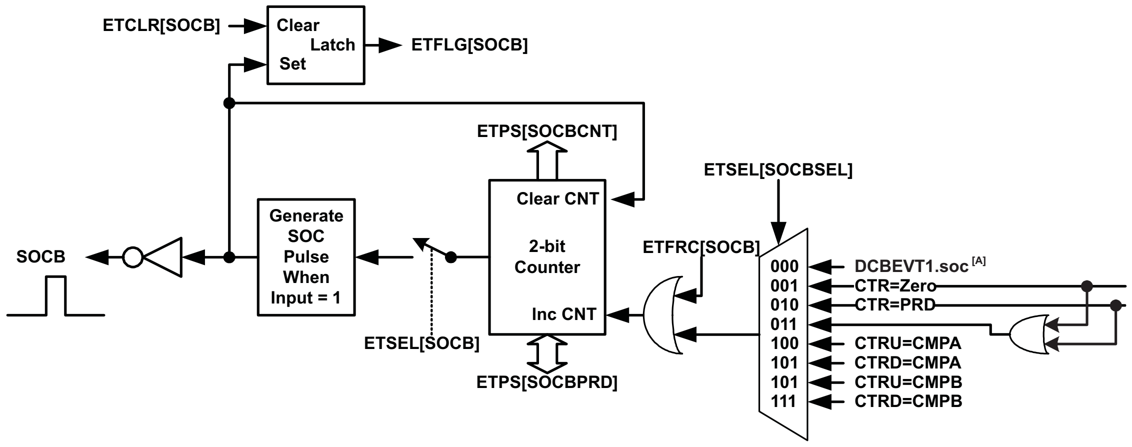
Figure 3-45. Event-Trigger SOCA Pulse Generator



A The DCAEVT1.soc signals are signals generated by the Digital compare (DC) submodule described later in Section 3.2.9

Figure 3-46 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.

Figure 3-46. Event-Trigger SOCB Pulse Generator

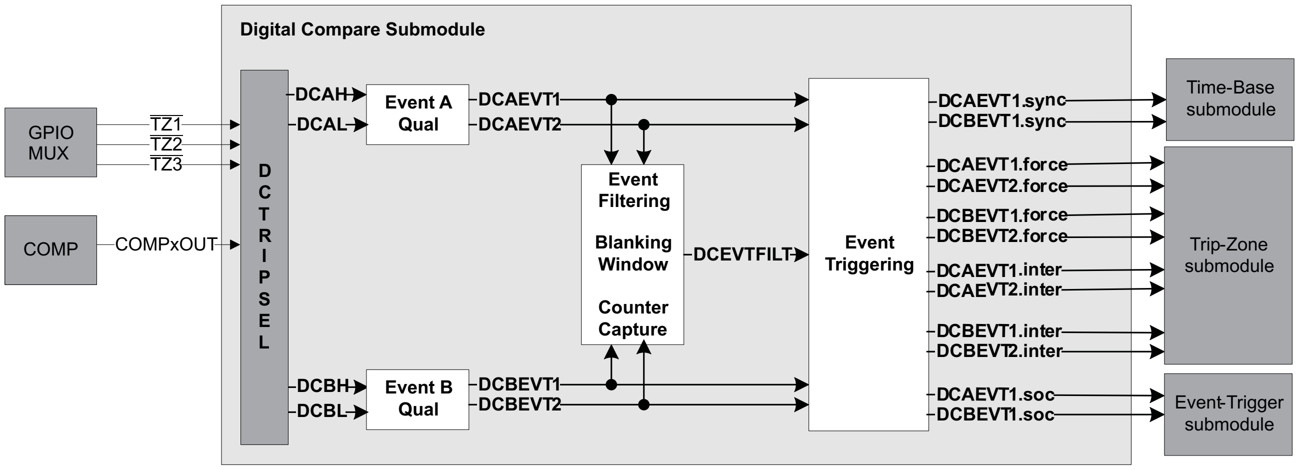


A The DCBEVT1.soc signals are signals generated by the Digital compare (DC) submodule described later in Section 3.2.9

3.2.9 Digital Compare (DC) Submodule

Figure 3-47 illustrates where the digital compare (DC) submodule signals interface to other submodules in the ePWM system.

Figure 3-47. Digital-Compare Submodule High-Level Block Diagram



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.

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 3-21. Digital Compare Submodule Registers

Register Name	Address offset	Shadowed	Description
TZDCSEL[(1)] [(2)]	0x13	No	Trip Zone Digital Compare Select Register
DCTRIPSEL[(1)]	0x30	No	Digital Compare Trip Select Register
DCACTL[(1)]	0x31	No	Digital Compare A Control Register
DCBCTL[(1)]	0x32	No	Digital Compare B Control Register
DCFCTL[(1)]	0x33	No	Digital Compare Filter Control Register
DCCAPCTL[(1)]	0x34	No	Digital Compare Capture Control Register
DCFOFFSET	0x35	Writes	Digital Compare Filter Offset Register
DCFOFFSETCNT	0x36	No	Digital Compare Filter Offset Counter Register
DCFWINDOW	0x37	No	Digital Compare Filter Window Register

(1) These registers are EALLOW protected and can be modified only after executing the EALLOW instruction. For more information, see the device-specific version of the System Control and Interrupts Reference Guide.

(2) The TZDCSEL register is part of the trip-zone submodule but is mentioned again here because of its functional significance to the digital compare submodule.

Table 3-21. Digital Compare Submodule Registers (continued)

Register Name	Address offset	Shadowed	Description
DCFWINDOWCNT	0x38	No	Digital Compare Filter Window Counter Register
DCCAP	0x39	Yes	Digital Compare Counter Capture Register

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 illustrated in Figure 3-47 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 DCTRIPSEL 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*TBCLK sync pulse width is required. If pulse width is < 3*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 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 EPWMxSYNCI input signal and the TBCTL[SWFSYNC] signal to generate a synchronization pulse to the time-base counter.

The diagrams below show how the DCAEVT1, DCAEVT2 or DCEVTFLT signals are processed to generate the digital compare A event force, interrupt, soc and sync signals.

Figure 3-48. DCAEVT1 Event Triggering

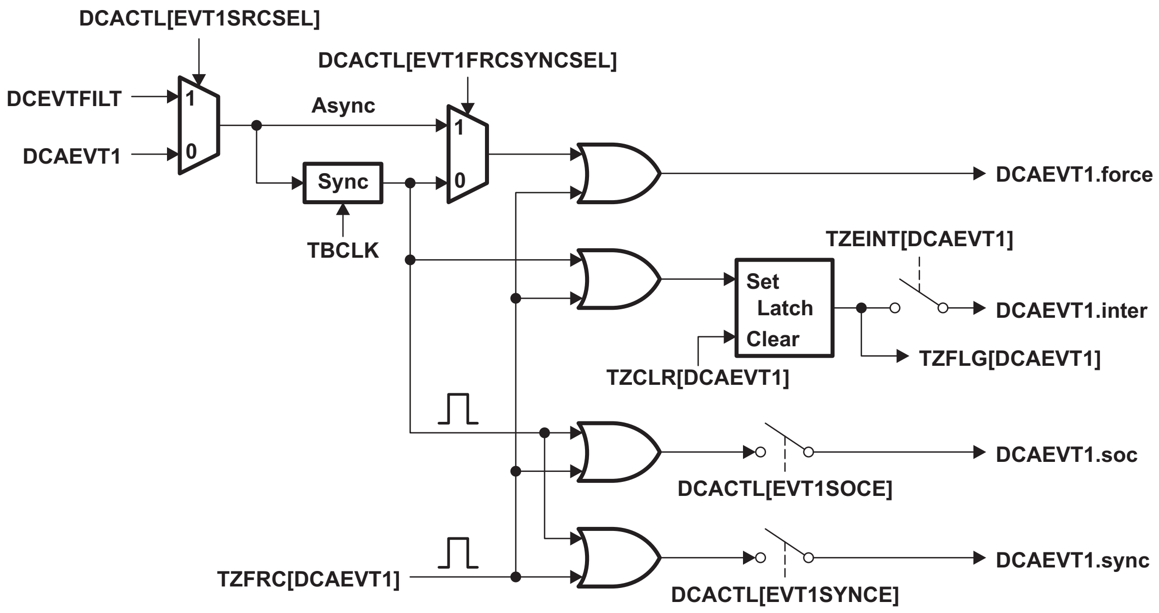
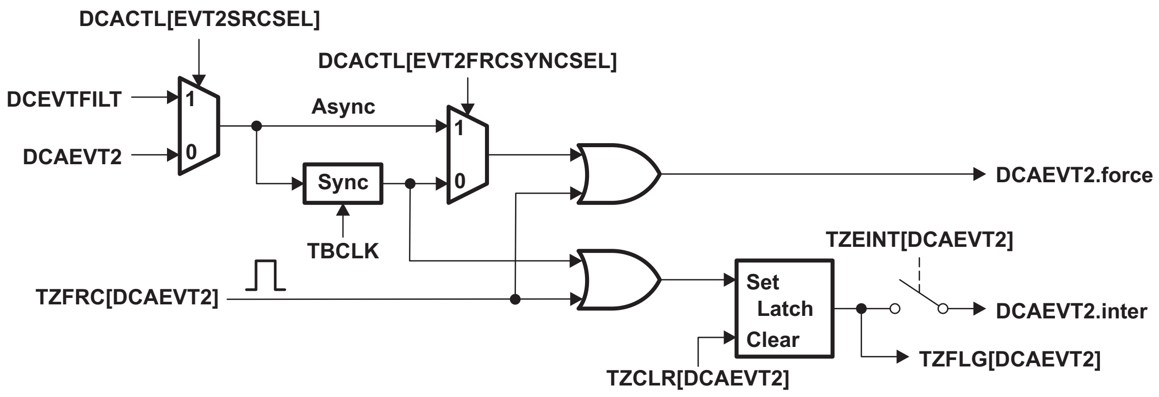


Figure 3-49. DCAEVT2 Event Triggering



The diagrams below show how the DCBEVT1, DCBEVT2 or DCEVTFLT signals are processed to generate the digital compare B event force, interrupt, soc and sync signals.

Figure 3-50. DCBEVT1 Event Triggering

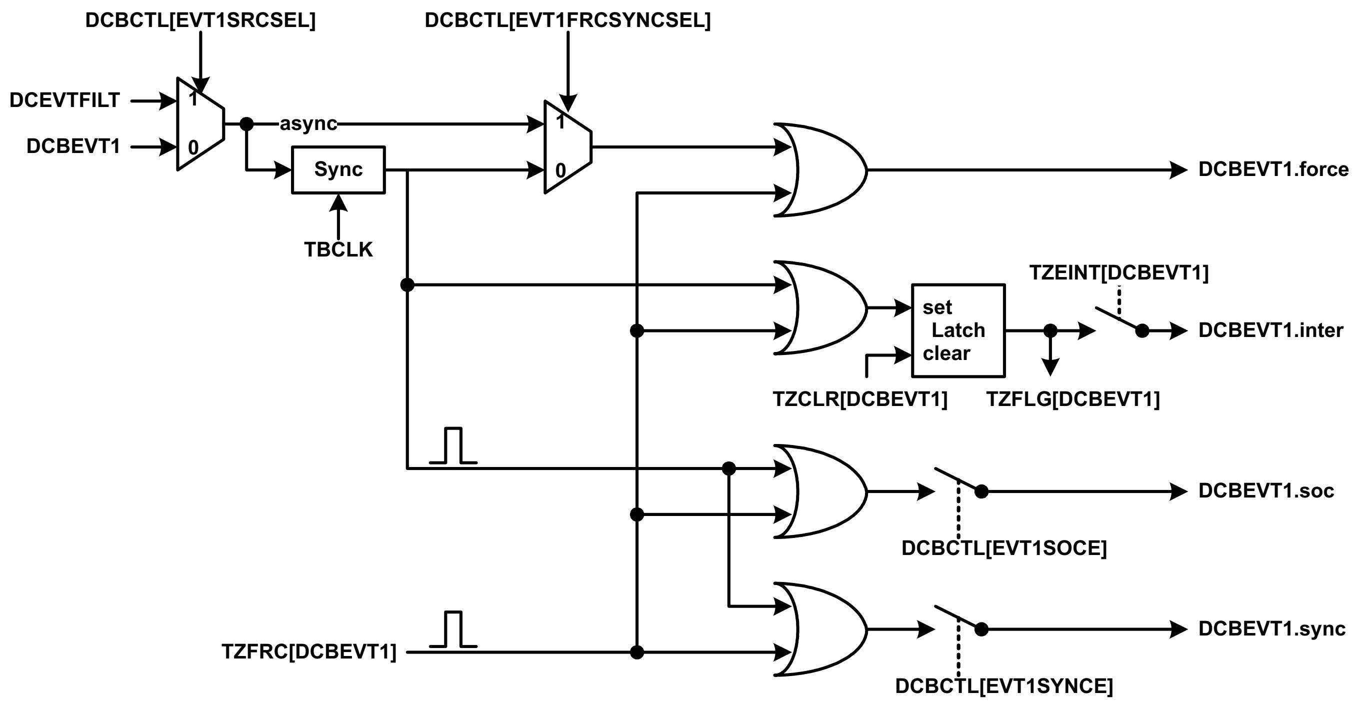
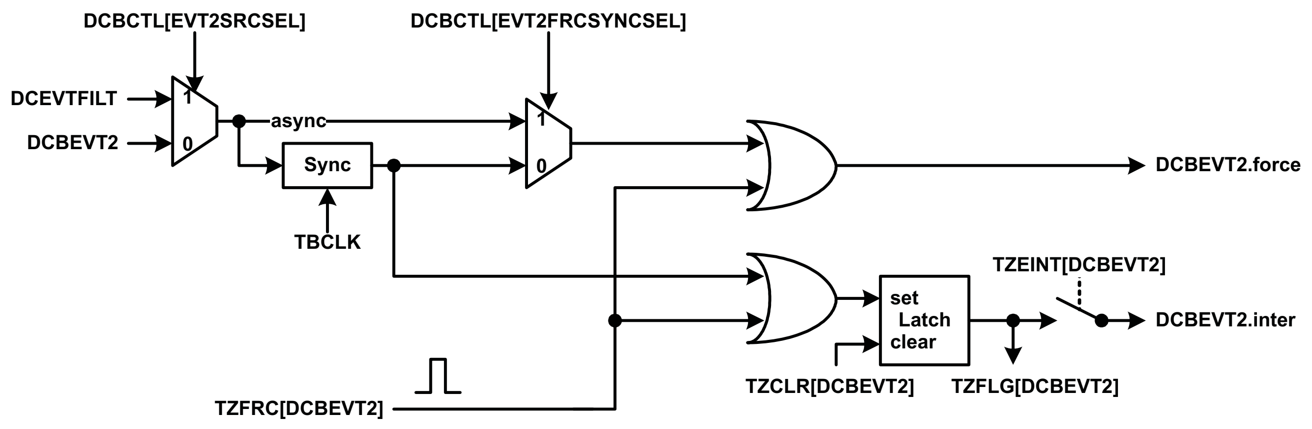


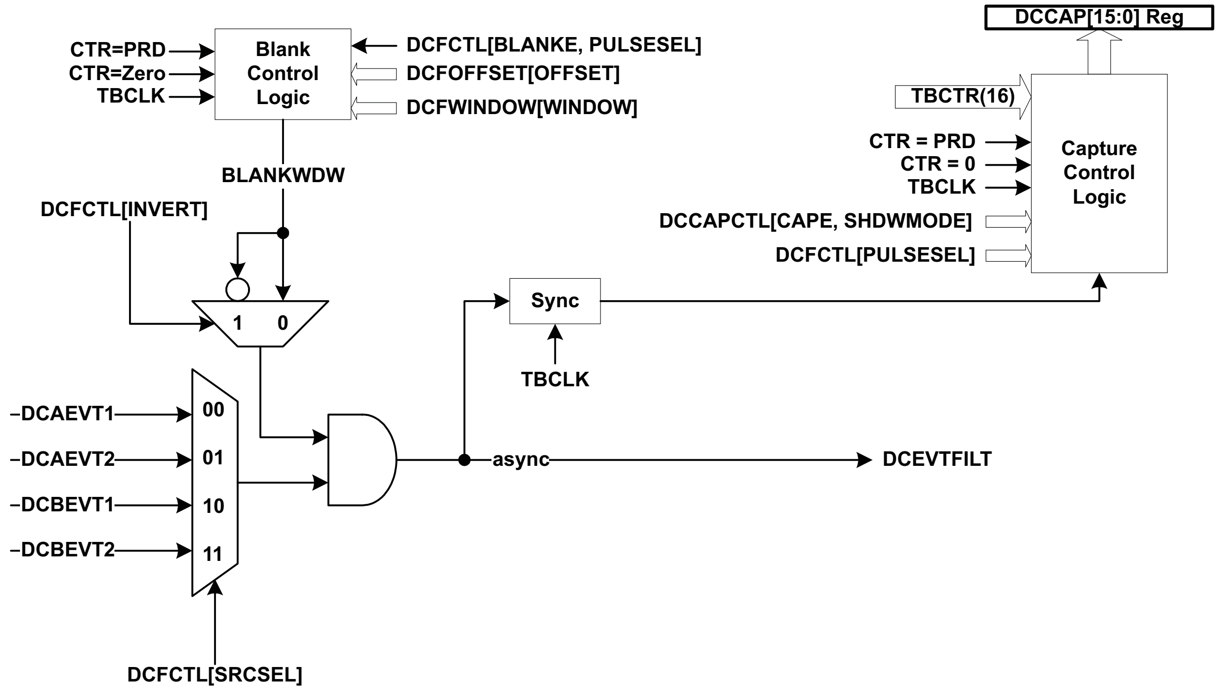
Figure 3-51. DCBEVT2 Event Triggering



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. The diagram below shows the details of the event filtering logic.

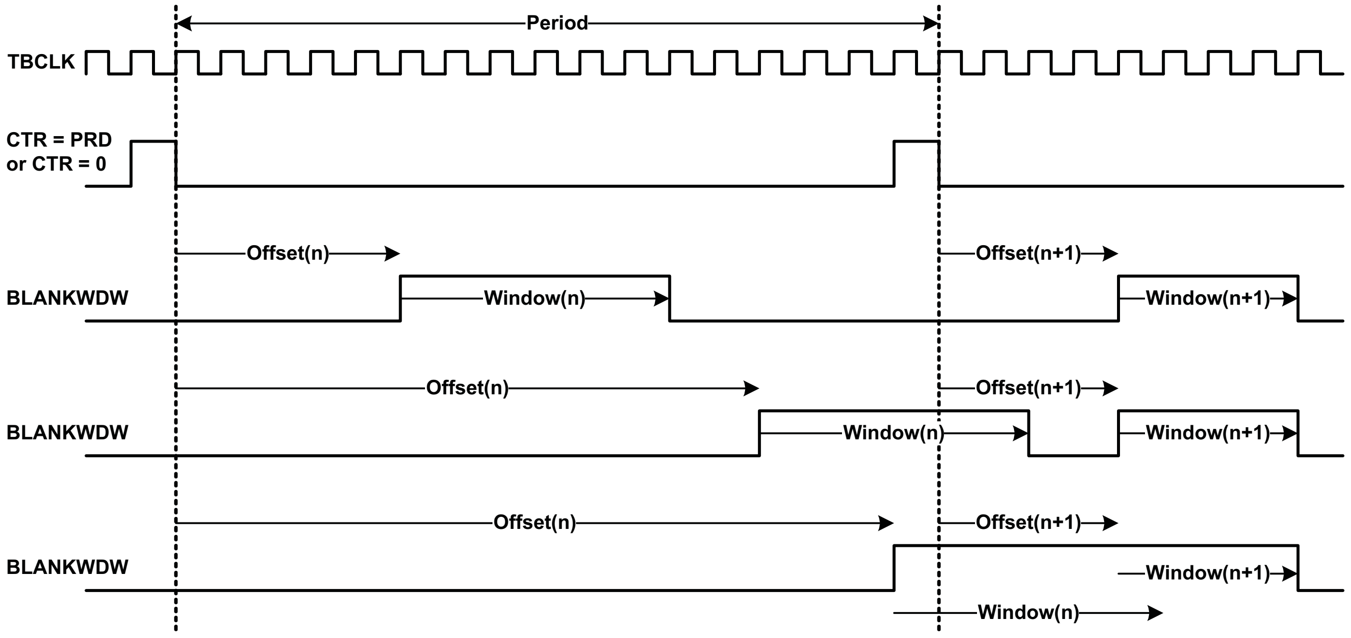
Figure 3-52. Event Filtering



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.

The diagram below 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 3-53. Blanking Window Timing Diagram



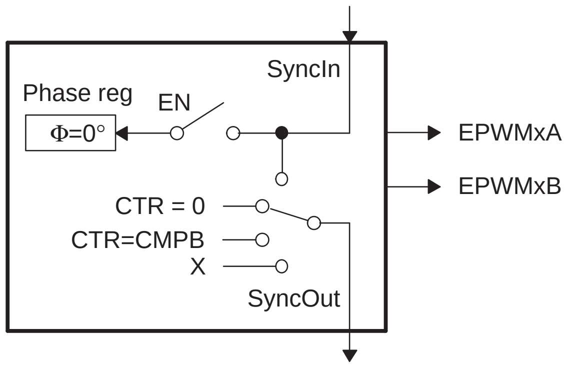
3.3 Applications to Power Topologies

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 in this user's guide, 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 3-54. 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 3-54. Simplified ePWM Module



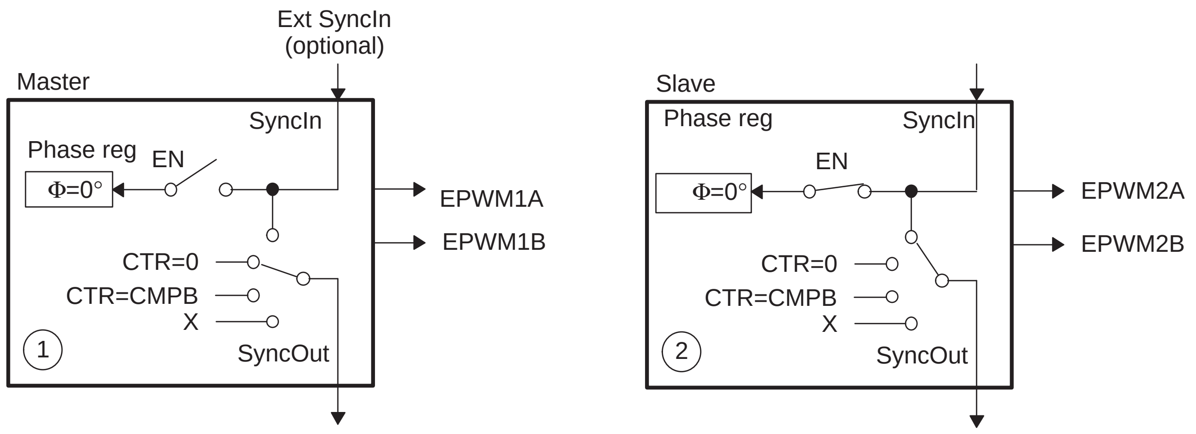
3.3.2 Key Configuration Capabilities

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, i.e., via the enable switch. Although various combinations are possible, the two most common—master module and slave module modes—are shown in Figure 3-55.

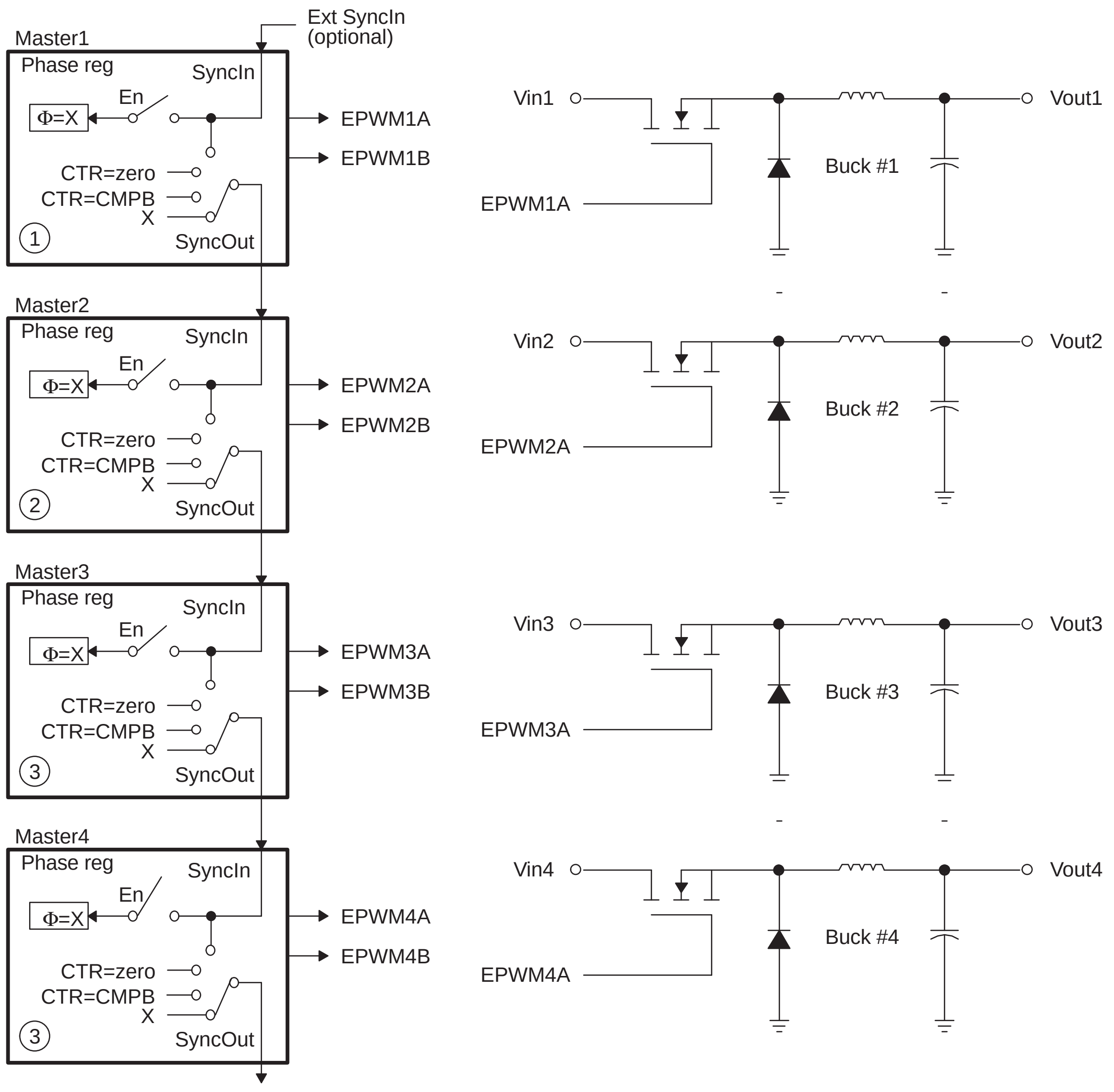
Figure 3-55. EPWM1 Configured as a Typical Master, EPWM2 Configured as a Slave



3.3.3 *Controlling Multiple Buck Converters With Independent Frequencies*

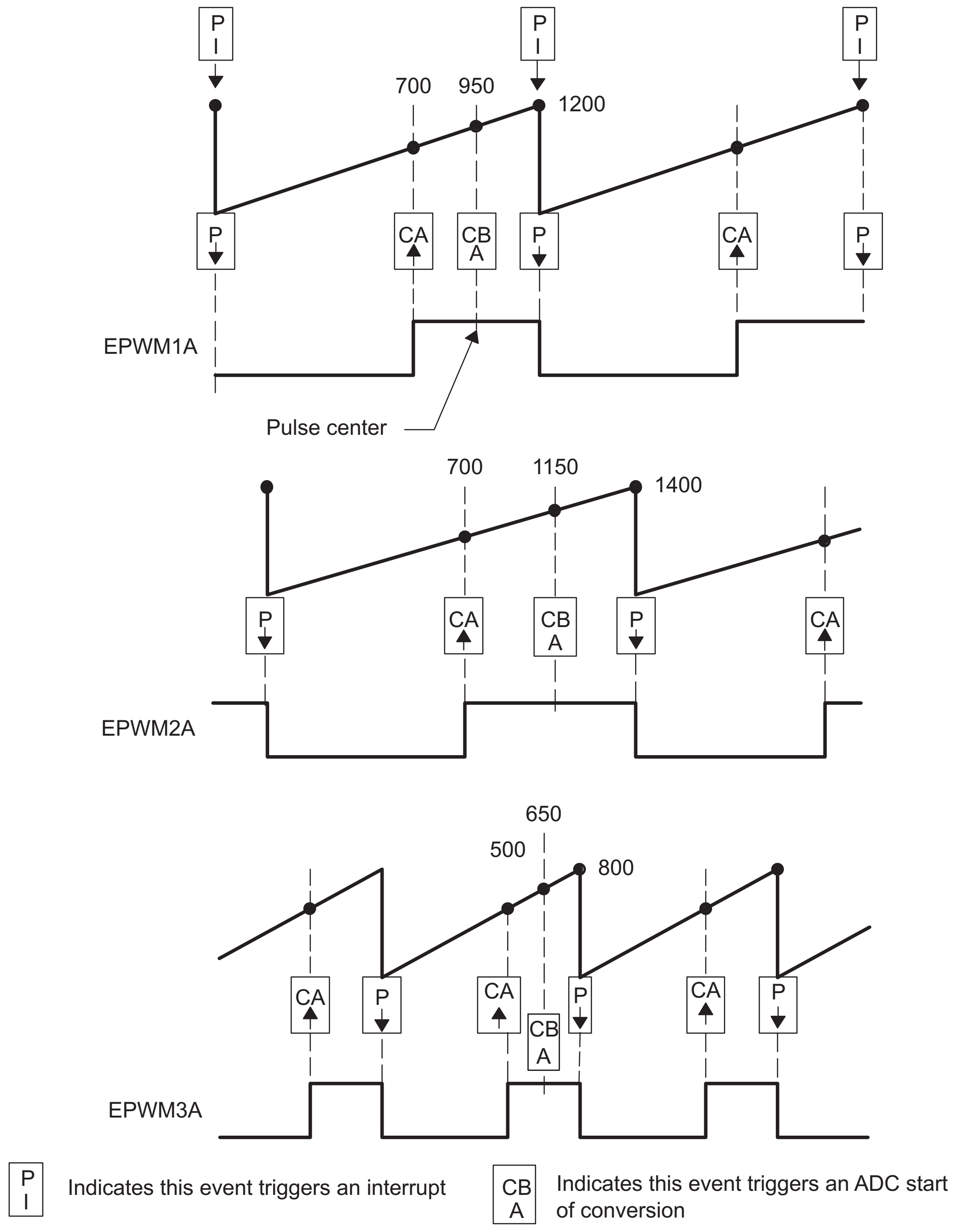
One of the simplest power converter topologies is the buck. A single ePWM module configured as a master can control two buck stages with the same PWM frequency. If independent frequency control is required for each buck converter, then one ePWM module must be allocated for each converter stage. Figure 3-56 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 3-57 shows the waveforms generated by the setup shown in Figure 3-56; note that only three waveforms are shown, although there are four stages.

Figure 3-56. Control of Four Buck Stages. Here $F_{PWM1} \neq F_{PWM2} \neq F_{PWM3} \neq F_{PWM4}$



NOTE: Θ = X indicates value in phase register is a "don't care"

Figure 3-57. Buck Waveforms for Figure 3-56 (Note: Only three bucks shown here)



P I
700
950
1200
P
CA
CB A
EPWM1A
Pulse center
700
1150
1400
EPWM2A
650
500
800
EPWM3A
P I
Indicates this event triggers an interrupt
CB A
Indicates this event triggers an ADC start of conversion

Example 3-8. Configuration for Example in Figure 3-57

```
//=====================================================================
// (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.PRDLD = TB_SHADOW;
   EPwm1Regs.TBCTL.bit.SYNCOSEL = 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.PRD = 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.PRDLD = TB_SHADOW;
   EPwm2Regs.TBCTL.bit.SYNCOSEL = 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.PRD = 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.PRDLD = TB_SHADOW;
   EPwm3Regs.TBCTL.bit.SYNCOSEL = 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.PRD = 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 3-58 shows such a configuration; Figure 3-59 shows the waveforms generated by the configuration.

Figure 3-58. Control of Four Buck Stages. (Note: $F_{PWM2} = N \times F_{PWM1}$)

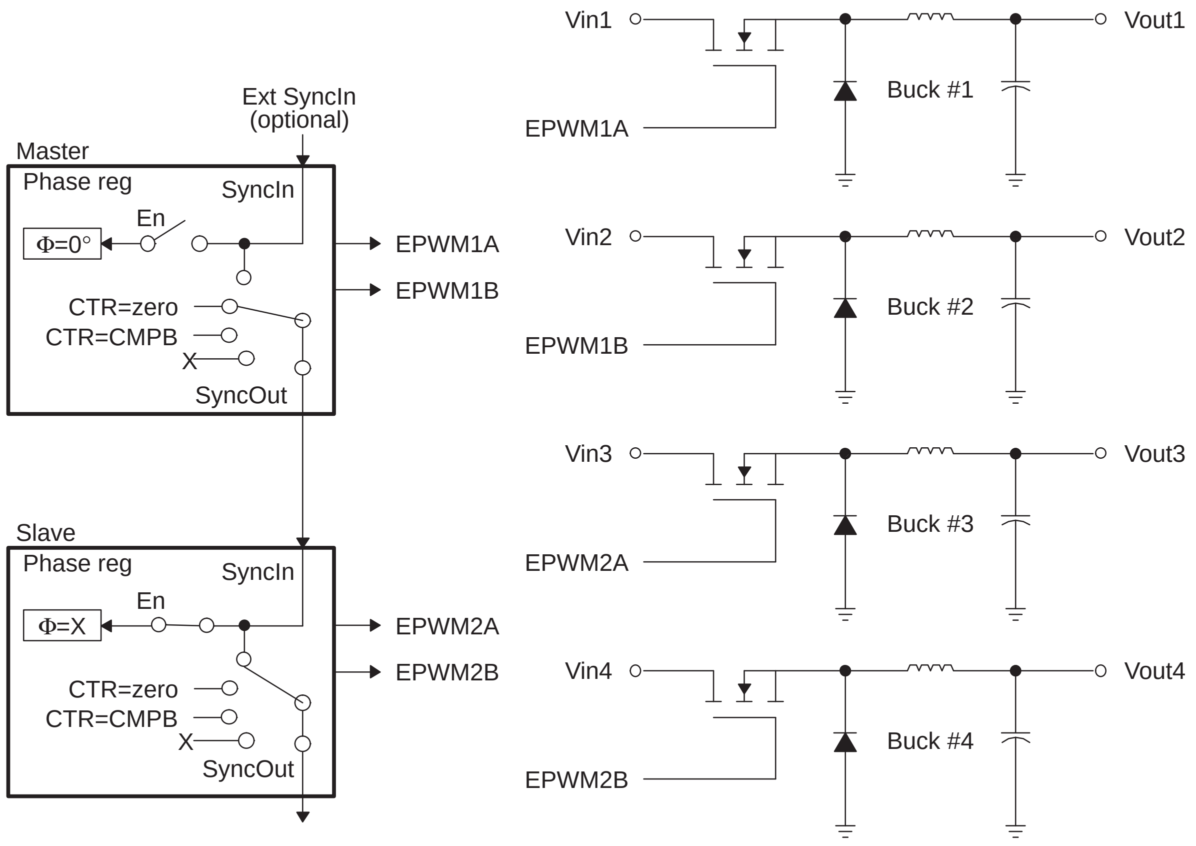
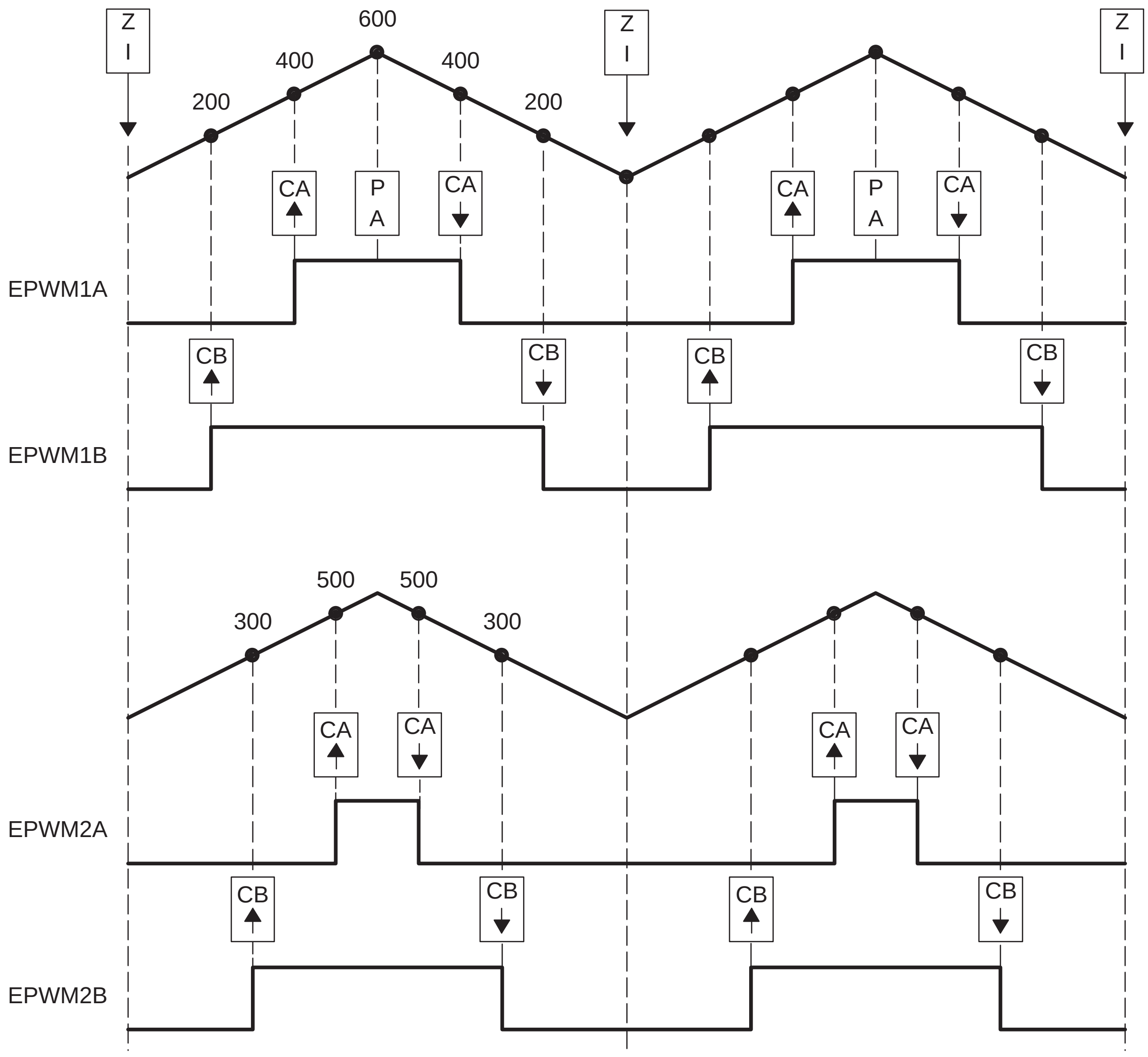


Figure 3-59. Buck Waveforms for Figure 3-58 (Note: $F_{PWM2} = F_{PWM1}$)



Example 3-9. Code Snippet for Configuration in Figure 3-58

```
//=========================
// 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.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.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.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.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 3-60 shows control of two synchronized Half-H bridge stages where stage 2 can operate at integer multiple (N) frequencies of stage 1. Figure 3-61 shows the waveforms generated by the configuration shown in Figure 3-60.

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 3-60. Control of Two Half-H Bridge Stages (F_{PWM2} = N x F_{PWM1})

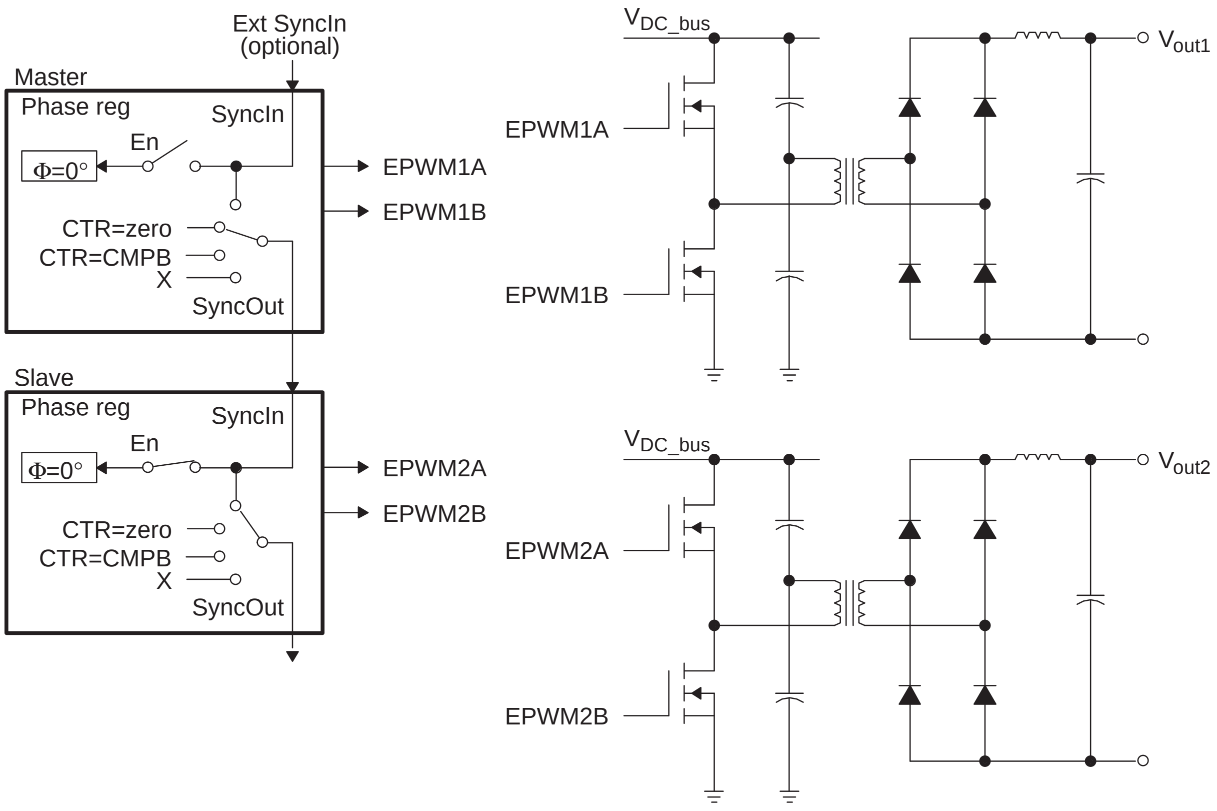
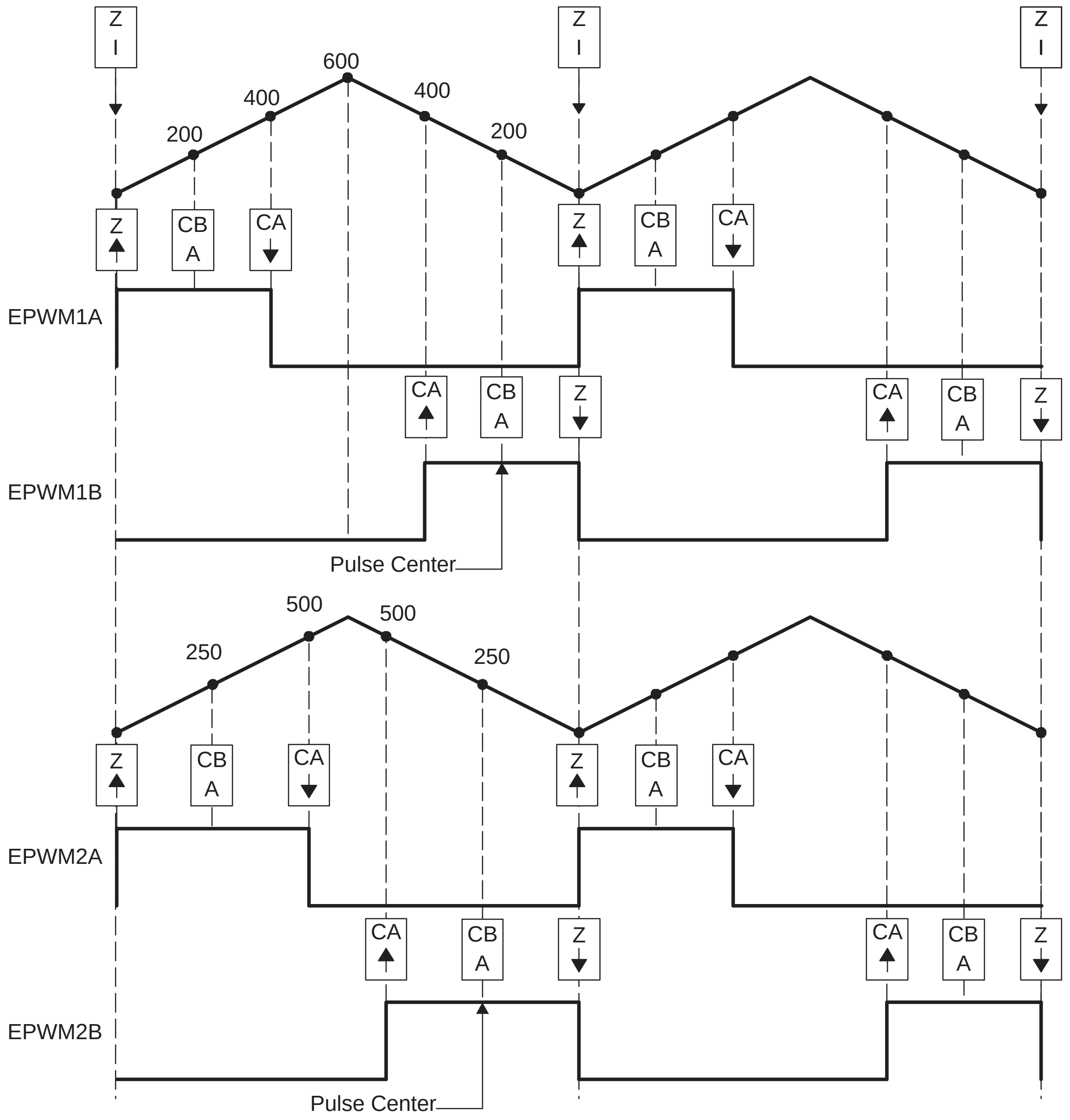


Figure 3-61. Half-H Bridge Waveforms for Figure 3-60 (Note: Here F_{PWM2} = F_{PWM1})



Example 3-10. Code Snippet for Configuration in Figure 3-60

```
//=====================================================================
// 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.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.AQCTLB.bit.ZRO = AQ_CLEAR;           // set actions for EPWM1B
     EPwm1Regs.AQCTLB.bit.CAD = AQ_SET;
// 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.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 EPWM1A
     EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;
     EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR;           // set actions for EPWM1B
     EPwm2Regs.AQCTLB.bit.CAD = AQ_SET;
//=============================================================
     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 3-62 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 3-62), 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 3-62. Control of Dual 3-Phase Inverter Stages as Is Commonly Used in Motor Control

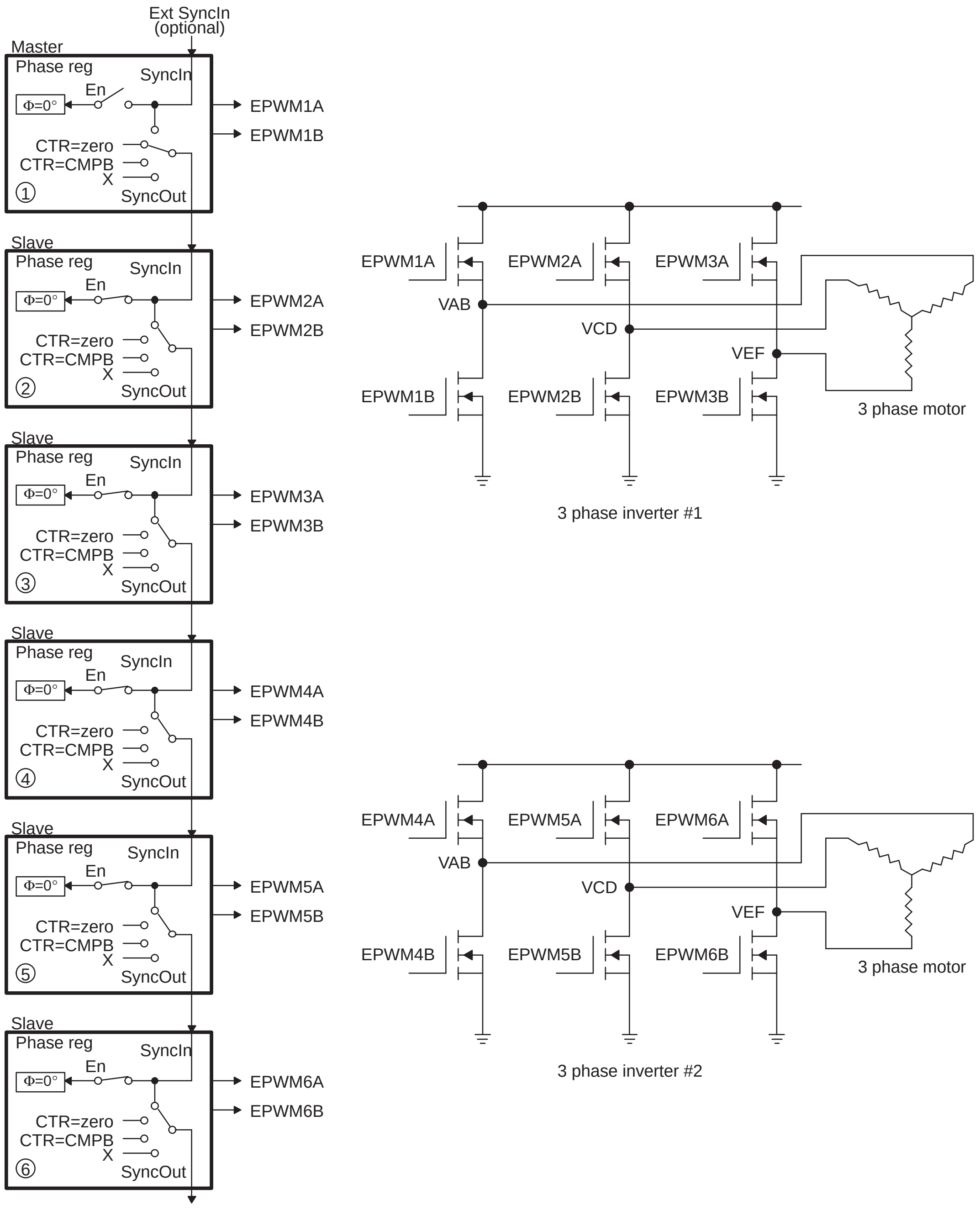
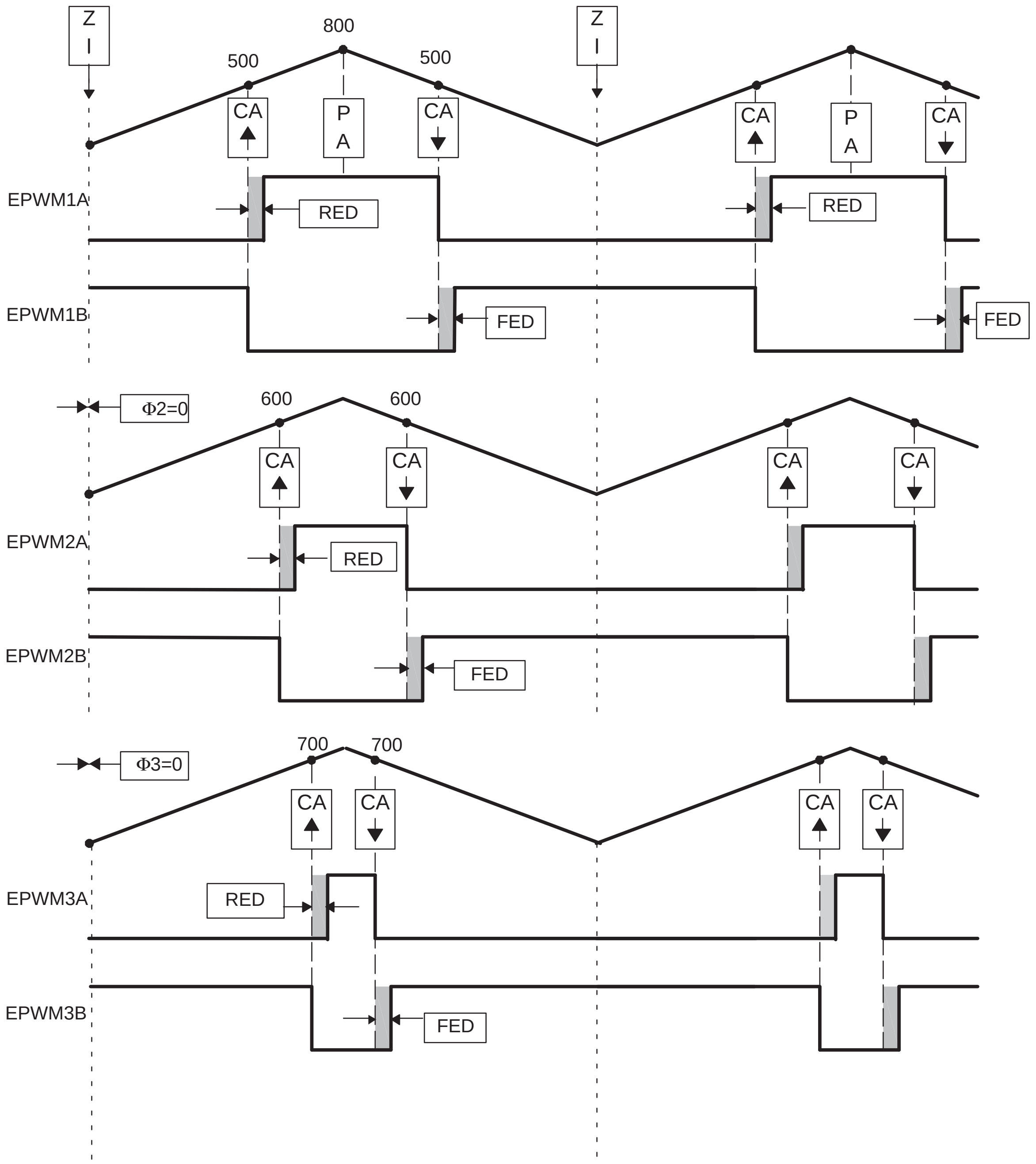


Figure 3-63. 3-Phase Inverter Waveforms for Figure 3-62 (Only One Inverter Shown)



Example 3-11. Code Snippet for Configuration in Figure 3-62

```
//=====================================================================
// 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.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.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.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.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.PRDLD = 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.bitMODE = 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 3-64 shows a master and slave module with a phase relationship of 120°, i.e., the slave leads the master.

Figure 3-64. Configuring Two PWM Modules for Phase Control

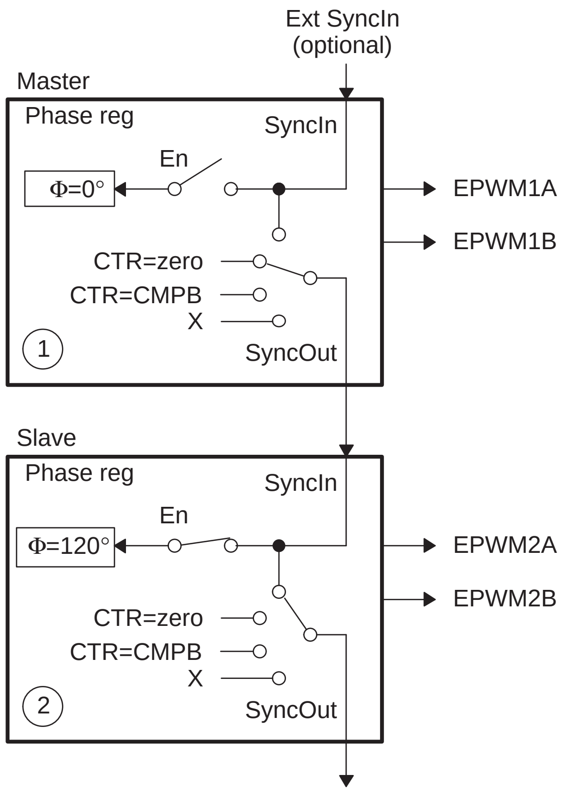
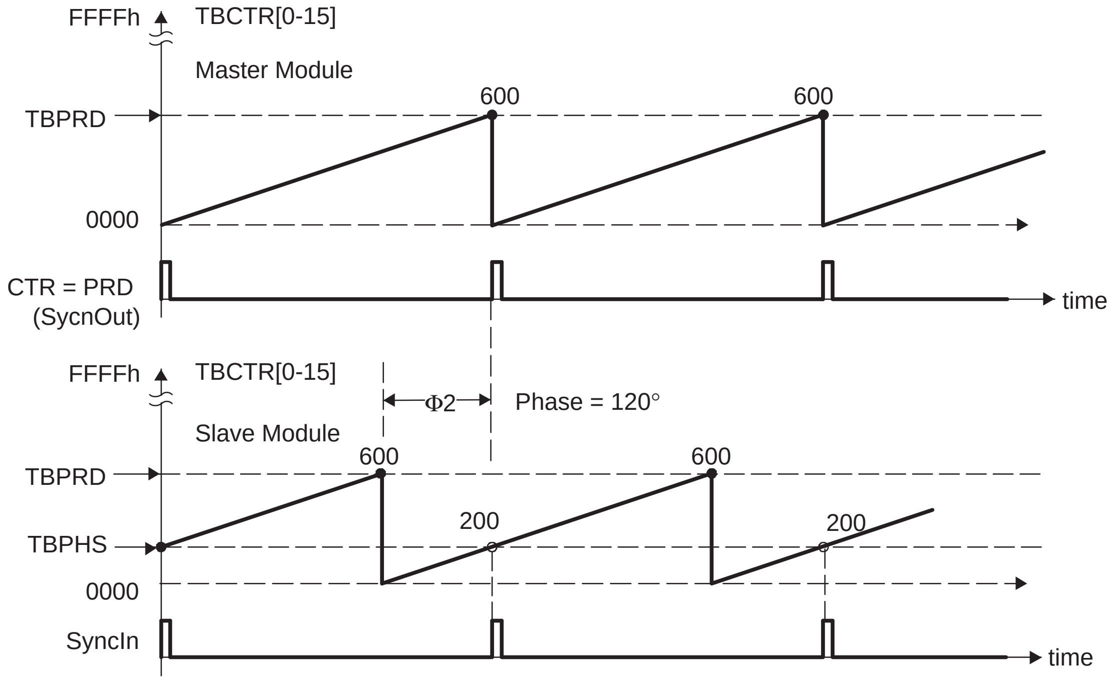


Figure 3-65 shows the associated timing waveforms for this configuration. Here, TBPRD = 600 for both master and slave. For the slave, TBPHS = 200 (200/600 X 360° = 120°). Whenever the master generates a SyncIn pulse (CTR = PRD), the value of TBPHS = 200 is loaded into the slave TBCTR register so the slave time-base is always leading the master's time-base by 120°.

Figure 3-65. 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 3-66. 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°. 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) x (—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) x (2-1) = 200 (i.e., Phase value for Slave module 2)

TBPHS(3,3) = 400 (i.e., Phase value for Slave module 3)

Figure 3-67 shows the waveforms for the configuration in Figure 3-66.

Figure 3-66. Control of a 3-Phase Interleaved DC/DC Converter

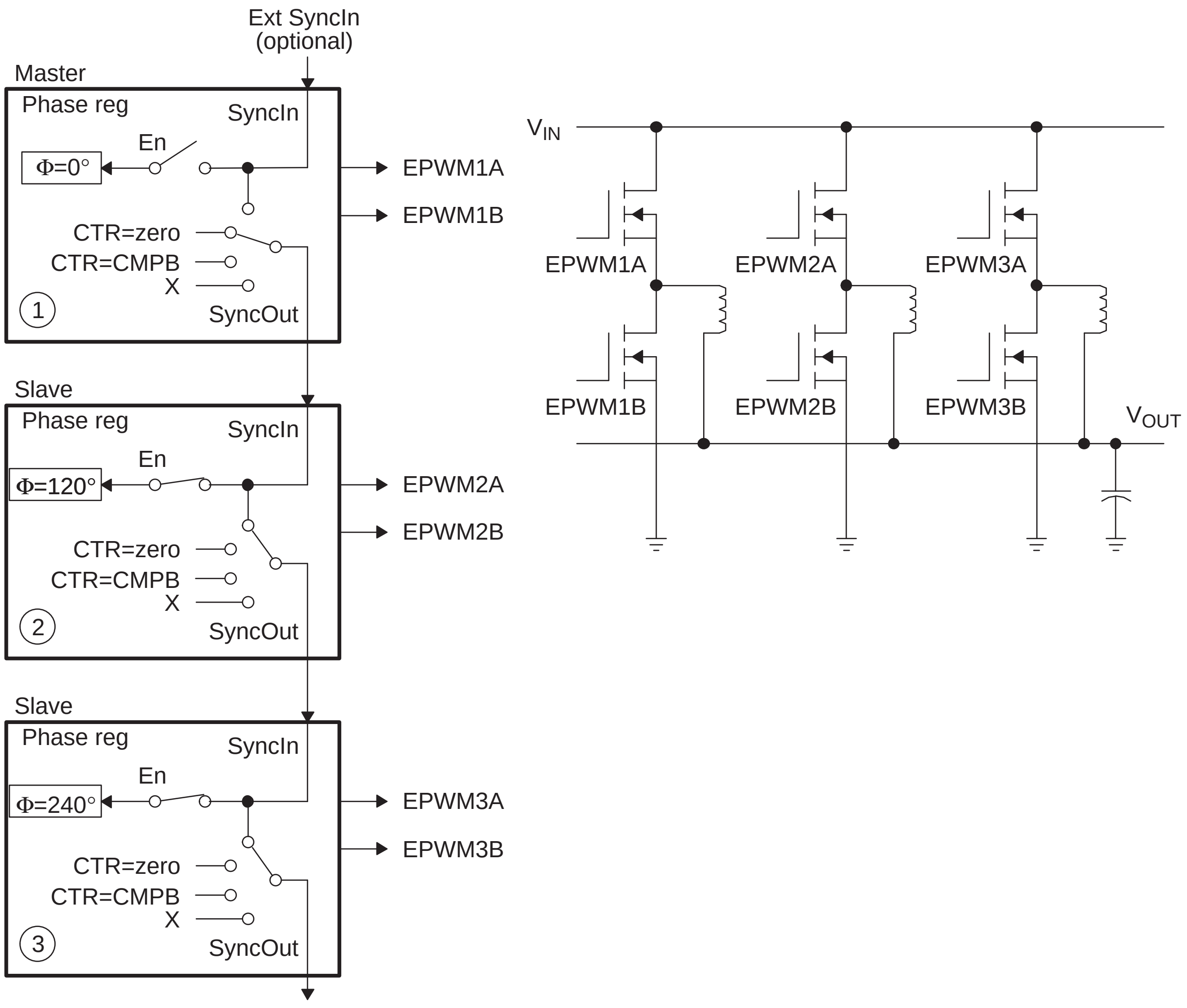
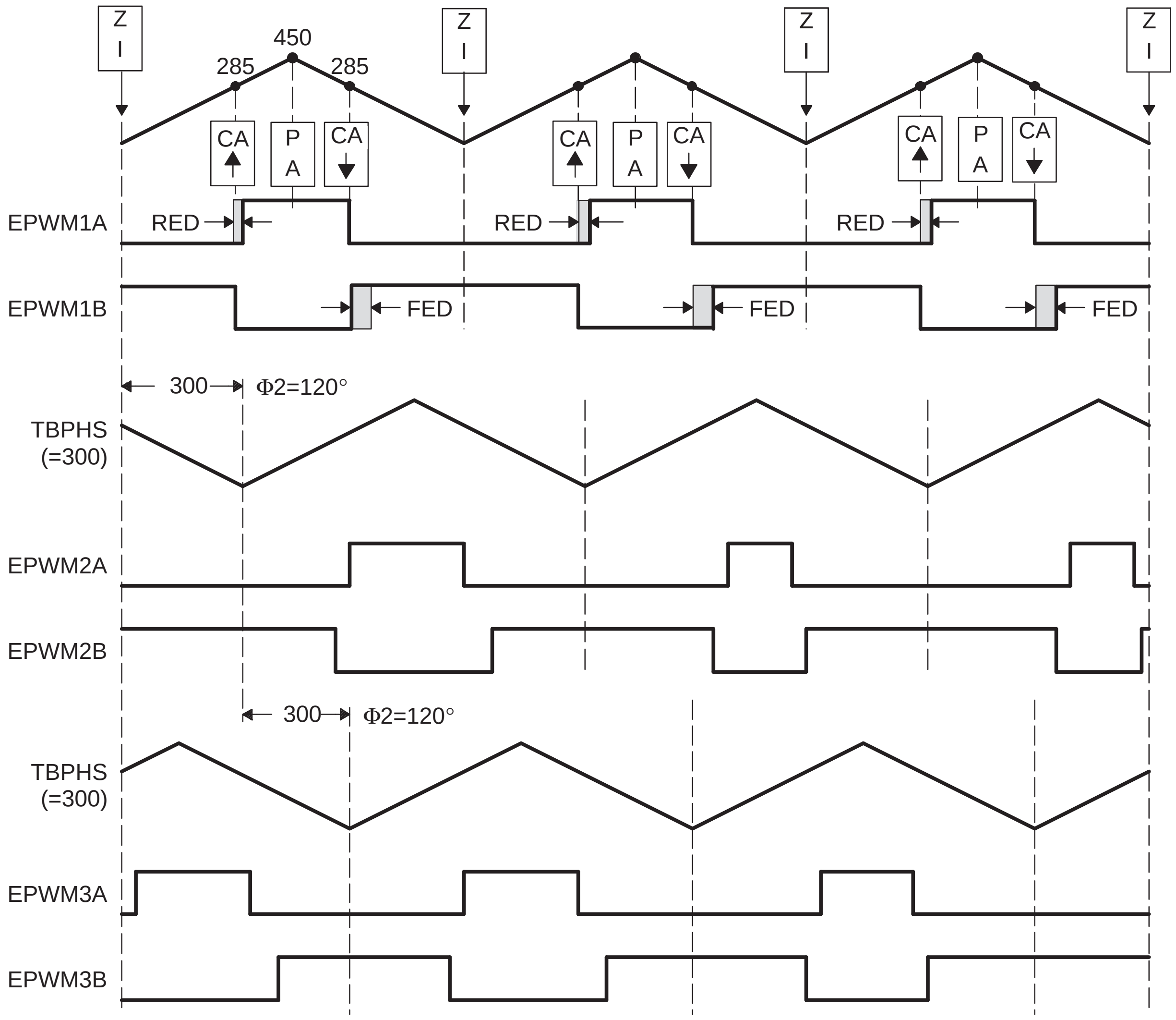


Figure 3-67. 3-Phase Interleaved DC/DC Converter Waveforms for Figure 3-66



Example 3-12. Code Snippet for Configuration in Figure 3-66

```
//=====================================================================
// 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.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.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
// EPWM Module 2 config
EPwm2Regs.TBPRD = 450;                          // Period = 900 TBCLK counts
EPwm2Regs.TBPHS.half.TBPHS = 300;               // Phase = 300/900 * 360 = 120 deg
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Symmetrical mode
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;          // Slave module
EPwm2Regs.TBCTL.bit.PHSDIR = TB_DOWN;           // Count DOWN on sync (=120 deg)
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.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
// EPWM Module 3 config
EPwm3Regs.TBPRD = 450;                          // Period = 900 TBCLK counts
EPwm3Regs.TBPHS.half.TBPHS = 300;               // Phase = 300/900 * 360 = 120 deg
EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Symmetrical mode
EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;          // Slave module
EPwm2Regs.TBCTL.bit.PHSDIR = TB_UP;             // Count UP on sync (=240 deg)
EPwm3Regs.TBCTL.bit.PRDLD = 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 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 3-68 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 3-69 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 3-68. Controlling a Full-H Bridge Stage (F_{PWM2} = F_{PWM1})

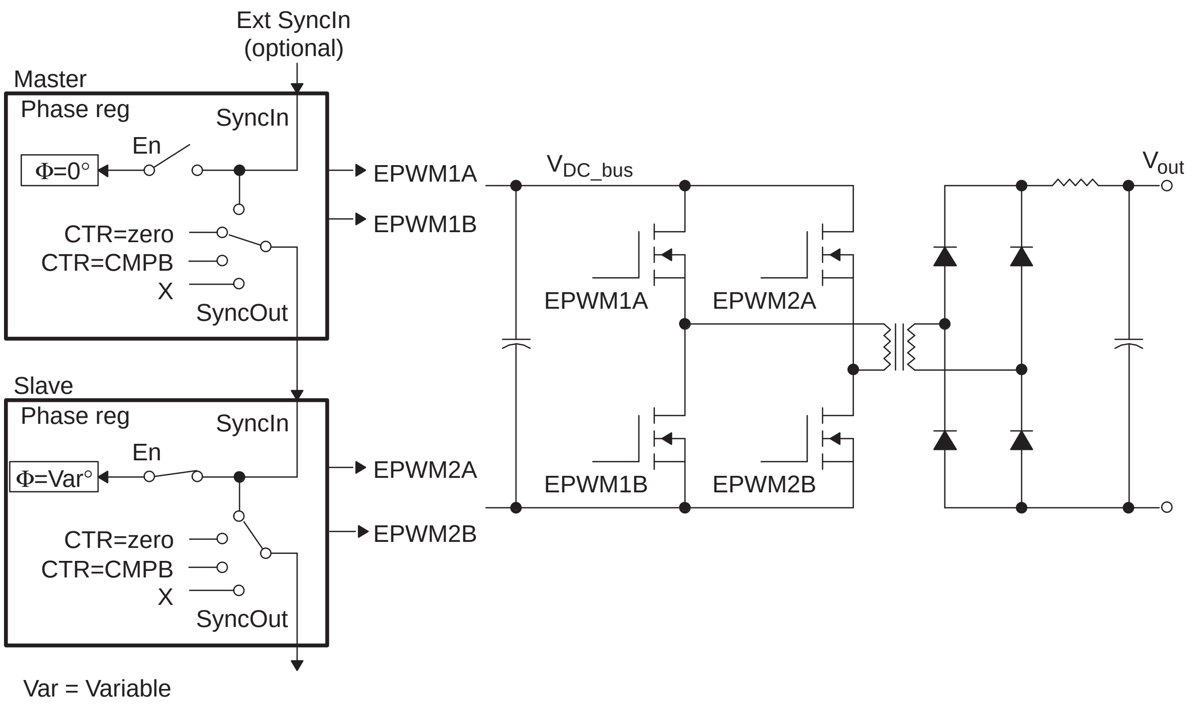
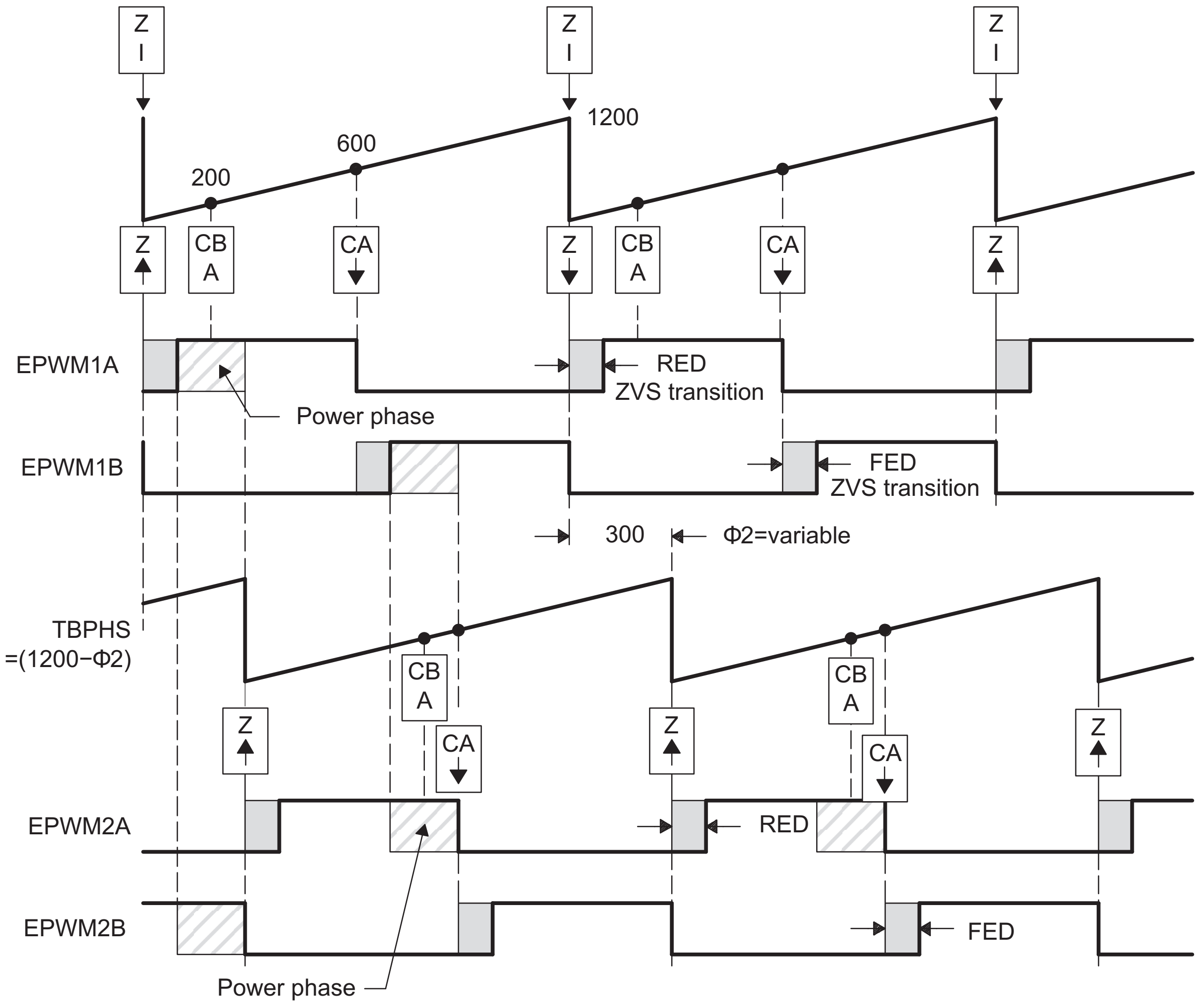


Figure 3-69. ZVS Full-H Bridge Waveforms



Example 3-13. Code Snippet for Configuration in Figure 3-68

```
//=====================================================================
// Config
//=====================================================================
// Initialization Time
//========================
// EPWM Module 1 config
EPwm1Regs.TBPRD = 1200;                               // Period = 1201 TBCLK counts
EPwm1Regs.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 3-70 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 sub-module. 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 3-71 shows the waveforms generated by the configuration.

Figure 3-70. Peak Current Mode Control of a Buck Converter

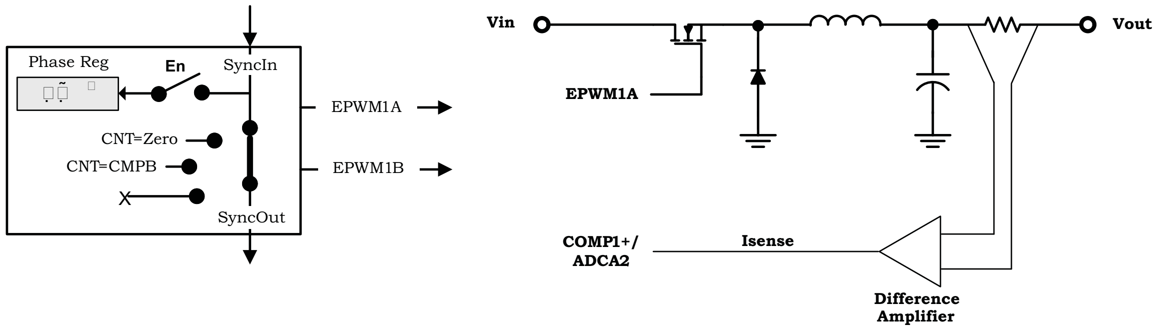
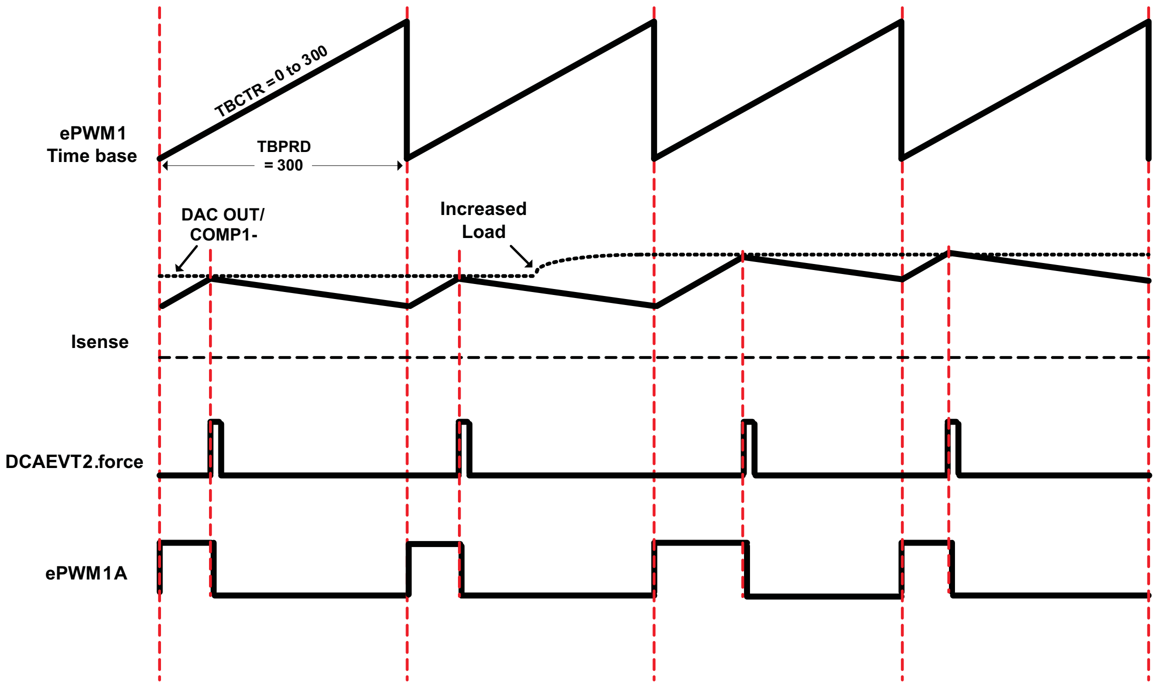


Figure 3-71. Peak Current Mode Control Waveforms for Figure 3-70



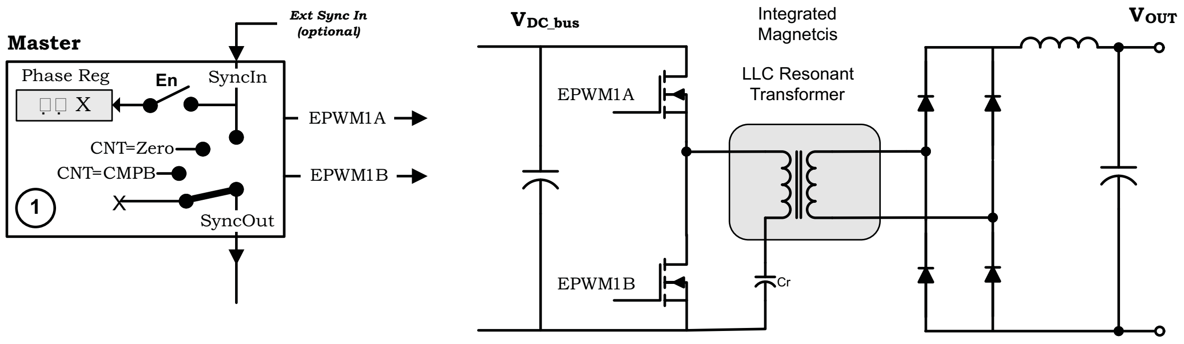
Example 3-14. Code Snippet for Configuration in Figure 3-70

```
//=============================================================================
// 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

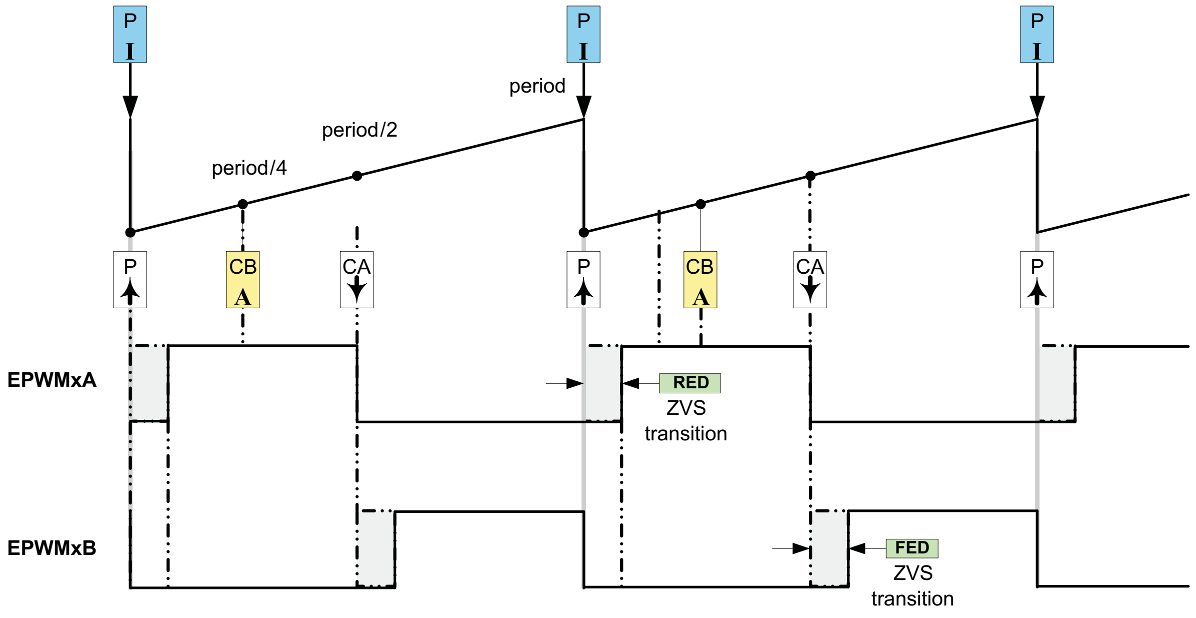
Various topologies of resonant converters are well-known in the field of power electronics for many years. In addition to these, H-bridge LLC resonant converter topology has recently gained popularity in many consumer electronics applications where high efficiency and power density are required. In this example single channel configuration of ePWM1 is detailed, yet the configuration can easily be extended to multi channel. 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 (i.e 30 @100MHz TBCLK), it is up to user to update it in real time to enhance the efficiency by adjusting enough time delay for soft switching.

Figure 3-72. Control of Two Resonant Converter Stages



NOTE: Θ = X indicates value in phase register is a "don't care"

Figure 3-73. H-Bridge LLC Resonant Converter PWM Waveforms



P I — Indicates this event triggers an interrupt

CB A — Indicates this event triggers an ADC start of conversion

Example 3-15. Code Snippet for Configuration in Figure 3-72

```
//=====================================================================
// Config
//===================================================================== //
Initialization Time
//======================== //
EPWMxA & EPWMxB config
EPwm1Regs.TBCTL.bit.PRDLD = 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.PRD = 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;
EPwm1Regs.TZSEL.bit.OSHT1 = 1;                                  // one-
shot source EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
                                                                // set EPWM1A to low at fault
EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_LO;                          // set EPWM1B to low at fault
instant
EPwm1Regs.TZEINT.bit.OST = 1;                                   // 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
```

3.4 Registers

This chapter includes the register layouts and bit description for the submodules.

3.4.1 Time-Base Submodule Registers

Figure 3-74 through Figure 3-82 and Table 3-22 through Table 3-30 provide the time-base register definitions.

Figure 3-74. Time-Base Period Register (TBPRD)

15	0
TBPRD	
R/W-0	

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

Table 3-22. Time-Base Period Register (TBPRD) Field Descriptions

Bits	Name	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. • 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. • 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. • The active and shadow registers share the same memory map address.

Figure 3-75. Time Base Period High Resolution Register (TBPRDHR)

15	8
TBPRDHR	
R/W-0	

7	0
Reserved	
R-0	

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

Table 3-23. 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[PRDLD] 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

Figure 3-76. Time Base Period Mirror Register (TBPRDM)

15	0
TBPRD	
R/W-0	

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

Table 3-24. Time Base Period Mirror Register (TBPRDM) Field Descriptions

Bit	Field	Value	Description
15-0	TBPRD	0000-FFFFh	TBPRDM and TBPRD can both be used to access the time-base period. 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. 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[PRDLD] bit. • If TBCTL[PRDLD] = 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. • If TBCTL[PRDLD] = 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.

Figure 3-77. Time-Base Period High Resolution Mirror Register (TBPRDHRM)

15 — 8
TBPRDHR
R/W-0

7 — 0
Reserved
R-0

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

Table 3-25. 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[PRDLD] 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

Figure 3-78. Time-Base Phase Register (TBPHS)

15 — 0
TBPHS
R/W-0

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

Table 3-26. Time-Base Phase Register (TBPHS) Field Descriptions

Bits	Name	Value	Description
15-0	TBPHS	0000-FFFF	These bits set time-base counter phase of the selected ePWM relative to the time-base that is supplying the synchronization input signal. • If TBCTL[PHSEN] = 0, then the synchronization event is ignored and the time-base counter is not loaded with the phase. • 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 (EPWMxSYNCI) or by a software forced synchronization.

Figure 3-79. Time-Base Phase High Resolution Register (TBPHSHR)

15 — 8
TBPHSHR
R/W-0

7 — 0
Reserved
R-0

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

Table 3-27. 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

Figure 3-80. Time-Base Counter Register (TBCTR)

15 — 0
TBCTR
R/W-0

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

Table 3-28. 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.

Figure 3-81. 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	PRDLD	PHSEN	CTRMODE
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 3-29. Time-Base Control Register (TBCTL) Field Descriptions

Bit	Field	Value	Description
15:14	FREE, SOFT		Emulation Mode Bits. These bits select the behavior of the ePWM time-base counter during emulation events:
		00	Stop after the next time-base counter increment or decrement
		01	Stop when counter completes a whole cycle: • Up-count mode: stop when the time-base counter = period (TBCTR = TBPRD) • Down-count mode: stop when the time-base counter = 0x0000 (TBCTR = 0x0000) • Up-down-count mode: stop when the time-base counter = 0x0000 (TBCTR = 0x0000)
		1X	Free run

Table 3-29. Time-Base Control Register (TBCTL) Field Descriptions (continued)

Bit	Field	Value	Description
13	PHSDIR		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.
		0	Count down after the synchronization event.
		1	Count up after the synchronization event.
12:10	CLKDIV		Time-base Clock Prescale Bits
			These bits determine part of the time-base clock prescale value. TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
		000	/1 (default on reset)
		001	/2
		010	/4
		011	/8
		100	/16
		101	/32
		110	/64
		111	/128
9:7	HSPCLKDIV		High Speed Time-base Clock Prescale Bits
			These bits determine part of the time-base clock prescale value. TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
			This divisor emulates the HSPCLK in the TMS320x281x system as used on the Event Manager (EV) peripheral.
		000	/1
		001	/2 (default on reset)
		010	/4
		011	/6
		100	/8
		101	/10
		110	/12
		111	/14
6	SWFSYNC		Software Forced Synchronization Pulse
		0	Writing a 0 has no effect and reads always return a 0.
		1	Writing a 1 forces a one-time synchronization pulse to be generated.
			This event is ORed with the EPWMxSYNCI input of the ePWM module.
			SWFSYNC is valid (operates) only when EPWMxSYNCI is selected by SYNCOSEL = 00.
5:4	SYNCOSEL		Synchronization Output Select. These bits select the source of the EPWMxSYNCO signal.
		00	EPWMxSYNC:
		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	PRDLD		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.

Table 3-29. Time-Base Control Register (TBCTL) Field Descriptions (continued)

Bit	Field	Value	Description
2	PHSEN		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 EPWMxSYNCI input signal occurs or when a software synchronization is forced by the SWFSYNC bit, or when a digital compare sync event occurs.
1:0	CTRMODE		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:
		00	Up-count mode
		01	Down-count mode
		10	Up-down-count mode
		11	Stop-freeze counter operation (default on reset)

Figure 3-82. Time-Base Status Register (TBSTS)

15–8
Reserved
R-0

7–3	2	1	0
Reserved	CTRMAX	SYNCI	CTRDIR
R-0	R/W1C-0	R/W1C-0	R-1

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

Table 3-30. Time-Base Status Register (TBSTS) Field Descriptions

Bit	Field	Value	Description
15:3	Reserved		Reserved
2	CTRMAX		Time-Base Counter Max Latched Status Bit
		0	Reading a 0 indicates the time-base counter never reached its maximum value. Writing a 0 will have no effect.
		1	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		Input Synchronization Latched Status Bit
		0	Writing a 0 will have no effect. Reading a 0 indicates no external synchronization event has occurred.
		1	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		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.

Figure 3-83. High Resolution Period Control Register (HRPCTL)

15–8
Reserved
R-0

7–3	2	1	0
Reserved	TBPHSHR LOADE	Reserved	HRPE
R-0	R/W-0	R-0	R/W-0

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

Table 3-31. High Resolution Period Control Register (HRPCTL) Field Descriptions

Bit	Field	Value	Description[(1)] [(2)]
15-3	Reserved		Reserved

(1) This register is EALLOW protected.
(2) This register is used with Type 1 ePWM modules (support high-resolution period) only.

Table 3-31. High Resolution Period Control Register (HRPCTL) Field Descriptions (continued)

Bit	Field	Value	Description[(1)] [(2)]
2	TBPHSHRLOADE		TBPHSHR Load Enable
			This bit allows you to synchronize ePWM modules with a high-resolution phase on a SYNCIN, TBCTL[SWFSYNC], or digital compare event. This allows for multiple ePWM modules operating at the same frequency to be phase aligned with high-resolution.
		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.
			Note: This bit and the TBCTL[PHSEN] bit must be set to 1 when high resolution period control is enabled for up-down count mode even if TBPHSHR = 0x0000.
1	Reserved		Reserved
0	HRPE		High Resolution Period Enable Bit
		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.

3.4.2 Counter-Compare Submodule Registers

Figure 3-84 through Figure 3-86 and Table 3-32 through Table 3-34 illustrate the counter-compare submodule control and status registers.

Figure 3-84. Counter-Compare A Register (CMPA)

15 … 0
CMPA
R/W-0

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

Table 3-32. Counter-Compare A Register (CMPA) Field Descriptions

Bits	Name	Description
15-0	CMPA	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: • Do nothing; the event is ignored. • Clear: Pull the EPWMxA and/or EPWMxB signal low • Set: Pull the EPWMxA and/or EPWMxB signal high • Toggle the EPWMxA and/or EPWMxB signal Shadowing of this register is enabled and disabled by the CMPCTL[SHDWAMODE] bit. By default this register is shadowed. • 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. • Before a write, the CMPCTL[SHDWAFULL] bit can be read to determine if the shadow register is currently full. • 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. • In either mode, the active and shadow registers share the same memory map address.

Figure 3-85. Counter-Compare B Register (CMPB)

15 … 0
CMPB
R/W-0

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

Table 3-33. Counter-Compare B Register (CMPB) Field Descriptions

Bits	Name	Description
15-0	CMPB	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: • Do nothing. event is ignored. • Clear: Pull the EPWMxA and/or EPWMxB signal low • Set: Pull the EPWMxA and/or EPWMxB signal high • Toggle the EPWMxA and/or EPWMxB signal Shadowing of this register is enabled and disabled by the CMPCTL[SHDWBMODE] bit. By default this register is shadowed. • 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: • Before a write, the CMPCTL[SHDWBFULL] bit can be read to determine if the shadow register is currently full. • 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. • In either mode, the active and shadow registers share the same memory map address.

Figure 3-86. Counter-Compare Control Register (CMPCTL)

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 3-34. Counter-Compare Control Register (CMPCTL) Field Descriptions

Bits	Name	Value	Description
15-10	Reserved		Reserved
9	SHDWBFULL		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		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		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		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		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)
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)

Figure 3-87. Compare A High Resolution Register (CMPAHR)

15	8
CMPAHR	
R/W-0	

7	0
Reserved	
R-0	

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

Table 3-35. Compare A High Resolution Register (CMPAHR) 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

Figure 3-88. Counter-Compare A Mirror Register (CMPAM)

15	0
CMPA	
R/W-0	

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

Table 3-36. Counter-Compare A Mirror Register (CMPAM) Field Descriptions

Bit	Field	Value	Description
15-0	CMPA	0000-FFFFh	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. 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. • 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. • Before a write, the CMPCTL[SHDWAFULL] bit can be read to determine if the shadow register is currently full. • 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.

Figure 3-89. Compare A High Resolution Mirror Register

15	8
CMPAHR	
R/W-0	

7	0
Reserved	
R-0	

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

Table 3-37. Compare A High-Resolution Mirror Register (CMPAHRM) 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

3.4.3 *Action-Qualifier Submodule Registers*

Figure 3-90 through Figure 3-93 and Table 3-38 through Table 3-41 provide the action-qualifier submodule register definitions.

Figure 3-90. Action-Qualifier Output A Control Register (AQCTLA)

15-12	11-10	9-8
Reserved	CBD	CBU
R-0	R/W-0	R/W-0

7-6	5-4	3-2	1-0
CAD	CAU	PRD	ZRO
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 3-38. Action-Qualifier Output A Control Register (AQCTLA) Field Descriptions

Bits	Name	Value	Description
15-12	Reserved		Reserved
11-10	CBD		Action when the time-base counter equals the active CMPB register and the counter is decrementing.
		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.
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 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.
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 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.
5-4	CAU		Action when the counter equals the active CMPA register and the counter is incrementing.
		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.

Table 3-38. Action-Qualifier Output A Control Register (AQCTLA) Field Descriptions (continued)

Bits	Name	Value	Description
3-2	PRD		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.
		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.
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.

Figure 3-91. Action-Qualifier Output B Control Register (AQCTLB)

15-12	11-10	9-8
Reserved	CBD	CBU
R-0	R/W-0	R/W-0

7-6	5-4	3-2	1-0
CAD	CAU	PRD	ZRO
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 3-39. 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.
5-4	CAU		Action when the counter equals the active CMPA 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.

Table 3-39. Action-Qualifier Output B Control Register (AQCTLB) Field Descriptions (continued)

Bits	Name	Value	Description
3-2	PRD		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.
		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.
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 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.

Figure 3-92. Action-Qualifier Software Force Register (AQSFRC)

15		8
	Reserved	
	R-0	

7	6	5	4	3	2	1	0
RLDCSF		OTSFB	ACTSFB		OTSFA	ACTSFA	
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 3-40. Action-Qualifier Software Force Register (AQSFRC) Field Descriptions

Bit	Field	Value	Description
15:8	Reserved		
7:6	RLDCSF		AQCSFRC Active Register Reload From Shadow Options
		00	Load on event counter equals zero
		01	Load on event counter equals period
		10	Load on event counter equals zero or counter equals period
		11	Load immediately (the active register is directly accessed by the CPU and is not loaded from the shadow register).
5	OTSFB		One-Time Software Forced Event on Output B
		0	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, i.e., a forced event is initiated.)
			This is a one-shot forced event. It can be overridden by another subsequent event on output B.
		1	Initiates a single s/w forced event
4:3	ACTSFB		Action when One-Time Software Force B Is invoked
		00	Does nothing (action disabled)
		01	Clear (low)
		10	Set (high)
		11	Toggle (Low -> High, High -> Low)
			Note: This action is not qualified by counter direction (CNT_dir)

Table 3-40. Action-Qualifier Software Force Register (AQSFRC) Field Descriptions (continued)

Bit	Field	Value	Description
2	OTSFA		One-Time Software Forced Event on Output A
		0	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 (i.e., a forced event is initiated).
		1	Initiates a single software forced event
1:0	ACTSFA		Action When One-Time Software Force A Is Invoked
		00	Does nothing (action disabled)
		01	Clear (low)
		10	Set (high)
		11	Toggle (Low → High, High → Low)
			Note: This action is not qualified by counter direction (CNT_dir)

Figure 3-93. Action-Qualifier Continuous Software Force Register (AQCSFRC)

15 - 8
Reserved
R-0

7 - 4	3 - 2	1 - 0
Reserved	CSFB	CSFA
R-0	R/W-0	R/W-0

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

Table 3-41. Action-qualifier Continuous Software Force Register (AQCSFRC) Field Descriptions

Bits	Name	Value	Description
15-4	Reserved		Reserved
3-2	CSFB		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].
		00	Forcing disabled, i.e., has no effect
		01	Forces a continuous low on output B
		10	Forces a continuous high on output B
		11	Software forcing is disabled and has no effect
1-0	CSFA		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.
		00	Forcing disabled, i.e., has no effect
		01	Forces a continuous low on output A
		10	Forces a continuous high on output A
		11	Software forcing is disabled and has no effect

3.4.4 Dead-Band Submodule Registers

Figure 3-94 through Table 3-44 provide the register definitions.

Figure 3-94. Dead-Band Generator Control Register (DBCTL)

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 3-42. Dead-Band Generator Control Register (DBCTL) Field Descriptions

Bits	Name	Value	Description
15	HALFCYCLE		Half Cycle Clocking Enable Bit:
		0	Full cycle clocking enabled. The dead-band counters are clocked at the TBCLK rate.
		1	Half cycle clocking enabled. The dead-band counters are clocked at TBCLK*2.
14-6	Reserved		Reserved
5-4	IN_MODE		Dead Band Input Mode Control
			Bit 5 controls the S5 switch and bit 4 controls the S4 switch shown in Figure 3-31.
			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.
		00	EPWMxA In (from the action-qualifier) is the source for both falling-edge and rising-edge delay.
		01	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.
		10	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.
		11	EPWMxB In (from the action-qualifier) is the source for both rising-edge delay and falling-edge delayed signal.
3-2	POLSEL		Polarity Select Control
			Bit 3 controls the S3 switch and bit 2 controls the S2 switch shown in Figure 3-31.
			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.
		00	Active high (AH) mode. Neither EPWMxA nor EPWMxB is inverted (default).
		01	Active low complementary (ALC) mode. EPWMxA is inverted.
		10	Active high complementary (AHC). EPWMxB is inverted.
		11	Active low (AL) mode. Both EPWMxA and EPWMxB are inverted.

Table 3-42. Dead-Band Generator Control Register (DBCTL) Field Descriptions (continued)

Bits	Name	Value	Description
1-0	OUT_MODE		Dead-band Output Mode Control
			Bit 1 controls the S1 switch and bit 0 controls the S0 switch shown in Figure 3-31.
			This allows you to selectively enable or bypass the dead-band generation for the falling-edge and rising-edge delay.
		00	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.
			In this mode, the POLSEL and IN_MODE bits have no effect.
		01	Disable rising-edge delay. The EPWMxA signal from the action-qualifier is passed straight through to the EPWMxA input of the PWM-chopper submodule.
			The falling-edge delayed signal is seen on output EPWMxB. The input signal for the delay is determined by DBCTL[IN_MODE].
		10	The rising-edge delayed signal is seen on output EPWMxA. The input signal for the delay is determined by DBCTL[IN_MODE].
			Disable falling-edge delay. The EPWMxB signal from the action-qualifier is passed straight through to the EPWMxB input of the PWM-chopper submodule.
		11	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].

Figure 3-95. Dead-Band Generator Rising Edge Delay Register (DBRED)

15-10	9-8
Reserved	DEL
R-0	R/W-0

7-0
DEL
R/W-0

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

Table 3-43. Dead-Band Generator Rising Edge Delay Register (DBRED) Field Descriptions

Bits	Name	Value	Description
15-10	Reserved		Reserved
9-0	DEL		Rising Edge Delay Count. 10-bit counter.

Figure 3-96. Dead-Band Generator Falling Edge Delay Register (DBFED)

15-10	9-8
Reserved	DEL
R-0	R/W-0

7-0
DEL
R/W-0

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

Table 3-44. Dead-Band Generator Falling Edge Delay Register (DBFED) Field Descriptions

Bits	Name	Description
15-10	Reserved	Reserved
9-0	DEL	Falling Edge Delay Count. 10-bit counter

3.4.5 PWM-Chopper Submodule Control Register

Figure 3-97 and Table 3-45 provide the definitions for the PWM-chopper submodule control register.

Figure 3-97. PWM-Chopper Control Register (PCCTL)

15-11	10-8
Reserved	CHPDUTY
R-0	R/W-0

7-5	4-1	0
CHPFREQ	OSHTWTH	CHPEN
R/W-0	R/W-0	R/W-0

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

Table 3-45. PWM-Chopper Control Register (PCCTL) Bit Descriptions

Bits	Name	Value	Description
15-11	Reserved		Reserved
10-8	CHPDUTY		Chopping Clock Duty Cycle
		000	Duty = 1/8 (12.5%)
		001	Duty = 2/8 (25.0%)
		010	Duty = 3/8 (37.5%)
		011	Duty = 4/8 (50.0%)
		100	Duty = 5/8 (62.5%)
		101	Duty = 6/8 (75.0%)
		110	Duty = 7/8 (87.5%)
		111	Reserved
7:5	CHPFREQ		Chopping Clock Frequency
		000	Divide by 1 (no prescale, = 11.25 MHz at 90 MHz SYSCLKOUT)
		001	Divide by 2 (5.63 MHz at 90 MHz SYSCLKOUT)
		010	Divide by 3 (3.75 MHz at 90 MHz SYSCLKOUT)
		011	Divide by 4 (2.81 MHz at 90 MHz SYSCLKOUT)
		100	Divide by 5 (2.25 MHz at 90 MHz SYSCLKOUT)
		101	Divide by 6 (1.88 MHz at 90 MHz SYSCLKOUT)
		110	Divide by 7 (1.61 MHz at 90 MHz SYSCLKOUT)
		111	Divide by 8 (1.41 MHz at 90 MHz SYSCLKOUT)
4:1	OSHTWTH		One-Shot Pulse Width
		0000	1 x SYSCLKOUT / 8 wide (= 80 nS at 100 MHz SYSCLKOUT)
		0001	2 x SYSCLKOUT / 8 wide (= 160 nS at 100 MHz SYSCLKOUT)
		0010	3 x SYSCLKOUT / 8 wide (= 240 nS at 100 MHz SYSCLKOUT)
		0011	4 x SYSCLKOUT / 8 wide (= 320 nS at 100 MHz SYSCLKOUT)
		0100	5 x SYSCLKOUT / 8 wide (= 400 nS at 100 MHz SYSCLKOUT)
		0101	6 x SYSCLKOUT / 8 wide (= 480 nS at 100 MHz SYSCLKOUT)
		0110	7 x SYSCLKOUT / 8 wide (= 560 nS at 100 MHz SYSCLKOUT)
		0111	8 x SYSCLKOUT / 8 wide (= 640 nS at 100 MHz SYSCLKOUT)
		1000	9 x SYSCLKOUT / 8 wide (= 720 nS at 100 MHz SYSCLKOUT)
		1001	10 x SYSCLKOUT / 8 wide (= 800 nS at 100 MHz SYSCLKOUT)
		1010	11 x SYSCLKOUT / 8 wide (= 880 nS at 100 MHz SYSCLKOUT)
		1011	12 x SYSCLKOUT / 8 wide (= 960 nS at 100 MHz SYSCLKOUT)
		1100	13 x SYSCLKOUT / 8 wide (= 1040 nS at 100 MHz SYSCLKOUT)
		1101	14 x SYSCLKOUT / 8 wide (= 1120 nS at 100 MHz SYSCLKOUT)
		1110	15 x SYSCLKOUT / 8 wide (= 1200 nS at 100 MHz SYSCLKOUT)
		1111	16 x SYSCLKOUT / 8 wide (= 1280 nS at 100 MHz SYSCLKOUT)

Table 3-45. PWM-Chopper Control Register (PCCTL) Bit Descriptions (continued)

Bits	Name	Value	Description
0	CHPEN		PWM-chopping Enable
		0	Disable (bypass) PWM chopping function
		1	Enable chopping function

3.4.6 Trip-Zone Submodule Control and Status Registers

Figure 3-98. 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 3-46. Trip-Zone Submodule Select Register (TZSEL) Field Descriptions

Bits	Name	Value	Description
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 (Figure 3-99) 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 (Figure 3-102).			
15	DCBEVT1		Digital Compare Output B Event 1 Select
		0	Disable DCBEVT1 as one-shot-trip source for this ePWM module.
		1	Enable DCBEVT1 as one-shot-trip source for this ePWM module.
14	DCAEVT1		Digital Compare Output A Event 1 Select
		0	Disable DCAEVT1 as one-shot-trip source for this ePWM module.
		1	Enable DCAEVT1 as one-shot-trip source for this ePWM module.
13	OSHT6		Trip-zone 6 ($\overline{\text{TZ6}}$) Select
		0	Disable $\overline{\text{TZ6}}$ as a one-shot trip source for this ePWM module.
		1	Enable $\overline{\text{TZ6}}$ as a one-shot trip source for this ePWM module.
12	OSHT5		Trip-zone 5 ($\overline{\text{TZ5}}$) Select
		0	Disable $\overline{\text{TZ5}}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{\text{TZ5}}$ as a one-shot trip source for this ePWM module
11	OSHT4		Trip-zone 4 ($\overline{\text{TZ4}}$) Select
		0	Disable $\overline{\text{TZ4}}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{\text{TZ4}}$ as a one-shot trip source for this ePWM module
10	OSHT3		Trip-zone 3 ($\overline{\text{TZ3}}$) Select
		0	Disable $\overline{\text{TZ3}}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{\text{TZ3}}$ as a one-shot trip source for this ePWM module
9	OSHT2		Trip-zone 2 ($\overline{\text{TZ2}}$) Select
		0	Disable $\overline{\text{TZ2}}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{\text{TZ2}}$ as a one-shot trip source for this ePWM module
8	OSHT1		Trip-zone 1 ($\overline{\text{TZ1}}$) Select
		0	Disable $\overline{\text{TZ1}}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{\text{TZ1}}$ as a one-shot trip source for this ePWM module
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 (Figure 3-99) is taken on the EPWMxA and EPWMxB outputs. A cycle-by-cycle trip condition is automatically cleared when the time-base counter reaches zero.			
7	DCBEVT2		Digital Compare Output B Event 2 Select
		0	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		Digital Compare Output A Event 2 Select
		0	Disable DCAEVT2 as a CBC trip source for this ePWM module
		1	Enable DCAEVT2 as a CBC trip source for this ePWM module

Table 3-46. Trip-Zone Submodule Select Register (TZSEL) Field Descriptions (continued)

Bits	Name	Value	Description
5	CBC6		Trip-zone 6 ($\overline{TZ6}$) Select
		0	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		Trip-zone 5 ($\overline{TZ5}$) Select
		0	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		Trip-zone 4 ($\overline{TZ4}$) Select
		0	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		Trip-zone 3 ($\overline{TZ3}$) Select
		0	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		Trip-zone 2 ($\overline{TZ2}$) Select
		0	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		Trip-zone 1 ($\overline{TZ1}$) Select
		0	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

Figure 3-99. 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	1-0
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 3-47. Trip-Zone Control Register Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Reserved
11-10	DCBEVT2		Digital Compare Output B Event 2 Action On EPWMxB:
		00	High-impedance (EPWMxB = High-impedance state)
		01	Force EPWMxB to a high state.
		10	Force EPWMxB to a low state.
		11	Do Nothing, trip action is disabled
9-8	DCBEVT1		Digital Compare Output B Event 1 Action On EPWMxB:
		00	High-impedance (EPWMxB = High-impedance state)
		01	Force EPWMxB to a high state.
		10	Force EPWMxB to a low state.
		11	Do Nothing, trip action is disabled
7-6	DCAEVT2		Digital Compare Output A Event 2 Action On EPWMxA:
		00	High-impedance (EPWMxA = High-impedance state)
		01	Force EPWMxA to a high state.
		10	Force EPWMxA to a low state.
		11	Do Nothing, trip action is disabled

Table 3-47. Trip-Zone Control Register Field Descriptions (continued)

Bit	Field	Value	Description
5-4	DCAEVT1		Digital Compare Output A Event 1 Action On EPWMxA:
		00	High-impedance (EPWMxA = High-impedance state)
		01	Force EPWMxA to a high state.
		10	Force EPWMxA to a low state.
		11	Do Nothing, trip action is disabled
3-2	TZB		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.
		00	High-impedance (EPWMxB = High-impedance state)
		01	Force EPWMxB to a high state
		10	Force EPWMxB to a low state
		11	Do nothing, no action is taken on EPWMxB.
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.

Figure 3-100. Trip-Zone Enable Interrupt Register (TZEINT)

15–8
Reserved
R -0

7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	Reserved
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 3-48. Trip-Zone Enable Interrupt Register (TZEINT) Field Descriptions

Bits	Name	Value	Description
15-3	Reserved		Reserved
6	DCBEVT2		Digital Comparator Output B Event 2 Interrupt Enable
		0	Disabled
		1	Enabled
5	DCBEVT1		Digital Comparator Output B Event 1 Interrupt Enable
		0	Disabled
		1	Enabled
4	DCAEVT2		Digital Comparator Output A Event 2 Interrupt Enable
		0	Disabled
		1	Enabled
3	DCAEVT1		Digital Comparator Output A Event 1 Interrupt Enable
		0	Disabled
		1	Enabled
2	OST		Trip-zone One-Shot Interrupt Enable
		0	Disable one-shot interrupt generation
		1	Enable Interrupt generation; a one-shot trip event will cause a EPWMx_TZINT PIE interrupt.
1	CBC		Trip-zone Cycle-by-Cycle Interrupt Enable
		0	Disable cycle-by-cycle interrupt generation.

Table 3-48. Trip-Zone Enable Interrupt Register (TZEINT) Field Descriptions (continued)

Bits	Name	Value	Description
		1	Enable interrupt generation; a cycle-by-cycle trip event will cause an EPWMx_TZINT PIE interrupt.
0	Reserved		Reserved

Figure 3-101. Trip-Zone Flag Register (TZFLG)

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	INT
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 3-49. Trip-Zone Flag Register Field Descriptions

Bit	Field	Value	Description
15:7	Reserved		Reserved
6	DCBEVT2		Latched Status Flag for Digital Compare Output B Event 2
		0	Indicates no trip event has occurred on DCBEVT2
		1	Indicates a trip event has occurred for the event defined for DCBEVT2
5	DCBEVT1		Latched Status Flag for Digital Compare Output B Event 1
		0	Indicates no trip event has occurred on DCBEVT1
		1	Indicates a trip event has occurred for the event defined for DCBEVT1
4	DCAEVT2		Latched Status Flag for Digital Compare Output A Event 2
		0	Indicates no trip event has occurred on DCAEVT2
		1	Indicates a trip event has occurred for the event defined for DCAEVT2
3	DCAEVT1		Latched Status Flag for Digital Compare Output A Event 1
		0	Indicates no trip event has occurred on DCAEVT1
		1	Indicates a trip event has occurred for the event defined for DCAEVT1
2	OST		Latched Status Flag for A One-Shot Trip Event
		0	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		Latched Status Flag for Cycle-By-Cycle Trip Event
		0	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 .
0	INT		Latched Trip Interrupt Status Flag
		0	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 .

Figure 3-102. Trip-Zone Clear Register (TZCLR)

15–8
Reserved
R-0

7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	INT
R-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W-0	R/W-0	R/W-0

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

Table 3-50. Field Descriptions

Bit	Field	Value	Description
15-7	Reserved		Reserved
6	DCBEVT2		Clear Flag for Digital Compare Output B Event 2
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCBEVT2 event trip condition.
5	DCBEVT1		Clear Flag for Digital Compare Output B Event 1
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCBEVT1 event trip condition.
4	DCAEVT2		Clear Flag for Digital Compare Output A Event 2
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCAEVT2 event trip condition.
3	DCAEVT1		Clear Flag for Digital Compare Output A Event 1
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCAEVT1 event trip condition.
2	OST		Clear Flag for One-Shot Trip (OST) Latch
		0	Has no effect. Always reads back a 0.
		1	Clears this Trip (set) condition.
1	CBC		Clear Flag for Cycle-By-Cycle (CBC) Trip Latch
		0	Has no effect. Always reads back a 0.
		1	Clears this Trip (set) condition.
0	INT		Global Interrupt Clear Flag
		0	Has no effect. Always reads back a 0.
		1	Clears the trip-interrupt flag for this ePWM module (TZFLG[INT]).
			NOTE: 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.

Figure 3-103. Trip-Zone Force Register (TZFRC)

15–8
Reserved
R-0

7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	Reserved
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R- 0

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

Table 3-51. Trip-Zone Force Register (TZFRC) Field Descriptions

Bits	Name	Value	Description
15- 7	Reserved		Reserved
6	DCBEVT2		Force Flag for Digital Compare Output B Event 2

Table 3-51. Trip-Zone Force Register (TZFRC) Field Descriptions (continued)

Bits	Name	Value	Description
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 forces the DCBEVT2 event trip condition and sets the TZFLG[DCBEVT2] bit.
5	DCBEVT1		Force Flag for Digital Compare Output B Event 1
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 forces the DCBEVT1 event trip condition and sets the TZFLG[DCBEVT1] bit.
4	DCAEVT2		Force Flag for Digital Compare Output A Event 2
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 forces the DCAEVT2 event trip condition and sets the TZFLG[DCAEVT2] bit.
3	DCAEVT1		Force Flag for Digital Compare Output A Event 1
		0	Writing 0 has no effect. This bit always reads back 0
		1	Writing 1 forces the DCAEVT1 event trip condition and sets the TZFLG[DCAEVT1] bit.
2	OST		Force a One-Shot Trip Event via Software
		0	Writing of 0 is ignored. Always reads back a 0.
		1	Forces a one-shot trip event and sets the TZFLG[OST] bit.
1	CBC		Force a Cycle-by-Cycle Trip Event via Software
		0	Writing of 0 is ignored. Always reads back a 0.
		1	Forces a cycle-by-cycle trip event and sets the TZFLG[CBC] bit.
0	Reserved		Reserved

Figure 3-104. 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

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

Table 3-52. Trip Zone Digital Compare Event Select Register (TZDCSEL) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Reserved
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
		111	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
		111	reserved

Table 3-52. Trip Zone Digital Compare Event Select Register (TZDCSEL) Field Descriptions (continued)

Bit	Field	Value	Description
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
		111	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

3.4.7 Digital Compare Submodule Registers

Figure 3-105. Digital Compare Trip Select (DCTRIPSEL)

15 – 12	11 – 8
DCBLCOMPSEL	DCBHCOMPSEL
R/W-0	R/W-0

7 – 4	3 – 0
DCALCOMPSEL	DCAHCOMPSEL
R/W-0	R/W-0

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

Table 3-53. Digital Compare Trip Select (DCTRIPSEL) Field Descriptions

Bit	Field	Value	Description
15-12	DCBLCOMPSEL		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.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.
11-8	DCBHCOMPSEL		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.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.
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
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.

Table 3-53. Digital Compare Trip Select (DCTRIPSEL) Field Descriptions (continued)

Bit	Field	Value	Description
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
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.

Figure 3-106. Digital Compare A Control Register (DCACTL)

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	EVT1SRCSEL
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 3-54. Digital Compare A Control Register (DCACTL) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved		Reserved
9	EVT2FRC SYNCSEL		DCAEVT2 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
8	EVT2SRCSEL		DCAEVT2 Source Signal Select
		0	Source Is DCAEVT2 Signal
		1	Source Is DCEVTFILT Signal
7-4	Reserved		Reserved
3	EVT1SYNCE		DCAEVT1 SYNC, Enable/Disable
		0	SYNC Generation Disabled
		1	SYNC Generation Enabled
2	EVT1SOCE		DCAEVT1 SOC, Enable/Disable
		0	SOC Generation Disabled
		1	SOC Generation Enabled
1	EVT1FRC SYNCSEL		DCAEVT1 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
0	EVT1SRCSEL		DCAEVT1 Source Signal Select
		0	Source Is DCAEVT1 Signal
		1	Source Is DCEVTFILT Signal

Figure 3-107. Digital Compare B Control Register (DCBCTL)

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	EVT1SRCSEL
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 3-55. Digital Compare B Control Register (DCBCTL) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved		Reserved
9	EVT2FRC SYNCSEL		DCBEVT2 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
8	EVT2SRCSEL		DCBEVT2 Source Signal Select
		0	Source Is DCBEVT2 Signal
		1	Source Is DCEVTFILT Signal
7-4	Reserved		Reserved
3	EVT1SYNCE		DCBEVT1 SYNC, Enable/Disable
		0	SYNC Generation Disabled
		1	SYNC Generation Enabled
2	EVT1SOCE		DCBEVT1 SOC, Enable/Disable
		0	SOC Generation Disabled
		1	SOC Generation Enabled
1	EVT1FRC SYNCSEL		DCBEVT1 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
0	EVT1SRCSEL		DCBEVT1 Source Signal Select
		0	Source Is DCBEVT1 Signal
		1	Source Is DCEVTFILT Signal

Figure 3-108. Digital Compare Filter Control Register (DCFCTL)

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 3-56. Digital Compare Filter Control Register (DCFCTL) 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

Table 3-56. Digital Compare Filter Control Register (DCFCTL) Field Descriptions (continued)

Bit	Field	Value	Description
5-4	PULSESEL		Pulse Select For Blanking & Capture Alignment
		00	Time-base counter equal to period (TBCTR = TBPRD)
		01	Time-base counter equal to zero (TBCTR = 0x0000)
		10	Reserved
		11	Reserved
3	BLANKINV		Blanking Window Inversion
		0	Blanking window not inverted
		1	Blanking window inverted
2	BLANKE		Blanking Window Enable/Disable
		0	Blanking window is disabled
		1	Blanking window is enabled
1-0	SRCSEL		Filter Block Signal Source Select
		00	Source Is DCAEVT1 Signal
		01	Source Is DCAEVT2 Signal
		10	Source Is DCBEVT1 Signal
		11	Source Is DCBEVT2 Signal

Figure 3-109. Digital Compare Capture Control Register (DCCAPCTL)

15-8
Reserved
R-0

7-2	1	0
Reserved	SHDWMODE	CAPE
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 Capture Control Register (DCCAPCTL) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved		Reserved
1	SHDWMODE		TBCTR Counter Capture Shadow Select Mode
		0	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		TBCTR Counter Capture Enable/Disable
		0	Disable the time-base counter capture.
		1	Enable the time-base counter capture.

Figure 3-110. Digital Compare Counter Capture Register (DCCAP)

15-0
DCCAP
R-0

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

Table 3-58. Digital Compare Counter Capture Register (DCCAP) Field Descriptions

Bit	Field	Value	Description
15-0	DCCAP	0000-FFFFh	Digital Compare Time-Base Counter Capture To enable time-base counter capture, set the DCCAPCLT[CAPE] bit to 1. 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. Shadowing of DCCAP is enabled and disabled by the DCCAPCTL[SHDWMODE] bit. By default this register is shadowed. • 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. • If DCCAPCTL[SHDWMODE] = 1, then the shadow register is disabled. In this mode, CPU reads will return the active register value. The active and shadow registers share the same memory map address.

Figure 3-111. Digital Compare Filter Offset Register (DCFOFFSET)

15 0
DCOFFSET
R-0

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

Table 3-59. Digital Compare Filter Offset Register (DCFOFFSET) Field Descriptions

Bit	Field	Value	Description
15-0	OFFSET	0000- FFFFh	Blanking Window Offset 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. 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.

Figure 3-112. Digital Compare Filter Offset Counter Register (DCFOFFSETCNT)

15 0
OFFSETCNT
R-0

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

Table 3-60. Digital Compare Filter Offset Counter Register (DCFOFFSETCNT) Field Descriptions

Bit	Field	Value	Description
15-0	OFFSETCNT	0000- FFFFh	Blanking Offset Counter 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. 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 a emulation stop.

Figure 3-113. Digital Compare Filter Window Register (DCFWINDOW)

15 — 8
Reserved
R-0

7 — 0
WINDOW
R/W-0

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

Table 3-61. Digital Compare Filter Window Register (DCFWINDOW) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		Reserved
7-0	WINDOW		Blanking Window Width
		00h	No blanking window is generated.
		01-FFh	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.

Figure 3-114. Digital Compare Filter Window Counter Register (DCFWINDOWCNT)

15 — 8
Reserved
R-0

7 — 0
WINDOWCNT
R-0

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

Table 3-62. Digital Compare Filter Window Counter Register (DCFWINDOWCNT) 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 *Event-Trigger Submodule Registers*

Figure 3-115 through Figure 3-119 and Table 3-63 through Table 3-67 describe the registers for the event-trigger submodule.

Figure 3-115. 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 3-63. Event-Trigger Selection Register (ETSEL) Field Descriptions

Bits	Name	Value	Description
15	SOCBEN		Enable the ADC Start of Conversion B (EPWMxSOCB) Pulse
		0	Disable EPWMxSOCB.
		1	Enable EPWMxSOCB pulse.
14-12	SOCBSEL		EPWMxSOCB Selection Options
			These bits determine when a EPWMxSOCB pulse will be generated.
		000	Enable DCBEVT1.soc event
		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.
11	SOCAEN		Enable the ADC Start of Conversion A (EPWMxSOCA) Pulse
		0	Disable EPWMxSOCA.
		1	Enable EPWMxSOCA pulse.
10-8	SOCASEL		EPWMxSOCA Selection Options
			These bits determine when a EPWMxSOCA pulse will be generated.
		000	Enable DCAEVT1.soc event
		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.
7-4	Reserved		Reserved
3	INTEN		Enable ePWM Interrupt (EPWMx_INT) Generation
		0	Disable EPWMx_INT generation
		1	Enable EPWMx_INT generation
2-0	INTSEL		ePWM Interrupt (EPWMx_INT) Selection Options
		000	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.

Figure 3-116. 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 3-64. 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
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[SOCAEN] = 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.

Table 3-64. Event-Trigger Prescale Register (ETPS) Field Descriptions (continued)

Bits	Name		Description
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)

Figure 3-117. Event-Trigger Flag Register (ETFLG)

15 - 8
Reserved
R-0

7 - 4	3	2	1	0
Reserved	SOCB	SOCA	Reserved	INT
R-0	R-0	R-0	R-0	R-0

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

Table 3-65. Event-Trigger Flag Register (ETFLG) Field Descriptions

Bits	Name	Value	Description
15-4	Reserved		Reserved
3	SOCB		Latched ePWM ADC Start-of-Conversion B (EPWMxSOCB) Status Flag
		0	Indicates no EPWMxSOCB event occurred
		1	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		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.
		0	Indicates no event occurred
		1	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		Latched ePWM Interrupt (EPWMx_INT) Status Flag
		0	Indicates no event occurred
		1	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. Refer to Figure 3-44.

Figure 3-118. Event-Trigger Clear Register (ETCLR)

15 – 8
Reserved
R = 0

7 – 4	3	2	1	0
Reserved	SOCB	SOCA	Reserved	INT
R-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 3-66. Event-Trigger Clear Register (ETCLR) Field Descriptions

Bits	Name	Value	Description
15-4	Reserved		Reserved
3	SOCB		ePWM ADC Start-of-Conversion B (EPWMxSOCB) Flag Clear Bit
		0	Writing a 0 has no effect. Always reads back a 0
		1	Clears the ETFLG[SOCB] flag bit
2	SOCA		ePWM ADC Start-of-Conversion A (EPWMxSOCA) Flag Clear Bit
		0	Writing a 0 has no effect. Always reads back a 0
		1	Clears the ETFLG[SOCA] flag bit
1	Reserved		Reserved
0	INT		ePWM Interrupt (EPWMx_INT) Flag Clear Bit
		0	Writing a 0 has no effect. Always reads back a 0
		1	Clears the ETFLG[INT] flag bit and enable further interrupts pulses to be generated

Figure 3-119. Event-Trigger Force Register (ETFRC)

15 – 8
Reserved
R-0

7 – 4	3	2	1	0
Reserved	SOCB	SOCA	Reserved	INT
R-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 3-67. Event-Trigger Force Register (ETFRC) Field Descriptions

Bits	Name	Value	Description
15-4	Reserved		Reserved
3	SOCB		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		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		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.

3.4.9 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

Chapter 4

SPRUH18E – January 2011 – Revised March 2014

High-Resolution Pulse Width Modulator (HRPWM)

This document is used in conjunction with the device-specific *Enhanced Pulse Width Modulator (ePWM) Module Reference Guide.* The HRPWM module described in this reference guide is a Type 1 HRPWM. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566) 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:

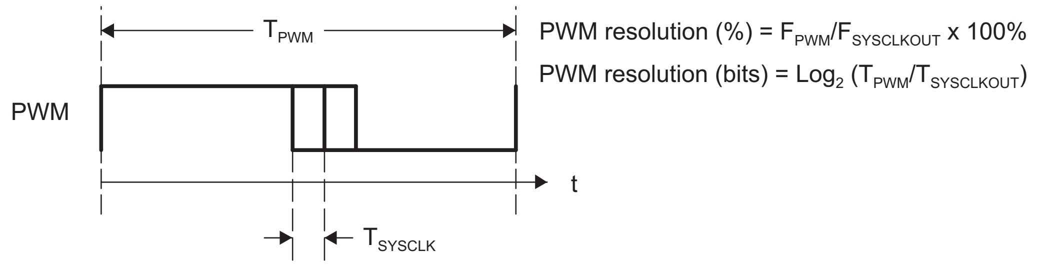
- Extended time resolution capability
- Used in both duty cycle and phase-shift control methods
- Finer time granularity control or edge positioning using extensions to the Compare A and Phase registers
- Implemented using the A signal path of PWM, i.e., on the EPWMxA output.
- Self-check diagnostics 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 +/- 1-2 cycle jitter in this mode.

Topic **Page**

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


T_{PWM}
PWM
t
T_{SYSCLK}
PWM resolution (%) = $F_{PWM}/F_{SYSCLKOUT}$ x 100%
PWM resolution (bits) = Log_2 $(T_{PWM}/T_{SYSCLKOUT})$

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 the device-specific datasheet for typical and maximum performance specifications for the MEP.

Table 4-1. Resolution for PWM and HRPWM

PWM Freq	Regular Resolution (PWM)		High Resolution (HRPWM)	
	90 MHz SYSCLKOUT			
(kHz)	Bits	%	Bits	%
20	12.1	0.0	18.1	0.000
50	10.8	0.1	16.8	0.001
100	9.8	0.1	15.8	0.002
150	9.2	0.2	15.2	0.003
200	8.8	0.2	14.8	0.004
250	8.5	0.3	14.4	0.005
500	7.5	0.6	13.4	0.009
1000	6.5	1.1	12.4	0.018
1500	5.9	1.7	11.9	0.027
2000	5.5	2.2	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:

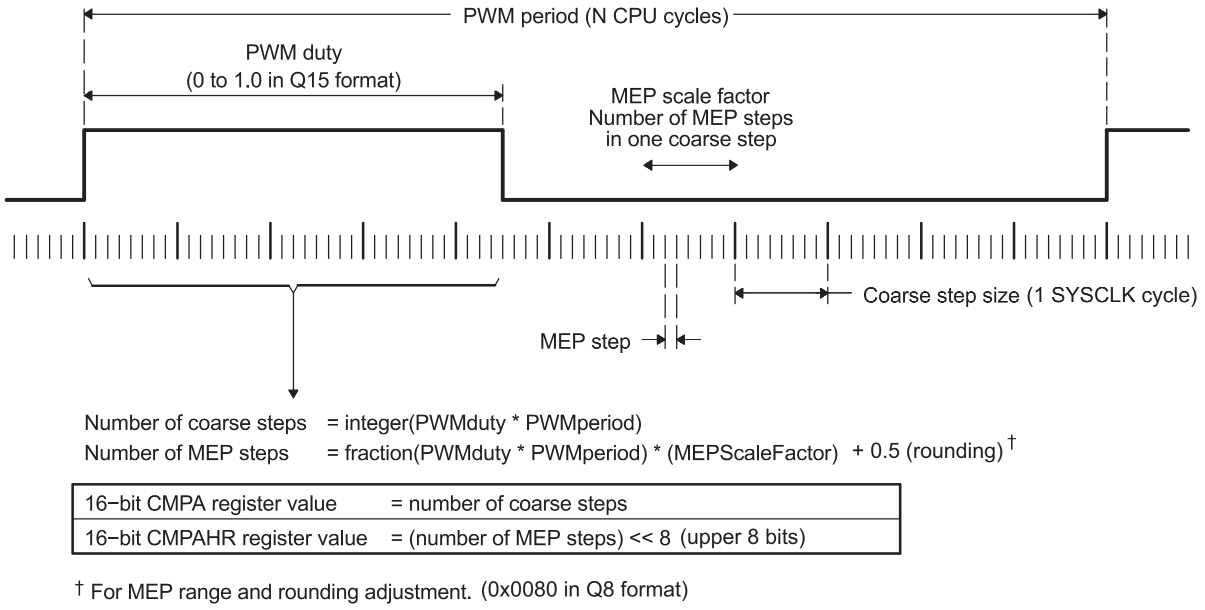
- Single-phase buck, boost, and flyback
- Multi-phase buck, boost, and flyback
- Phase-shifted full bridge
- Direct modulation of D-Class power amplifiers

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 via an 8-bit field in the Compare A extension register (CMPAHR).

Figure 4-2. Operating Logic Using MEP


PWM period (N CPU cycles)
PWM duty
(0 to 1.0 in Q15 format)
MEP scale factor
Number of MEP steps
in one coarse step
Coarse step size (1 SYSCLK cycle)
MEP step
Number of coarse steps = integer(PWMduty * PWMperiod)
Number of MEP steps = fraction(PWMduty * PWMperiod) * (MEPScaleFactor) + 0.5 (rounding) †
16−bit CMPA register value = number of coarse steps
16−bit CMPAHR register value = (number of MEP steps) << 8 (upper 8 bits)
† For MEP range and rounding adjustment. (0x0080 in Q8 format)

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 document can be seen in the device-specific *Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*.

The HRPWM operation is controlled and monitored using the following registers:

Table 4-2. HRPWM Registers

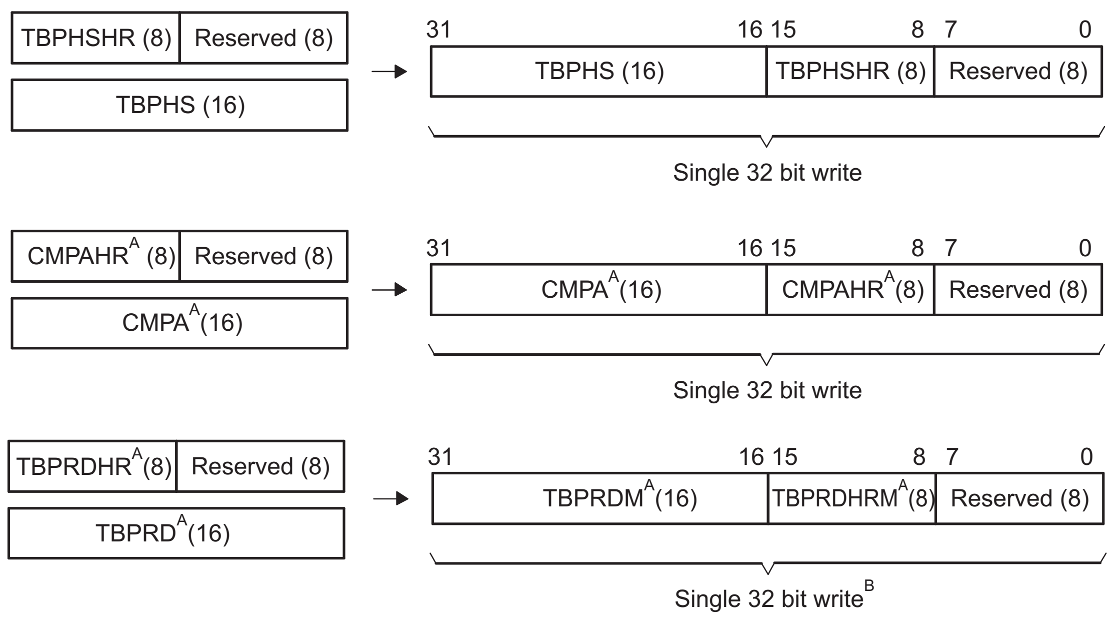
mnemonic	Address Offset	Shadowed	Description
TBPHSHR	0x0002	No	Extension Register for HRPWM Phase (8 bits)
TBPRDHR	0x0006	Yes	Extension Register for HRPWM Period (8 bits)
CMPAHR	0x0008	Yes	Extension Register for HRPWM Duty (8 bits)
HRCNFG	0x0020	No	HRPWM Configuration Register
HRMSTEP	0x0026	No	HRPWM MEP Step 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.

- 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)

Figure 4-3. HRPWM Extension Registers and Memory Configuration



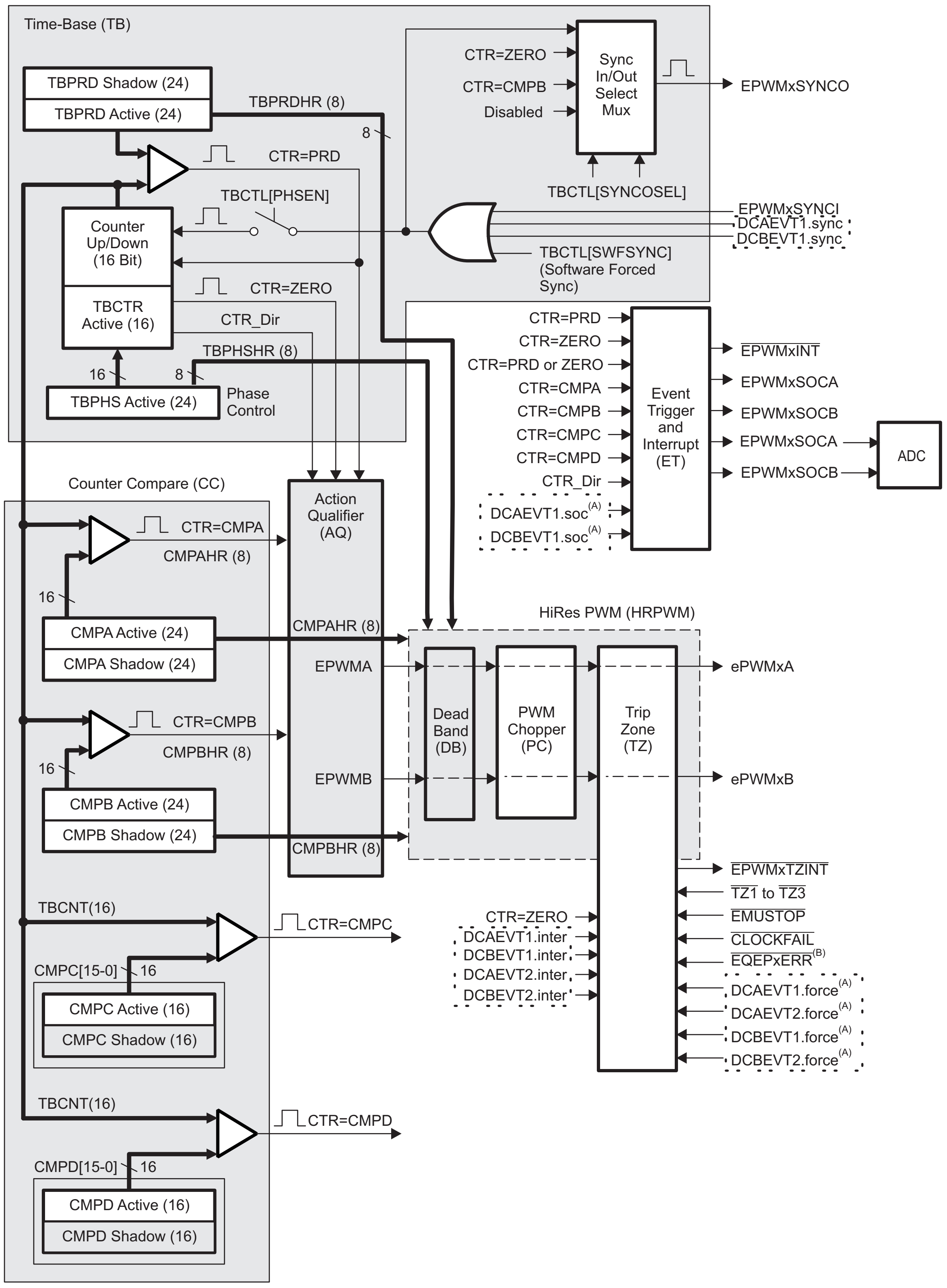
A These registers are mirrored and can be written to at two different memory locations (mirrored registers have an "M" suffix (i.e. CMPA mirror = CMPAM). Reads of the high-resolution mirror registers will result in indeterminate values.

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

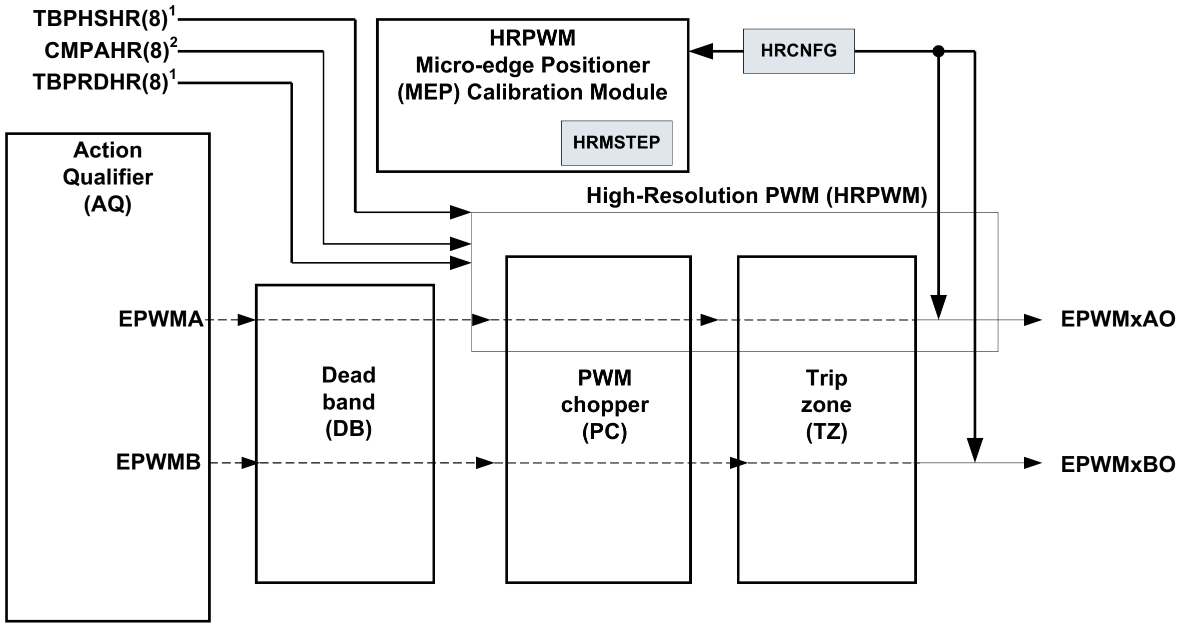
HRPWM capabilities are controlled using the Channel A PWM signal path. HRPWM support on the channel B signal path is available by properly configuring the HRCNFG register. Figure 4-4 shows how the HRPWM interfaces with the 8-bit extension registers.

Figure 4-4. HRPWM System Interface



A These events are generated by the type 1 ePWM digital compare (DC) submodule based on the levels of the COMPxOUT and $\overline{\text{TZ}}$ signals.

Figure 4-5. HRPWM Block Diagram



(1) From ePWM Time-base (TB) submodule

(2) From ePWM counter-compare (CC) submodule

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, e.g., 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 = fraction(PWMduty*PWMperiod)<<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 = (fraction(PWMduty * PWMperiod) * MEP Scale Factor + 0.5)<<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 [1][2][3]	PWM MIN (Hz) [4]	PWM MAX (MHz)	Res. @ MAX (Bits) [5]
50.0	111	763	2.50	11.1
60.0	93	916	3.00	10.9
70.0	79	1068	3.50	10.6
80.0	69	1221	4.00	10.4
90.0	62	1373	4.50	10.3
100.0	56	1526	5.00	10.1

(1) System frequency = SYSCLKOUT, i.e., 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, i.e., TBPRD = 65535. PWM mode is asymmetrical up-count.
(5) Resoluton 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 0.300 or 30.0% on time and the required converter PWM frequency is 1.25 MHz. In conventional PWM generation with a system clock of 90 MHz, the duty cycle choices are in the vicinity of 30.0%. In Figure 4-6, a compare value of 22 counts (that is, duty = 30.6%) is the closest to 30.0% that you can attain. This is equivalent to an edge position of 244.4 ns instead of the desired 240.0 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 240 ns. Table 4-4 shows that in addition to the CMPA value of 21 (that is, duty = 29.2% and edge positioning at 233.3 ns), 37 steps of the MEP (CMPAHR register) will position the edge at 239.96 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%

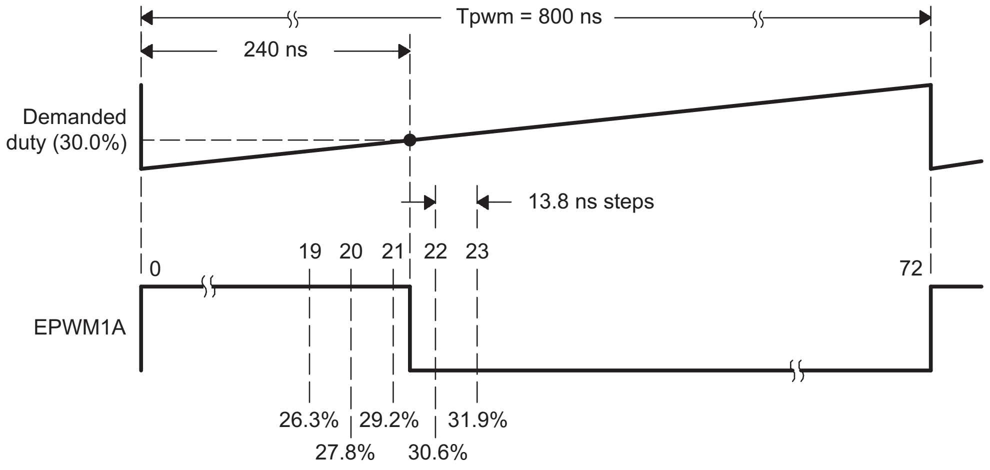


Table 4-4. CMPA vs Duty (left), and [CMPA:CMPAHR] vs Duty (right)

CMPA (count)[1] [2] [3]	Duty %	High Time (ns)	CMPA (count)	CMPAHR (count)	Duty (%)	High Time (ns)
17	23.61%	188.9	21	31	29.86%	238.88
18	25.0%	200.0	21	32	29.88%	239.06
19	26.39%	211.1	21	33	29.91%	239.24
20	27.78%	222.2	21	34	29.93%	239.42
21	29.17%	233.4	21	35	29.95%	239.60
22	30.56%	244.5	21	36	29.97%	239.78
23	31.94%	255.5	21	37	30.00%	239.96
24	33.33%	266.6	21	38	30.02%	240.14
25	34.72%	277.8	21	39	30.04%	240.32
Required			21	40	30.06%	240.50
21.6	30.0%	240.0	21	41	30.09%	240.68

[1] System clock, SYSCLKOUT and TBCLK = 90 MHz, 11.1 ns
[2] For a PWM Period register value of 72 counts, PWM Period =72 x 11.1 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	=	11.1 ns (90 MHz)
PWM frequency	=	1.25 MHz (1/800 ns)
Required PWM duty cycle, **PWMDuty**	=	0.300 (30.0%)
PWM period in terms of coarse steps, **PWMperiod** (800 ns/ 11.1 ns)	=	72
Number of MEP steps per coarse step at 180 ps (11.1 ns/ 180 ps), **MEP_ScaleFactor**	=	61
Value to keep CMPAHR within the range of 1-255 and fractional rounding constant (default value). In the event that frac(**PWMDuty * PWMperiod**) * MEP_ScaleFactor results in a value with a decimal portion ≥ 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	=	int(**PWMDuty***PWMperiod**); int means integer part
	=	int(0.300* 72)
CMPA register value	=	21 (15h)

Step 2: Fractional value conversion for CMPAHR register

CMPAHR register value	=	(frac(**PWMDuty***PWMperiod**)***MEP_ScaleFactor**+0.5) << 8; frac means fractional part
	=	(frac(21.6)* 72+ 0.5) <<8; Shift is to move the value as CMPAHR high byte
	=	((0.6 * 72 + 0.5) << 8)
	=	(43.2 + 0.5) <<8
	=	43.7 * 256 ; Shifting left by 8 is the same as multiplying by 256.
	=	11,187
CMPAHR value	=	CMPAHR value = 2BB3h; 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 = frac (PWMDuty*PWMperiod<<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, i.e., [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.

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. To better understand the useable duty cycle range, see Table 4-5. 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.

Figure 4-7. Low % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0)

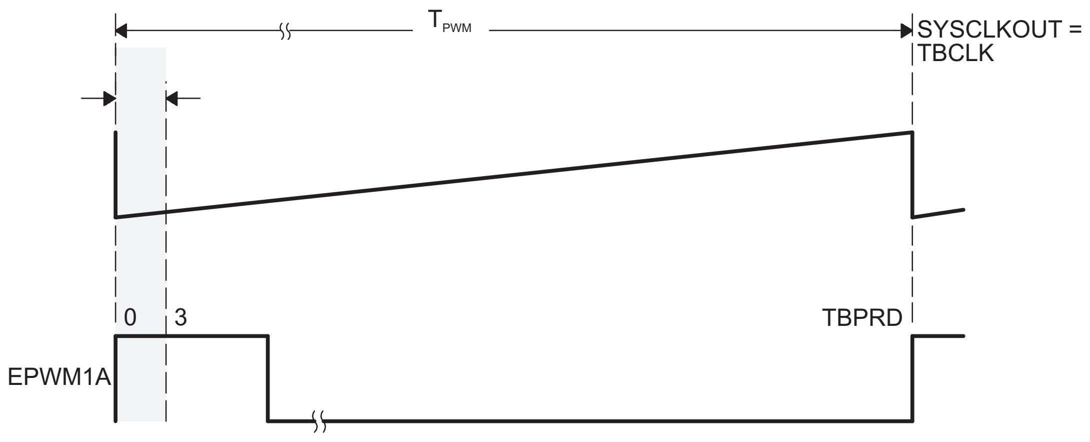


Table 4-5. Duty Cycle Range Limitation for 3 SYSCLK/TBCLK Cycles

PWM Frequency [(1)] (kHz)	3 Cycles Minimum Duty	3 Cycles Maximum Duty[(2)]
200	0.67%	99.33%
400	1.33%	98.67%
600	2.00%	98.00%
800	2.67%	97.33%
1000	3.33%	97.67%
1200	4.00%	96.00%
1400	4.67%	95.33%
1600	5.33%	95.67%
1800	6.00%	94.00%
2000	6.67%	93.33%

(1) System clock - $T_{SYSCLKOUT}$ = 11.1 ns System clock = TBCLK = 90 MHz
(2) This limitation applies only if high-resolution period (TBPRDHR) control is enabled.

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)

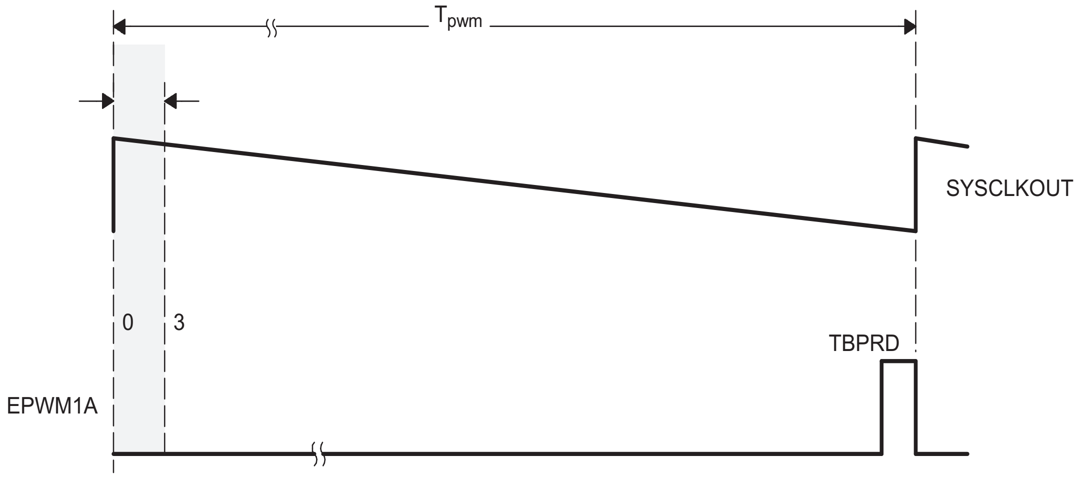


Figure 4-9. Up-Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)

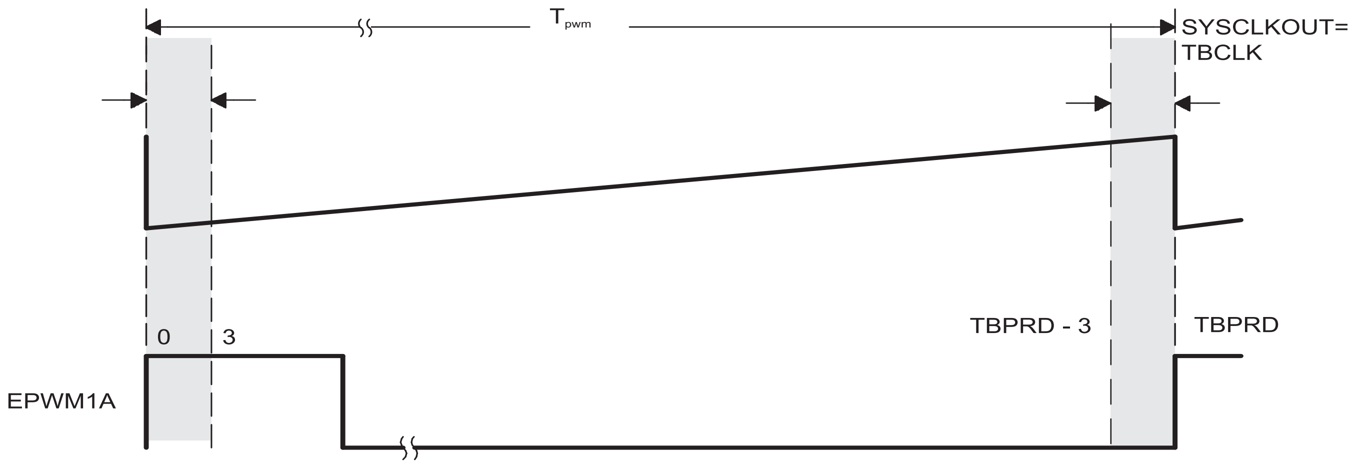
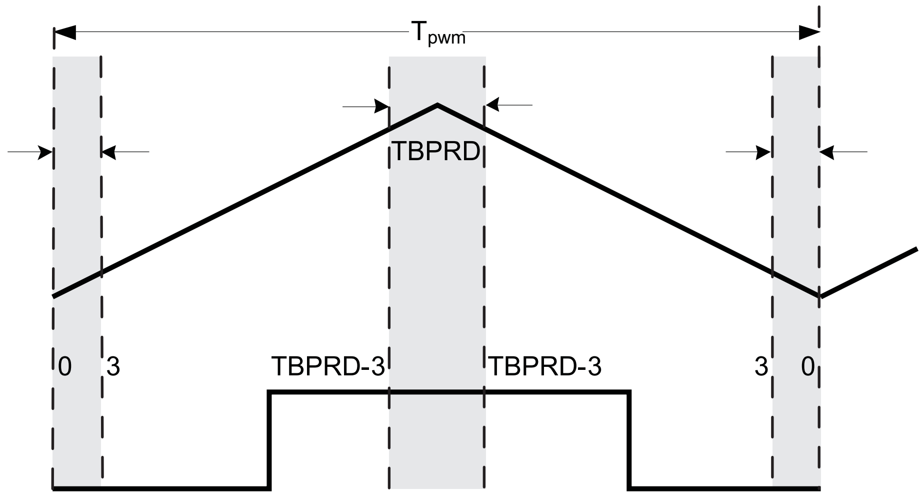


Figure 4-10. Up-Down Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)



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	= 11.1 ns (90 MHz)
Required PWM frequency	= 175 kHz (TBPRD value of 514.286)
Number of MEP steps per coarse step at 180 ps (MEP_ScaleFactor)	= 61 (11.1 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 = 514, then PWM frequency = 174.75 kHz (period = (514+1)* T_{TBCLK}).

TBPRD = 513, then PWM frequency = 175.10 kHz (period = (513+1)* T_{TBCLK}).

In up-down count mode:

If TBPRD = 258, then PWM frequency = 174.42 kHz (period = (258*2)* T_{TBCLK}).

If TBPRD = 257, then PWM frequency = 175.10 kHz (period = (257*2)* T_{TBCLK}).

Solution:

With 61 MEP steps per coarse step at 180 ps each:

Step 1: Percentage Integer Period value conversion for TBPRD register

Integer period value	= 514 * T_{TBCLK}
	= int (514.286) * T_{TBCLK}
	= int (PWMperiod) * T_{TBCLK}
In up-count mode:	
TBPRD register value	= 513 (TBPRD = period value - 1)
	= 0201h
In up-down count mode:	= 257 (TBPRD = period value / 2)
TBPRD register value	= 0101h

Step 2: Fractional value conversion for TBPRDHR register

TBPRDHR register value	= (frac(PWMperiod) * MEP_ScaleFactor + 0.5) (shift is to move the value as TBPRDHR high byte)
If auto-conversion enabled and HRMSTEP =	
MEP_ScaleFactor value (61):	=frac (PWMperiod)<<8
TBPRDHR register value	=frac (514.286)<<8
	=0.286 × 256
	=0049h

The autoconversion will then automatically perform the calculation such that TBPRDHR MEP delay is scaled by hardware to:	=((TBPRDHR(15:0) >> 8) × HRMSTEP + 80h)>>8
	= (0049h × 61 + 80h) >> 8
	=(11E5h) >> 8
Period MEP delay	=0011h 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 4.4.
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 (EPWMxSYNCI 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 4.4.

4.2.5 HRPWM Examples Using Optimized Assembly Code.

The best way to understand how to use the HRPWM capabilities is through 2 real examples:

1. Simple buck converter using asymmetrical PWM (i.e. 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, #defines introduced in the device-specific *Pulse Width Modulator (ePWM) Module Reference Guide* are also used.

Example 4-1 This example assumes 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 = 80 MHz are:

- PWM frequency = 800 kHz (i.e., 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

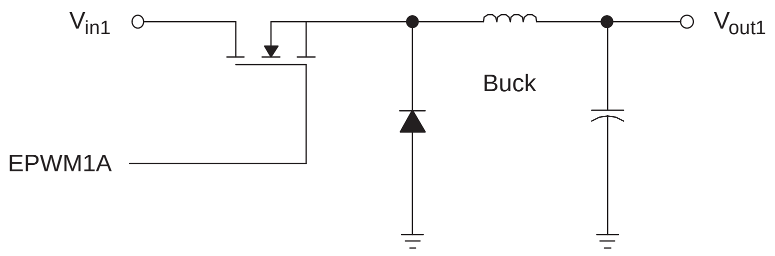
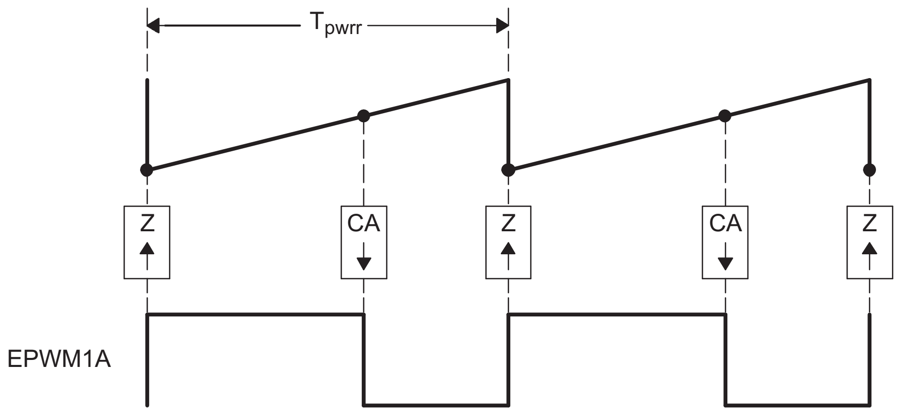


Figure 4-12. PWM Waveform Generated for Simple Buck Controlled Converter



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 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.PRDLD = TB_IMMEDIATE;          // set Immediate load
EPwm1Regs.TBPRD = 100;                             // Period set for 800 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.SYNCOSEL = 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 = 83*256;                         // Start with typical Scale Factor
                                                   // value for 80 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 = 533 kHz (i.e. 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

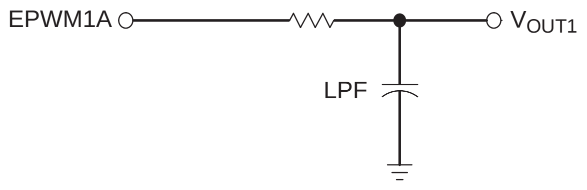
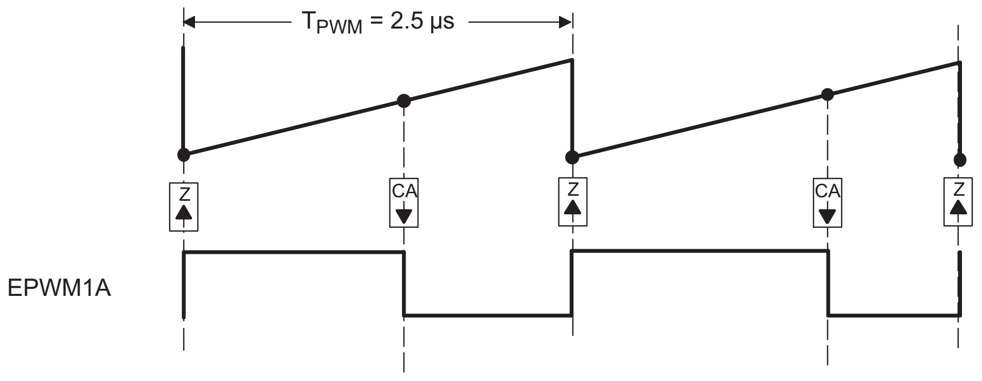


Figure 4-14. PWM Waveform Generated for the PWM DAC Function



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_SP 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.PRDLD = TB_IMMEDIATE;      // Set Immediate load
EPwm1Regs.TBPRD = 150;                         // Period set for 533 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.SYNCOSEL = 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 = 83*256;                      // Start with typical Scale Factor
                                               // value for 80 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 HRPWM Register Descriptions

This section describes the applicable HRPWM registers.

4.3.1 Register Summary

A summary of the registers required for the HRPWM is shown in the table below.

Table 4-6. Register Descriptions

Name	Offset	Size(x16)/Shadow	Description
Time Base Registers			
TBCTL	0x0000	1/0	Time Base Control Register
TBSTS	0x0001	1/0	Time Base Status Register
TBPHSHR	0x0002	1/0	Time Base Phase High Resolution Register
TBPHS	0x0003	1/0	Time Base Phase Register
TBCNT	0x0004	1/0	Time Base Counter Register
TBPRD	0x0005	1/1	Time Base Period Register Set
TBPRDHR	0x0006	1/1	Time Base Period High Resolution Register Set
Compare Registers			
CMPCTL	0x0007	1/0	Counter Compare Control Register
CMPAHR	0x0008	1/1	Counter Compare A High Resolution Register Set
CMPA	0x0009	1/1	Counter Compare A Register Set
CMPB	0x000A	1/1	Counter Compare B Register Set
HRPWM Registers			
HRCNFG	0x0020	1/0	HRPWM Configuration Register
HRMSTEP	0x0026	1/0	HRPWM MEP Step Register
High Resolution Period & Mirror Registers			
HRPCTL	0x0028	1/0	High Resolution Period Control Register
TBPRDHRM	0x002A	1/1	Time Base Period High Resolution Mirror Register Set
TBPRDM	0x002B	1/1	Time Base Period Mirror Register Set
CMPAHRM	0x002C	1/1	Counter Compare A High Resolution Mirror Register Set
CMPAM	0x002D	1/1	Counter Compare A Mirror Register Set

4.3.2 Registers and Field Descriptions

Figure 4-15. HRPWM Configuration Register (HRCNFG)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
SWAPAB	AUTOCONV	SELOUTB	HRLOAD		CTLMODE	EDGMODE	
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 4-7. HRPWM Configuration Register (HRCNFG) Field Descriptions

Bit	Field	Value	Description[(1)]
15-8	Reserved		Reserved
7	SWAPAB		Swap ePWM A & B Output Signals
			This bit enables the swapping of the A & 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		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 (i.e. software sets CMPAHR = (fraction(PWMduty * PWMperiod) * MEP Scale Factor)<<8 + 0x080 for duty cycle), then this mode must be disabled.
5	SELOUTB		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		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		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 (i.e., this is duty or period control mode). (Default on Reset)
		1	TBPHSHR(8) Register controls the edge position (i.e., this is phase control mode).
1-0	EDGMODE		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.

Figure 4-16. Counter Compare A High Resolution Register (CMPAHR)

15 — 8	7 — 0
CMPAHR	Reserved
R/W-0	R-0

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

Table 4-8. Counter Compare A High Resolution Register (CMPAHR) Field Descriptions

Bit	Field	Value	Description
15-8	CMPAHR	00-FEh	Compare A High Resolution register bits for MEP step control. 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.
7-0	Reserved	00-FFh	Any writes to these bit(s) must always have a value of 0.

Figure 4-17. TB Phase High Resolution Register (TBPHSHR)

15 — 8	7 — 0
TBPHSH	Reserved
R/W-0	R-0

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

Table 4-9. TB Phase High Resolution Register (TBPHSHR) Field Descriptions

Bit	Field	Value	Description
15-8	TBPHSH	00-FEh	Time base phase high resolution bits
7-0	Reserved	00-FFh	Any writes to these bit(s) must always have a value of 0.

Figure 4-18. Time Base Period High Resolution Register

15 — 8
TBPRDHR
R/W-0

7 — 0
Reserved
R-0

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

Table 4-10. Time Base Period High-Resolution Register (TBPRDHR) Field Descriptions

Bit	Field	Value	Description
15-8	PRDHR	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[PRDLD] 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		Reserved for TI Test

Figure 4-19. Compare A High Resolution Mirror Register

15 ... 8
CMPAHR
R/W-0

7 ... 0
Reserved
R-0

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

Table 4-11. Compare A High-Resolution Mirror Register (CMPAHRM) 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	00-FFh	Reserved for TI Test

Figure 4-20. Time-Base Period High Resolution Mirror Register

15 ... 8
TBPRDHR
R/W-0

7 ... 0
Reserved
R-0

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

Table 4-12. 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[PRDLD] 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 registerTBPRDHRM, are indeterminate (reserved for TI Test). The TBPRDHRM register is available with ePWM modules which support high-respolution period control and is used only when the high resolution period feature is enabled.
7-0	Reserved		Reserved

Figure 4-21. High Resolution Period Control Register (HRPCTL)

15 ... 8
Reserved
R-0

7 ... 3	2	1	0
Reserved	TBPHSHR LOADE	Reserved	HRPE
R-0	R/W-0	R-0	R/W-0

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

Table 4-13. High Resolution Period Control Register (HRPCTL) Field Descriptions

Bit	Field	Value	Description[1] [2]
15-3	Reserved		Reserved
2	TBPHSHRLOADE		TBPHSHR Load Enable
			This bit allows you to synchronize ePWM modules with a high-resolution phase on a SYNCIN, TBCTL[SWFSYNC] or digital compare event. This allows for multiple ePWM modules operating at the same frequency to be phase aligned with high-resolution.
		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	Reserved		Reserved
0	HRPE		High Resolution Period Enable Bit
		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.

Figure 4-22. High Resolution Micro Step Register (HRMSTEP) (EALLOW protected):

15 – 8	7 – 0
Reserved	HRMSTEP
R-0	R/W-0

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

Table 4-14. High Resolution Micro Step Register (HRMSTEP) Field Descriptions

Bit	Field	Value	Description
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.

4.4 Appendix A: SFO Library Software - SFO_TI_Build_V6.lib

The following table lists several features of the SFO_TI_Build_V6.lib library.

Table 4-15. SFO Library Features

	SYSCLK Freq	SFO_TI_Build_V6.lib	Unit
Max. 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

A functional description of the SFO library routine, SFO(), is found below.

4.4.1 *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 = 80 MHz and assuming the MEP step size is 150 ps, the typical scale factor value at 80 MHz = 83 MEP steps per TBCLK unit (12.5 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 = (fraction(PWMduty * PWMperiod) * MEP Scale Factor)<<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 CMPA 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 CMPA register value must not fall below 3 or above TBPRD-3.This would avoid any unexpected transitions on the PWM signal.

4.4.2 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 4-16.

Table 4-16. 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 & 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 TMS320F2806x devices, the F2806x C/C++ Header Files and Peripheral Examples in controlSUITE F2806x_Device.h and F2806x_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 "F2806x_Device.h"       // F2806x Headerfile
#include "F2806x_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 below.

Example 4-7. Declaring an Element

```
int MEP_ScaleFactor = 0;   //scale factor value
volatile struct EPWM_REGS *ePWM[] = {0, &EPwm1Regs, &EPwm2Regs, &EPwm3Regs,
```

Example 4-7. Declaring an Element (continued)

```
&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:

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 here.

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)
 }
```

4.4.3 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 below.

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;

 }
```

High Resolution Capture

This chapter describes the operation of the high-resolution capture (HRCAP) module on the TMS320xF2806x Piccolo™ devices. The HRCAP module described here is a Type 0 HRCAP. HRCAP measures the width of external pulses with a typical resolution within hundreds of picoseconds. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566) for a list of all devices with an HRCAP module of the same type, to determine the differences between types, and for a list of device-specific differences within a type.

Topic **Page**

5.1 Introduction

Uses for the HRCAP include:

- Capactive touch applications
- High-resolution period and duty cycle measurements of pulse train cycles
- Instantaneous speed measurements
- Instantaneous frequency measurements
- Voltage measurements across an isolation boundary
- Distance/sonar measurement and scanning

The HRCAP module includes the following features:

- Pulse-width capture in either non-high-resolution or high-resolution modes
- Difference (Delta) mode pulse-width capture
- Typical high-resolution capture on the order of 300 ps resolution on each edge
- Interrupt on either falling or rising edge
- Continuous mode capture of pulse widths in 2-deep buffer
- Calibration logic for precision high-resolution capture
- All of the above resources are dedicated to a single input pin.

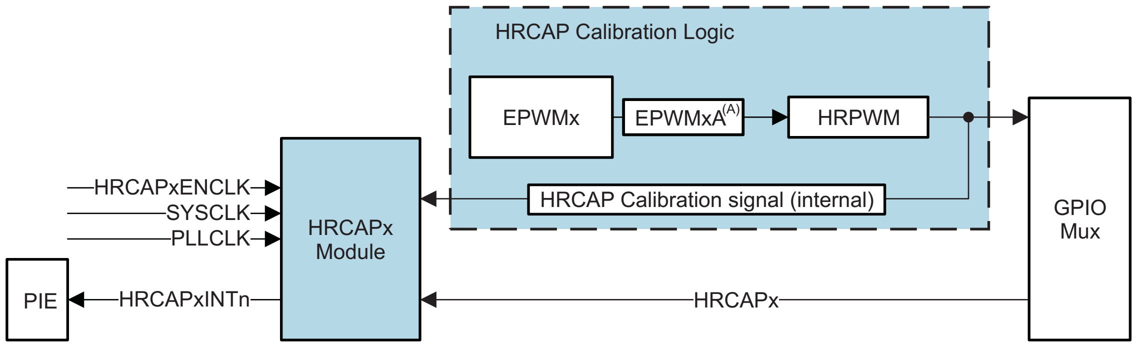
5.2 Description

The HRCAP module includes one capture channel in addition to a high-resolution calibration block, which connects internally to an HRPWM channel during calibration. See the device-specific data manual to determine which HRPWM channel output the HRCAP module is internally tied to during calibration.

Each HRCAP channel has the following independent key resources:

- Dedicated input capture pin
- 16-bit HRCAP capture clock (HCCAPCLK) which is either equal to the PLL output frequency (asynchronous to SYSCLKOUT) or equal to the SYSCLKOUT frequency (synchronous to SYSCLKOUT).
- High-resolution pulse width capture in a 2-deep buffer
- High-resolution calibration logic utilizing an internal connection to an HRPWM output

Figure 5-1. HRCAP Module System Block Diagram

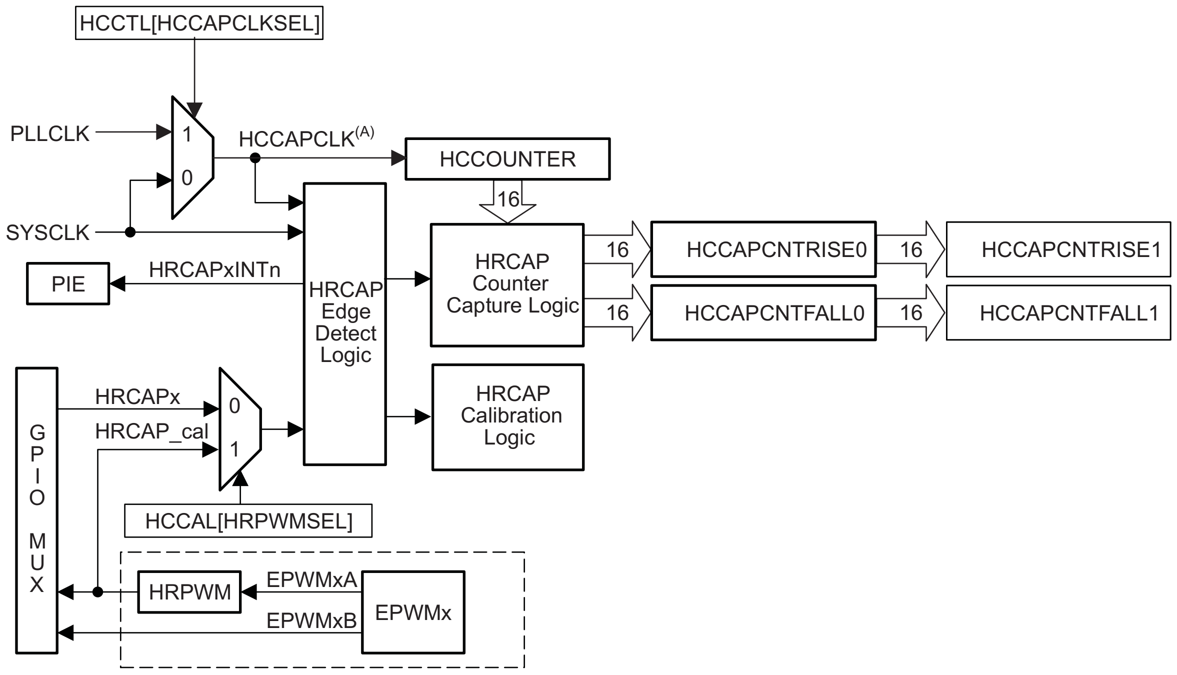


A In general, the largest numerical instance of an HRPWM module channel A output is the internal HRCAP calibration signal input. For instance, on devices where there are eight HRPWM instances, ePWM8A HRPWM output is the internal HRCAP calibration signal input.

5.3 Operational Details

Figure 5-2 shows the various components that implement the high-resolution, pulse-width capture functionality of the module.

Figure 5-2. HRCAP Block Diagram



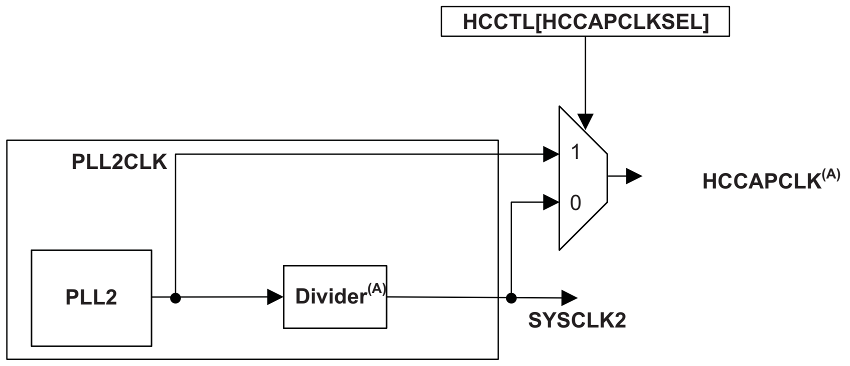
A If PLLCLK is selected as the source for HCCAPCLK, HCCAPCLK is asynchronous to SYSCLK.

5.3.1 *HRCAP Clocking*

Although the HRCAP module is clocked by the system clock, the 16-bit counter (HCCOUNTER) and edge detection logic used for capturing high-resolution pulses is clocked by HCCAPCLK. HCCAPCLK must fall within the frequency range specified in the *Electricals* section of the device-specific data manual. HCCAPCLK can either be clocked by the system clock (SYSCLK), or the output of the PLL (PLLCLK) before the divider is applied. If HCCAPCLK is fed from the PLLCLK (HCCTL[HCCAPCLKSEL] = 1), then HCCAPCLK will be asynchronous to SYSCLK. On this device, HCCAPCLK is clocked by SYSCLK2 or PLL2 instead of the main device SYSCLK or PLL.

Figure 5-3 shows how the HCCAPCLK that clocks the HCCOUNTER and edge detection logic is generated.

Figure 5-3. HCCAPCLK Generation



A On this device, the clock divider is a default value of /2 when DEVICECNF[SYSCLK2DIV2DIS] =0.

5.3.2 HRCAP Modes of Operation

The HRCAP module has two modes of operation:

- **Normal capture mode:** The HRCAP module captures pulse widths in normal resolution within +/- 1 SYSCLK (where SYSCLK is sourced from the same PLL output clock that sources HCCAPCLK — on this device, this is SYSCLK2). This mode requires less software overhead than high-resolution capture mode.
- **High-resolution capture mode:** The HRCAP module captures pulse widths with the resolution of each edge captured within +/- 300 ps typical and requires the usage of the HCCal calibration library provided by Texas Instruments. In this mode, one HRCAP channel and the ePWM module connected to the HRCAP calibration input must be dedicated to HRCAP calibration and are not functionally available to the application during calibration.

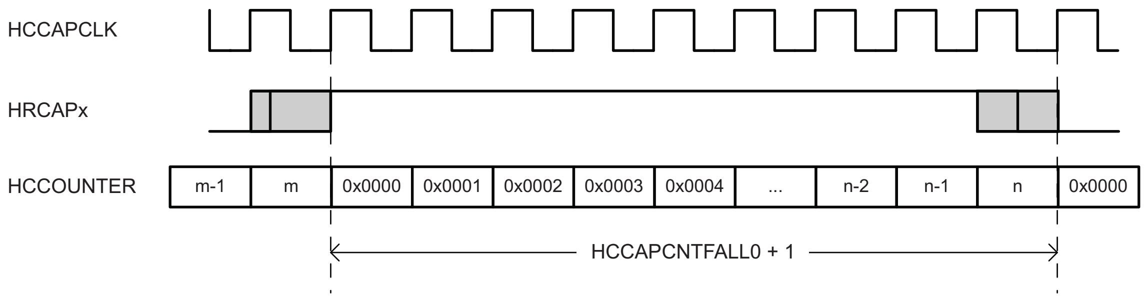
5.3.2.1 HRCAP Counter

Both modes of operation utilize HCCOUNTER, which resets to 0 and starts counting HCCAPCLK cycles again under the following conditions:

- SOFTRESET
- Detection of rising edge
- Detection of falling edge
- Device reset and reenable of HRCAP clock

When a rising edge is detected, the value in HCCOUNTER is captured into the 16-bit HCCAPCNTRISE0 register before the counter resets to 0. When a falling edge is detected, the value in HCCOUNTER is captured into the 16-bit HCCAPCNTFALL0 register before the counter resets to 0. Because the HCCOUNTER starts counting at 0 after an edge is detected, the actual low and high pulse widths (non-high-resolution) are HCCAPCNTFALL0 + 1 and HCCAPCNTRISE0 + 1, respectively, where the “+1” is added to account for the “0” HCCAPCLK cycle. This behavior is illustrated for high pulse width capture in Figure 5-4.

Figure 5-4. HCCOUNTER Behavior During High Pulse Width Capture



Because the HCCOUNTER starts counting immediately after SOFTRESET, the first capture result into the capture “0” registers should be discarded. The value captured will be the number of HCCAPCLK cycles since the last SOFTRESET or the HRCAPCLKEN bit was set rather than an actual pulse width measurement.

5.3.2.2 HCCAP0 - HCCAP1 Registers

The HRCAP capture registers include a 2-deep FIFO buffer to store two pulse widths’ worth of data.

When a rising edge event occurs, HCCAPCNTRISE0 is loaded with the pulse width data from the last falling edge to the current rising edge (low pulse width). At the next rising edge event, the value in the HCCAPCNTRISE0 register is loaded into HCCAPCNTRISE1.

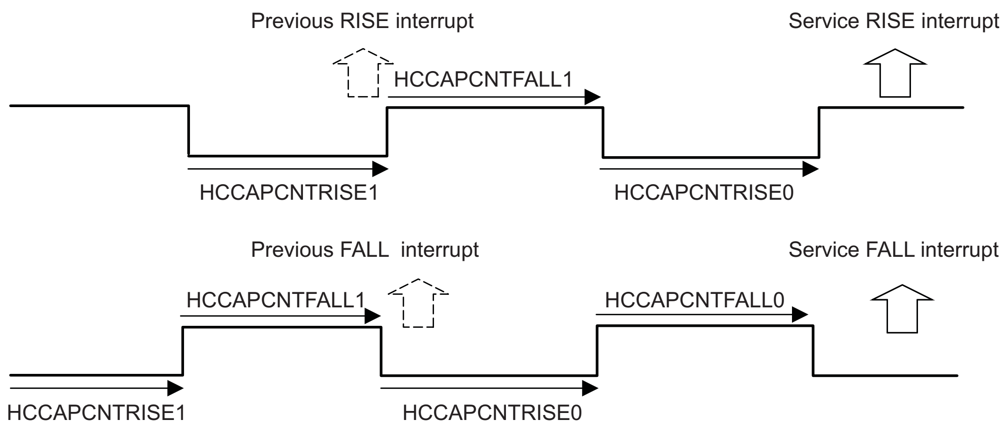
When a falling edge event occurs, HCCAPCNTFALL0 is loaded with the pulse width data from the last rising edge to the current falling edge (high pulse width). For falling edge events, the HRCAP logic operates such that the HCCAPCNTFALL0 register value is then loaded into the HCCAPCNTFALL1 register at the next rising edge event rather than waiting until the next falling edge event.

5.3.2.3 RISE vs FALL Capture Events

HRCAP capture registers can be read either during rising edge capture events or during falling edge capture events. They should not be read during both events.

There are a number of differences with regard to using RISE events to read captured registers vs. using FALL events to read captured registers as shown below in Figure 5-5.

Figure 5-5. Rise vs. Fall Capture Events



5.3.2.3.1 RISE Capture Events

When a RISE event occurs, the application code has access to full valid capture data for two pulse widths (1 period) in high-resolution capture mode and three pulse widths (1.5 periods) in normal capture mode. HCCAPCNTFALL0 does not have valid data available on RISE events, as this values are not captured until the falling edge event after the current rising edge event (event has not yet occurred).

The application code has until the next RISE event to read all relevant capture data and clear the RISE event. Otherwise, the data will be overwritten and invalid. Therefore RISE events are generally used to capture data for period signals where duty cycle may vary significantly.

NOTE: Because HCCOUNTER starts counting immediately after SOFTRESET, the first RISE capture result into HCCAPCNTRISE0 does not include valid pulse width data and should be discarded. When the second RISE capture event occurs, this invalid data is transferred to HCCAPCNTRISE1, and therefore the data in this register should also be discarded. After the second rise interrupt, all capture data is valid and can be used normally.

5.3.2.3.2 FALL Capture Events

When a FALL event occurs, the application code has access to full valid capture data for three pulse widths (1.5 periods) in high-resolution capture mode and four pulse widths (2 full periods) in normal capture mode.

The application code has only until the next RISE event to read all relevant registers and clear the FALL event. Otherwise, the data will be overwritten and invalid. Therefore FALL events are generally used to capture short pulse widths spaced far enough apart to read the registers safely.

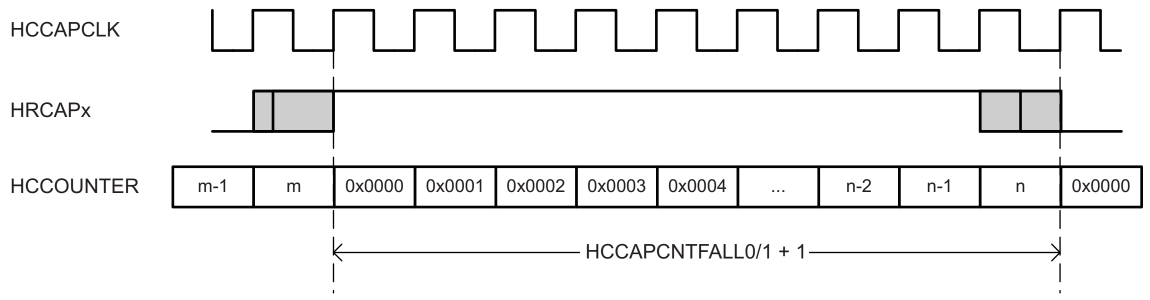
NOTE: Because HCCOUNTER starts counting immediately after SOFTRESET, the first FALL capture result into HCCAPCNTFALL0 does not include valid pulse width data and should be discarded. By the next FALL capture event, the invalid data in the “0” register has been transferred into HCCAPCNTFALL1. Therefore the data in this register should also be discarded. After the second FALL interrupt, all capture data is valid and can be used normally.

5.3.2.4 Normal Capture Mode

In normal capture mode, when a rise event (HCIFR[RISE]=1) or a fall event (HCIFR[FALL]=1) occurs, the application code reads the HCCAPCNTRISE0/1 and HCCAPCNTFALL0/1 registers and does not require the HCCal HRCAP calibration library. The resolution of the captured result will be accurate within +/- 1 SYSCLK cycles (where SYSCLK is sourced by the same PLLCLK that generates HCCAPCLK – on this device, it is within +/- 1 SYSCLK2 cycles).

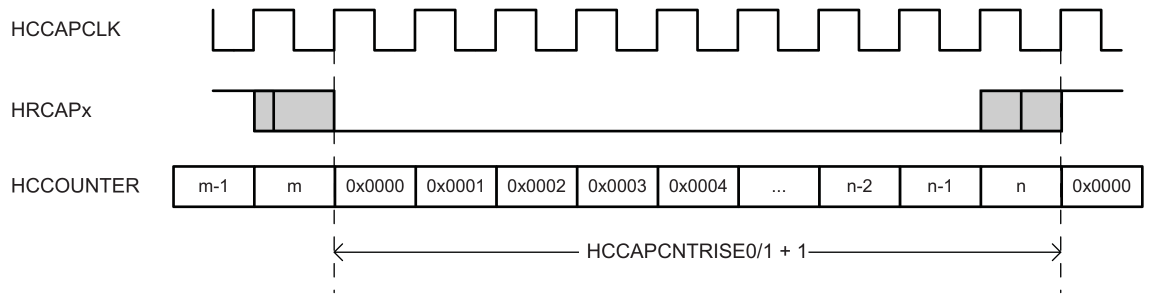
High pulse widths are measured in number of HCCAPCLK cycles equal to 1 + HCCAPCNTFALL0 or 1 + HCCAPCNTFALL1 as shown in Figure 5-6.

Figure 5-6. High Pulse Width Normal Mode Capture



Low pulse widths are measured in number of HCCAPCLK cycles equal to 1 + HCCAPCNTRISE0 or 1 + HCCAPCNTRISE1 as shown in Figure 5-7.

Figure 5-7. Low Pulse Width Normal Mode Capture



In both cases, 1 is added to the value in the HCCAPCNT registers to account for the HCCAPCLK cycle in which HCCOUNTER = 0.

5.3.2.5 High Resolution Capture Mode

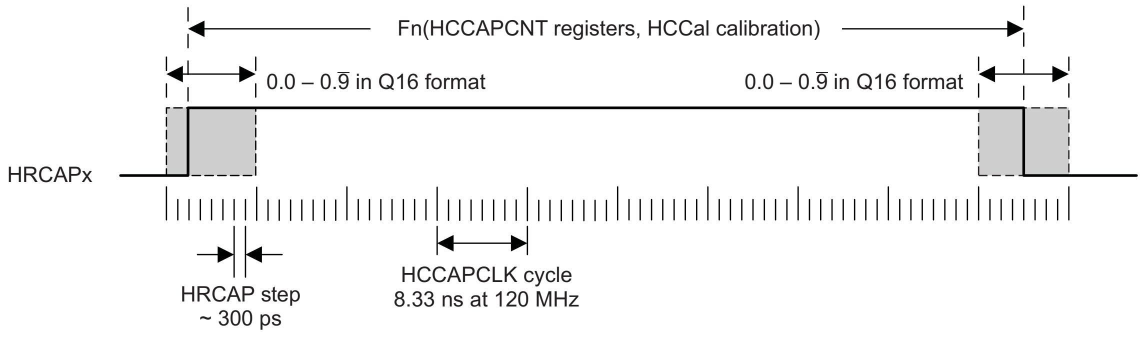
In high-resolution capture mode, the application code utilizes the HCCal HRCAP calibration library functions to capture the high-resolution pulse width with each edge captured at a typical resolution of +/- 300 ps (with two edges, the resolution of the measured pulse width could vary by +/- 600 ps). Note that although the HRCAP logic itself can be calibrated to capture high-resolution pulse widths, if the jitter on the input signal is greater than +/- 300 ps, the captured value will also vary according to the jitter on the input signal.

In order to use high-resolution capture mode, the high-resolution capture logic must be calibrated to scale the HRCAP step size to a Q16 fraction of the HCCAPCLK. One HRCAP module and the ePWM module internally connected to the HRCAP calibration input must be dedicated only to calibration and cannot be used functionally in the application during calibration.

Texas Instruments provides a calibration function in the HCCal HRCAP calibration library to perform this calibration once prior to using the HRCAP in high-resolution capture mode and periodically in a slow loop to account for changes in the HRCAP step size due to voltage and temperature changes while the application is running.

The library also provides functions to measure the high resolution high pulse width, low pulse width, and period. The pulse and period width measurement results are returned in Q16 fixed-point format with the fractional portion of the result representing a fraction of an HCCAPCLK cycle. (For instance a pulse width may appear in the form of 500.25 HCCAPCLK cycles). Figure 5-8 shows how the high-resolution pulse width is a function of the calibration of the HRCAP step size and the values in the HCCAPCNT registers.

Figure 5-8. HRCAP High-Resolution Mode Operating Logic

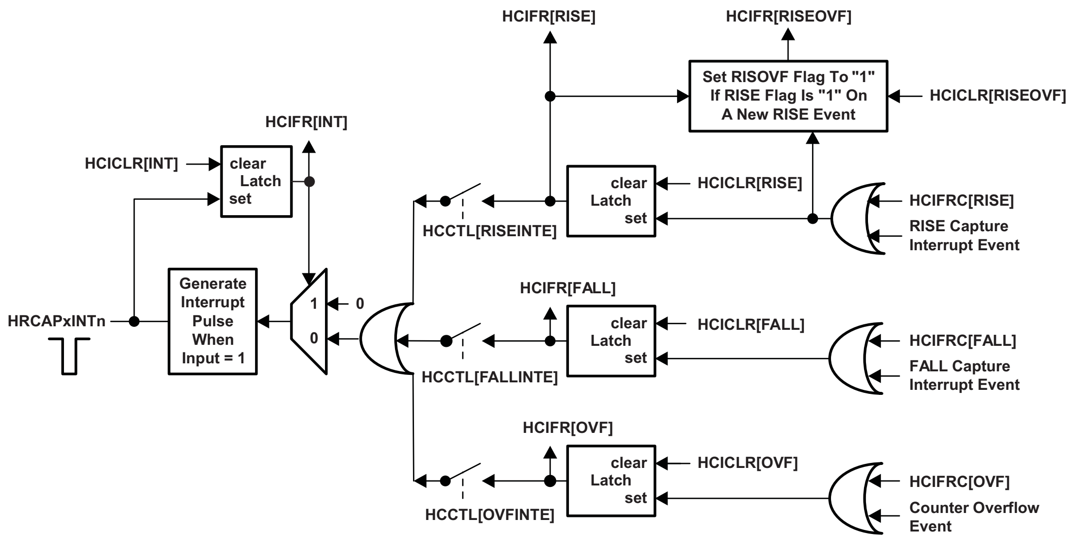

Fn(HCCAPCNT registers, HCCal calibration)
0.0 – 0.$\bar{9}$ in Q16 format
0.0 – 0.$\bar{9}$ in Q16 format
HRCAPx
HRCAP step
~ 300 ps
HCCAPCLK cycle
8.33 ns at 120 MHz

For details on using the HCCal HRCAP calibration library in high-resolution capture mode, see Section 5.5.

5.3.3 HRCAP Interrupts

Rising edge capture (RISE), falling edge capture (FALL), and HCCOUNTER overflow (OVF) events can generate interrupts to the PIE from the HRCAP module. Additionally, if the rising edge capture flag is set (HCIFR[RISE]) when another rising edge capture event occurs, a rising edge overflow (RISEOVF) interrupt can also generate an interrupt to the PIE. The HRCAP interrupt logic is shown below in Figure 5-9.

Figure 5-9. Interrupts in HRCAP Module



RISE/RISEOVF, FALL, and OVF events will only generate an interrupt if the corresponding interrupt enable bits in the HCCTL register are set to 1. Interrupt events can be cleared by writing a 1 to the corresponding bits in the HCICLR register. For testing purposes, interrupt events can be forced by writing a 1 to the corresponding bits in the HCIFRC register.

For proper operation, RISE and FALL interrupts should not be enabled at the same time. Capture registers should be read during rising edge interrupt events only, or during falling edge interrupt events only, and not during both interrupt events simultaneously. If RISEOVF interrupts are enabled, the RISE flag must always be acknowledged after a RISE event, otherwise a rise overflow condition will occur.

5.4 Register Descriptions

The complete HRCAP register set is shown in Table 5-1.

Table 5-1. HRCAP Register Summary

Name	Address	Description
	Offset	
HCCTL	0x00	HRCAP Control Register
HCIFR	0x01	HRCAP Interrupt Flag Register
HCICLR	0x02	HRCAP Interrupt Clear Register
HCIFRC	0x03	HRCAP Interrupt Force Register
HCCOUNTER	0x04	HRCAP 16-bit Counter Register
HCCAPCNTRISE0	0x10	HRCAP Capture Counter On Rising Edge 0 Register
HCCAPCNTFALL0	0x12	HRCAP Capture Counter On Falling Edge 0 Register
HCCAPCNTRISE1	0x18	HRCAP Capture Counter On Rising Edge 1 Register
HCCAPCNTFALL1	0x1A	HRCAP Capture Counter On Falling Edge 1 Register

5.4.1 *HRCAP Control Register (HCCTL) – EALLOW protected*

The HRCAP control register (HCCTL) is shown and described in the figure and table below.

Figure 5-10. HRCAP Control Register (HCCTL)

15-9	8
Reserved	HCCAPCLKSEL
R-0	R/W-0

7-4	3	2	1	0
Reserved	OVFINTE	FALLINTE	RISEINTE	SOFTRESET
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 5-2. HRCAP Control Register (HCCTL) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved		Reserved
8	HCCAPCLKSEL		Capture clock select bit. This bit is used to select the clock source for HCCAPCLK. This bit should be set such that HCCAPCLK falls between the frequency range limits specified in the HRCAP *Electricals* section of the device-specific data manual.
		0	HCCAPCLK = SYSCLK
		1	HCCAPCLK = PLLCLK (bypasses the clock divider).
7-4	Reserved		Reserved
3	OVFINTE		Counter overflow interrupt enable bit
		0	Disable counter overflow interrupt
		1	Enable counter overflow interrupt
2	FALLINTE		Falling edge capture interrupt enable bit
		0	Disable falling edge capture interrupt
		1	Enable rising edge capture interrupt
1	RISEINTE		Rising edge capture interrupt enable bit
		0	Disable rising edge capture interrupt
		1	Enable rising edge capture interrupt
0	SOFTRESET		Soft reset
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear HCCOUNTER, all capture registers, and the IFR register bits.

5.4.2 *HRCAP Interrupt Flag Register (HCIFR)*

The HRCAP interrupt flag register (HCIFR) is shown and described in the figure and table below.

Figure 5-11. HRCAP Interrupt Flag Register (HCIFR)

15 - 8
Reserved
R-0

7 - 5	4	3	2	1	0
Reserved	RISEOVF	COUNTEROVF	FALL	RISE	INT
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 5-3. HRCAP Interrupt Flag Register (HCIFR) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved		Reserved
4	RISEOVF		Rising edge interrupt overflow event flag
		0	No rising edge interrupt overflow event has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	This bit is set to "1" if the RISE flag is "1" when a new RISE event occurs.
3	COUNTEROVF		Counter overflow interrupt flag
		0	The HCCOUNTER has not overflowed. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	This bit is set to 1 when the 16-bit HCCOUNTER overflows (from 0xFFFF to 0x0000). This bit can also be set to 1 by writing to the corresponding bit in the HCIFRC register.
2	FALL		Falling edge capture interrupt flag:
		0	No falling edge interrupt has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	A falling edge input capture event has occurred. This bit can also be set to 1 by writing to the corresponding bit in the HCIFRC register.
1	RISE		Rising edge capture interrupt flag
		0	No rising edge interrupt has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	A rising edge input capture event has occurred. This bit can also be set to 1 by writing to the corresponding bit in the HCIFRC register.
0	INT		Global interrupt flag
		0	No HRCAP interrupt has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	An enabled RISE, FALL or COUNTEROVF interrupt has been generated. No further interrupts are generated until this bit is cleared.

5.4.3 HRCAP Interrupt Clear Register (HCICLR)

The HRCAP interrupt clear register (HCICLR) is shown and described in the figure and table below.

Figure 5-12. HRCAP Interrupt Clear Register (HCICLR)

15–8
Reserved
R-0

7–5	4	3	2	1	0
Reserved	RISEOVF	COUNTEROVF	FALL	RISE	INT
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 5-4. HRCAP Interrupt Clear Register (HCICLR) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved		Reserved
4	RISEOVF		Rising edge interrupt overflow clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding RISEOVF flag bit in the HCIFR register to "0". The hardware setting of HCIFR[RISEOVF] flag bit has priority over the software clear if both happen on the same cycle.
3	COUNTEROVF		Counter overflow interrupt clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding COUNTEROVF flag bit in the HCIFR register to "0". The hardware setting of HCIFR[COUNTEROVF] flag bit has priority over the software clear if both happen on the same cycle.
2	FALL		Falling edge capture interrupt clear bit:
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding FALL flag bit in the HCIFR register to "0". The hardware setting of HCIFR[FALL] flag bit has priority over the software clear if both happen on the same cycle.
1	RISE		Rising edge capture interrupt clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding RISE flag bit in the HCIFR register to "0". The hardware setting of HCIFR[RISE] flag bit has priority over the software clear if both happen on the same cycle.
0	INT		Global interrupt clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding INT flag bit in the HCIFR register to "0". The hardware setting of HCIFR[INT] flag bit has priority over the software clear if both happen on the same cycle.

5.4.4 HRCAP Interrupt Force Register (HCIFRC)

The HRCAP interrupt force register (HCIFRC) is shown and described in the figure and table below.

Figure 5-13. HRCAP Interrupt Force Register (HCIFRC)

15–8
Reserved
R-0

7–4	3	2	1	0
Reserved	COUNTEROVF	FALL	RISE	Reserved
R-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 5-5. HRCAP Interrupt Force Register (HCIFRC) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Reserved
3	COUNTEROVF		Counter overflow interrupt force bits
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will force the corresponding COUNTEROVF flag bit in HCIFR register to "1". The software HCCTL[SOFTRESET] clearing of the HCIFR[COUNTEROVF] bit has priority over the hardware trying to set the bit in the same cycle.
2	FALL		Falling edge interrupt force bits
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will force the corresponding FALL flag bit in HCIFR register to "1". The software HCCTL[SOFTRESET] clearing of the HCIFR[FALL] bit has priority over the hardware trying to set the bit in the same cycle.
1	RISE		Rising edge interrupt force bits
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will force the corresponding RISE flag bit in HCIFR register to "1". The software HCCTL[SOFTRESET] clearing of the HCIFR[RISE] bit has priority over the hardware trying to set the bit in the same cycle.
0	Reserved		Reserved

5.4.5 HRCAP Counter Register (HCCOUNTER)

The HRCAP counter register (HCCOUNTER) is shown and described in the figure and table below.

Figure 5-14. HRCAP Counter Register (HCCOUNTER)

15–0
COUNTER
R-0

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

Table 5-6. HRCAP Counter Register (HCCOUNTER) Field Descriptions

Bit	Field	Value	Description
15-0	COUNTER	0	16-bit capture counter This free running counter is used to capture rising and falling edge events. The HCCOUNTER is incremented on every HCCAPCLK cycle. When the counter reaches 0xFFFF, it will overflow to 0x0000 on the next cycle and generate a COUNTEROVF interrupt event. The counter is reset to 0x0000 on every rising and falling edge event. The counter can also be reset to 0x0000 by a system reset or by setting the HCCTL[SOFTRESET] bit. **NOTE:** Because the counter is clocked from HCCAPCLK, which can be asynchronous to SYSCLK, CPU reads to this register should not be performed unless the clocks to the HRCAP module are disabled (HRCAPxENCLK = 0).

5.4.6 HRCAP Capture Counter On Rising Edge 0 Register (HCCAPCNTRISE0)

The HRCAP capture counter on rising edge 0 register (HCCAPCNTRISE0) is shown and described in the figure and table below.

Figure 5-15. HRCAP Capture Counter On Rising Edge 0 Register (HCCAPCNTRISE0)

15 … 0
HCCAPCNTRISE0
R-0

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

Table 5-7. HRCAP Capture Counter On Rising Edge 0 Register (HCCAPCNTRISE0) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTRISE0	0	HRCAP capture counter on rising edge 0 register This register captures the16-bit HCCOUNTER value when a rising edge event is detected.

5.4.7 HRCAP Capture Counter On Rising Edge 1 Register (HCCAPCNTRISE1)

The HRCAP capture counter on rising edge 0 register (HCCAPCNTRISE1) is shown and described in the figure and table below.

Figure 5-16. HRCAP Capture Counter On Rising Edge 1 Register (HCCAPCNTRISE1)

15 … 0
HCCAPCNTRISE1
R-0

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

Table 5-8. HRCAP Capture Counter On Rising Edge 1 Register (HCCAPCNTRISE1) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTRISE1	0	HRCAP capture counter on rising edge 1 register On an input rising edge event, the value in the HCCAPCNTRISE0 register is copied into the HCCAPCNTRISE1 register before the HCCOUNTER value is captured into the HCCAPCNTRISE0 register.

5.4.8 HRCAP Capture Counter On Falling Edge 0 Register (HCCAPCNTFALL0)

The HRCAP capture counter on falling edge 0 register (HCCAPCNTFALL0) is shown and described in the figure and table below.

Figure 5-17. HRCAP Capture Counter On Falling Edge 0 Register (HCCAPCNTFALL0)

15 … 0
HCCAPCNTFALL0
R-0

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

Table 5-9. HRCAP Capture Counter On Falling Edge 0 Register (HCCAPCNTFALL0) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTFALL0	0	HRCAP capture counter on falling edge 0 register This register captures the16-bit HCCOUNTER value when a Falling edge event is detected.

5.4.9 HRCAP Capture Counter On Falling Edge 1 Register (HCCAPCNTFALL1)

The HRCAP capture counter on falling edge 1 register (HCCAPCNTFALL1) is shown and described in the figure and table below.

Figure 5-18. HRCAP Capture Counter On Falling Edge 1 Register (HCCAPCNTFALL1)

15 … 0
HCCAPCNTFALL1
R-0

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

Table 5-10. HRCAP Capture Counter On Falling Edge 1 Register (HCCAPCNTFALL1) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTFALL1	0	HRCAP capture counter on falling edge 1 register On an input falling edge event, the value in the HCCAPCNTFALL0 register is copied into the HCCAPCNTFALL1 register before the HCCOUNTER value is captured into the HCCAPCNTFALL0 register.

5.5 HRCAP Calibration Library

The HRCAP calibration (HCCal) logic is capable of capturing an edge in discrete time steps which subdivide an HCCAPCLK cycle. As previously mentioned, the size of each step is on the order of 300 ps (see device-specific data sheet for typical HRCAP step size on your device). The HRCAP_Cal() function in the HRCAP calibration library must be run periodically to be certain that time steps are optimally applied and that the edge capture accuracy is maintained over a wide range of PWM frequencies, system clock frequencies, voltages, and temperatures. The HRCAP step size varies based on worst-case process parameters, operating temperature, and voltage. HRCAP step size increases with decreasing voltage and increasing temperature and decreases with increasing voltage and decreasing temperature.

Applications that use the HRCAP in high-resolution capture mode should use the TI-supplied HRCAP calibration (HCCal) software HRCAP_Cal() function. The HRCAP_Cal function helps to dynamically scale the HRCAP step size to a fraction of the HCCAPCLK cycle while the HRCAP is in high-resolution mode. To utilize the HCCal capabilities effectively during HRCAP operation, the HRCAP calibration logic uses built-in self-check and diagnostics capabilities to scale the HRCAP step size appropriately for any operating condition.

TI provides a C-callable library containing one HRCAP calibration function that utilizes this hardware and properly calibrates the internal HRCAP step logic as a fraction of a HCCAPCLK cycle. The library supplies additional high-resolution capture functions to calculate pulse widths captured in Q16 integer + fractional HCCAPCLK cycles based on the values in the HCCAPCNT registers and the calibration results.

The contents of these functions are proprietary to Texas Instruments and will not be published.

Currently, there is 1 released version of the HCCal Type 0 library, HCCal_Type0_V1.lib, which is located in the controlSUITE software package under the /libs/utilities/hrcap_hccal/ directory.

5.5.1 HRCAP Calibration Library Functions

5.5.1.1 HRCAP_Cal

The HRCAP_Cal() function runs calibration and self-check diagnostics logic on a given HRCAP module to internally scale the HCCal step logic as a fraction of a HCCAPCLK cycle.

Prototype:

```
Uint16 HRCAP_Cal(Uint16 HRCAPModule, Uint16 PLLClk, volatile struct EPWM_REGS *ePWMModule);
```

Parameters:

HRCAPModule	The HRCAP module number as an integer value (i.e., “1” for HRCAP1, and “2” for HRCAP2). This HRCAP module will be dedicated to calibration only and cannot be used functionally to capture pulse widths.
PLLClk	If 0, then HCCAPCLK is clocked by SYSCLK, where SYSCLK is the system clock that clocks the HRCAP module. If 1, then HCCAPCLK is clocked by PLLCLK, where the PLLCLK frequency is a multiple of the system clock that clocks the HRCAP module.
ePWMModule	A pointer to the address of the EPWM_REGS structure for the HRPWM module used to calibrate the HRCAP (i.e., if HRPWM7A output is connected to the HRCAP’s internal calibration logic, then &EPwm7Regs is passed into this parameter, and if HRPWM8A output is connected to the HRCAP’s internal calibration logic, then &EPwm8Regs is passed into this parameter). The single ePWM module used for HRCAP calibration is device-dependent.

Returns:

0	If HCCal calibration is in progress without encountering errors.
1	If HCCal has exited with errors. User should check that PLL is configured such that the HCCAPCLK frequency falls within the frequency limits designated by the HRCAP Electricals section of the data manual.
2	If HCCal calibration has completed without errors.

Description:

This function drives the HRCAP calibration module logic to subdivide an HCCAPCLK cycle into HRCAP time steps equivalent to a fraction of an HCCAPCLK cycle at any given time.

HRCAP_Cal() can only be used with HCCAPCLK between 98 Mhz and 120 MHz (See your device-specific data manual’s HRCAP *Electricals* section for the device-specific HCCAPCLK frequency limits).

The function can be called at any time on a single HRCAP module which is dedicated to calibration only.

The calibration HRCAP module cannot be used functionally to capture pulse widths. The calibration logic driven by this function uses a single ePWMxA HRPWM channel output connected internally to the HRCAP input to run diagnostics. During calibration, that ePWM module cannot be used for normal ePWM functions in the application. For instance, on this device, while the HRCAP is in use in high-resolution capture mode, ePWM8 cannot be used functionally by the application during calibration.

5.5.1.2 LowPulseWidth0

The LowPulseWidth0() function captures the high-resolution low pulse width around HCCAPCNTRISE0 in HCCAPCLK cycles.

Prototype:

```
Uint32 LowPulseWidth0 (Uint16 * ptrHRCAPmodule);
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

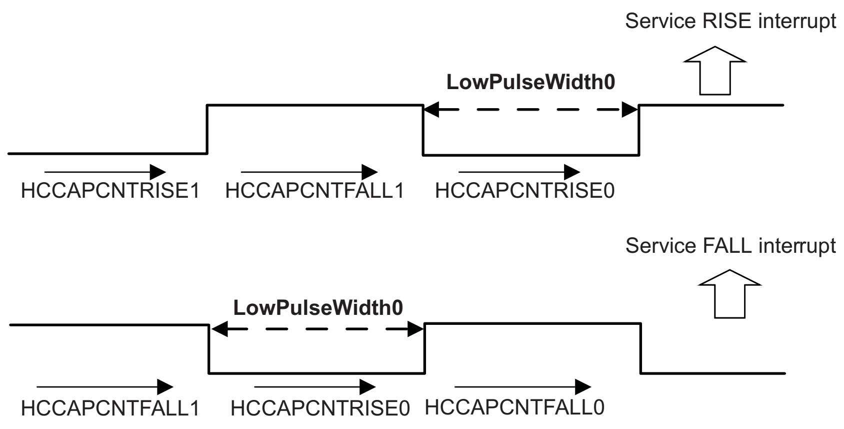
Returns:

32-bit high-resolution low pulse width around HCCAPCNTRISE0 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

This function can be called for any of the HRCAP modules not used for calibration to convert the HCCAPCNTRISE0 and HRCAP calibration results into a fixed-point Q16 integer + fractional high-resolution low-pulse width in HCCAPCLK cycles. Figure 5-19 shows which low pulse widths can be captured on a RISE and FALL event.

Figure 5-19. LowPulseWidth0 Capture on RISE and FALL Events



5.5.1.3 HighPulseWidth0 and HighPulseWidth1

The HighPulseWidth0()function captures the high-resolution high pulse width around HCCAPCNTFALL0 in HCCAPCLK cycles. The HighPulseWidth1()function captures the high-resolution high pulse width around HCCAPCNTFALL1 in HCCAPCLK cycles.

Prototypes:

```
Uint32 HighPulseWidth0 (Uint16 * ptrHRCAPmodule);
Uint32 HighPulseWidth1 (Uint16 * ptrHRCAPmodule);
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

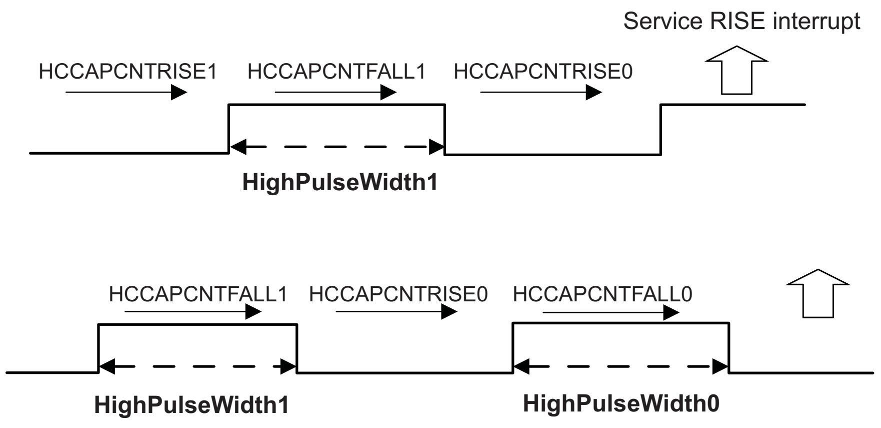
Returns:

32-bit high-resolution high pulse width around HCCAPCNTFALL0/1 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

These functions can be called for any of the HRCAP modules not used for calibration. They use the calibration logic to convert the HCCAPCNTFALL0/1 register value and HCCal calibration results into a fixed-point Q16 integer + fractional high-resolution high-pulse width in HCCAPCLK cycles. Figure 5-20 shows which high pulse widths can be captured on a RISE and FALL event.

Figure 5-20. HighPulseWidth0/1 Capture on RISE and FALL Events



5.5.1.4 PeriodWidthRise0

The PeriodWidthRise0() function captures the rising edge to rising edge high-resolution period width around HCCAPCNTRISE0 and HCCAPCNTFALL1 in HCCAPCLK cycles.

Prototype:

```
Uint32 PeriodWidthRise0 (Uint16 * ptrHRCAPmodule)
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

Returns:

32-bit high-resolution rising edge to rising edge period width around HCCAPCNTRISE0+HCCAPCNTFALL1 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

This function can be called for any of the HRCAP modules not used for calibration. It uses the calibration logic to convert the HCCAPCNTRISE0 and HCCAPCNTFALL1 register values and HCCal calibration results into the function into a fixed-point Q16 integer + fractional high-resolution period width in HCCAPCLK cycles. Figure 5-21 shows which period widths can be captured on a RISE and FALL event.

5.5.1.5 PeriodWidthFall0

The PeriodWidthFall0()function captures the falling edge to falling edge high-resolution period width around HCCAPCNTFALL0 and HCCAPCNTRISE0 in HCCAPCLK cycles.

Prototype:

```
Uint32 PeriodWidthFall0 (Uint16 * ptrHRCAPmodule);
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

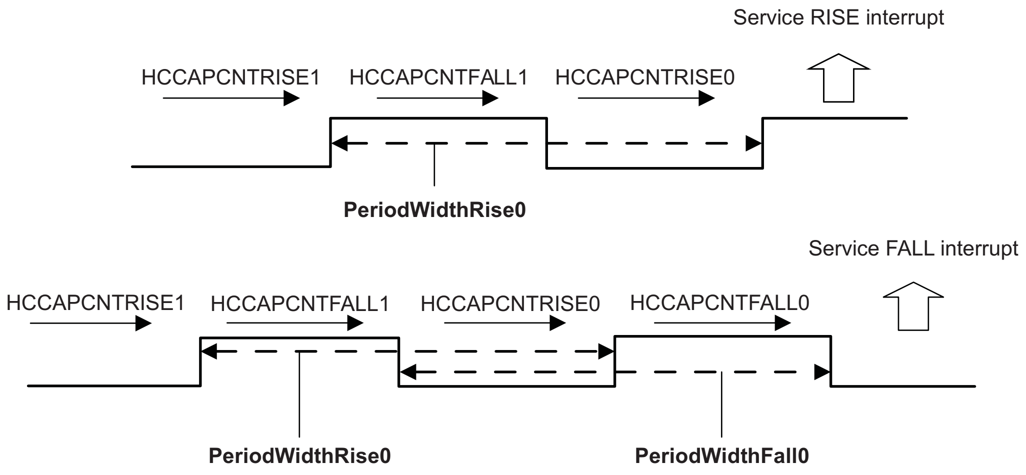
Returns:

32-bit high-resolution falling edge to falling edge period width around HCCAPCNTFALL0+HCCAPCNTRISE0 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

This function can be called for any of the HRCAP modules not used for calibration. It uses the calibration logic to convert the HCCAPCNTFALL0 and HCCAPCNTRISE0 register values and HCCal calibration results into a fixed-point Q16 integer + fractional high-resolution period width in HCCAPCLK cycles. Figure 5-21 shows which period widths can be captured on a RISE and FALL event.

Figure 5-21. PeriodWidthRise0 and PeriodWidthFall0 Capture on RISE and FALL Events



5.5.2 HRCAP Calibration Library Software Usage

To use the HRCAP in high-resolution mode, it is recommended that the HRCAP calibration functions be used as described here.

Step 1: Add "Include" Files

The HCCal_Type0_V1.h file needs to be included as follows. This include file is mandatory while using the HRCAP calibration library functions. For these devices, the F2806x_Device.h and F2806x_Examples.h files in the F2806x C/C++ Header Files and Peripheral Examples package in controlSUITE 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 1: A Sample of How to Add "Include" Files

```
#include "F2806x_Device.h"            // F2806x Headerfile
#include "F2806x_Examples.h"          // F2806x Examples Headerfile
#include "HCCal_Type0_V1.h"
```

Step 2: HRCAP Register Array Declaration

Declare an array of pointers to HRCAP_REGS structures which includes all available HRCAP modules on the device. Position 0 includes a 0 value which is not used by the HRCAP_Cal function.

Example 2: A Sample of How to Declare an HRCAP Register Array

```
#define NUM_HRCAP 5        // # of HRCAP modules on 2806x + 1

volatile struct HRCAP_REGS *HRCAP[NUM_HRCAP] = {0, &HRCap1Regs,
      &HRCap2Regs, &HRCap3Regs, &HRCap4Regs};
```

Step 3: HRCAP Pre-Calibration

Prior to using the HRCAP in high-resolution mode in application code, HRCAP_Cal() should be called to calibrate the HRCAP step size subdivision into HCCAPCLK.

As part of the one-time pre-calibration prior to using the HRCAP in high-resolution mode, include the following:

Example 3: A Sample of HRCAP Pre-Calibration

```
while (status!= HCCAL_COMPLETE)  // While calibration is incomplete
  {
                                 // Use HRCAP2 to calibrate with:
                                 // HCCAPCLK = PLL2CLK
                                 // ePWM8A = HRCAP calibration input
       status = HRCAP_Cal(2,HCCAPCLK_PLLCLK, &EPwm8Regs);
       if (status == HCCAL_ERROR)
       {
       ESTOP0;               // Error, stop and check HCCAPCLK frequency
       }
   }
```

Step 4: Application Code Calibration

While the application is running, fluctuations in both device temperature and supply voltage may be expected. To be sure that optimal HRCAP step size is used for each HRCAP module, the HRCAP_Cal function should be re-run periodically as part of a slower back-ground loop. Some examples of this are shown here.

NOTE: See the hrcap_capture_hrpwm example in the device-specific C/C++ header files and peripheral examples available in controlSUITE from the TI website

Example 4: HRCAP_Cal Function Calls

```
main ()
{
    int status;

    // User code
    // HRCAP 1, 3, and 4 are running in high-resolution mode
    // The status variable returns 2 once calibration has been
    // completed by the HCCal Calibration Module running
    // diagnostics.

    status = HRCAP_Cal(2,HCCAPCLK_PLLCLK, &EPwm8Regs);

    // The function returns a 2 if HCCAPCLK is not within the
    // appropriate frequency range.
    if(status==HCCAL_ERROR) {ESTOP0;}
}
```

Step 5: Application Code Pulse Width Measurement

While the application is running, when a RISE or FALL event occurs, pulse and period widths can be measured using the high resolution pulse width functions as shown below.

Example 5: LowPulseWidth, HighPulseWidth, and PeriodWidth Function Calls

```
interrupt void HRCAP1_Isr (void)
{
    EALLOW;
    if (HRCap1Regs.HCIFR.bit.RISEOVF == 1) {
        ESTOP0;         // Another rising edge detected
     }
        if (first < 1) {
            first++;            // Discard first data (because first interrupt
                                // after reset/clk enable measures time from
                                // clock start to edge - invalid pulse width)
    } else {

                  periodwidth =    PeriodWidthRise0((Uint16 *)&HRCap1Regs);
                  pulsewidthlow =  LowPulseWidth0((Uint16 *)&HRCap1Regs);
                  pulsewidthhigh = HighPulseWidth0((Uint16 *)&HRCap1Regs);
}

    HRCap1Regs.HCICLR.bit.RISE=1;
    HRCap1Regs.HCICLR.bit.INT=1;
    PieCtrlRegs.PIEACK.bit.ACK4=1;
    EDIS;
}
```

Enhanced Capture (eCAP) Module

The enhanced capture (eCAP) module is used in systems where accurate timing of external events is important. This guide describes the TMS320x280x Piccolo™ Enhanced Capture (eCAP) Module and how to use it.

The eCAP module described in this reference guide is a Type 0 eCAP. See the *TMS320C28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566) for a list of all devices with a eCAP module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

Topic **Page**

6.1 Introduction

Uses for eCAP include:

- Speed measurements of rotating machinery (e.g., toothed sprockets sensed via 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 described in this guide includes the following features:

- 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 dedicated to a single input pin
- When not used in capture mode, the ECAP module can be configured as a single channel PWM output

6.2 Description

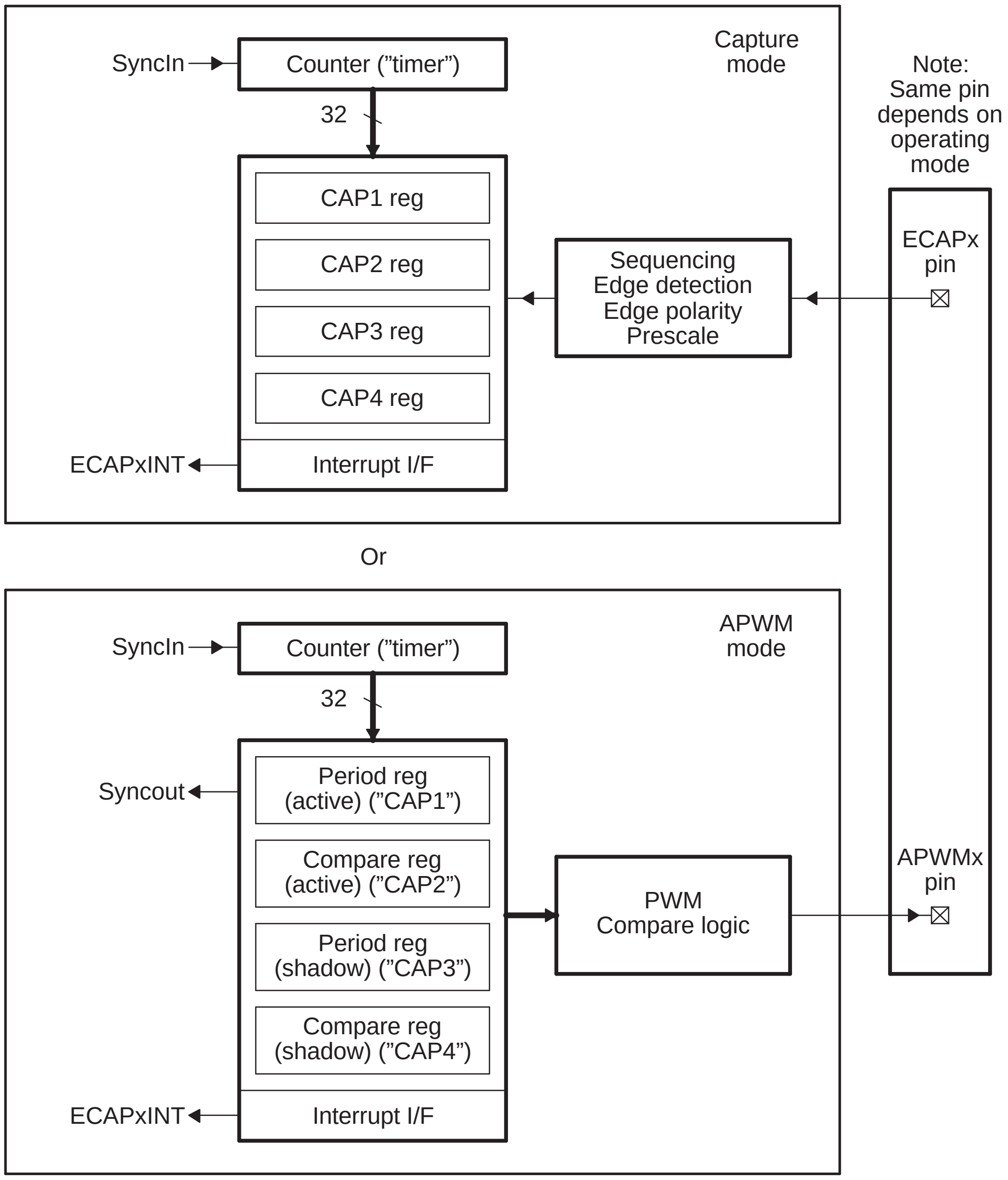
The eCAP module represents one complete capture channel that can be instantiated multiple times depending on the target device. In the context of this guide, one eCAP channel has the following independent key resources:

- Dedicated input capture pin
- 32-bit time base (counter)
- 4 x 32-bit time-stamp capture registers (CAP1-CAP4)
- 4-stage sequencer (Modulo4 counter) that is synchronized to external events, ECAP pin rising/falling edges.
- Independent edge polarity (rising/falling edge) selection for all 4 events
- Input capture signal prescaling (from 2-62)
- One-shot compare register (2 bits) to freeze captures after 1 to 4 time-stamp events
- Control for continuous time-stamp captures using a 4-deep circular buffer (CAP1-CAP4) scheme
- Interrupt capabilities on any of the 4 capture events

6.3 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 capture shadow registers, respectively. Figure 6-1 is a high-level view of both the capture and auxiliary pulse-width modulator (APWM) modes of operation.

Figure 6-1. Capture and APWM Modes of Operation



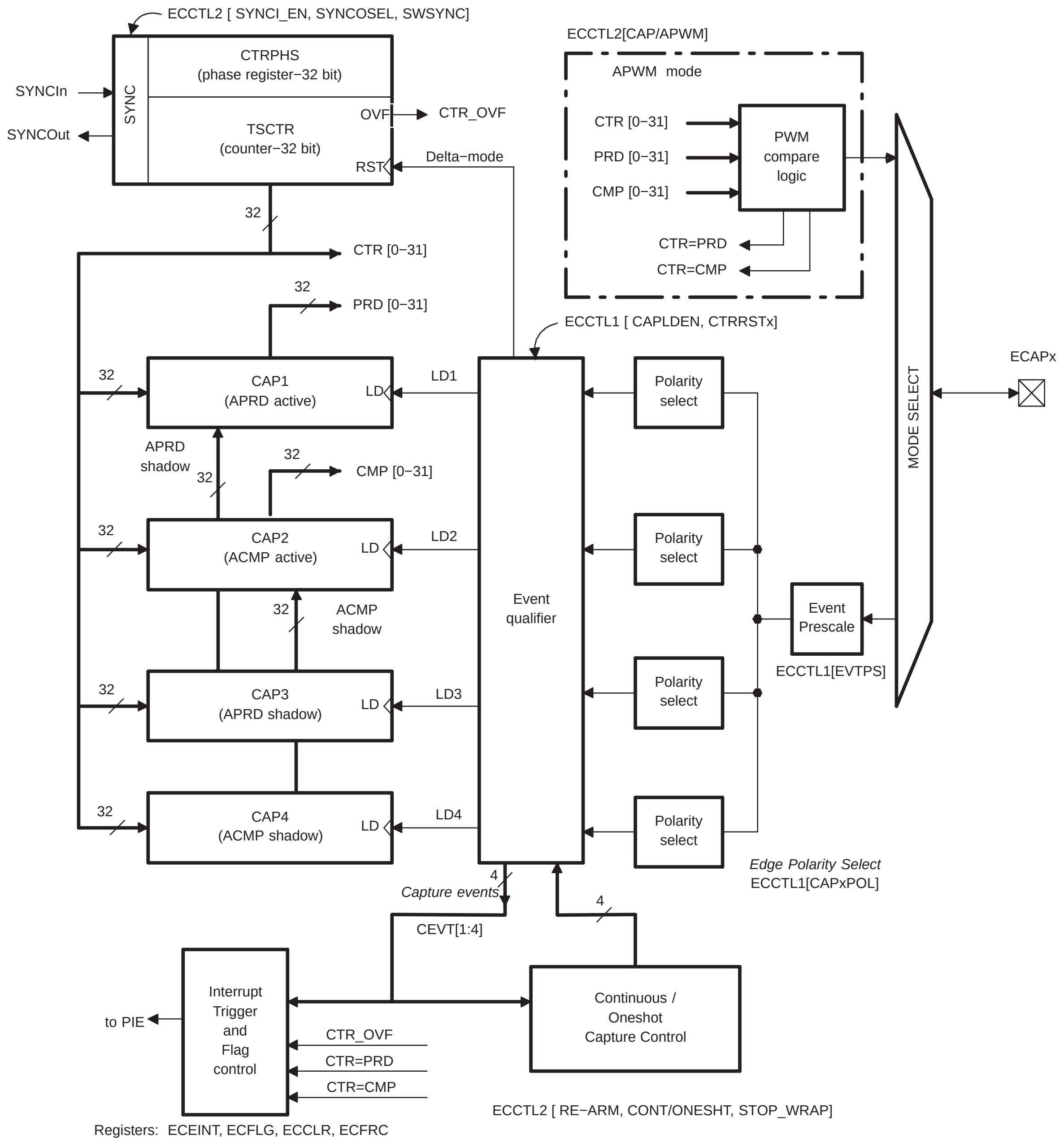
A A single pin is shared between CAP and APWM functions. In capture mode, it is an input; in APWM mode, it is an output.

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.

6.4 Capture Mode Description

Figure 6-2 shows the various components that implement the capture function.

Figure 6-2. Capture Function Diagram

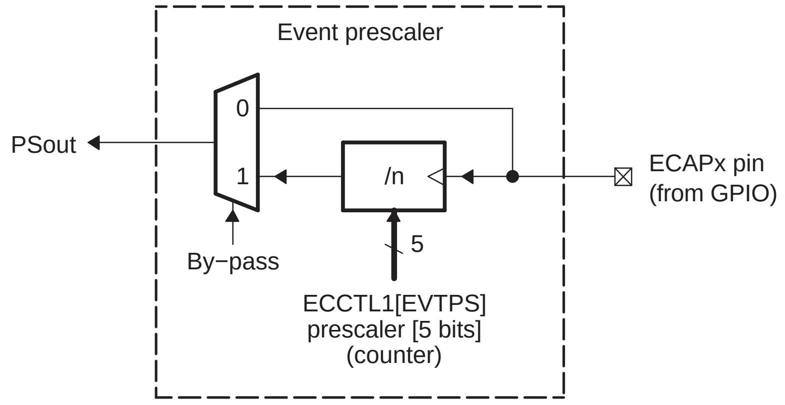


6.4.1 Event Prescaler

- An input capture signal (pulse train) can be prescaled by N = 2-62 (in multiples of 2) or can bypass the prescaler.

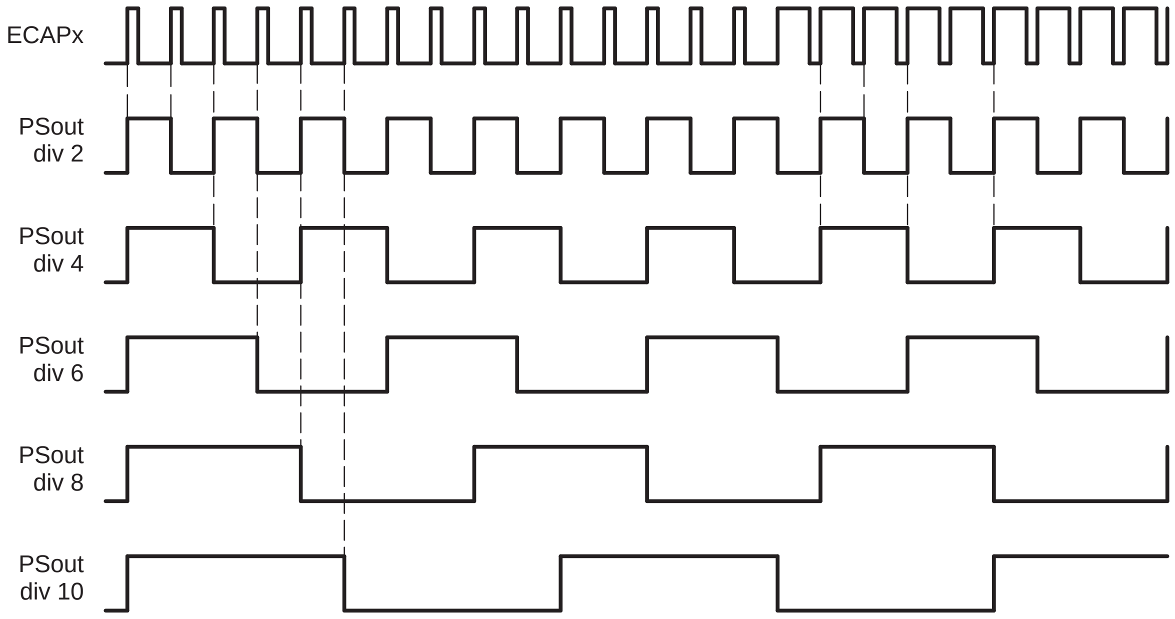
  This is useful when very high frequency signals are used as inputs. Figure 6-3 shows a functional diagram and Figure 6-4 shows the operation of the prescale function.

Figure 6-3. Event Prescale Control



A When a prescale value of 1 is chosen (i.e. ECCTL1[13:9] = 0,0,0,0,0) the input capture signal by-passes the prescale logic completely.

Figure 6-4. Prescale Function Waveforms



6.4.2 Edge Polarity Select and Qualifier

- 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.

6.4.3 Continuous/One-Shot Control

- 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.
- A 2-bit stop register is used to compare the Mod4 counter output, and when equal stops the Mod4 counter and inhibits further loads of the CAP1-CAP4 registers. This occurs during one-shot operation.

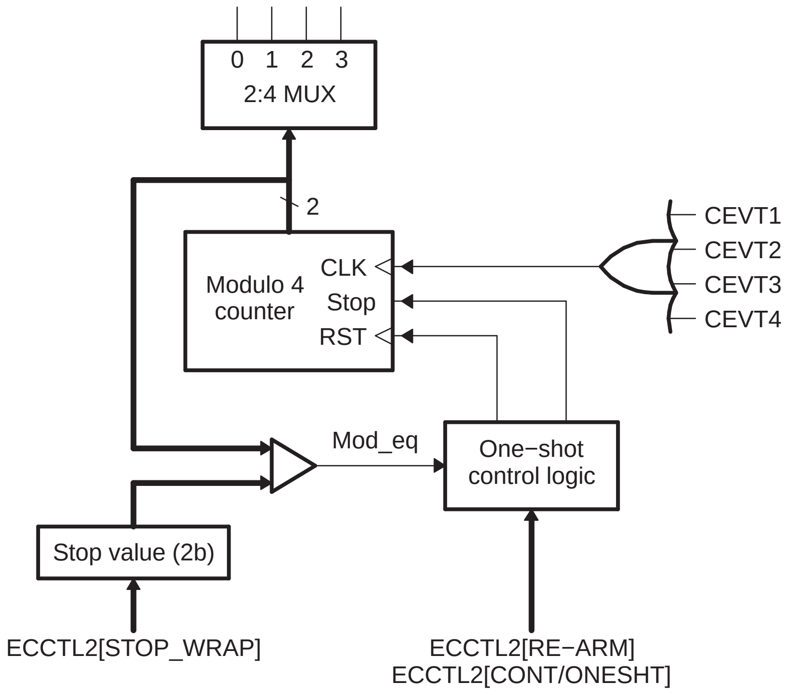
The continuous/one-shot block controls the start/stop and reset (zero) functions of the Mod4 counter via a mono-shot type of action that can be triggered by the stop-value comparator and re-armed via software control.

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 (i.e., 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 6-5. Details of the Continuous/One-shot Block



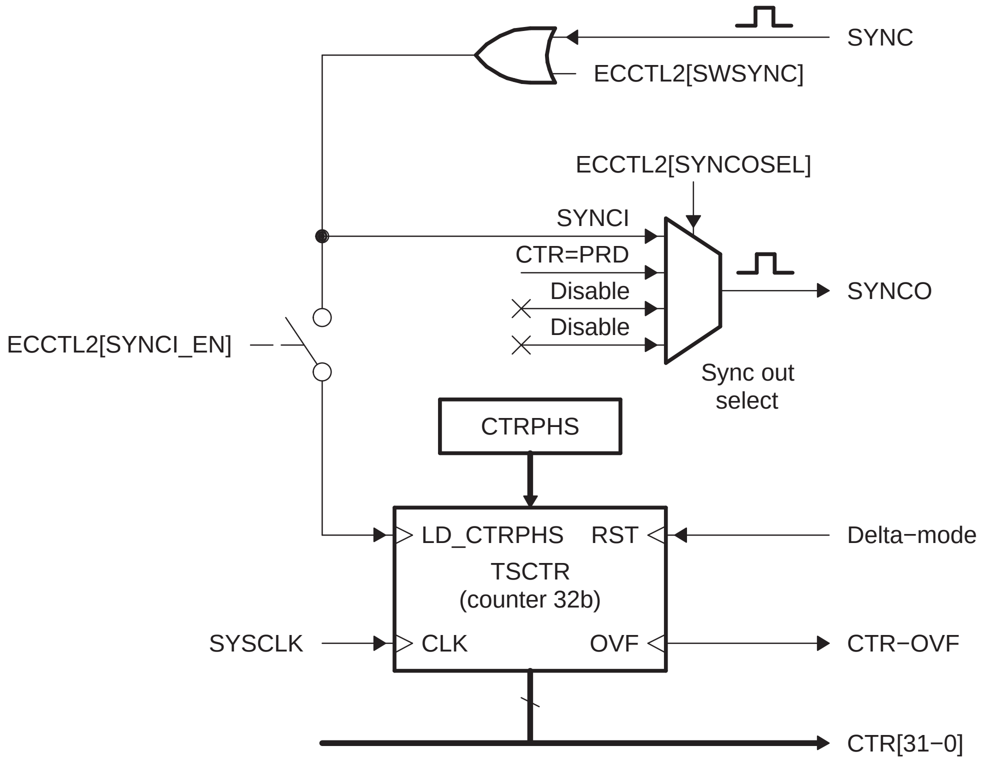
6.4.4 32-Bit Counter and Phase Control

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 6-6. Details of the Counter and Synchronization Block



6.4.5 CAP1-CAP4 Registers

These 32-bit registers are fed by the 32-bit counter timer bus, CTR[0-31] and are loaded (i.e., capture a time-stamp) when their respective LD inputs are strobed.

Loading of the capture registers can be inhibited via control bit CAPLDEN. During one-shot operation, this bit is cleared (loading is inhibited) automatically when a stop condition occurs, i.e. 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.

6.4.6 Interrupt Control

An Interrupt can be generated on capture events (CEVT1-CEVT4, CTROVF) or APWM events (CTR = PRD, CTR = CMP).

A counter overflow event (FFFFFFFF->00000000) is also provided as an interrupt source (CTROVF).

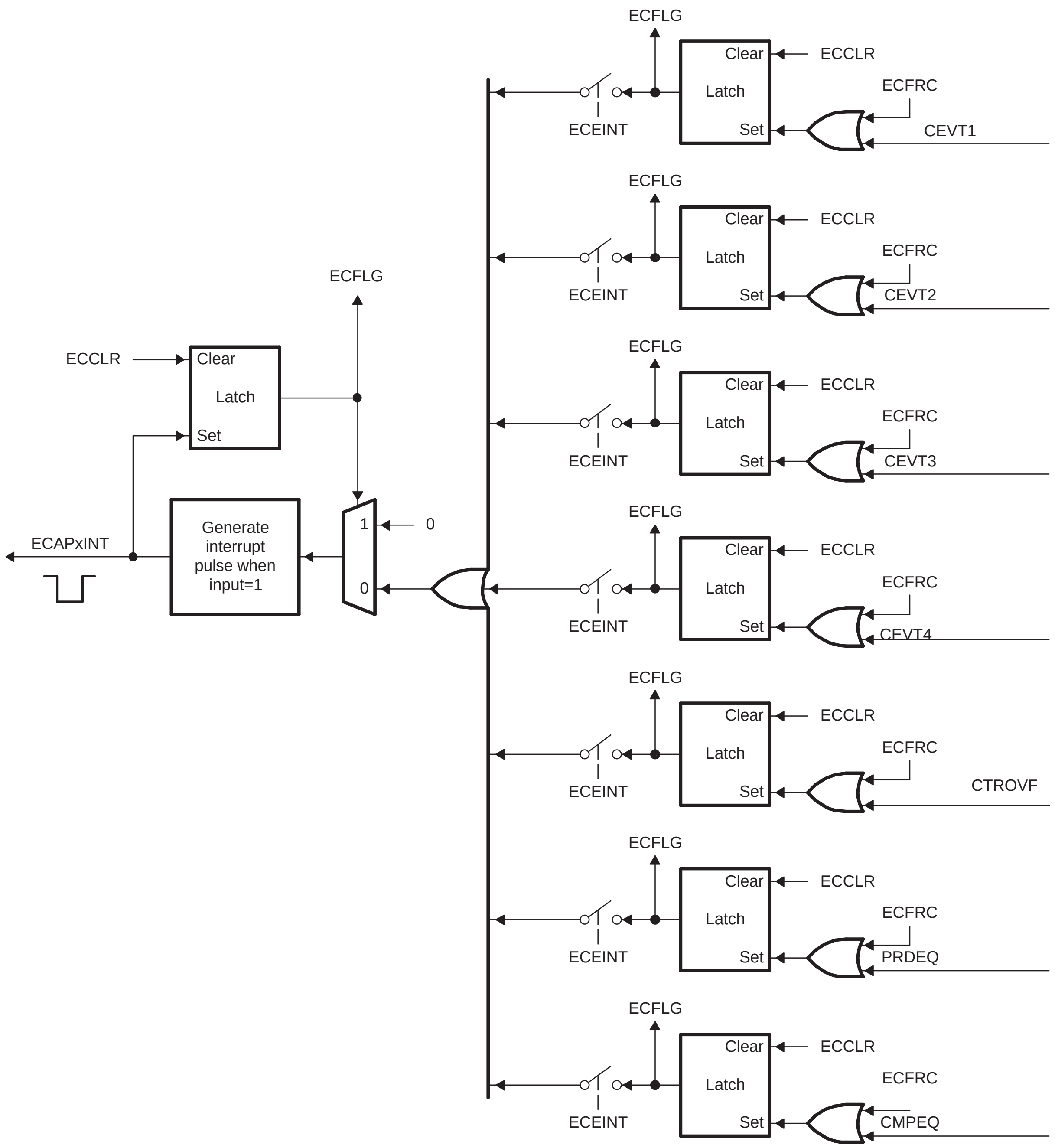
The capture events are edge and sequencer qualified (i.e., ordered in time) by the polarity select and Mod4 gating, respectively.

One of these events can be selected as the interrupt source (from the eCAPx module) going to the PIE.

Seven interrupt events (CEVT1, CEVT2, CEVT3, CEVT4, CNTOVF, CTR=PRD, CTR=CMP) can be generated. The interrupt enable register (ECEINT) is used to enable/disable individual interrupt event sources. The interrupt flag register (ECFLG) indicates if any interrupt event has been latched and contains the global interrupt flag bit (INT). An interrupt pulse is generated to the PIE only if any of the interrupt events are enabled, the flag bit is 1, and the INT flag bit is 0. The interrupt service routine must clear the global interrupt flag bit and the serviced event via the interrupt clear register (ECCLR) before any other interrupt pulses are generated. You can force an interrupt event via the interrupt force register (ECFRC). This is useful for test purposes.

Note: The CEVT1, CEVT2, CEVT3, CEVT4 flags are only active in capture mode (ECCTL2[CAP/APWM == 0]). The CTR=PRD, CTR=CMP flags are only valid in APWM mode (ECCTL2[CAP/APWM == 1]). CNTOVF flag is valid in both modes.

Figure 6-7. Interrupts in eCAP Module



6.4.7 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:

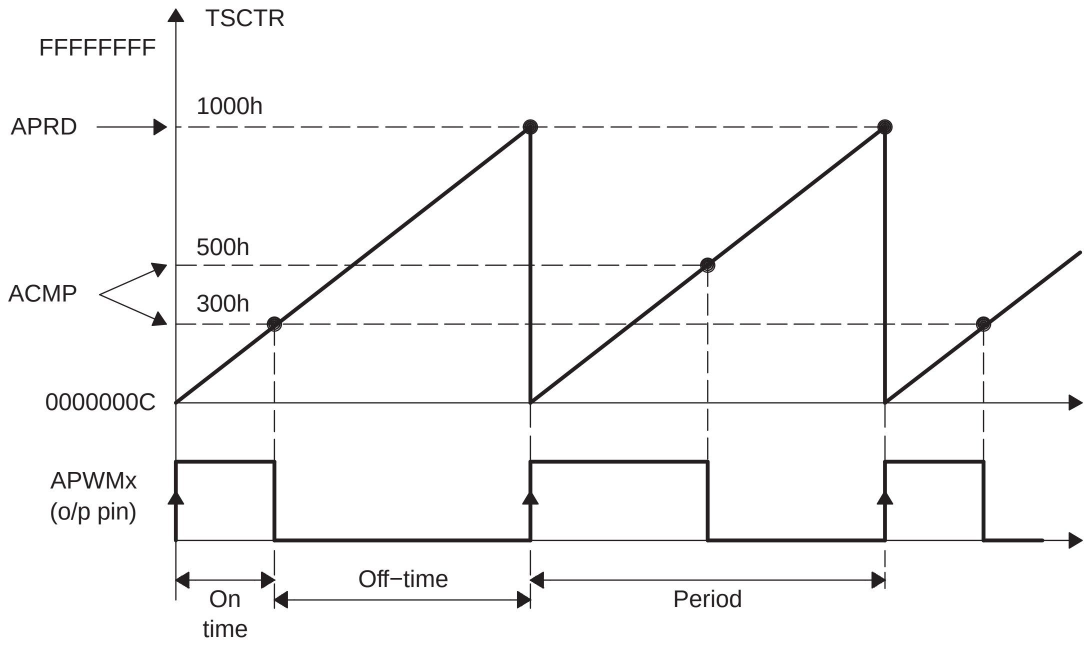
- Immediate - APRD or ACMP are transferred to CAP1 or CAP2 immediately upon writing a new value.
- On period equal, i.e., CTR[31:0] = PRD[31:0]

6.4.8 APWM Mode Operation

Main operating highlights of the APWM section:

- The time-stamp counter bus is made available for comparison via 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, i.e., during run-time, you only need to use the shadow registers.

Figure 6-8. 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
```

6.5 Capture Module - Control and Status Registers

Figure 6-9. Time-Stamp Counter Register (TSCTR)

31 … 0
TSCTR
R/W-0

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

Table 6-1. Time-Stamp Counter Register (TSCTR) Field Descriptions

Bit(s)	Field	Description
31:0	TSCTR	Active 32-bit counter register that is used as the capture time-base

Figure 6-10. Counter Phase Control Register (CTRPHS)

31 … 0
CTRPHS
R/W-0

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

Table 6-2. Counter Phase Control Register (CTRPHS) Field Descriptions

Bit(s)	Field	Description
31:0	CTRPHS	Counter phase value register that can be programmed for phase lag/lead. This register shadows TSCTR and 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 time-bases.

Figure 6-11. Capture-1 Register (CAP1)

31 … 0
CAP1
R/W-0

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

Table 6-3. Capture-1 Register (CAP1) Field Descriptions

Bit(s)	Field	Description
31:0	CAP1	This register can be loaded (written) by :) Time-Stamp (i.e., counter value TSCTR) during a capture event) Software - may be useful for test purposes / initialization) APRD shadow register (i.e., CAP3) when used in APWM mode

Figure 6-12. Capture-2 Register (CAP2)

31 … 0
CAP2
R/W-0

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

Table 6-4. Capture-2 Register (CAP2) Field Descriptions

Bit(s)	Field	Description
31:0	CAP2	This register can be loaded (written) by: • Time-Stamp (i.e., counter value) during a capture event • Software - may be useful for test purposes • APRD shadow register (i.e., CAP4) when used in APWM mode

NOTE: In APWM mode, writing 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 6-13. Capture-3 Register (CAP3)

31 … 0
CAP3
R/W-0

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

Table 6-5. Capture-3 Register (CAP3) Field Descriptions

Bit(s)	Field	Description
31:0	CAP3	In CMP mode, this is a time-stamp capture register. In APWM mode, this is the period shadow (APRD) register. You update the PWM period value through this register. In this mode, CAP3 (APRD) shadows CAP1.

Figure 6-14. Capture-4 Register (CAP4)

31 … 0
CAP4
R/W-0

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

Table 6-6. Capture-4 Register (CAP4) Field Descriptions

Bit(s)	Field	Description
31:0	CAP4	In CMP mode, this is a time-stamp capture register. In APWM mode, this is the compare shadow (ACMP) register. You update the PWM compare value via this register. In this mode, CAP4 (ACMP) shadows CAP2.

Figure 6-15. ECAP Control Register 1 (ECCTL1)

15 14	13 … 9	8
FREE/SOFT	PRESCALE	CAPLDEN
R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
CTRRST4	CAP4POL	CTRRST3	CAP3POL	CTRRST2	CAP2POL	CTRRST1	CAP1POL
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. ECAP Control Register 1 (ECCTL1) Field Descriptions

Bit(s)	Field	Value	Description
15:14	FREE/SOFT		Emulation Control
		00	TSCTR counter stops immediately on emulation suspend
		01	TSCTR counter runs until = 0
		1x	TSCTR counter is unaffected by emulation suspend (Run Free)
13:9	PRESCALE		Event Filter prescale select
		00000	Divide by 1 (ino prescale, by-pass the prescaler)
		00001	Divide by 2
		00010	Divide by 4
		00011	Divide by 6

Table 6-7. ECAP Control Register 1 (ECCTL1) Field Descriptions (continued)

Bit(s)	Field	Value	Description
		00100	Divide by 8
		00101	Divide by 10
		...	
		11110	Divide by 60
		11111	Divide by 62
8	CAPLDEN		Enable Loading of CAP1-4 registers on a capture event
		0	Disable CAP1-4 register loads at capture event time.
		1	Enable CAP1-4 register loads at capture event time.
7	CTRRST4		Counter Reset on Capture Event 4
		0	*Do not* reset counter on Capture Event 4 (absolute time stamp operation)
		1	Reset counter after Capture Event 4 time-stamp has been captured (used in difference mode operation)
6	CAP4POL		Capture Event 4 Polarity select
		0	Capture Event 4 triggered on a rising edge (RE)
		1	Capture Event 4 triggered on a falling edge (FE)
5	CTRRST3		Counter Reset on Capture Event 3
		0	*Do not* reset counter on Capture Event 3 (absolute time stamp)
		1	Reset counter after Event 3 time-stamp has been captured (used in difference mode operation)
4	CAP3POL		Capture Event 3 Polarity select
		0	Capture Event 3 triggered on a rising edge (RE)
		1	Capture Event 3 triggered on a falling edge (FE)
3	CTRRST2		Counter Reset on Capture Event 2
		0	*Do not* reset counter on Capture Event 2 (absolute time stamp)
		1	Reset counter after Event 2 time-stamp has been captured (used in difference mode operation)
2	CAP2POL		Capture Event 2 Polarity select
		0	Capture Event 2 triggered on a rising edge (RE)
		1	Capture Event 2 triggered on a falling edge (FE)
1	CTRRST1		Counter Reset on Capture Event 1
		0	*Do not* reset counter on Capture Event 1 (absolute time stamp)
		1	Reset counter after Event 1 time-stamp has been captured (used in difference mode operation)
0	CAP1POL		Capture Event 1 Polarity select
		0	Capture Event 1 triggered on a rising edge (RE)
		1	Capture Event 1 triggered on a falling edge (FE)

Figure 6-16. ECAP Control Register 2 (ECCTL2)

15	11	10	9	8
Reserved		APWMPOL	CAP/APWM	SWSYNC
R-0		R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
SYNCO_SEL		SYNCI_EN	TSCTRSTOP	REARM	STOP_WRAP		CONT/ONESHT
R/W-0		R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-0

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

Table 6-8. ECAP Control Register 2 (ECCTL2) Field Descriptions

Bit(s)	Field		Description
15:11	Reserved		Reserved
10	APWMPOL		APWM output polarity select. This is applicable only in APWM operating mode
		0	Output is active high (i.e., Compare value defines high time)
		1	Output is active low (i.e., Compare value defines low time)
9	CAP/APWM		CAP/APWM operating mode select
		0	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
		1	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		Software-forced Counter (TSCTR) Synchronizing. This provides a convenient software method to synchronize some or all ECAP time bases. In APWM mode, the synchronizing can also be done via the CTR = PRD event.
		0	Writing a zero has no effect. Reading always returns a zero
		1	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		Sync-Out Select
		00	Select sync-in event to be the sync-out signal (pass through)
		01	Select CTR = PRD event to be the sync-out signal
		10	Disable sync out signal
		11	Disable sync out signal
5	SYNCI_EN		Counter (TSCTR) Sync-In select mode
		0	Disable sync-in option
		1	Enable counter (TSCTR) to be loaded from CTRPHS register upon either a SYNCI signal or a S/W force event.
4	TSCTRSTOP		Time Stamp (TSCTR) Counter Stop (freeze) Control
		0	TSCTR stopped
		1	TSCTR free-running
3	RE-ARM		Re-Arming Control. Note: The re-arm function is valid in one shot or continuous mode.
		0	Has no effect (reading always returns a 0)
		1	Arms the sequence as follows: 1) Resets the Mod4 counter to zero 2) Unfreezes the Mod4 counter 3) Enables capture register loads
2:1	STOP_WRAP		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, i.e., 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.
		00	Stop after Capture Event 1 in one-shot mode Wrap after Capture Event 1 in continuous mode.
		01	Stop after Capture Event 2 in one-shot mode Wrap after Capture Event 2 in continuous mode.

Table 6-8. ECAP Control Register 2 (ECCTL2) Field Descriptions (continued)

Bit(s)	Field		Description
		10	Stop after Capture Event 3 in one-shot mode Wrap after Capture Event 3 in continuous mode.
		11	Stop after Capture Event 4 in one-shot mode Wrap after Capture Event 4 in continuous mode.
			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.
0	CONT/ONESHT		Continuous or one-shot mode control (applicable only in capture mode)
		0	Operate in continuous mode
		1	Operate in one-Shot mode

Figure 6-17. ECAP Interrupt Enable Register (ECEINT)

15							8
Reserved							

7	6	5	4	3	2	1	0
CTR=CMP	CTR=PRD	CTROVF	CEVT4	CEVT3	CEVT2	CETV1	Reserved
		R/W	R/W	R/W	R/W	R/W	

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

Table 6-9. ECAP Interrupt Enable Register (ECEINT) Field Descriptions

Bits	Field	Value	Description
15:8	Reserved		
7	CTR=CMP		Counter Equal Compare Interrupt Enable
		0	Disable Compare Equal as an Interrupt source
		1	Enable Compare Equal as an Interrupt source
6	CTR=PRD		Counter Equal Period Interrupt Enable
		0	Disable Period Equal as an Interrupt source
		1	Enable Period Equal as an Interrupt source
5	CTROVF		Counter Overflow Interrupt Enable
		0	Disabled counter Overflow as an Interrupt source
		1	Enable counter Overflow as an Interrupt source
4	CEVT4		Capture Event 4 Interrupt Enable
		0	Disable Capture Event 4 as an Interrupt source
		1	Capture Event 4 Interrupt Enable
3	CEVT3		Capture Event 3 Interrupt Enable
		0	Disable Capture Event 3 as an Interrupt source
		1	Enable Capture Event 3 as an Interrupt source
2	CEVT2		Capture Event 2 Interrupt Enable
		0	Disable Capture Event 2 as an Interrupt source
		1	Enable Capture Event 2 as an Interrupt source
1	CEVT1		Capture Event 1 Interrupt Enable
		0	Disable Capture Event 1 as an Interrupt source
		1	Enable Capture Event 1 as an Interrupt source
0	Reserved		

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:

- Disable global interrupts
- Stop eCAP counter
- Disable eCAP interrupts
- Configure peripheral registers
- Clear spurious eCAP interrupt flags
- Enable eCAP interrupts
- Start eCAP counter
- Enable global interrupts

Figure 6-18. ECAP Interrupt Flag Register (ECFLG)

15–8
Reserved
R-0

7	6	5	4	3	2	1	0
CTR=CMP	CTR=PRD	CTROVF	CEVT4	CETV3	CEVT2	CETV1	INT
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 6-10. ECAP Interrupt Flag Register (ECFLG) Field Descriptions

Bits	Field	Value	Description
15:8	Reserved		
7	CTR=CMP		Compare Equal Compare Status Flag. This flag is active only in APWM mode.
		0	Indicates no event occurred
		1	Indicates the counter (TSCTR) reached the compare register value (ACMP)
6	CTR=PRD		Counter Equal Period Status Flag. This flag is only active in APWM mode.
		0	Indicates no event occurred
		1	Indicates the counter (TSCTR) reached the period register value (APRD) and was reset.
5	CTROVF		Counter Overflow Status Flag. This flag is active in CAP and APWM mode.
		0	Indicates no event occurred.
		1	Indicates the counter (TSCTR) has made the transition from FFFFFFFF " 00000000
4	CEVT4		Capture Event 4 Status Flag This flag is only active in CAP mode.
		0	Indicates no event occurred
		1	Indicates the fourth event occurred at ECAPx pin
3	CEVT3		Capture Event 3 Status Flag. This flag is active only in CAP mode.
		0	Indicates no event occurred.
		1	Indicates the third event occurred at ECAPx pin.
2	CEVT2		Capture Event 2 Status Flag. This flag is only active in CAP mode.
		0	Indicates no event occurred.
		1	Indicates the second event occurred at ECAPx pin.
1	CEVT1		Capture Event 1 Status Flag. This flag is only active in CAP mode.
		0	Indicates no event occurred.
		1	Indicates the first event occurred at ECAPx pin.
0	INT		Global Interrupt Status Flag
		0	Indicates no interrupt generated.
		1	Indicates that an interrupt was generated.

Figure 6-19. ECAP Interrupt Clear Register (ECCLR)

15–8
Reserved
R-0

7	6	5	4	3	2	1	0
CTR=CMP	CTR=PRD	CTROVF	CEVT4	CETV3	CETV2	CETV1	INT
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 6-11. ECAP Interrupt Clear Register (ECCLR) Field Descriptions

Bits	Field		Description
15:8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	CTR=CMP		Counter Equal Compare Status Flag
		0	Writing a 0 has no effect. Always reads back a 0
		1	Writing a 1 clears the CTR=CMP flag condition
6	CTR=PRD		Counter Equal Period Status Flag
		0	Writing a 0 has no effect. Always reads back a 0
		1	Writing a 1 clears the CTR=PRD flag condition
5	CTROVF		Counter Overflow Status Flag
		0	Writing a 0 has no effect. Always reads back a 0
		1	Writing a 1 clears the CTROVF flag condition
4	CEVT4		Capture Event 4 Status Flag
		0	Writing a 0 has no effect. Always reads back a 0.
		1	Writing a 1 clears the CEVT4 flag condition.
3	CEVT3		Capture Event 3 Status Flag
		0	Writing a 0 has no effect. Always reads back a 0.
		1	Writing a 1 clears the CEVT3 flag condition.
2	CEVT2		Capture Event 2 Status Flag
		1	Writing a 0 has no effect. Always reads back a 0.
		0	Writing a 1 clears the CEVT2 flag condition.
1	CEVT1		Capture Event 1 Status Flag
		0	Writing a 0 has no effect. Always reads back a 0.
		1	Writing a 1 clears the CEVT1 flag condition.
0	INT		Global Interrupt Clear Flag
		0	Writing a 0 has no effect. Always reads back a 0.
		1	Writing a 1 clears the INT flag and enable further interrupts to be generated if any of the event flags are set to 1.

Figure 6-20. ECAP Interrupt Forcing Register (ECFRC)

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

7	6	5	4	3	2	1	0
CTR=CMP	CTR=PRD	CTROVF	CEVT4	CETV3	CETV2	CETV1	reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 6-12. ECAP Interrupt Forcing Register (ECFRC) Field Descriptions

Bits	Field	Value	Description
15:8	Reserved	0	Any writes to these bit(s) must always have a value of 0.
7	CTR=CMP		Force Counter Equal Compare Interrupt
		0	No effect. Always reads back a 0.
		1	Writing a 1 sets the CTR=CMP flag bit.
6	CTR=PRD		Force Counter Equal Period Interrupt
		0	No effect. Always reads back a 0.
		1	Writing a 1 sets the CTR=PRD flag bit.

Table 6-12. ECAP Interrupt Forcing Register (ECFRC) Field Descriptions (continued)

Bits	Field	Value	Description
5	CTROVF		Force Counter Overflow
		0	No effect. Always reads back a 0.
		1	Writing a 1 to this bit sets the CTROVF flag bit.
4	CEVT4		Force Capture Event 4
		0	No effect. Always reads back a 0.
		1	Writing a 1 sets the CEVT4 flag bit
3	CEVT3		Force Capture Event 3
		0	No effect. Always reads back a 0.
		1	Writing a 1 sets the CEVT3 flag bit
2	CEVT2		Force Capture Event 2
		0	No effect. Always reads back a 0.
		1	Writing a 1 sets the CEVT2 flag bit.
1	CEVT1		Force Capture Event 1
		1	No effect. Always reads back a 0.
		0	Sets the CEVT1 flag bit.
0	Reserved	0	Any writes to these bit(s) must always have a value of 0.

6.6 Register Mapping

Table 6-13 shows the eCAP module control and status register set.

Table 6-13. Control and Status Register Set

Name	Offset	Size (x16)	Description
Time Base Module Registers			
TSCTR	0x0000	2	Time-Stamp Counter
CTRPHS	0x0002	2	Counter Phase Offset Value Register
CAP1	0x0004	2	Capture 1 Register
CAP2	0x0006	2	Capture 2 Register
CAP3	0x0008	2	Capture 3 Register
CAP4	0x000A	2	Capture 4 Register
reserved	0x000C - 0x0013	8	
ECCTL1	0x0014	1	Capture Control Register 1
ECCTL2	0x0015	1	Capture Control Register 2
ECEINT	0x0016	1	Capture Interrupt Enable Register
ECFLG	0x0017	1	Capture Interrupt Flag Register
ECCLR	0x0018	1	Capture Interrupt Clear Register
ECFRC	0x0019	1	Capture Interrupt Force Register
Reserved	0x001A - 0x001F	6	

6.7 Application of the ECAP Module

The following sections will provide Applications examples and code snippets to show how to configure and operate the eCAP module. For clarity and ease of use, the examples use the eCAP “C” header files. Below are useful #defines which will help in the understanding of the examples.

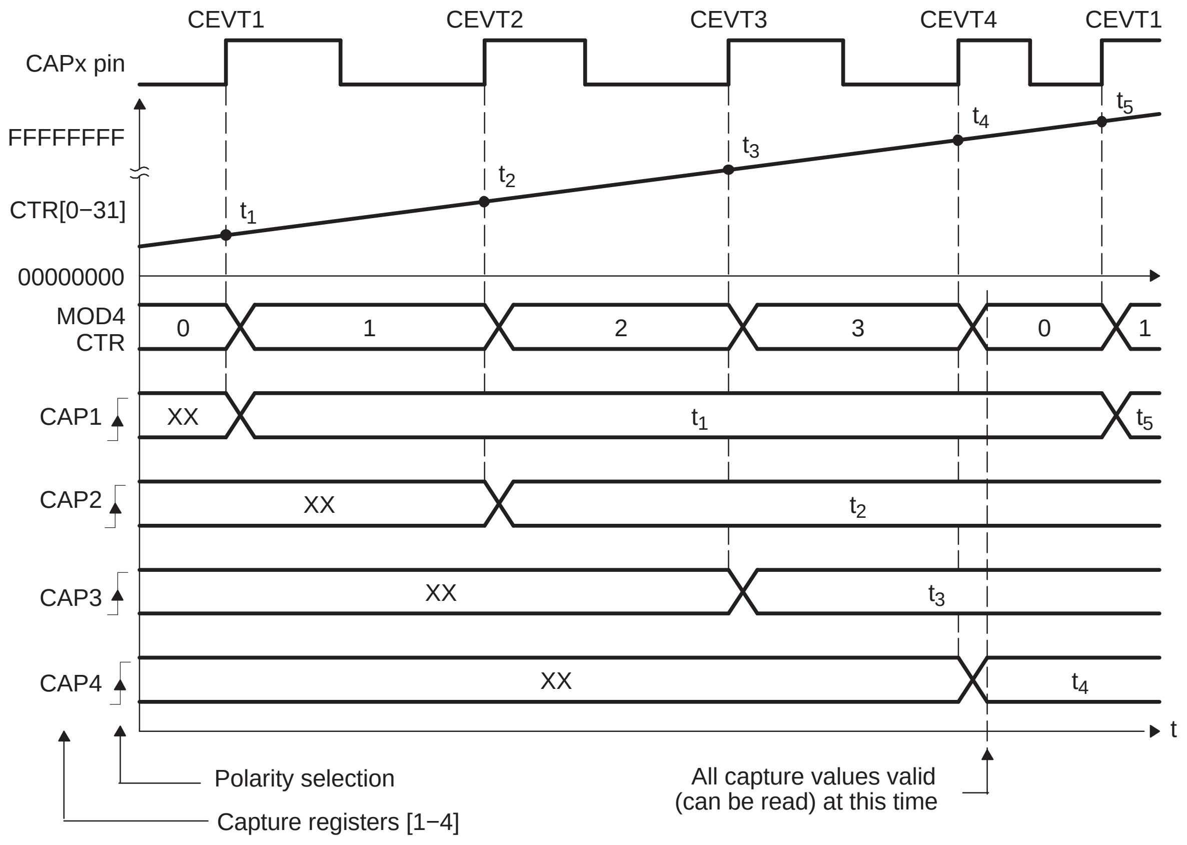
```
// ECCTL1 ( ECAP Control Reg 1)
//==========================

// CAPxPOL bits       #define EC_RISING 0x0   #define EC_FALLING 0x1
// CTRRSTx bits       #define EC_ABS_MODE 0x0 #define EC_DELTA_MODE 0x1
// PRESCALE bits      #define EC_BYPASS 0x0   #define EC_DIV1 0x0
                                              #define EC_DIV2 0x1
                                              #define EC_DIV4 0x2
                                              #define EC_DIV6 0x3
                                              #define EC_DIV8 0x4
                                              #define EC_DIV10 0x5
// ECCTL2 ( ECAP Control Reg 2)
//==========================
// CONT/ONESHOT bit                           #define EC_CONTINUOUS 0x0
                                              #define EC_ONESHOT 0x1
// STOPVALUE bit     #define EC_EVENT1 0x0    #define EC_EVENT2 0x1 #define EC_EVENT3 0x2
                                              #define EC_EVENT4 0x3
// RE-ARM bit        #define EC_ARM 0x1
// TSCTRSTOP bit     #define EC_FREEZE 0x0   #define EC_RUN 0x1
// SYNCO_SEL bit     #define EC_SYNCIN 0x0   #define EC_CTR_PRD 0x1
                                             #define EC_SYNCO_DIS 0x2
// CAP/APWM mode bit #define EC_CAP_MODE 0x0 #define EC_APWM_MODE 0x1
// APWMPOL bit       #define EC_ACTV_HI 0x0  #define EC_ACTV_LO 0x1
// Generic           #define EC_DISABLE 0x0  #define EC_ENABLE 0x1 #define EC_FORCE 0x1
```

6.7.1 Example 1 - Absolute Time-Stamp Operation Rising Edge Trigger

Figure 6-21 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 (i.e., time-stamp) is first captured, then Mod4 counter is incremented to the next state. When the TSCTR reaches FFFFFFFF (i.e. maximum value), it wraps around to 00000000 (not shown in Figure 6-21), 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, i.e. 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 6-21. Capture Sequence for Absolute Time-stamp and Rising Edge Detect



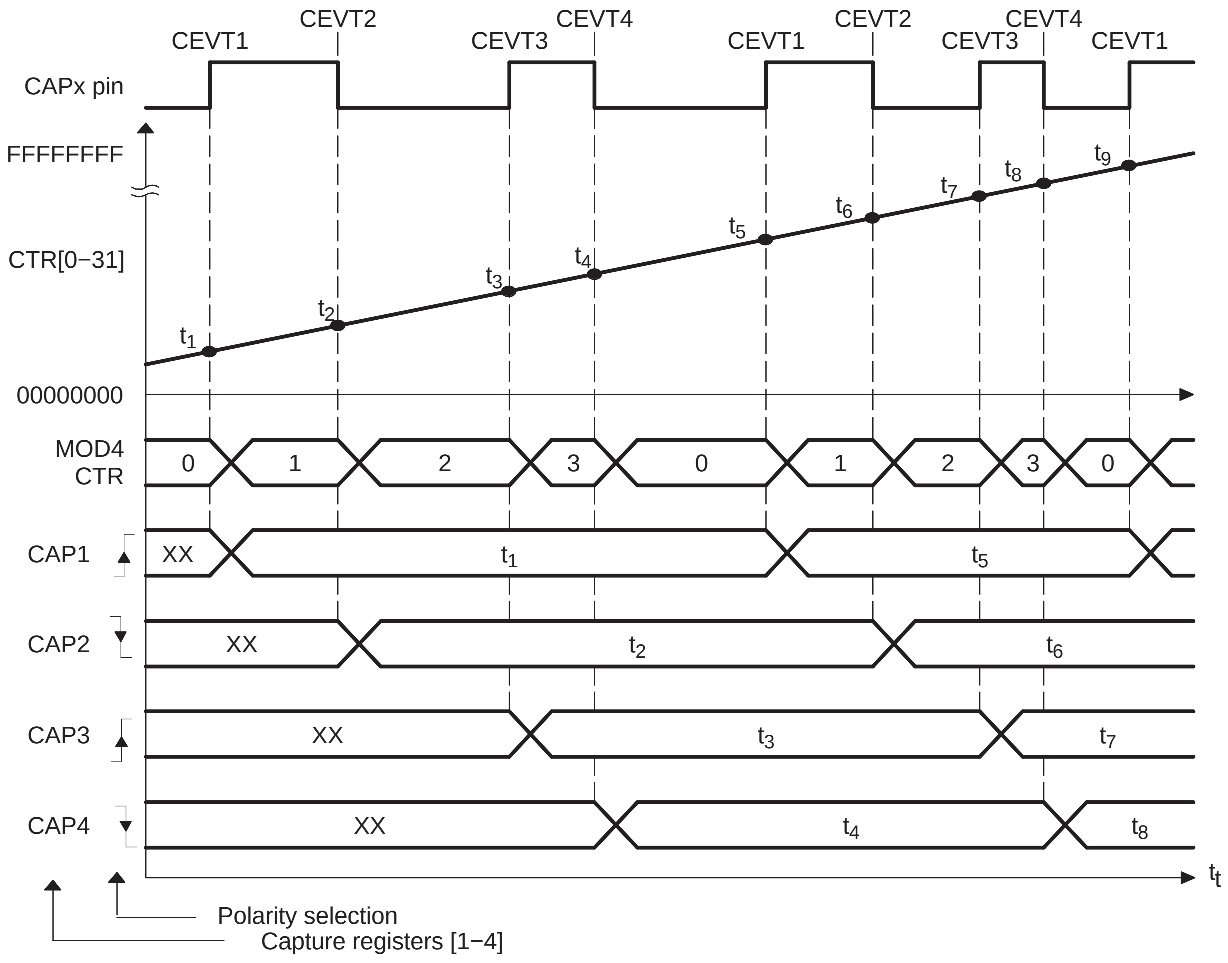
6.7.1.1 Code snippet for CAP mode Absolute Time, Rising Edge Trigger

```
// Code snippet for CAP mode Absolute Time, Rising edge trigger
// Initialization Time
//=======================

// ECAP module 1 config ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;

   ECap1Regs.ECCTL1.bit.CAP2POL = EC_RISING;
   ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
   ECap1Regs.ECCTL1.bit.CAP4POL = EC_RISING;
   ECap1Regs.ECCTL1.bit.CTRRST1 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST2 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST3 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST4 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
   ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
   ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
   ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
   ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
   ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
   ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN;

// Allow TSCTR to run

// Run Time (CEVT4 triggered ISR call)
//==========================================

TSt1 = ECap1Regs.CAP1;
// Fetch Time-Stamp captured at t1 TSt2 = ECap1Regs.CAP2;
// Fetch Time-Stamp captured at t2 TSt3 = ECap1Regs.CAP3;
// Fetch Time-Stamp captured at t3 TSt4 = ECap1Regs.CAP4;
// Fetch Time-Stamp captured at t4 Period1 = TSt2-TSt1;
// Calculate 1st period Period2 = TSt3-TSt2;
// Calculate 2nd period Period3 = TSt4-TSt3;
// Calculate 3rd period
```

6.7.2 Example 2 - Absolute Time-Stamp Operation Rising and Falling Edge Trigger

In Figure 6-22 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, i.e: Period1 = $t_3 – t_1$, Period2 = $t_5 – t_3$, …etc. Duty Cycle1 (on-time %) = ($t_2 – t_1$) / Period1 x 100%, etc. Duty Cycle1 (off-time %) = ($t_3 – t_2$) / Period1 x 100%, etc.

Figure 6-22. Capture Sequence for Absolute Time-stamp With Rising and Falling Edge Detect



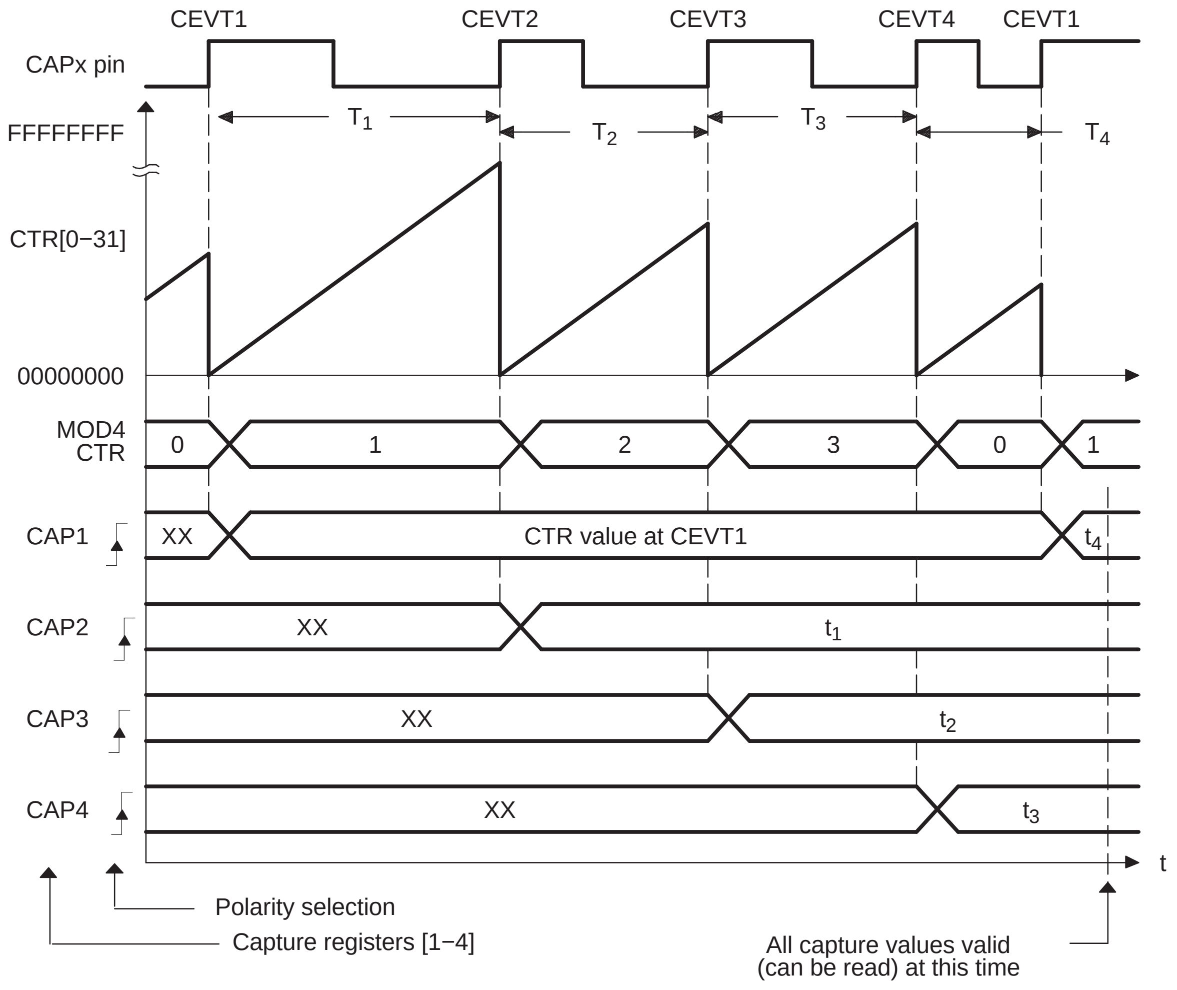
6.7.2.1 Code snippet for CAP mode Absolute Time, Rising & Falling Edge Triggers

```
// Code snippet for CAP mode Absolute Time, Rising and Falling
// edge triggers // Initialization Time
//=======================

// ECAP module 1 config ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
   ECap1Regs.ECCTL1.bit.CAP2POL = EC_FALLING;
   ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
   ECap1Regs.ECCTL1.bit.CAP4POL = EC_FALLING;
   ECap1Regs.ECCTL1.bit.CTRRST1 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST2 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST3 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST4 = EC_ABS_MODE;
   ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
   ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
   ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
   ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
   ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
   ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
   ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN;

// Allow TSCTR to run
// Run Time (CEVT4 triggered ISR call)

//==========================================
TSt1 = ECap1Regs.CAP1;
// Fetch Time-Stamp captured at t1 TSt2 = ECap1Regs.CAP2;
// Fetch Time-Stamp captured at t2 TSt3 = ECap1Regs.CAP3;
// Fetch Time-Stamp captured at t3 TSt4 = ECap1Regs.CAP4;
// Fetch Time-Stamp captured at t4 Period1 = TSt3-TSt1;
// Calculate 1st period DutyOnTime1 = TSt2-TSt1;
// Calculate On time DutyOffTime1 = TSt3-TSt2;
// Calculate Off time
```

6.7.3 Example 3 - Time Difference (Delta) Operation Rising Edge Trigger

This example Figure 6-23 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 (that is, 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 (i.e. Max value), before the next event, it wraps around to 00000000 and continues, a CNTOVF (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, i.e. Period1 = T_1, Period2 = T_2,…etc. As shown in the diagram, 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 6-23. Capture Sequence for Delta Mode Time-stamp and Rising Edge Detect



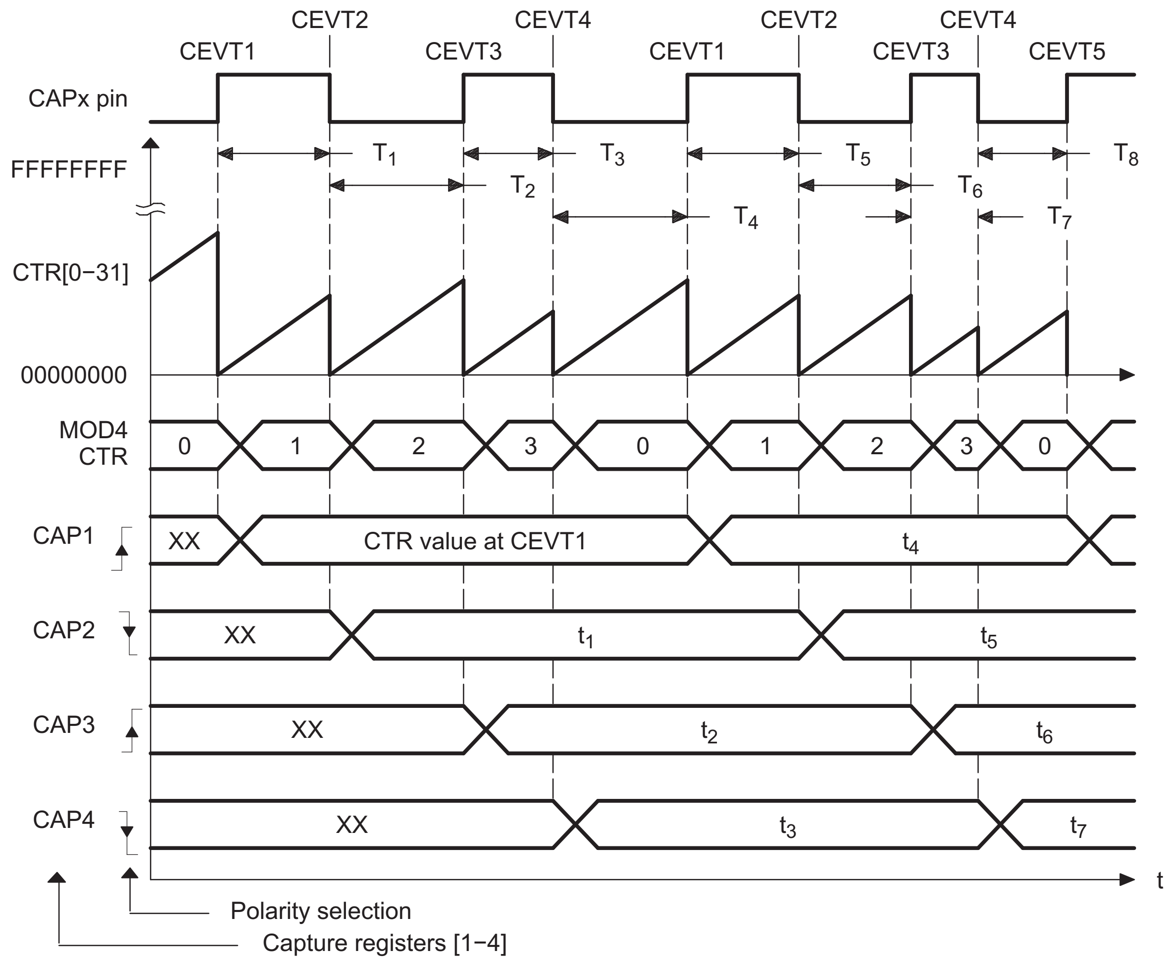
6.7.3.1 Code Snippet for CAP mode Delta Time, Rising Edge Trigger

```
// Code snippet for CAP mode Delta Time, Rising edge trigger
// Initialization Time

//=======================
// ECAP module 1 config ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP2POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP4POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CTRRST1 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST2 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST3 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST4 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN;

// Allow TSCTR to run
// Run Time (CEVT1 triggered ISR call)

//==========================================
// Note: here Time-stamp directly represents the Period value. Period4 = ECap1Regs.CAP1;
// Fetch Time-Stamp captured at T1 Period1 = ECap1Regs.CAP2;
// Fetch Time-Stamp captured at T2 Period2 = ECap1Regs.CAP3;
// Fetch Time-Stamp captured at T3 Period3 = ECap1Regs.CAP4;
// Fetch Time-Stamp captured at T4
```

6.7.4 Example 4 - Time Difference (Delta) Operation Rising and Falling Edge Trigger

In Figure 6-24 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, i.e: Period1 = T_1+T_2, Period2 = T_3+T_4, …etc Duty Cycle1 (on-time %) = T_1 / Period1 x 100%, etc Duty Cycle1 (off-time %) = T_2 / Period1 x 100%, etc

Figure 6-24. Capture Sequence for Delta Mode Time-stamp With Rising and Falling Edge Detect



During initialization, you must write to the active registers for both period and compare. This will then automatically copy the init values into the shadow values. For subsequent compare updates, i.e. during run-time, only the shadow registers must be used.

6.7.4.1 Code snippet for CAP mode Delta Time, Rising and Falling Edge Triggers

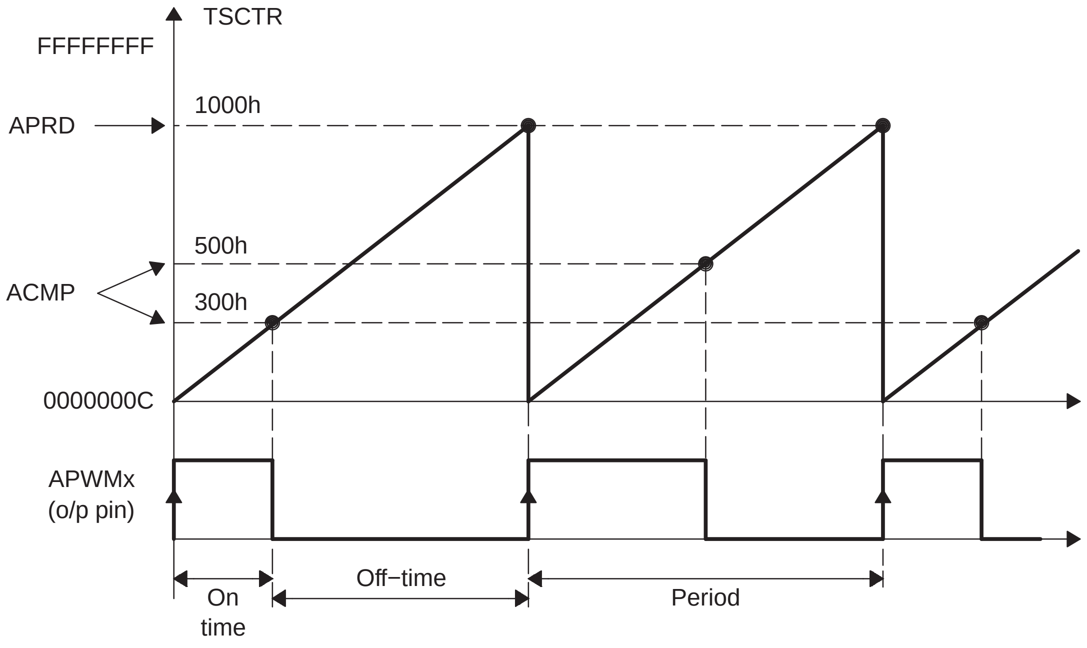
```
// Code snippet for CAP mode Delta Time, Rising and Falling
// edge triggers
// Initialization Time

//=======================
// ECAP module 1 config ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
   ECap1Regs.ECCTL1.bit.CAP2POL = EC_FALLING;
   ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
   ECap1Regs.ECCTL1.bit.CAP4POL = EC_FALLING;
   ECap1Regs.ECCTL1.bit.CTRRST1 = EC_DELTA_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST2 = EC_DELTA_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST3 = EC_DELTA_MODE;
   ECap1Regs.ECCTL1.bit.CTRRST4 = EC_DELTA_MODE;
   ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
   ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
   ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
   ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
   ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
   ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
   ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN;

// Allow TSCTR to run

// Run Time (CEVT1 triggered ISR call)
//==========================================
//
Note: here Time-stamp directly represents the Duty cycle values. DutyOnTime1 = ECap1Regs.CAP2;
// Fetch Time-Stamp captured at T2 DutyOffTime1 = ECap1Regs.CAP3;
// Fetch Time-Stamp captured at T3 DutyOnTime2 = ECap1Regs.CAP4;
// Fetch Time-Stamp captured at T4 DutyOffTime2 = ECap1Regs.CAP1;
// Fetch Time-
Stamp captured at T1 Period1 = DutyOnTime1 + DutyOffTime1; Period2 = DutyOnTime2 + DutyOffTime2;
```

6.8 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 output pin APWMx. 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. Note here values are in hexadecimal (“h”) notation.

6.8.1 Example 1 - Simple PWM Generation (Independent Channel/s)

Figure 6-25. PWM Waveform Details of APWM Mode Operation



Example 6-1. COh,ode Snippet for APWM Mode

```
// Code snippet for APWM mode Example 1

// Initialization Time

//=======================

// ECAP module 1 config ECap1Regs.CAP1 = 0x1000;
// Set period value ECap1Regs.CTRPHS = 0x0;
// make phase zero ECap1Regs.ECCTL2.bit.CAP_APWM = EC_APWM_MODE;

// Active high ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;

// Synch not used ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;

 // Synch not used ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN;

// Allow TSCTR to run

// Run Time (Instant 1, e.g. ISR call)

 //======================

ECap1Regs.CAP2 = 0x300;

// Set Duty cycle i.e. compare value
```

Example 6-1. COh,ode Snippet for APWM Mode (continued)

```
// Run Time (Instant 2, e.g. another ISR call)

//======================

ECap1Regs.CAP2 = 0x500;

// Set Duty cycle i.e. compare value
```

Chapter 7

SPRUH18E – January 2011 – Revised March 2014

Enhanced QEP (eQEP) Module

The eQEP module described here is a Type 0 eQEP. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566) for a list of all devices with a module of the same type to determine the differences between types and for a list of device-specific differences within a type.

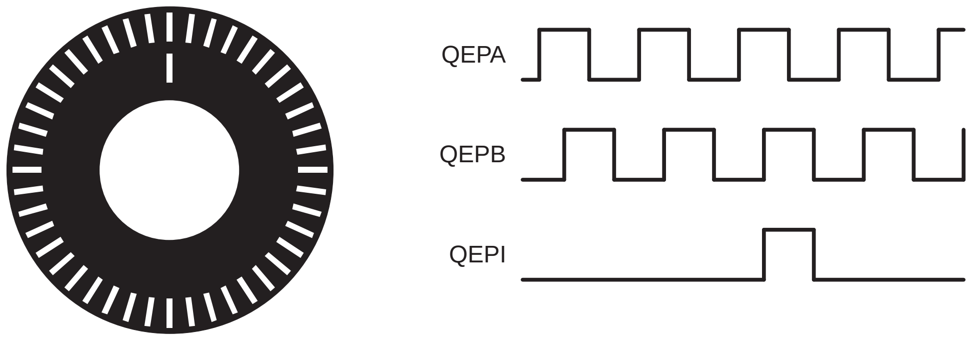
The enhanced quadrature encoder pulse (eQEP) module is used for direct interface with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine for use in a high-performance motion and position-control system.

Topic **Page**

7.1 Introduction

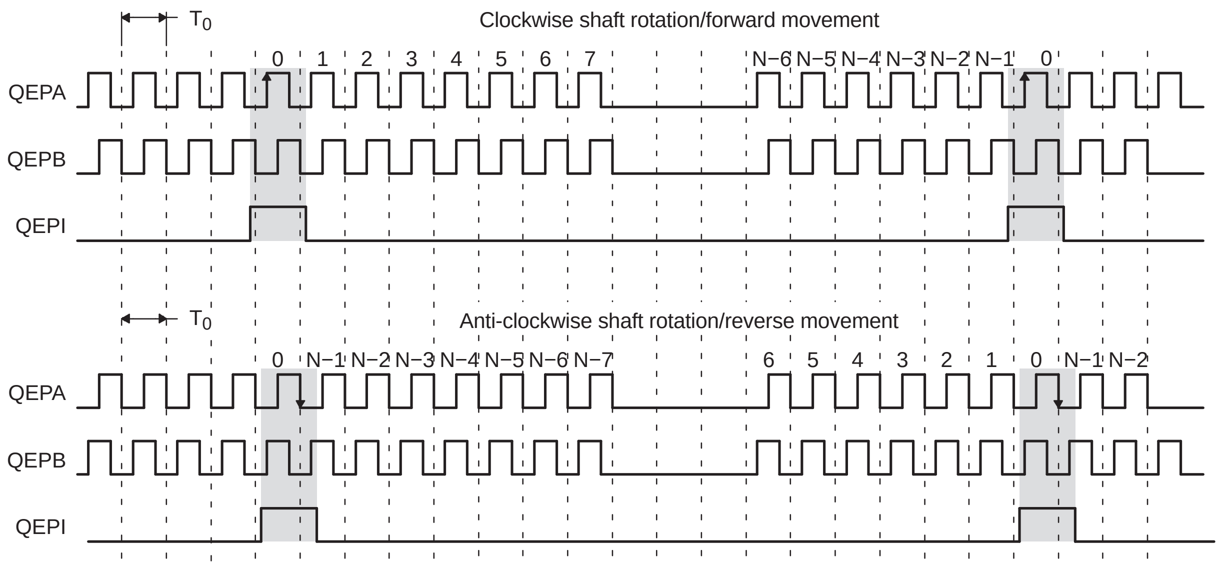
A single track of slots patterns the periphery of an incremental encoder disk, as shown in Figure 7-1. These slots create an alternating pattern of dark and light lines. The disk count is defined as the number of dark/light line pairs that occur per revolution (lines per revolution). As a rule, a second track is added to generate a signal that occurs once per revolution (index signal: QEPI), which can be used to indicate an absolute position. Encoder manufacturers identify the index pulse using different terms such as index, marker, home position, and zero reference

Figure 7-1. Optical Encoder Disk



To derive direction information, the lines on the disk are read out by two different photo-elements that "look" at the disk pattern with a mechanical shift of 1/4 the pitch of a line pair between them. This shift is realized with a reticle or mask that restricts the view of the photo-element to the desired part of the disk lines. As the disk rotates, the two photo-elements generate signals that are shifted 90° out of phase from each other. These are commonly called the quadrature QEPA and QEPB signals. The clockwise direction for most encoders is defined as the QEPA channel going positive before the QEPB channel and vise versa as shown in Figure 7-2.

Figure 7-2. QEP Encoder Output Signal for Forward/Reverse Movement

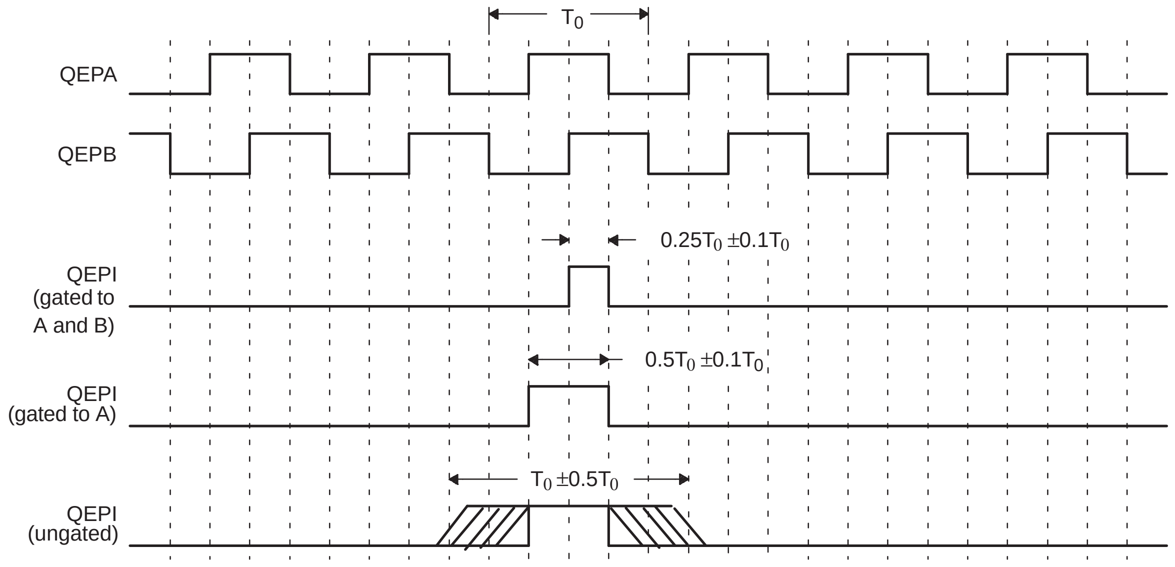

T0
Clockwise shaft rotation/forward movement
0 1 2 3 4 5 6 7 N−6 N−5 N−4 N−3 N−2 N−1 0
QEPA
QEPB
QEPI
T0
Anti-clockwise shaft rotation/reverse movement
0 N−1 N−2 N−3 N−4 N−5 N−6 N−7 6 5 4 3 2 1 0 N−1 N−2
QEPA
QEPB
QEPI

Legend: N = lines per revolution

The encoder wheel typically makes one revolution for every revolution of the motor or the wheel may be at a geared rotation ratio with respect to the motor. Therefore, the frequency of the digital signal coming from the QEPA and QEPB outputs varies proportionally with the velocity of the motor. For example, a 2000-line encoder directly coupled to a motor running at 5000 revolutions per minute (rpm) results in a frequency of 166.6 KHz, so by measuring the frequency of either the QEPA or QEPB output, the processor can determine the velocity of the motor.

Quadrature encoders from different manufacturers come with two forms of index pulse (gated index pulse or ungated index pulse) as shown in Figure 7-3. A nonstandard form of index pulse is ungated. In the ungated configuration, the index edges are not necessarily coincident with A and B signals. The gated index pulse is aligned to any of the four quadrature edges and width of the index pulse and can be equal to a quarter, half, or full period of the quadrature signal.

Figure 7-3. Index Pulse Example



Some typical applications of shaft encoders include robotics and even computer input in the form of a mouse. Inside your mouse you can see where the mouse ball spins a pair of axles (a left/right, and an up/down axle). These axles are connected to optical shaft encoders that effectively tell the computer how fast and in what direction the mouse is moving.

General Issues: Estimating velocity from a digital position sensor is a cost-effective strategy in motor control. Two different first order approximations for velocity may be written as:

$$v(k) \approx \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \tag{1}$$

$$v(k) \approx \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \tag{2}$$

where

v(k): Velocity at time instant k

x(k): Position at time instant k

x(k-1): Position at time instant k-1

T: Fixed unit time or inverse of velocity calculation rate

ΔX: Incremental position movement in unit time

t(k): Time instant "k"

t(k-1): Time instant "k-1"

X: Fixed unit position

ΔT: Incremental time elapsed for unit position movement.

Equation 1 is the conventional approach to velocity estimation and it requires a time base to provide unit time event for velocity calculation. Unit time is basically the inverse of the velocity calculation rate.

The encoder count (position) is read once during each unit time event. The quantity [x(k) - x(k-1)] is formed by subtracting the previous reading from the current reading. Then the velocity estimate is computed by multiplying by the known constant 1/T (where T is the constant time between unit time events and is known in advance).

Estimation based on Equation 1 has an inherent accuracy limit directly related to the resolution of the position sensor and the unit time period T. For example, consider a 500-line per revolution quadrature encoder with a velocity calculation rate of 400 Hz. When used for position the quadrature encoder gives a four-fold increase in resolution, in this case, 2000 counts per revolution. The minimum rotation that can be detected is therefore 0.0005 revolutions, which gives a velocity resolution of 12 rpm when sampled at 400 Hz. While this resolution may be satisfactory at moderate or high speeds, e.g. 1% error at 1200 rpm, it would clearly prove inadequate at low speeds. In fact, at speeds below 12 rpm, the speed estimate would erroneously be zero much of the time.

At low speed, Equation 2 provides a more accurate approach. It requires a position sensor that outputs a fixed interval pulse train, such as the aforementioned quadrature encoder. The width of each pulse is defined by motor speed for a given sensor resolution. Equation 2 can be used to calculate motor speed by measuring the elapsed time between successive quadrature pulse edges. However, this method suffers from the opposite limitation, as does Equation 1. A combination of relatively large motor speeds and high sensor resolution makes the time interval ΔT small, and thus more greatly influenced by the timer resolution. This can introduce considerable error into high-speed estimates.

For systems with a large speed range (that is, speed estimation is needed at both low and high speeds), one approach is to use Equation 2 at low speed and have the DSP software switch over to Equation 1 when the motor speed rises above some specified threshold.

7.2 Description

This section provides the eQEP inputs, memory map, and functional description.

7.2.1 EQEP Inputs

The eQEP inputs include two pins for quadrature-clock mode or direction-count mode, an index (or 0 marker), and a strobe input. The eQEP module requires that the QEPA, QEPB, and QEPI inputs are synchronized to SYSCLK prior to entering the module. The application code should enable the synchronous GPIO input feature on any eQEP-enabled GPIO pins (See the System Control and Interrupts user guide for your device for more details).

- *QEPA/XCLK and QEPB/XDIR*

  These two pins can be used in quadrature-clock mode or direction-count mode.

  – *Quadrature-clock Mode*

    The eQEP encoders provide two square wave signals (A and B) 90 electrical degrees out of phase whose phase relationship is used to determine the direction of rotation of the input shaft and number of eQEP pulses from the index position to derive the relative position information. For forward or clockwise rotation, QEPA signal leads QEPB signal and vice versa. The quadrature decoder uses these two inputs to generate quadrature-clock and direction signals.

  – *Direction-count Mode*

    In direction-count mode, direction and clock signals are provided directly from the external source. Some position encoders have this type of output instead of quadrature output. The QEPA pin provides the clock input and the QEPB pin provides the direction input.

- *eQEPI: Index or Zero Marker*

  The eQEP encoder uses an index signal to assign an absolute start position from which position information is incrementally encoded using quadrature pulses. This pin is connected to the index output of the eQEP encoder to optionally reset the position counter for each revolution. This signal can be used to initialize or latch the position counter on the occurrence of a desired event on the index pin.

- QEPS: *Strobe Input*

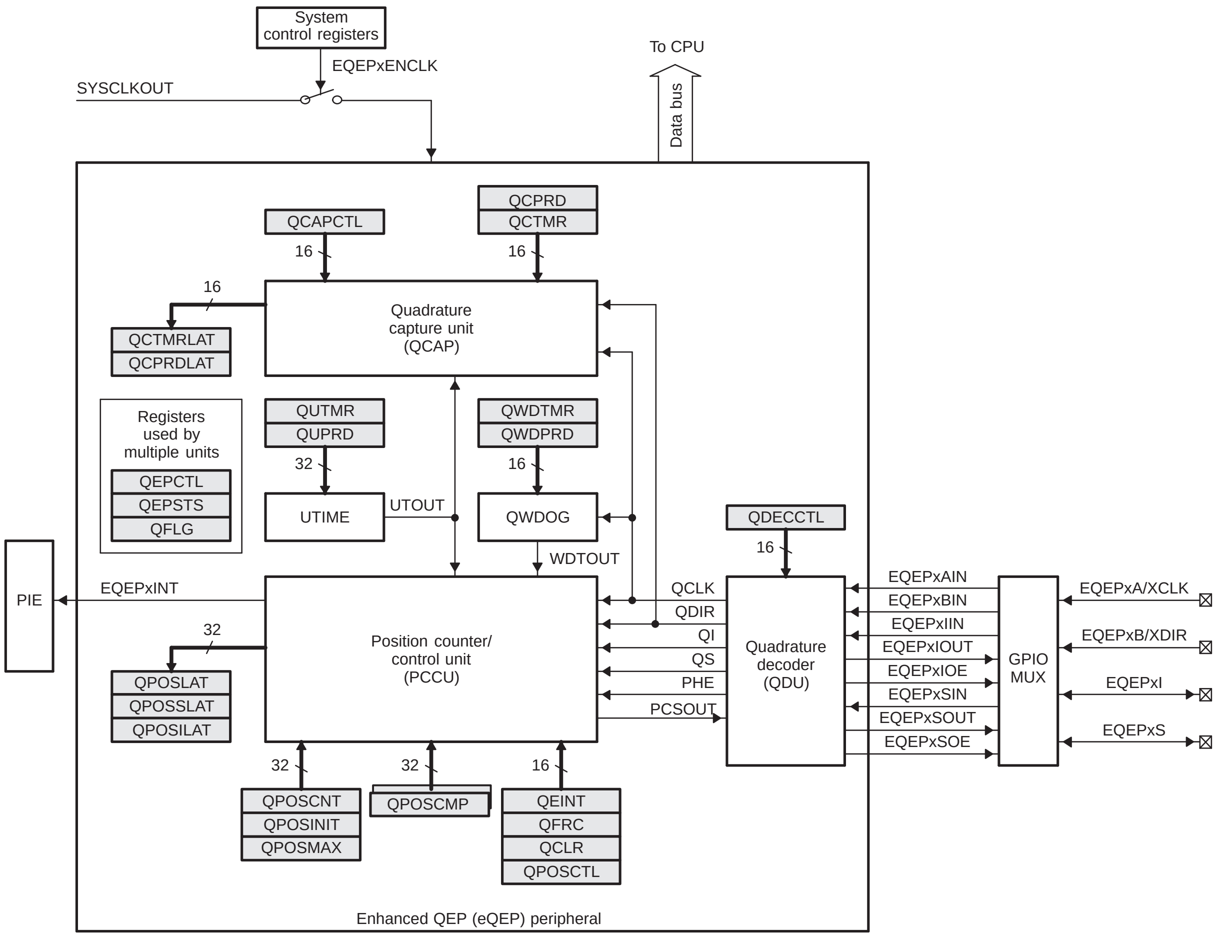
  This general-purpose strobe signal can initialize or latch the position counter on the occurrence of a desired event on the strobe pin. This signal is typically connected to a sensor or limit switch to notify that the motor has reached a defined position.

7.2.2 Functional Description

The eQEP peripheral contains the following major functional units (as shown in Figure 7-4):

- Programmable input qualification for each pin (part of the GPIO MUX)
- Quadrature decoder unit (QDU)
- Position counter and control unit for position measurement (PCCU)
- Quadrature edge-capture unit for low-speed measurement (QCAP)
- Unit time base for speed/frequency measurement (UTIME)
- Watchdog timer for detecting stalls (QWDOG)

Figure 7-4. Functional Block Diagram of the eQEP Peripheral



7.2.3 eQEP Memory Map

Table 7-1 lists the registers with their memory locations, sizes, and reset values.

Table 7-1. EQEP Memory Map

Name	Offset	Size(x16)/ #shadow	Reset	Register Description
QPOSCNT	0x00	2/0	0x00000000	eQEP Position Counter
QPOSINIT	0x02	2/0	0x00000000	eQEP Initialization Position Count
QPOSMAX	0x04	2/0	0x00000000	eQEP Maximum Position Count
QPOSCMP	0x06	2/1	0x00000000	eQEP Position-compare
QPOSILAT	0x08	2/0	0x00000000	eQEP Index Position Latch

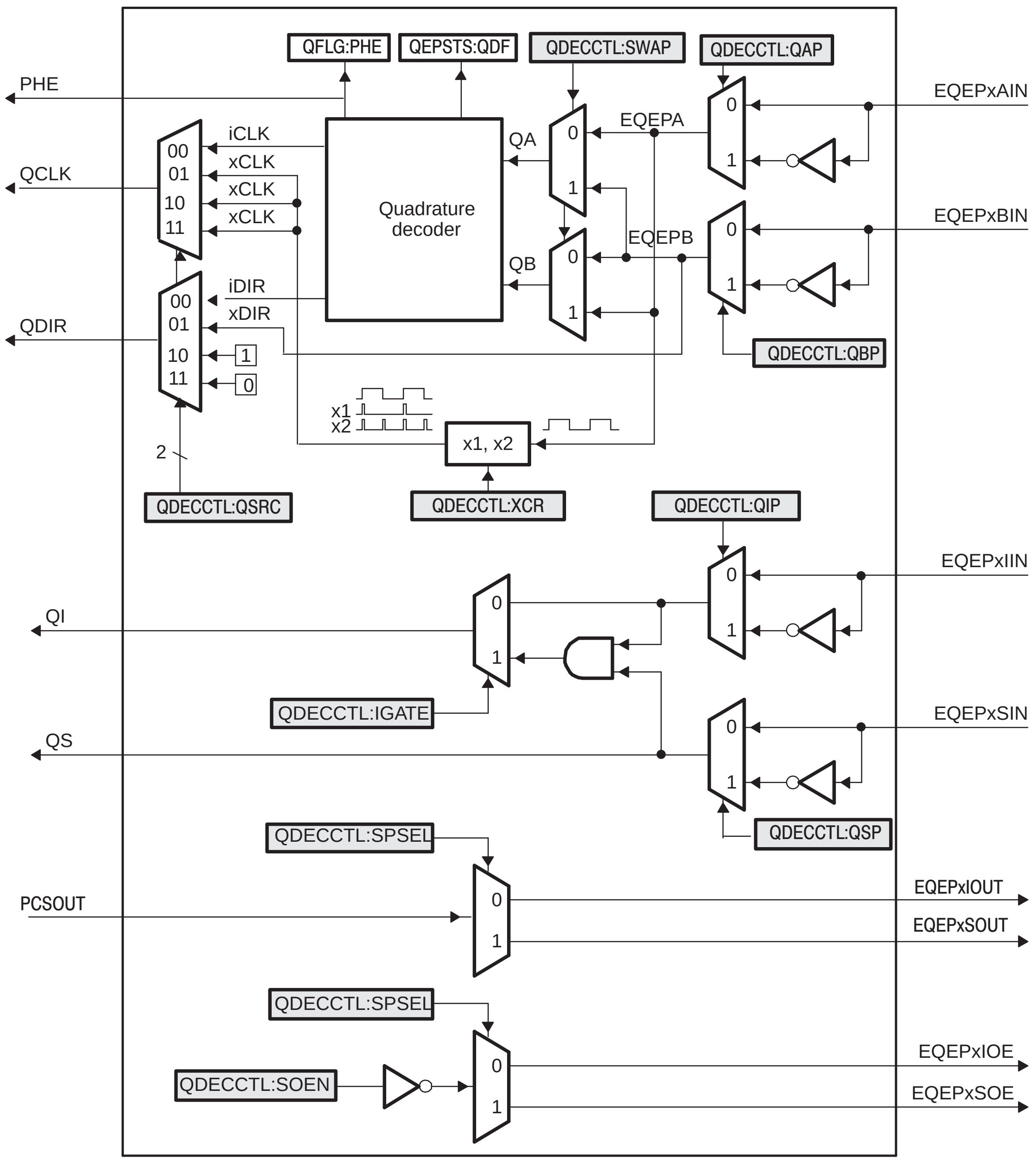
Table 7-1. EQEP Memory Map (continued)

Name	Offset	Size(x16)/ #shadow	Reset	Register Description
QPOSSLAT	0x0A	2/0	0x00000000	eQEP Strobe Position Latch
QPOSLAT	0x0C	2/0	0x00000000	eQEP Position Latch
QUTMR	0x0E	2/0	0x00000000	QEP Unit Timer
QUPRD	0x10	2/0	0x00000000	eQEP Unit Period Register
QWDTMR	0x12	1/0	0x0000	eQEP Watchdog Timer
QWDPRD	0x13	1/0	0x0000	eQEP Watchdog Period Register
QDECCTL	0x14	1/0	0x0000	eQEP Decoder Control Register
QEPCTL	0x15	1/0	0x0000	eQEP Control Register
QCAPCTL	0x16	1/0	0x0000	eQEP Capture Control Register
QPOSCTL	0x17	1/0	0x00000	eQEP Position-compare Control Register
QEINT	0x18	1/0	0x0000	eQEP Interrupt Enable Register
QFLG	0x19	1/0	0x0000	eQEP Interrupt Flag Register
QCLR	0x1A	1/0	0x0000	eQEP Interrupt Clear Register
QFRC	0x1B	1/0	0x0000	eQEP Interrupt Force Register
QEPSTS	0x1C	1/0	0x0000	eQEP Status Register
QCTMR	0x1D	1/0	0x0000	eQEP Capture Timer
QCPRD	0x1E	1/0	0x0000	eQEP Capture Period Register
QCTMRLAT	0x1F	1/0	0x0000	eQEP Capture Timer Latch
QCPRDLAT	0x20	1/0	0x0000	eQEP Capture Period Latch
reserved	0x21 to 0x3F	31/0		

7.3 Quadrature Decoder Unit (QDU)

Figure 7-5 shows a functional block diagram of the QDU.

Figure 7-5. Functional Block Diagram of Decoder Unit



7.3.1 Position Counter Input Modes

Clock and direction input to position counter is selected using QDECCTL[QSRC] bits, based on interface input requirement as follows:

- Quadrature-count mode
- Direction-count mode
- UP-count mode
- DOWN-count mode

7.3.1.1 Quadrature Count Mode

The quadrature decoder generates the direction and clock to the position counter in quadrature count mode.

Direction Decoding— The direction decoding logic of the eQEP circuit determines which one of the sequences (QEPA, QEPB) is the leading sequence and accordingly updates the direction information in QEPSTS[QDF] bit. Table 7-2 and Figure 7-6 show the direction decoding logic in truth table and state machine form. Both edges of the QEPA and QEPB signals are sensed to generate count pulses for the position counter. Therefore, the frequency of the clock generated by the eQEP logic is four times that of each input sequence. Figure 7-7 shows the direction decoding and clock generation from the eQEP input signals.

Table 7-2. Quadrature Decoder Truth Table

Previous Edge	Present Edge	QDIR	QPOSCNT
QA↑	QB↑	UP	Increment
	QB↓	DOWN	Decrement
	QA↓	TOGGLE	Increment or Decrement
QA↓	QB↓	UP	Increment
	QB↑	DOWN	Decrement
	QA↑	TOGGLE	Increment or Decrement
QB↑	QA↑	DOWN	Increment
	QA↓	UP	Decrement
	QB↓	TOGGLE	Increment or Decrement
QB↓	QA↓	DOWN	Increment
	QA↑	UP	Decrement
	QB↑	TOGGLE	Increment or Decrement

Figure 7-6. Quadrature Decoder State Machine

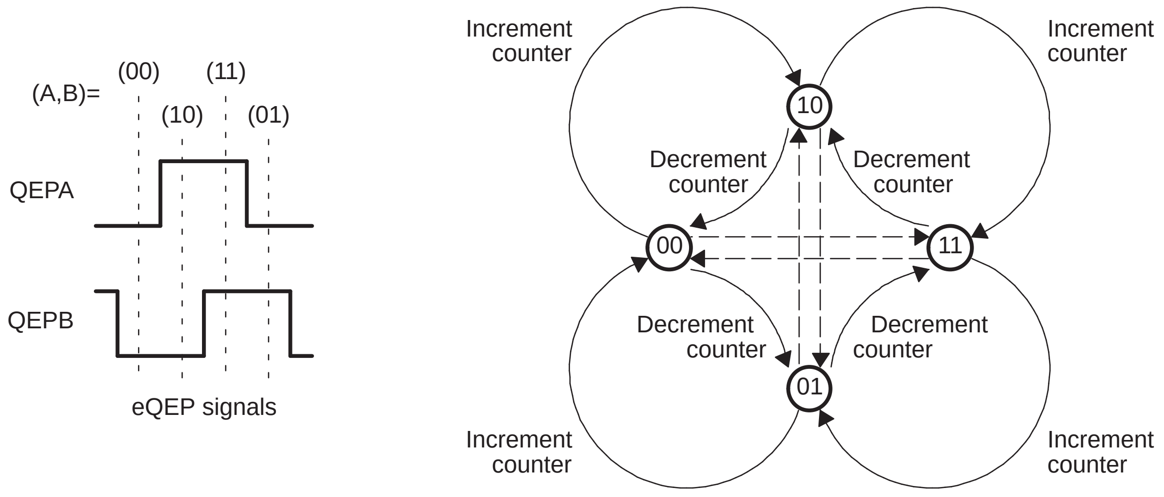
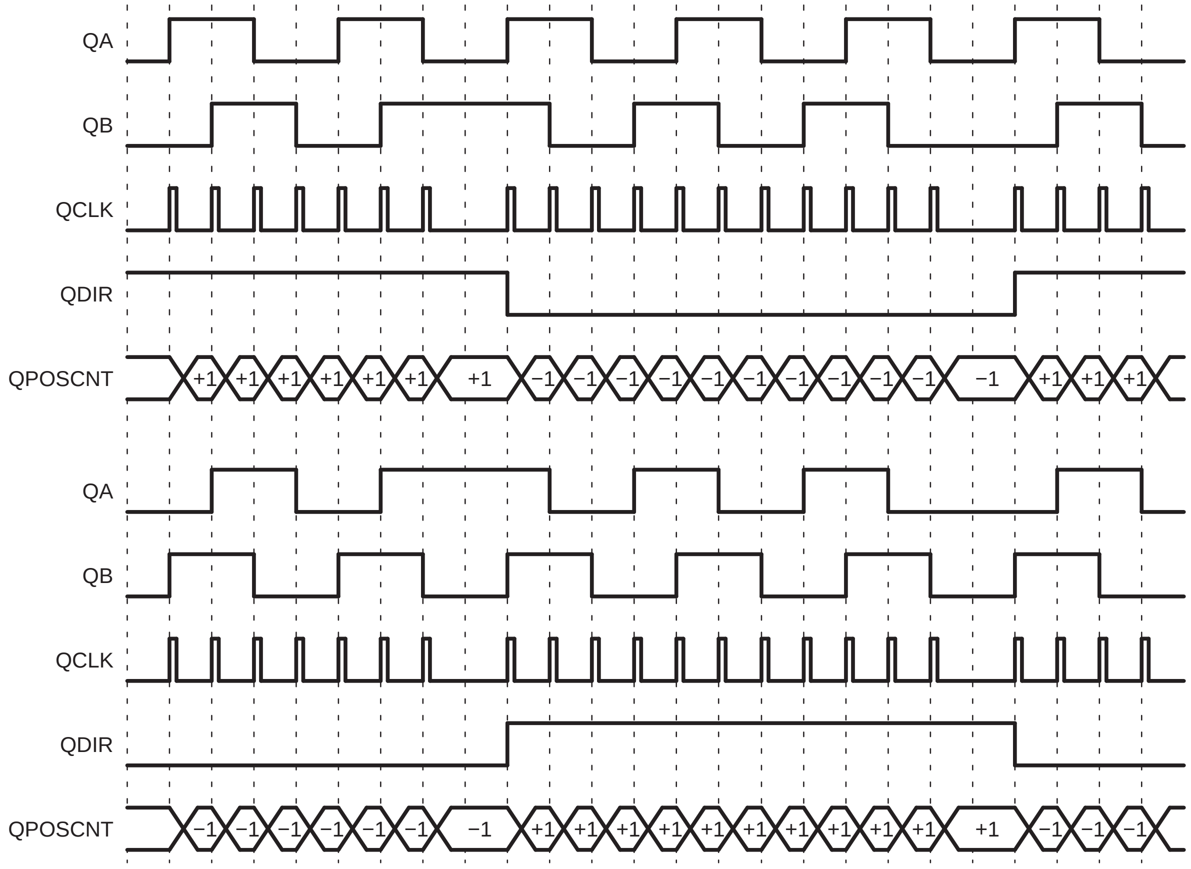


Figure 7-7. Quadrature-clock and Direction Decoding



Phase Error Flag— In normal operating conditions, quadrature inputs QEPA and QEPB will be 90 degrees out of phase. The phase error flag (PHE) is set in the QFLG register when edge transition is detected simultaneously on the QEPA and QEPB signals to optionally generate interrupts. State transitions marked by dashed lines in Figure 7-6 are invalid transitions that generate a phase error.

Count Multiplication— The eQEP position counter provides 4x times the resolution of an input clock by generating a quadrature-clock (QCLK) on the rising/falling edges of both eQEP input clocks (QEPA and QEPB) as shown in Figure 7-7¤.

Reverse Count— In normal quadrature count operation, QEPA input is fed to the QA input of the quadrature decoder and the QEPB input is fed to the QB input of the quadrature decoder. Reverse counting is enabled by setting the SWAP bit in the QDECCTL register. This will swap the input to the quadrature decoder thereby reversing the counting direction.

7.3.1.2 Direction-count Mode

Some position encoders provide direction and clock outputs, instead of quadrature outputs. In such cases, direction-count mode can be used. QEPA input will provide the clock for position counter and the QEPB input will have the direction information. The position counter is incremented on every rising edge of a QEPA input when the direction input is high and decremented when the direction input is low.

7.3.1.3 Up-Count Mode

The counter direction signal is hard-wired for up count and the position counter is used to measure the frequency of the QEPA input. Clearing of the QDECCTL[XCR] bit enables clock generation to the position counter on both edges of the QEPA input, thereby increasing the measurement resolution by 2x factor.

7.3.1.4 Down-Count Mode

The counter direction signal is hardwired for a down count and the position counter is used to measure the frequency of the QEPA input. Clearing of the QDECCTL[XCR] bit enables clock generation to the position counter on both edges of a QEPA input, thereby increasing the measurement resolution by 2x factor.

7.3.2 eQEP Input Polarity Selection

Each eQEP input can be inverted using QDECCTL[8:5] control bits. As an example, setting of QDECCTL[QIP] bit will invert the index input.

7.3.3 Position-Compare Sync Output

The enhanced eQEP peripheral includes a position-compare unit that is used to generate the position-compare sync signal on compare match between the position counter register (QPOSCNT) and the position-compare register (QPOSCMP). This sync signal can be output using an index pin or strobe pin of the EQEP peripheral.

Setting the QDECCTL[SOEN] bit enables the position-compare sync output and the QDECCTL[SPSEL] bit selects either an eQEP index pin or an eQEP strobe pin.

7.4 Position Counter and Control Unit (PCCU)

The position counter and control unit provides two configuration registers (QEPCTL and QPOSCTL) for setting up position counter operational modes, position counter initialization/latch modes and position-compare logic for sync signal generation.

7.4.1 Position Counter Operating Modes

Position counter data may be captured in different manners. In some systems, the position counter is accumulated continuously for multiple revolutions and the position counter value provides the position information with respect to the known reference. An example of this is the quadrature encoder mounted on the motor controlling the print head in the printer. Here the position counter is reset by moving the print head to the home position and then position counter provides absolute position information with respect to home position.

In other systems, the position counter is reset on every revolution using index pulse and position counter provides rotor angle with respect to index pulse position.

Position counter can be configured to operate in following four modes

- Position Counter Reset on Index Event
- Position Counter Reset on Maximum Position
- Position Counter Reset on the first Index Event
- Position Counter Reset on Unit Time Out Event (Frequency Measurement)

In all the above operating modes, position counter is reset to 0 on overflow and to QPOSMAX register value on underflow. Overflow occurs when the position counter counts up after QPOSMAX value. Underflow occurs when position counter counts down after "0". Interrupt flag is set to indicate overflow/underflow in QFLG register.

7.4.1.1 Position Counter Reset on Index Event (QEPCTL[PCRM]=00)

If the index event occurs during the forward movement, then position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse movement, then the position counter is reset to the value in the QPOSMAX register on the next eQEP clock.

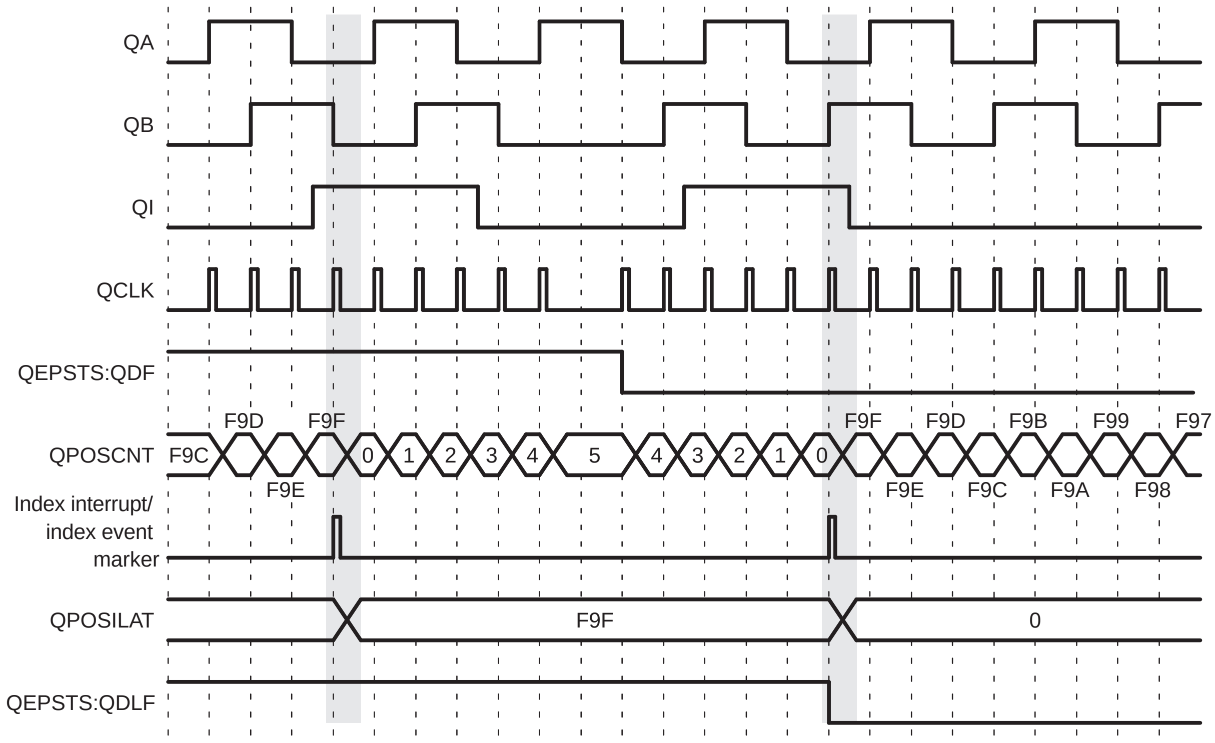
First index marker is defined as the quadrature edge following the first index edge. The eQEP peripheral records the occurrence of the first index marker (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in QEPSTS registers, it also remembers the quadrature edge on the first index marker so that same relative quadrature transition is used for index event reset operation.

For example, if the first reset operation occurs on the falling edge of QEPB during the forward direction, then all the subsequent reset must be aligned with the falling edge of QEPB for the forward rotation and on the rising edge of QEPB for the reverse rotation as shown in Figure 7-8.

The position-counter value is latched to the QPOSILAT register and direction information is recorded in the QEPSTS[QDLF] bit on every index event marker. The position-counter error flag (QEPSTS[PCEF]) and error interrupt flag (QFLG[PCE]) are set if the latched value is not equal to 0 or QPOSMAX. The position-counter error flag (QEPSTS[PCEF]) is updated on every index event marker and an interrupt flag (QFLG[PCE]) will be set on error that can be cleared only through software.

The index event latch configuration QEPCTL[IEL] bits are ignored in this mode and position counter error flag/interrupt flag are generated only in index event reset mode.

Figure 7-8. Position Counter Reset by Index Pulse for 1000 Line Encoder (QPOSMAX = 3999 or 0xF9F)

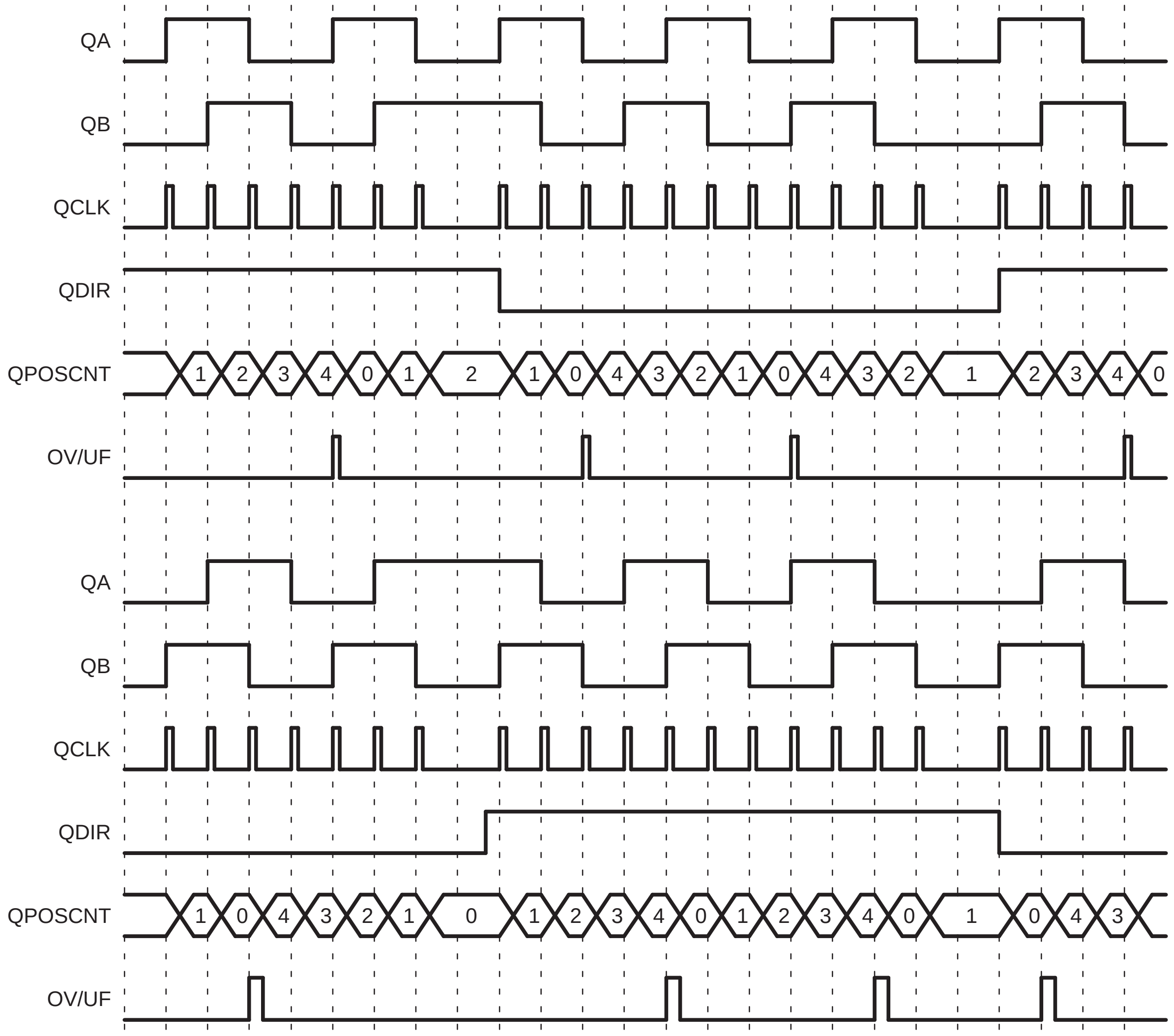


7.4.1.2 Position Counter Reset on Maximum Position (QEPCTL[PCRM]=01)

If the position counter is equal to QPOSMAX, then the position counter is reset to 0 on the next eQEP clock for forward movement and position counter overflow flag is set. If the position counter is equal to ZERO, then the position counter is reset to QPOSMAX on the next QEP clock for reverse movement and position counter underflow flag is set. Figure 7-9shows the position counter reset operation in this mode.

First index marker is defined as the quadrature edge following the first index edge. The eQEP peripheral records the occurrence of the first index marker (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in the QEPSTS registers; it also remembers the quadrature edge on the first index marker so that the same relative quadrature transition is used for the software index marker (QEPCTL[IEL]=11).

Figure 7-9. Position Counter Underflow/Overflow (QPOSMAX = 4)



7.4.1.3 Position Counter Reset on the First Index Event (QEPCTL[PCRM] = 10)

If the index event occurs during forward movement, then the position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse movement, then the position counter is reset to the value in the QPOSMAX register on the next eQEP clock. Note that this is done only on the first occurrence and subsequently the position counter value is not reset on an index event; rather, it is reset based on maximum position as described in Section Section 7.4.1.2.

First index marker is defined as the quadrature edge following the first index edge. The eQEP peripheral records the occurrence of the first index marker (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in QEPSTS registers, it also remembers the quadrature edge on the first index marker so that same relative quadrature transition is used for software index marker (QEPCTL[IEL]=11).

7.4.1.4 Position Counter Reset on Unit Time out Event (QEPCTL[PCRM] = 11)

In this mode, the QPOSCNT value is latched to the QPOSLAT register and then the QPOSCNT is reset (to 0 or QPOSMAX, depending on the direction mode selected by QDECCTL[QSRC] bits on a unit time event). This is useful for frequency measurement.

7.4.2 Position Counter Latch

The eQEP index and strobe input can be configured to latch the position counter (QPOSCNT) into QPOSILAT and QPOSSLAT, respectively, on occurrence of a definite event on these pins.

7.4.2.1 Index Event Latch

In some applications, it may not be desirable to reset the position counter on every index event and instead it may be required to operate the position counter in full 32-bit mode (QEPCTL[PCRM] = 01 and QEPCTL[PCRM] = 10 modes).

In such cases, the eQEP position counter can be configured to latch on the following events and direction information is recorded in the QEPSTS[QDLF] bit on every index event marker.

- Latch on Rising edge (QEPCTL[IEL]=01)
- Latch on Falling edge (QEPCTL[IEL]=10)
- Latch on Index Event Marker (QEPCTL[IEL]=11)

This is particularly useful as an error checking mechanism to check if the position counter accumulated the correct number of counts between index events. As an example, the 1000-line encoder must count 4000 times when moving in the same direction between the index events.

The index event latch interrupt flag (QFLG[IEL]) is set when the position counter is latched to the QPOSILAT register.

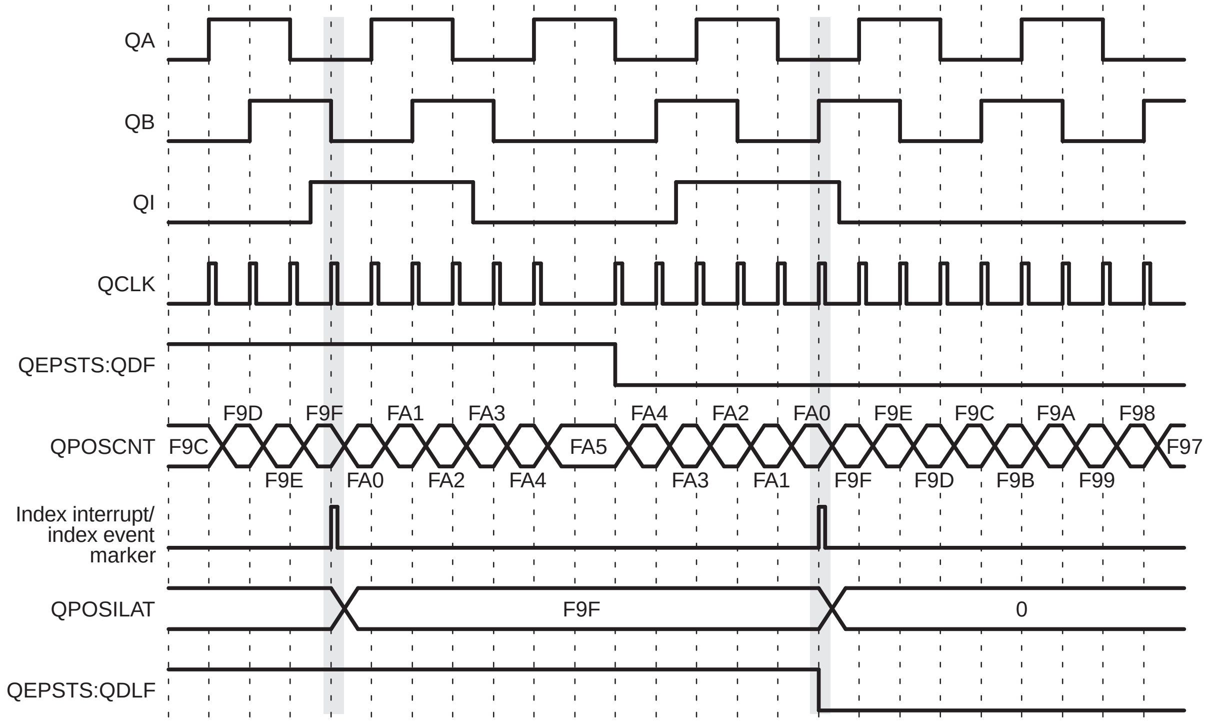
Latch on Rising Edge (QEPCTL[IEL]=01)— The position counter value (QPOSCNT) is latched to the QPOSILAT register on every rising edge of an index input.

Latch on Falling Edge (QEPCTL[IEL] = 10)— The position counter value (QPOSCNT) is latched to the QPOSILAT register on every falling edge of index input.

Latch on Index Event Marker/Software Index Marker (QEPCTL[IEL] = 11— The first index marker is defined as the quadrature edge following the first index edge. The eQEP peripheral records the occurrence of the first index marker (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in the QEPSTS registers. It also remembers the quadrature edge on the first index marker so that same relative quadrature transition is used for latching the position counter (QEPCTL[IEL]=11).

Figure 7-10 shows the position counter latch using an index event marker.

Figure 7-10. Software Index Marker for 1000-line Encoder (QEPCTL[IEL] = 1)



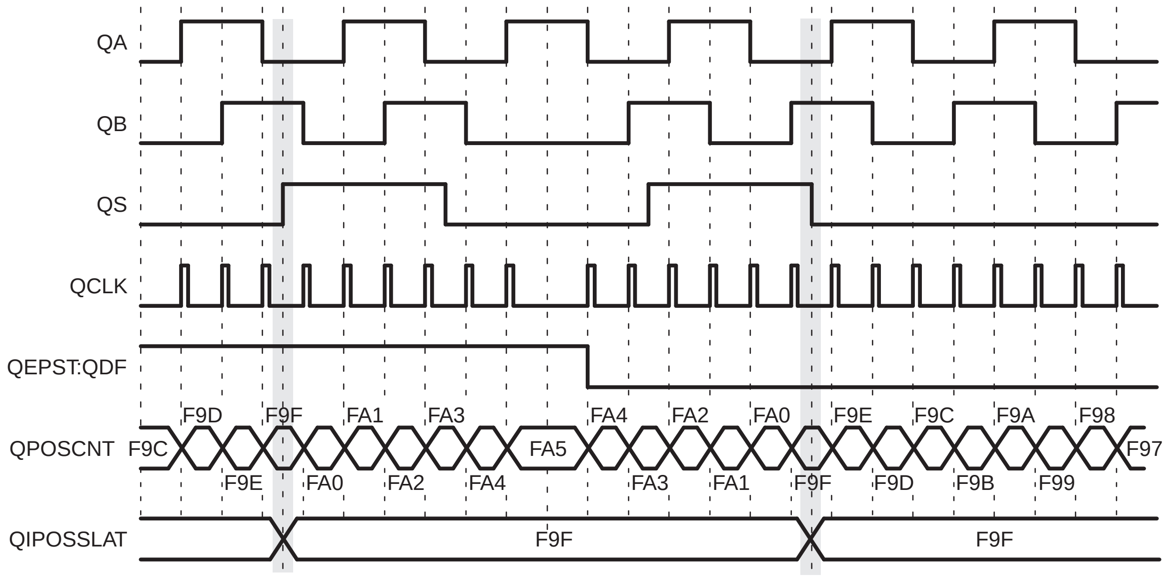
7.4.2.2 Strobe Event Latch

The position-counter value is latched to the QPOSSLAT register on the rising edge of the strobe input by clearing the QEPCTL[SEL] bit.

If the QEPCTL[SEL] bit is set, then the position counter value is latched to the QPOSSLAT register on the rising edge of the strobe input for forward direction and on the falling edge of the strobe input for reverse direction as shown in Figure 7-11.

The strobe event latch interrupt flag (QFLG[SEL) is set when the position counter is latched to the QPOSSLAT register.

Figure 7-11. Strobe Event Latch (QEPCTL[SEL] = 1)



7.4.3 Position Counter Initialization

The position counter can be initialized using following events:

- Index event
- Strobe event
- Software initialization

Index Event Initialization (IEI)— The QEPI index input can be used to trigger the initialization of the position counter at the rising or falling edge of the index input. If the QEPCTL[IEI] bits are 10, then the position counter (QPOSCNT) is initialized with a value in the QPOSINIT register on the rising edge of index input. Conversely, if the QEPCTL[IEI] bits are 11, initialization will be on the falling edge of the index input.

Strobe Event Initialization (SEI)— If the QEPCTL[SEI] bits are 10, then the position counter is initialized with a value in the QPOSINIT register on the rising edge of strobe input.

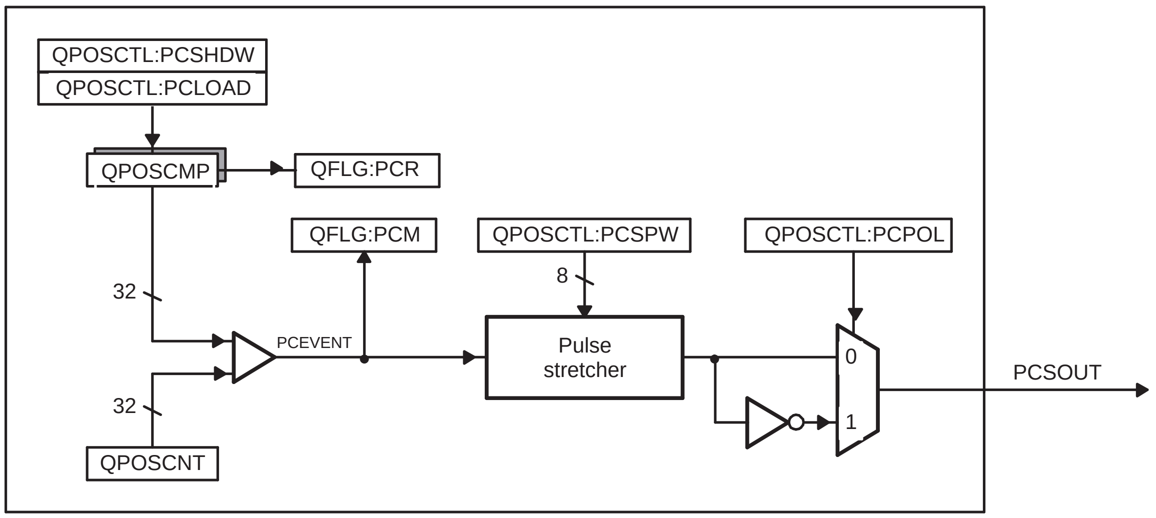
If QEPCTL[SEL] bits are 11, then the position counter is initialized with a value in the QPOSINIT register on the rising edge of strobe input for forward direction and on the falling edge of strobe input for reverse direction.

Software Initialization (SWI)— The position counter can be initialized in software by writing a 1 to the QEPCTL[SWI] bit. This bit is not automatically cleared. While the bit is still set, if a 1 is written to it again, the position counter will be re-initialized.

7.4.4 eQEP Position-compare Unit

The eQEP peripheral includes a position-compare unit that is used to generate a sync output and/or interrupt on a position-compare match. Figure 7-12 shows a diagram. The position-compare (QPOSCMP) register is shadowed and shadow mode can be enabled or disabled using the QPOSCTL[PSSHDW] bit. If the shadow mode is not enabled, the CPU writes directly to the active position compare register.

Figure 7-12. eQEP Position-compare Unit



In shadow mode, you can configure the position-compare unit (QPOSCTL[PCLOAD]) to load the shadow register value into the active register on the following events and to generate the position-compare ready (QFLG[PCR]) interrupt after loading.

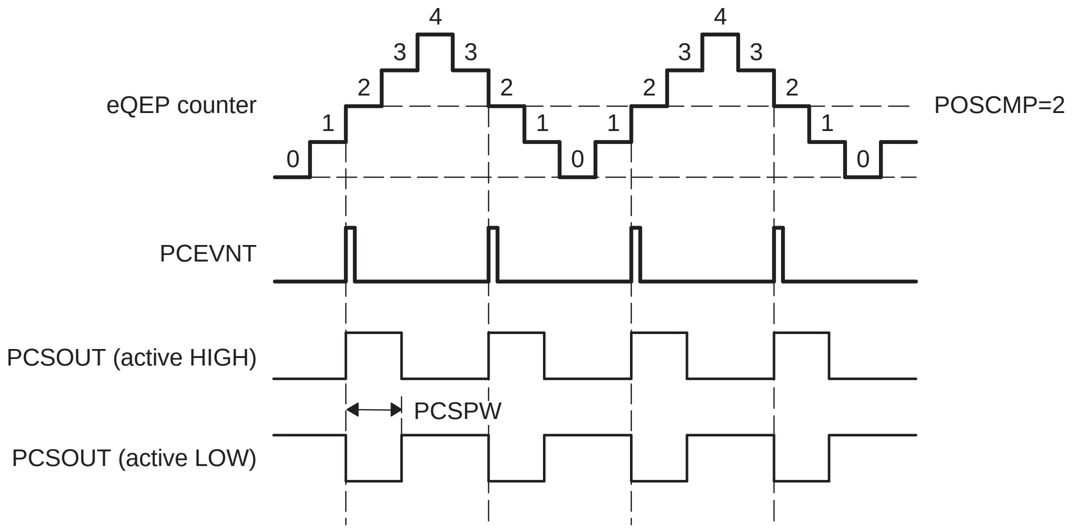
- Load on compare match
- Load on position-counter zero event

The position-compare match (QFLG[PCM]) is set when the position-counter value (QPOSCNT) matches with the active position-compare register (QPOSCMP) and the position-compare sync output of the programmable pulse width is generated on compare match to trigger an external device.

For example, if QPOSCMP = 2, the position-compare unit generates a position-compare event on 1 to 2 transitions of the eQEP position counter for forward counting direction and on 3 to 2 transitions of the eQEP position counter for reverse counting direction (see Figure 7-13).

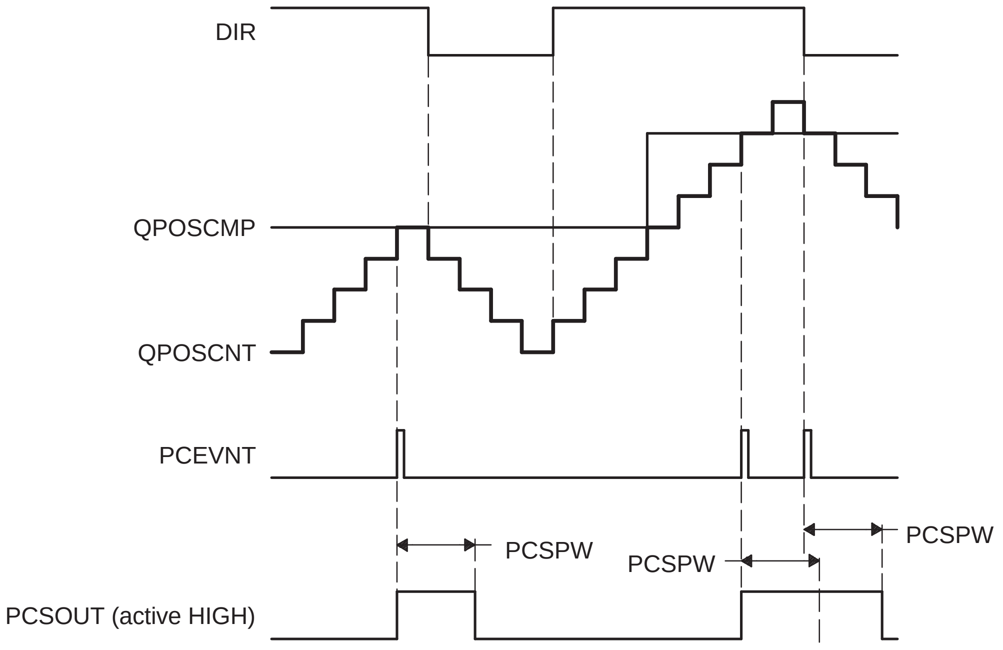
Figure 7-23 shows the layout of the eQEP Position-Compare Control Register (QPOSCTL) and Table 7-5 describes the QPOSCTL bit fields.

Figure 7-13. eQEP Position-compare Event Generation Points



The pulse stretcher logic in the position-compare unit generates a programmable position-compare sync pulse output on the position-compare match. In the event of a new position-compare match while a previous position-compare pulse is still active, then the pulse stretcher generates a pulse of specified duration from the new position-compare event as shown in Figure 7-14.

Figure 7-14. eQEP Position-compare Sync Output Pulse Stretcher



7.5 eQEP Edge Capture Unit

The eQEP peripheral includes an integrated edge capture unit to measure the elapsed time between the unit position events as shown in Figure 7-15. This feature is typically used for low speed measurement using the following equation:

$$v(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \quad (3)$$

where,

- X - Unit position is defined by integer multiple of quadrature edges (see Figure 7-16)
- ΔT - Elapsed time between unit position events
- v(k) - Velocity at time instant "k"

The eQEP capture timer (QCTMR) runs from prescaled SYSCLKOUT and the prescaler is programmed by the QCAPCTL[CCPS] bits. The capture timer (QCTMR) value is latched into the capture period register (QCPRD) on every unit position event and then the capture timer is reset, a flag is set in QEPSTS:UPEVNT to indicate that new value is latched into the QCPRD register. Software can check this status flag before reading the period register for low speed measurement and clear the flag by writing 1.

Time measurement (ΔT) between unit position events will be correct if the following conditions are met:

- No more than 65,535 counts have occurred between unit position events.
- No direction change between unit position events.

The capture unit sets the eQEP overflow error flag (QEPSTS[COEF]) in the event of capture timer overflow between unit position events. If a direction change occurs between the unit position events, then an error flag is set in the status register (QEPSTS[CDEF]).

Capture Timer (QCTMR) and Capture period register (QCPRD) can be configured to latch on following events.

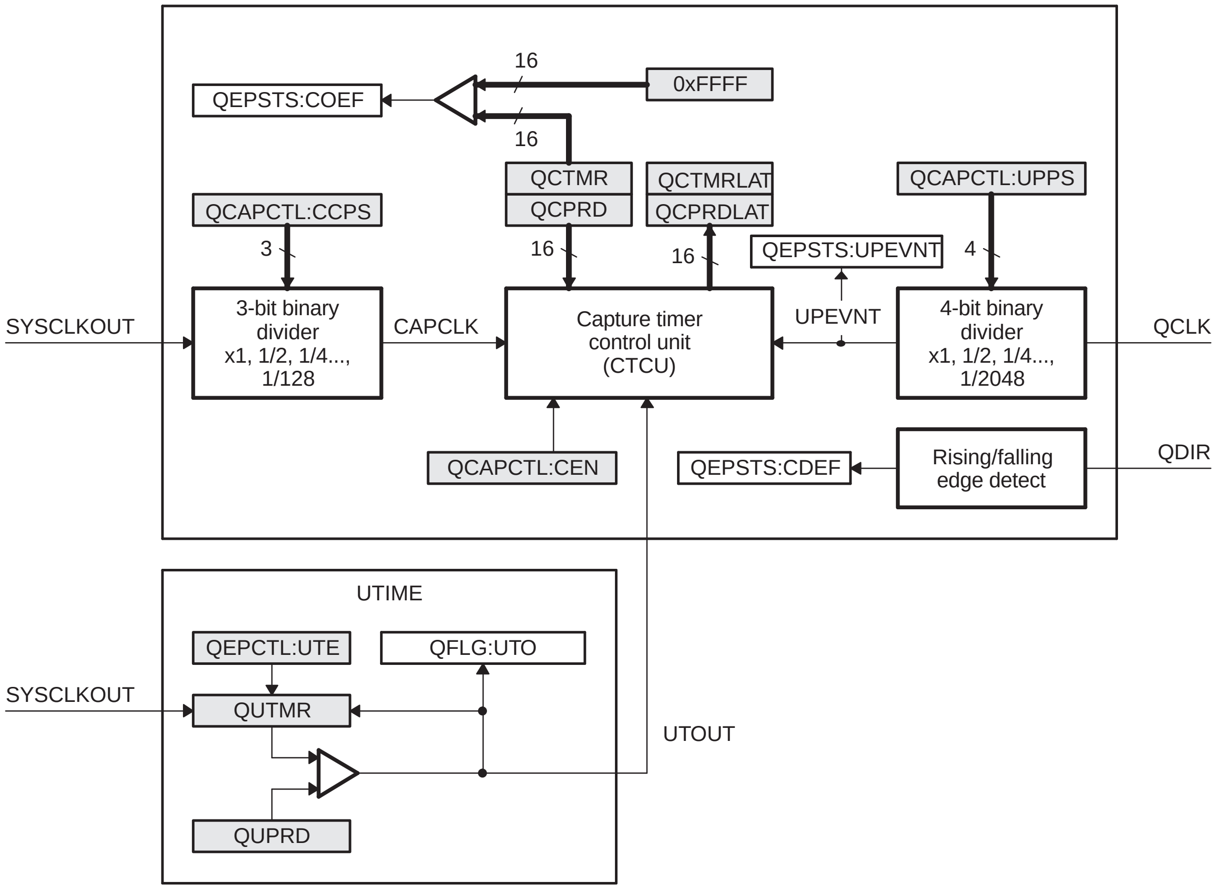
- CPU read of QPOSCNT register
- Unit time-out event

If the QEPCTL[QCLM] bit is cleared, then the capture timer and capture period values are latched into the QCTMRLAT and QCPRDLAT registers, respectively, when the CPU reads the position counter (QPOSCNT).

If the QEPCTL[QCLM] bit is set, then the position counter, capture timer, and capture period values are latched into the QPOSLAT, QCTMRLAT and QCPRDLAT registers, respectively, on unit time out.

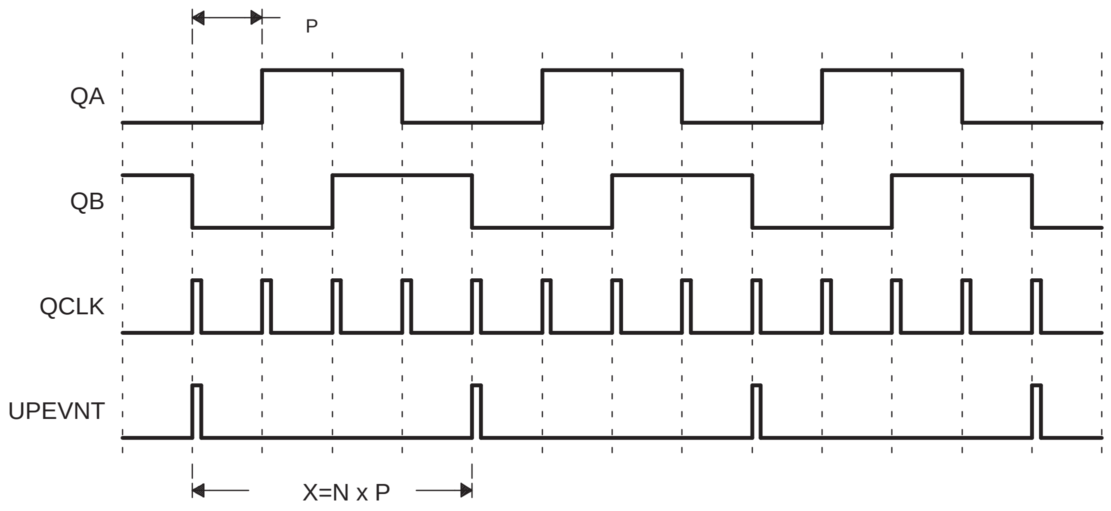
Figure 7-17 shows the capture unit operation along with the position counter.

Figure 7-15. eQEP Edge Capture Unit



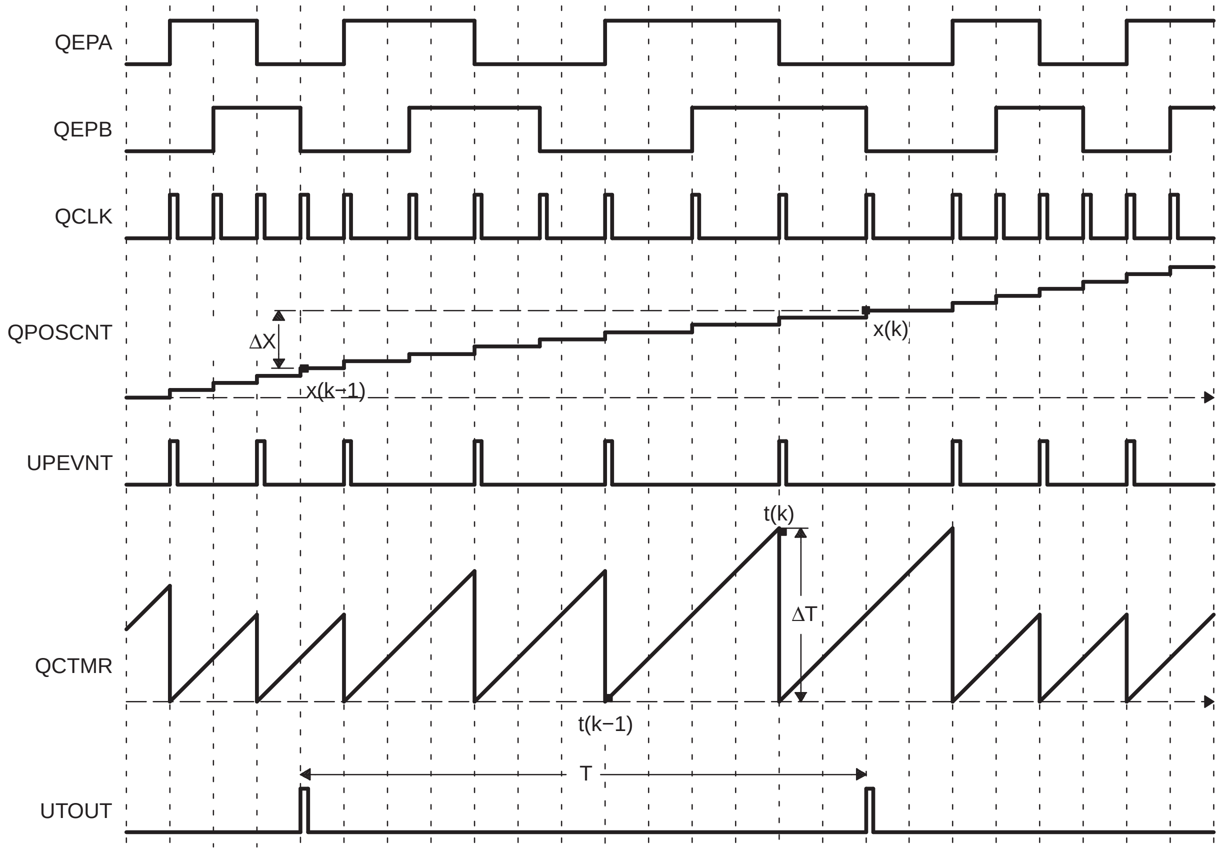
NOTE: The QCAPCTL[UPPS] prescaler should not be modified dynamically (such as switching the unit event prescaler from QCLK/4 to QCLK/8). Doing so may result in undefined behavior. The QCAPCTL[CPPS] prescaler can be modified dynamically (such as switching CAPCLK prescaling mode from SYSCLK/4 to SYSCLK/8) only after the capture unit is disabled.

Figure 7-16. Unit Position Event for Low Speed Measurement (QCAPCTL[UPPS] = 0010)



A N - Number of quadrature periods selected using QCAPCTL[UPPS] bits

Figure 7-17. eQEP Edge Capture Unit - Timing Details



Velocity Calculation Equations:

$$v(k) = \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \quad (4)$$

where

v(k): Velocity at time instant k

x(k): Position at time instant k

x(k-1): Position at time instant k-1

T: Fixed unit time or inverse of velocity calculation rate

ΔX: Incremental position movement in unit time

X: Fixed unit position

ΔT: Incremental time elapsed for unit position movement

t(k): Time instant "k"

t(k-1): Time instant "k-1"

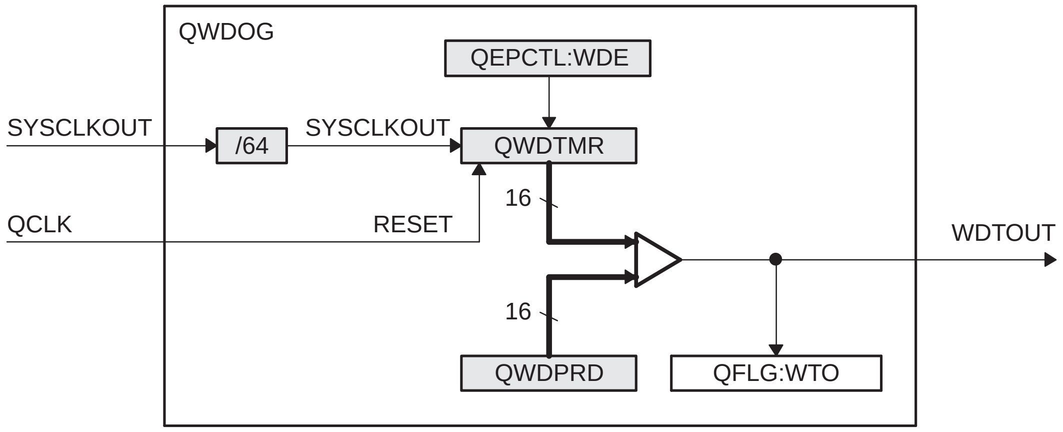
Unit time (T) and unit period(X) are configured using the QUPRD and QCAPCTL[UPPS] registers. Incremental position output and incremental time output is available in the QPOSLAT and QCPRDLAT registers.

Parameter	Relevant Register to Configure or Read the Information
T	Unit Period Register (QUPRD)
ΔX	Incremental Position = QPOSLAT(k) - QPOSLAT(K-1)
X	Fixed unit position defined by sensor resolution and ZCAPCTL[UPPS] bits
ΔT	Capture Period Latch (QCPRDLAT)

7.6 eQEP Watchdog

The eQEP peripheral contains a 16-bit watchdog timer that monitors the quadrature-clock to indicate proper operation of the motion-control system. The eQEP watchdog timer is clocked from SYSCLKOUT/64 and the quadrate clock event (pulse) resets the watchdog timer. If no quadrature-clock event is detected until a period match (QWDPRD = QWDTMR), then the watchdog timer will time out and the watchdog interrupt flag will be set (QFLG[WTO]). The time-out value is programmable through the watchdog period register (QWDPRD).

Figure 7-18. eQEP Watchdog Timer

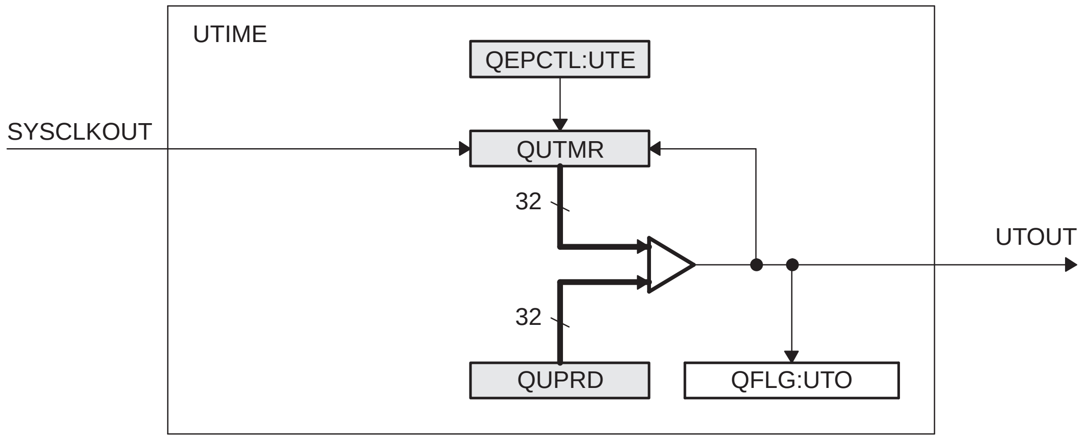


7.7 Unit Timer Base

The eQEP peripheral includes a 32-bit timer (QUTMR) that is clocked by SYSCLKOUT to generate periodic interrupts for velocity calculations. The unit time out interrupt is set (QFLG[UTO]) when the unit timer (QUTMR) matches the unit period register (QUPRD).

The eQEP peripheral can be configured to latch the position counter, capture timer, and capture period values on a unit time out event so that latched values are used for velocity calculation as described in Section Section 7.5.

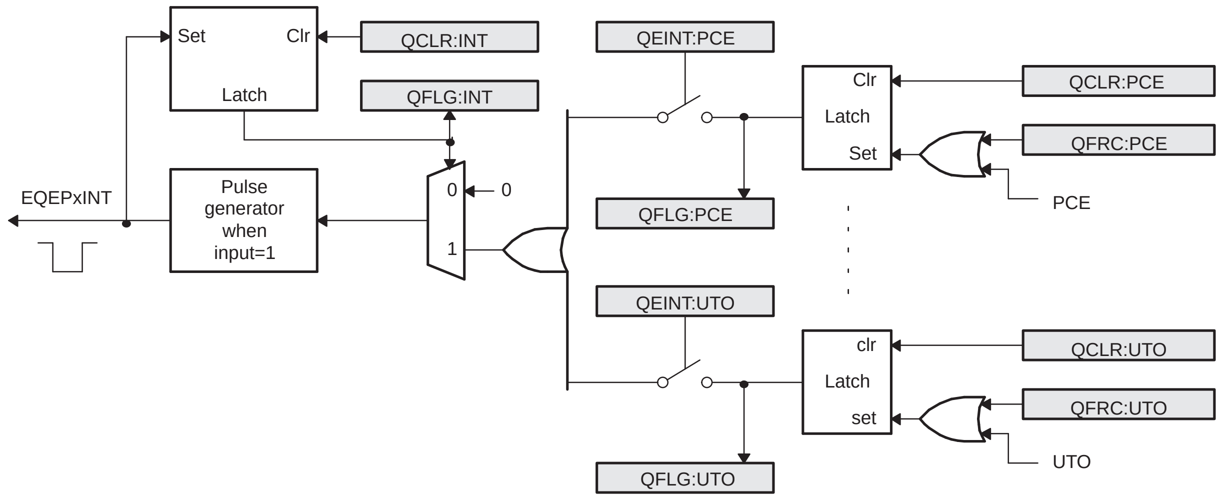
Figure 7-19. eQEP Unit Time Base



7.8 eQEP Interrupt Structure

Figure 7-20 shows how the interrupt mechanism works in the EQEP module.

Figure 7-20. EQEP Interrupt Generation



Eleven interrupt events (PCE, PHE, QDC, WTO, PCU, PCO, PCR, PCM, SEL, IEL and UTO) can be generated. The interrupt control register (QEINT) is used to enable/disable individual interrupt event sources. The interrupt flag register (QFLG) indicates if any interrupt event has been latched and contains the global interrupt flag bit (INT). An interrupt pulse is generated only to the PIE if any of the interrupt events is enabled, the flag bit is 1 and the INT flag bit is 0. The interrupt service routine will need to clear the global interrupt flag bit and the serviced event, via the interrupt clear register (QCLR), before any other interrupt pulses are generated. You can force an interrupt event by way of the interrupt force register (QFRC), which is useful for test purposes.

7.9 eQEP Registers

Figure 7-21. QEP Decoder Control (QDECCTL) Register

15-14	13	12	11	10	9	8
QSRC	SOEN	SPSEL	XCR	SWAP	IGATE	QAP
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4-0
QBP	QIP	QSP	Reserved
R/W-0	R/W-0	R/W-0	R-0

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

Table 7-3. eQEP Decoder Control (QDECCTL) Register Field Descriptions

Bits	Name	Value	Description
15-14	QSRC		Position-counter source selection
		00	Quadrature count mode (QCLK = iCLK, QDIR = iDIR)
		01	Direction-count mode (QCLK = xCLK, QDIR = xDIR)
		10	UP count mode for frequency measurement (QCLK = xCLK, QDIR = 1)
		11	DOWN count mode for frequency measurement (QCLK = xCLK, QDIR = 0)
13	SOEN		Sync output-enable
		0	Disable position-compare sync output
		1	Enable position-compare sync output

Table 7-3. eQEP Decoder Control (QDECCTL) Register Field Descriptions (continued)

Bits	Name	Value	Description
12	SPSEL		Sync output pin selection
		0	Index pin is used for sync output
		1	Strobe pin is used for sync output
11	XCR		External clock rate
		0	2x resolution: Count the rising/falling edge
		1	1x resolution: Count the rising edge only
10	SWAP		Swap quadrature clock inputs. This swaps the input to the quadrature decoder, reversing the counting direction.
		0	Quadrature-clock inputs are not swapped
		1	Quadrature-clock inputs are swapped
9	IGATE		Index pulse gating option
		0	Disable gating of Index pulse
		1	Gate the index pin with strobe
8	QAP		QEPA input polarity
		0	No effect
		1	Negates QEPA input
7	QBP		QEPB input polarity
		0	No effect
		1	Negates QEPB input
6	QIP		QEPI input polarity
		0	No effect
		1	Negates QEPI input
5	QSP		QEPS input polarity
		0	No effect
		1	Negates QEPS input
4-0	Reserved		Always write as 0

Figure 7-22. eQEP Control (QEPCTL) Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FREE, SOFT		PCRM		SEI		IEI		SWI	SEL	IEL		QPEN	QCLM	UTE	WDE
R/W-0		R/W-0		R/W-0		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 7-4. eQEP Control (QEPCTL) Register Field Descriptions

Bits	Name	Value	Description
15-14	FREE, SOFT		Emulation Control Bits
			QPOSCNT behavior
		00	Position counter stops immediately on emulation suspend
		01	Position counter continues to count until the rollover
		1x	Position counter is unaffected by emulation suspend
			QWDTMR behavior
		00	Watchdog counter stops immediately
		01	Watchdog counter counts until WD period match roll over
		1x	Watchdog counter is unaffected by emulation suspend
			QUTMR behavior
		00	Unit timer stops immediately
		01	Unit timer counts until period rollover
		1x	Unit timer is unaffected by emulation suspend
			QCTMR behavior
		00	Capture Timer stops immediately
		01	Capture Timer counts until next unit period event
		1x	Capture Timer is unaffected by emulation suspend
13-12	PCRM		Position counter reset mode
		00	Position counter reset on an index event
		01	Position counter reset on the maximum position
		10	Position counter reset on the first index event
		11	Position counter reset on a unit time event
11-10	SEI		Strobe event initialization of position counter
		00	Does nothing (action disabled)
		01	Does nothing (action disabled)
		10	Initializes the position counter on rising edge of the QEPS signal
		11	Clockwise Direction: Initializes the position counter on the rising edge of QEPS strobe Counter Clockwise Direction: Initializes the position counter on the falling edge of QEPS strobe
9-8	IEI		Index event initialization of position counter
		00	Do nothing (action disabled)
		01	Do nothing (action disabled)
		10	Initializes the position counter on the rising edge of the QEPI signal (QPOSCNT = QPOSINIT)
		11	Initializes the position counter on the falling edge of QEPI signal (QPOSCNT = QPOSINIT)
7	SWI		Software initialization of position counter
		0	Do nothing (action disabled)
		1	Initialize position counter (QPOSCNT=QPOSINIT). This bit is not cleared automatically
6	SEL		Strobe event latch of position counter
		0	The position counter is latched on the rising edge of QEPS strobe (QPOSSLAT = POSCCNT). Latching on the falling edge can be done by inverting the strobe input using the QSP bit in the QDECCTL register.
		1	Clockwise Direction: Position counter is latched on rising edge of QEPS strobe Counter Clockwise Direction: Position counter is latched on falling edge of QEPS strobe
5-4	IEL		Index event latch of position counter (software index marker)
		00	Reserved
		01	Latches position counter on rising edge of the index signal
		10	Latches position counter on falling edge of the index signal
		11	Software index marker. Latches the position counter and quadrature direction flag on index event marker. The position counter is latched to the QPOSILAT register and the direction flag is latched in the QEPSTS[QDLF] bit. This mode is useful for software index marking.

Table 7-4. eQEP Control (QEPCTL) Register Field Descriptions (continued)

Bits	Name	Value	Description
3	QPEN		Quadrature position counter enable/software reset
		0	Reset the eQEP peripheral internal operating flags/read-only registers. Control/configuration registers are not disturbed by a software reset.
		1	eQEP position counter is enabled
2	QCLM		eQEP capture latch mode
		0	Latch on position counter read by CPU. Capture timer and capture period values are latched into QCTMRLAT and QCPRDLAT registers when CPU reads the QPOSCNT register.
		1	Latch on unit time out. Position counter, capture timer and capture period values are latched into QPOSLAT, QCTMRLAT and QCPRDLAT registers on unit time out.
1	UTE		eQEP unit timer enable
		0	Disable eQEP unit timer
		1	Enable unit timer
0	WDE		eQEP watchdog enable
		0	Disable the eQEP watchdog timer
		1	Enable the eQEP watchdog timer

Figure 7-23. eQEP Position-compare Control (QPOSCTL) Register

15	14	13	12	11 - 8
PCSHDW	PCLOAD	PCPOL	PCE	PCSPW
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7 - 0
PCSPW
R/W-0

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

Table 7-5. eQEP Position-compare Control (QPOSCTL) Register Field Descriptions

Bit	Name		Description
15	PCSHDW		Position-compare shadow enable
		0	Shadow disabled, load Immediate
		1	Shadow enabled
14	PCLOAD		Position-compare shadow load mode
		0	Load on QPOSCNT = 0
		1	Load when QPOSCNT = QPOSCMP
13	PCPOL		Polarity of sync output
		0	Active HIGH pulse output
		1	Active LOW pulse output
12	PCE		Position-compare enable/disable
		0	Disable position compare unit
		1	Enable position compare unit
11-0	PCSPW		Select-position-compare sync output pulse width
		0x000	1 * 4 * SYSCLKOUT cycles
		0x001	2 * 4 * SYSCLKOUT cycles
		0xFFF	4096 * 4 * SYSCLKOUT cycles

Figure 7-24. eQEP Capture Control (QCAPCTL) Register

15	14 ... 7	6 ... 4	3 ... 0
CEN	Reserved	CCPS	UPPS
R/W-0	R-0	R/W-0	R/W-0

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

Table 7-6. eQEP Capture Control (QCAPCTL) Register Field Descriptions

Bits	Name	Description	
15	CEN		Enable eQEP capture
		0	eQEP capture unit is disabled
		1	eQEP capture unit is enabled
14-7	Reserved		Always write as 0
6-4	CCPS		eQEP capture timer clock prescaler
		000	CAPCLK = SYSCLKOUT/1
		001	CAPCLK = SYSCLKOUT/2
		010	CAPCLK = SYSCLKOUT/4
		011	CAPCLK = SYSCLKOUT/8
		100	CAPCLK = SYSCLKOUT/16
		101	CAPCLK = SYSCLKOUT/32
		110	CAPCLK = SYSCLKOUT/64
		111	CAPCLK = SYSCLKOUT/128
3-0	UPPS		Unit position event prescaler
		0000	UPEVNT = QCLK/1
		0001	UPEVNT = QCLK/2
		0010	UPEVNT = QCLK/4
		0011	UPEVNT = QCLK/8
		0100	UPEVNT = QCLK/16
		0101	UPEVNT = QCLK/32
		0110	UPEVNT = QCLK/64
		0111	UPEVNT = QCLK/128
		1000	UPEVNT = QCLK/256
		1001	UPEVNT = QCLK/512
		1010	UPEVNT = QCLK/1024
		1011	UPEVNT = QCLK/2048
		11xx	Reserved

Figure 7-25. eQEP Position Counter (QPOSCNT) Register

31 ... 0
QPOSCNT
R/W-0

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

Table 7-7. eQEP Position Counter (QPOSCNT) Register Field Descriptions

Bits	Name	Description
31-0	QPOSCNT	This 32-bit position counter register counts up/down on every eQEP pulse based on direction input. This counter acts as a position integrator whose count value is proportional to position from a give reference point.

Figure 7-26. eQEP Position Counter Initialization (QPOSINIT) Register

31 ... 0
QPOSINIT
R/W-0

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

Table 7-8. eQEP Position Counter Initialization (QPOSINIT) Register Field Descriptions

Bits	Name	Description
31-0	QPOSINIT	This register contains the position value that is used to initialize the position counter based on external strobe or index event. The position counter can be initialized through software.

Figure 7-27. eQEP Maximum Position Count Register (QPOSMAX) Register

31 0
QPOSMAX
R/W-0

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

Table 7-9. eQEP Maximum Position Count (QPOSMAX) Register Field Descriptions

Bits	Name	Description
31-0	QPOSMAX	This register contains the maximum position counter value.

Figure 7-28. eQEP Position-compare (QPOSCMP) Register

31 0
QPOSCMP
R/W-0

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

Table 7-10. eQEP Position-compare (QPOSCMP) Register Field Descriptions

Bits	Name	Description
31-0	QPOSCMP	The position-compare value in this register is compared with the position counter (QPOSCNT) to generate sync output and/or interrupt on compare match.

Figure 7-29. eQEP Index Position Latch (QPOSILAT) Register

31 0
QPOSILAT
R-0

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

Table 7-11. eQEP Index Position Latch(QPOSILAT) Register Field Descriptions

Bits	Name	Description
31-0	QPOSILAT	The position-counter value is latched into this register on an index event as defined by the QEPCTL[IEL] bits.

Figure 7-30. eQEP Strobe Position Latch (QPOSSLAT) Register

31 0
QPOSSLAT
R-0

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

Table 7-12. eQEP Strobe Position Latch (QPOSSLAT) Register Field Descriptions

Bits	Name	Description
31-0	QPOSSLAT	The position-counter value is latched into this register on strobe event as defined by the QEPCTL[SEL] bits.

Figure 7-31. eQEP Position Counter Latch (QPOSLAT) Register

31 … 0
QPOSLAT
R-0

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

Table 7-13. eQEP Position Counter Latch (QPOSLAT) Register Field Descriptions

Bits	Name	Description
31-0	QPOSLAT	The position-counter value is latched into this register on unit time out event.

Figure 7-32. eQEP Unit Timer (QUTMR) Register

31 … 0
QUTMR
R/W-0

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

Table 7-14. eQEP Unit Timer (QUTMR) Register Field Descriptions

Bits	Name	Description
31-0	QUTMR	This register acts as time base for unit time event generation. When this timer value matches with unit time period value, unit time event is generated.

Figure 7-33. eQEP Register Unit Period (QUPRD) Register

31 … 0
QUPRD
R/W-0

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

Table 7-15. eQEP Unit Period (QUPRD) Register Field Descriptions

Bits	Name	Description
31-0	QUPRD	This register contains the period count for unit timer to generate periodic unit time events to latch the eQEP position information at periodic interval and optionally to generate interrupt.

Figure 7-34. eQEP Watchdog Timer (QWDTMR) Register

15 … 0
QWDTMR
R/W-0

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

Table 7-16. eQEP Watchdog Timer (QWDTMR) Register Field Descriptions

Bits	Name	Description
15-0	QWDTMR	This register acts as time base for watch dog to detect motor stalls. When this timer value matches with watch dog period value, watch dog timeout interrupt is generated. This register is reset upon edge transition in quadrature-clock indicating the motion.

Figure 7-35. eQEP Watchdog Period (QWDPRD) Register

15 … 0
QWDPRD
R/W-0

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

Table 7-17. eQEP Watchdog Period (QWDPRD) Register Field Description

Bits	Name	Value	Description
15-0	QWDPRD		This register contains the time-out count for the eQEP peripheral watch dog timer. When the watchdog timer value matches the watchdog period value, a watchdog timeout interrupt is generated.

Figure 7-36. eQEP Interrupt Enable (QEINT) Register

15-12	11	10	9	8
Reserved	UTO	IEL	SEL	PCM
R-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	QPE	PCE	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 7-18. eQEP Interrupt Enable(QEINT) Register Field Descriptions

Bits	Name	Value	Description
15-12	Reserved	0	Always write as 0
11	UTO		Unit time out interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
10	IEL		Index event latch interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
9	SEL		Strobe event latch interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
8	PCM		Position-compare match interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
7	PCR		Position-compare ready interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
6	PCO		Position counter overflow interrupt enable

Table 7-18. eQEP Interrupt Enable(QEINT) Register Field Descriptions (continued)

Bits	Name	Value	Description
		0	Interrupt is disabled
		1	Interrupt is enabled
5	PCU		Position counter underflow interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
4	WTO		Watchdog time out interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
3	QDC		Quadrature direction change interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
2	QPE		Quadrature phase error interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
1	PCE		Position counter error interrupt enable
		0	Interrupt is disabled
		1	Interrupt is enabled
0	Reserved		Reserved

Figure 7-37. eQEP Interrupt Flag (QFLG) Register

15-12	11	10	9	8
Reserved	UTO	IEL	SEL	PCM
R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	PHE	PCE	INT
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 7-19. eQEP Interrupt Flag (QFLG) Register Field Descriptions

Bits	Name	Value	Description
15-12	Reserved		Always write as 0
11	UTO		Unit time out interrupt flag
		0	No interrupt generated
		1	Set by eQEP unit timer period match
10	IEL		Index event latch interrupt flag
		0	No interrupt generated
		1	This bit is set after latching the QPOSCNT to QPOSILAT
9	SEL		Strobe event latch interrupt flag
		0	No interrupt generated
		1	This bit is set after latching the QPOSCNT to QPOSSLAT
8	PCM		eQEP compare match event interrupt flag
		0	No interrupt generated
		1	This bit is set on position-compare match
7	PCR		Position-compare ready interrupt flag
		0	No interrupt generated
		1	This bit is set after transferring the shadow register value to the active position compare register.

Table 7-19. eQEP Interrupt Flag (QFLG) Register Field Descriptions (continued)

Bits	Name	Value	Description
6	PCO		Position counter overflow interrupt flag
		0	No interrupt generated
		1	This bit is set on position counter overflow.
5	PCU		Position counter underflow interrupt flag
		0	No interrupt generated
		1	This bit is set on position counter underflow.
4	WTO		Watchdog timeout interrupt flag
		0	No interrupt generated
		1	Set by watch dog timeout
3	QDC		Quadrature direction change interrupt flag
		0	No interrupt generated
		1	This bit is set during change of direction
2	PHE		Quadrature phase error interrupt flag
		0	No interrupt generated
		1	Set on simultaneous transition of QEPA and QEPB
1	PCE		Position counter error interrupt flag
		0	No interrupt generated
		1	Position counter error
0	INT		Global interrupt status flag
		0	No interrupt generated
		1	Interrupt was generated

Figure 7-38. eQEP Interrupt Clear (QCLR) Register

15-12	11	10	9	8
Reserved	UTO	IEL	SEL	PCM
R-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	PHE	PCE	INT
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 7-20. eQEP Interrupt Clear (QCLR) Register Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Always write as 0s
11	UTO		Clear unit time out interrupt flag
		0	No effect
		1	Clears the interrupt flag
10	IEL		Clear index event latch interrupt flag
		0	No effect
		1	Clears the interrupt flag
9	SEL		Clear strobe event latch interrupt flag
		0	No effect
		1	Clears the interrupt flag
8	PCM		Clear eQEP compare match event interrupt flag
		0	No effect
		1	Clears the interrupt flag

Table 7-20. eQEP Interrupt Clear (QCLR) Register Field Descriptions (continued)

Bit	Field	Value	Description
7	PCR		Clear position-compare ready interrupt flag
		0	No effect
		1	Clears the interrupt flag
6	PCO		Clear position counter overflow interrupt flag
		0	No effect
		1	Clears the interrupt flag
5	PCU		Clear position counter underflow interrupt flag
		0	No effect
		1	Clears the interrupt flag
4	WTO		Clear watchdog timeout interrupt flag
		0	No effect
		1	Clears the interrupt flag
3	QDC		Clear quadrature direction change interrupt flag
		0	No effect
		1	Clears the interrupt flag
2	PHE		Clear quadrature phase error interrupt flag
		0	No effect
		1	Clears the interrupt flag
1	PCE		Clear position counter error interrupt flag
		0	No effect
		1	Clears the interrupt flag
0	INT		Global interrupt clear flag
		0	No effect
		1	Clears the interrupt flag and enables further interrupts to be generated if an event flags is set to 1.

Figure 7-39. eQEP Interrupt Force (QFRC) Register

15			12	11	10	9	8
Reserved				UTO	IEL	SEL	PCM
R-0				R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	PHE	PCE	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 7-21. eQEP Interrupt Force (QFRC) Register Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Always write as 0s
11	UTO		Force unit time out interrupt
		0	No effect
		1	Force the interrupt
10	IEL		Force index event latch interrupt
		0	No effect
		1	Force the interrupt
9	SEL		Force strobe event latch interrupt
		0	No effect
		1	Force the interrupt

Table 7-21. eQEP Interrupt Force (QFRC) Register Field Descriptions (continued)

Bit	Field	Value	Description
8	PCM		Force position-compare match interrupt
		0	No effect
		1	Force the interrupt
7	PCR		Force position-compare ready interrupt
		0	No effect
		1	Force the interrupt
6	PCO		Force position counter overflow interrupt
		0	No effect
		1	Force the interrupt
5	PCU		Force position counter underflow interrupt
		0	No effect
		1	Force the interrupt
4	WTO		Force watchdog time out interrupt
		0	No effect
		1	Force the interrupt
3	QDC		Force quadrature direction change interrupt
		0	No effect
		1	Force the interrupt
2	PHE		Force quadrature phase error interrupt
		0	No effect
		1	Force the interrupt
1	PCE		Force position counter error interrupt
		0	No effect
		1	Force the interrupt
0	Reserved	0 1	Always write as 0

Figure 7-40. eQEP Status (QEPSTS) Register

15–8
Reserved
R-0

7	6	5	4	3	2	1	0
UPEVNT	FIDF	QDF	QDLF	COEF	CDEF	FIMF	PCEF
R/w-1	R-0	R-0	R-0	R/W-1	R/W-1	R/W-1	R-0

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

Table 7-22. eQEP Status (QEPSTS) Register Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		Always write as 0
7	UPEVNT		Unit position event flag
		0	No unit position event detected
		1	Unit position event detected. Write 1 to clear.
6	FIDF		Direction on the first index marker Status of the direction is latched on the first index event marker.
		0	Counter-clockwise rotation (or reverse movement) on the first index event
		1	Clockwise rotation (or forward movement) on the first index event

Table 7-22. eQEP Status (QEPSTS) Register Field Descriptions (continued)

Bit	Field	Value	Description
5	QDF		Quadrature direction flag
		0	Counter-clockwise rotation (or reverse movement)
		1	Clockwise rotation (or forward movement)
4	QDLF		eQEP direction latch flag Status of direction is latched on every index event marker.
		0	Counter-clockwise rotation (or reverse movement) on index event marker
		1	Clockwise rotation (or forward movement) on index event marker
3	COEF		Capture overflow error flag
		0	Sticky bit, cleared by writing 1
		1	Overflow occurred in eQEP Capture timer (QEPCTMR)
2	CDEF		Capture direction error flag
		0	Sticky bit, cleared by writing 1
		1	Direction change occurred between the capture position event.
1	FIMF		First index marker flag
		0	Sticky bit, cleared by writing 1
		1	Set by first occurrence of index pulse
0	PCEF		Position counter error flag. This bit is not sticky and it is updated for every index event.
		0	No error occurred during the last index transition.
		1	Position counter error

Figure 7-41. eQEP Capture Timer (QCTMR) Register

15 … 0
QCTMR
R/W

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

Table 7-23. eQEP Capture Timer (QCTMR) Register Field Descriptions

Bits	Name	Description
15-0	QCTMR	This register provides time base for edge capture unit.

Figure 7-42. eQEP Capture Period (QCPRD) Register

15 … 0
QCPRD
R/W

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

Table 7-24. eQEP Capture Period Register (QCPRD) Register Field Descriptions

Bits	Name	Description
15-0	QCPRD	This register holds the period count value between the last successive eQEP position events

Figure 7-43. eQEP Capture Timer Latch (QCTMRLAT) Register

15 … 0
QCTMRLAT
R

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

Table 7-25. eQEP Capture Timer Latch (QCTMRLAT) Register Field Descriptions

Bits	Name	Description
15-0	QCTMRLAT	The eQEP capture timer value can be latched into this register on two events viz., unit timeout event, reading the eQEP position counter.

Figure 7-44. eQEP Capture Period Latch (QCPRDLAT) Register

15	0
QCPRDLAT	
R/W	

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

Table 7-26. eQEP Capture Period Latch (QCPRDLAT) Register Field Descriptions

Bits	Name	Description
15-0	QCPRDLAT	eQEP capture period value can be latched into this register on two events viz., unit timeout event, reading the eQEP position counter.

Analog-to-Digital Converter and Comparator

The ADC module described in this reference guide is a Type 3 ADC and exists on the Piccolo™ family of devices. The Comparator function described in this reference guide is a Type 0 Comparator. See the *TMS320C28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566) 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.

Topic **Page**

8.1 Analog-to-Digital Converter (ADC)

The ADC module described in this reference guide 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.

8.1.1 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 the device datasheet 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.

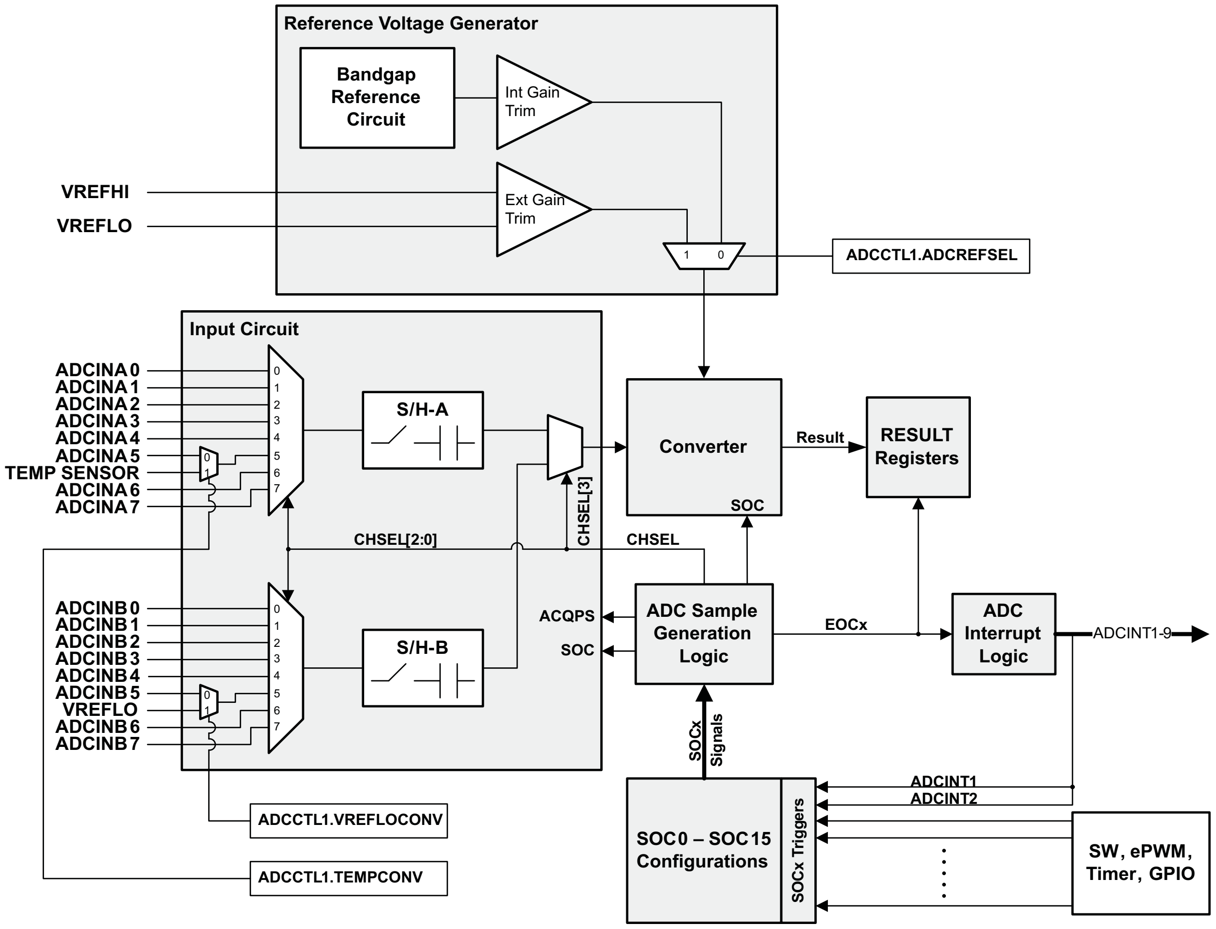
Functions of the ADC module include:

- 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 1-8
  - GPIO XINT2
  - CPU Timers 0/1/2
  - ADCINT1/2
- 9 flexible PIE interrupts, can configure interrupt request after any conversion

8.1.2 Block Diagram

Figure 8-1 shows the block diagram of the ADC module.

Figure 8-1. ADC Block Diagram

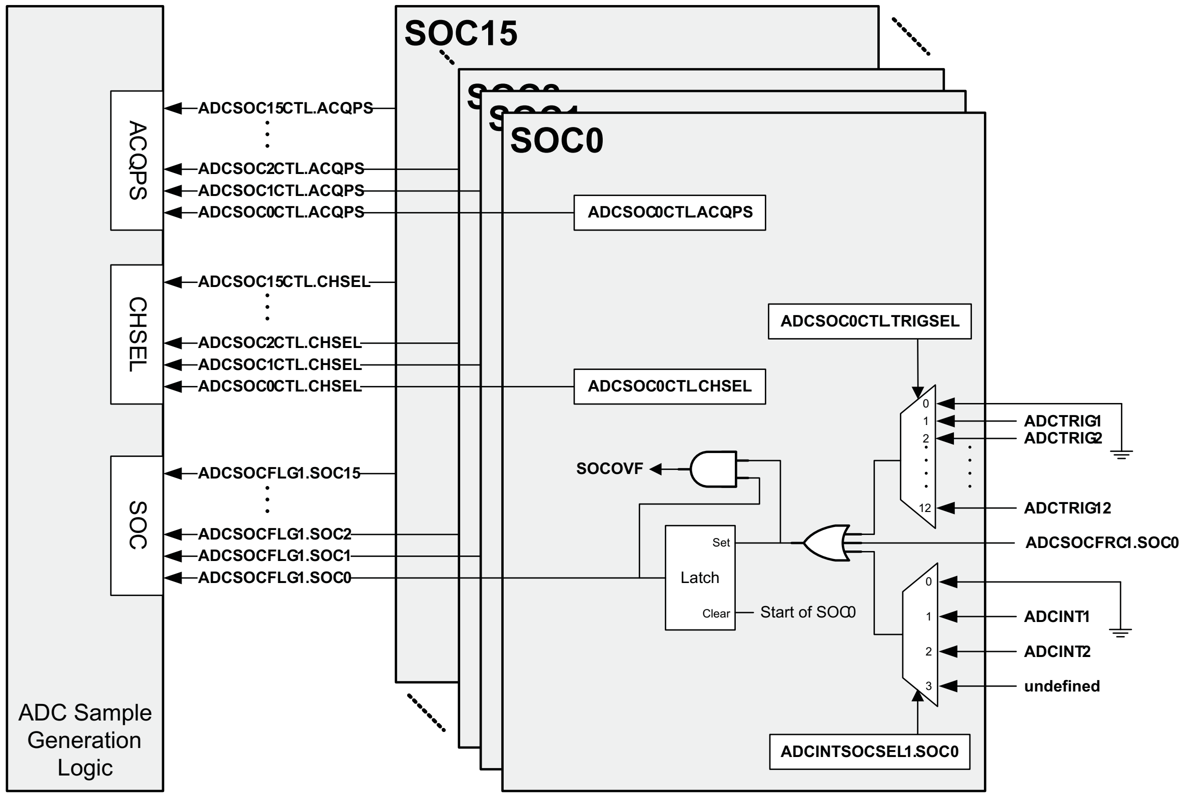


8.1.3 SOC Principle of Operation

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 SOC's can be configured for the same trigger, channel, and/or acquisition window as desired. This provides a very flexible means of configurating 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 8-2. 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 the Enhanced Pulse Width Modulator Module (ePWM) on how to do this. In this case, we will use SOCA. Then, setup 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 will be:

```
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* section in this manual. For the power-up sequence of the ADC, see Section 8.1.8. 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 please refer to Section 8.1.11

8.1.3.1 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 below in Table 8-1.

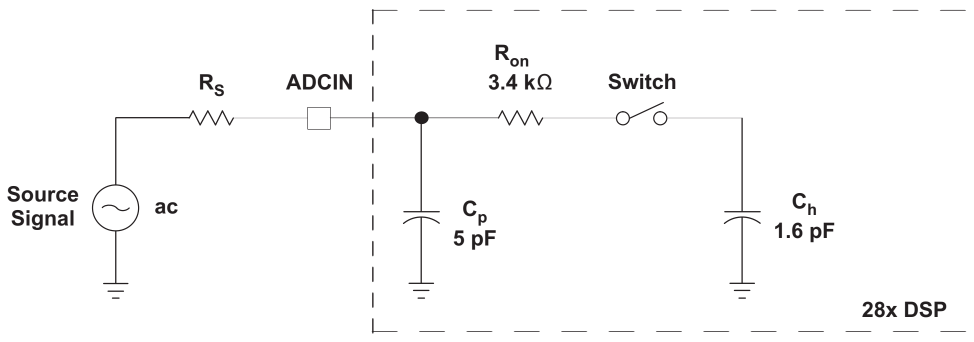
Table 8-1. Sample Timings with Different Values of ACQPS

SYSCLKOUT	ADC Clock	ACQPS	Sample Window	Conversion Time (13 cycles)	Total Time to Process Analog Voltage[(1)]
90Mhz	45MHz	6	155.56ns	288.89ns	444.44ns
90Mhz	45MHz	25	577.78ns	288.89ns	866.67ns

(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 8-3 , the ADCIN pins can be modeled as an RC circuit. With VREFLO connected to ground, a voltage swing from 0 to 3.3v on ADCIN yields a typical RC time constant of 2ns.

Figure 8-3. ADCINx Input Model



Typical Values of the Input Circuit Components:

Switch Resistance (R_{on}): 3.4 kΩ
Sampling Capacitor (C_h): 1.6 pF
Parasitic Capacitance (C_p): 5 pF
Source Resistance (R_S): 50 Ω

8.1.3.2 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
- ePWM1-8 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 1.6 for information on the ADC interrupt signals.

8.1.3.3 Channel Selection

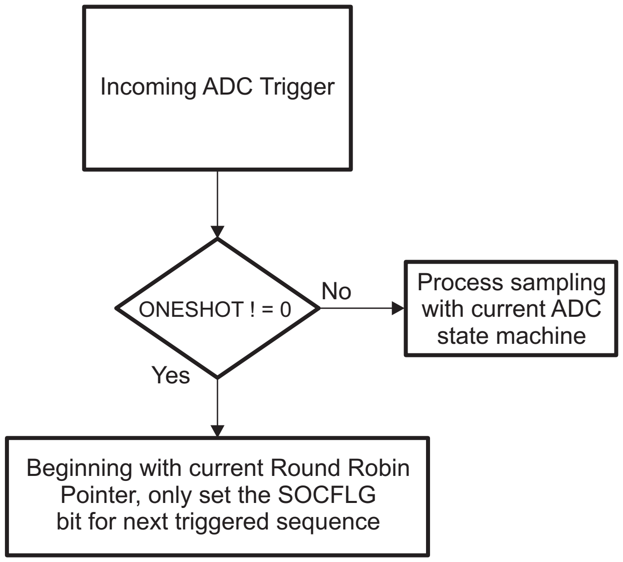
Each SOC can be configured to convert any of the available ADCIN input channels. When an SOC is configured for sequential sampling mode, the four bit CHSEL field of the ADCSOCxCTL register defines which channel to convert. When an SOC is configured for simultaneous sampling mode, the most significant 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 8.1.10 for details on this mode.

8.1.4 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 SOCPRIORITY field in the ADCSOCPRIORITYCTL register.

Figure 8-4. ONESHOT Single Conversion


Incoming ADC Trigger
ONESHOT ! = 0
No
Process sampling with current ADC state machine
Yes
Beginning with current Round Robin Pointer, only set the SOCFLG bit for next triggered sequence

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 SOCPRIORITY = 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 SOCs must eventually be triggered, or else the ADC will not generate new SOCs for other out-of-order triggers. Any non-orthogonal channels should be placed in the priority mode which is unaffected by ONESHOT mode

8.1.5 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 (RRPOINTER). The RRPOINTER reflected in the ADCSOCPRIORITYCTL register points to the last SOC converted. The highest priority SOC is given to the next value greater than the RRPOINTER value, wrapping around back to SOC0 after SOC15. At reset the value is 32 since 0 indicates a conversion has already occurred. When RRPOINTER equals 32 the highest priority is given to SOC0. The RRPOINTER is reset by a device reset, when the ADCCTL1.RESET bit is set, or when the SOCPRICTL register is written.

An example of the round robin priority method is given in Figure 8-5 .

Figure 8-5. Round Robin Priority Example

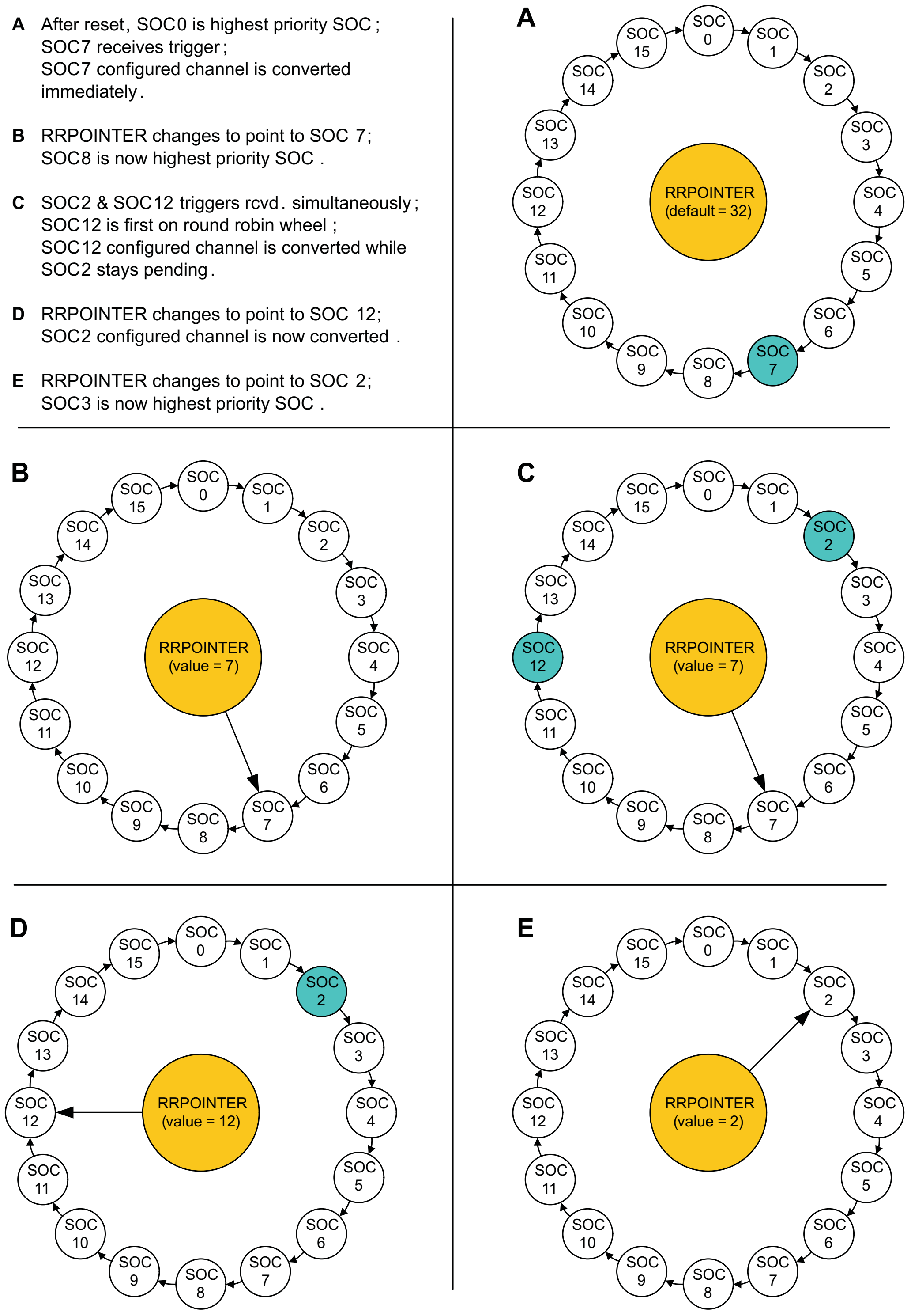
A After reset, SOC0 is highest priority SOC ;
SOC7 receives trigger ;
SOC7 configured channel is converted immediately .

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 .



The SOCPRIORITY field in the ADCSOCPRIORITYCTL 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 SOCPRIORITY field defines the first SOC that is not high priority. In other words, if a value of 4 is written into SOCPRIORITY, 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 8-6 .

Figure 8-6. High Priority Example

Example when SOCPRIORITY = 4

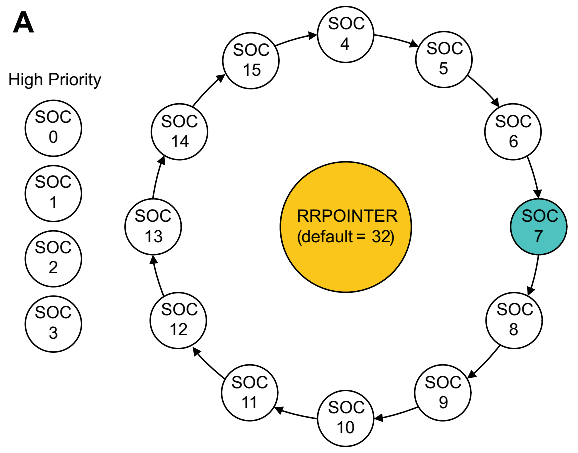
A After reset, SOC4 is 1st on round robin wheel ;
SOC7 receives trigger ;
SOC7 configured channel is converted immediately .

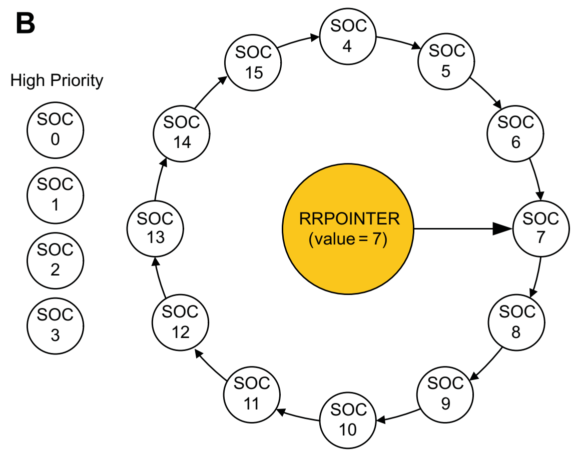
B RRPOINTER changes to point to SOC 7;
SOC8 is now 1st on round robin wheel .

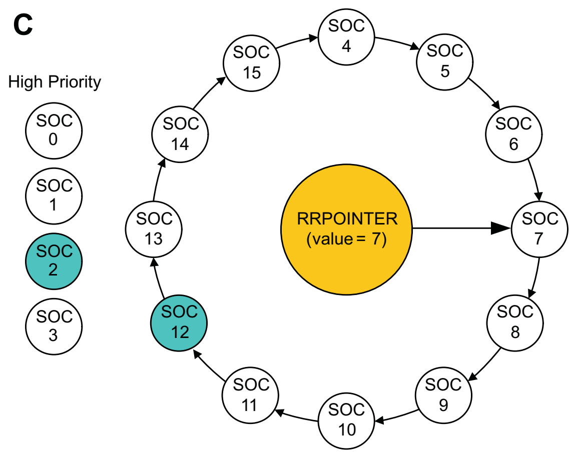
C SOC2 & SOC12 triggers rcvd. simultaneously ;
SOC2 interrupts round robin wheel and SOC 2 configured channel is converted while SOC 12 stays pending .

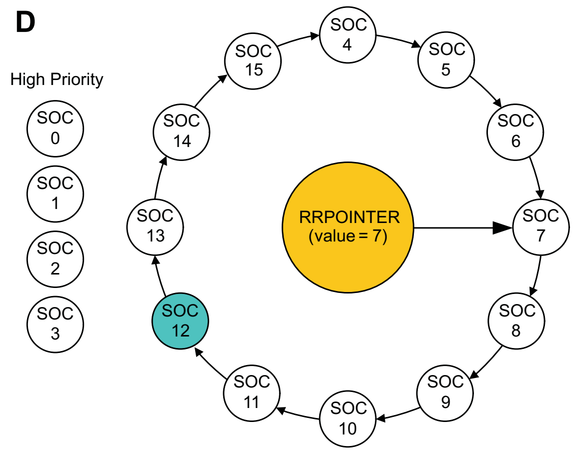
D RRPOINTER stays pointing to 7;
SOC12 configured channel is now converted .

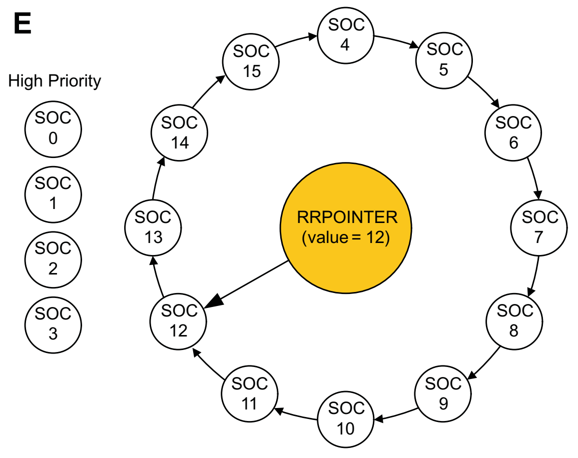
E RRPOINTER changes to point to SOC 12;
SOC13 is now 1st on round robin wheel .











8.1.6 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 8.1.12 shows the timing of simultaneous sampling mode.

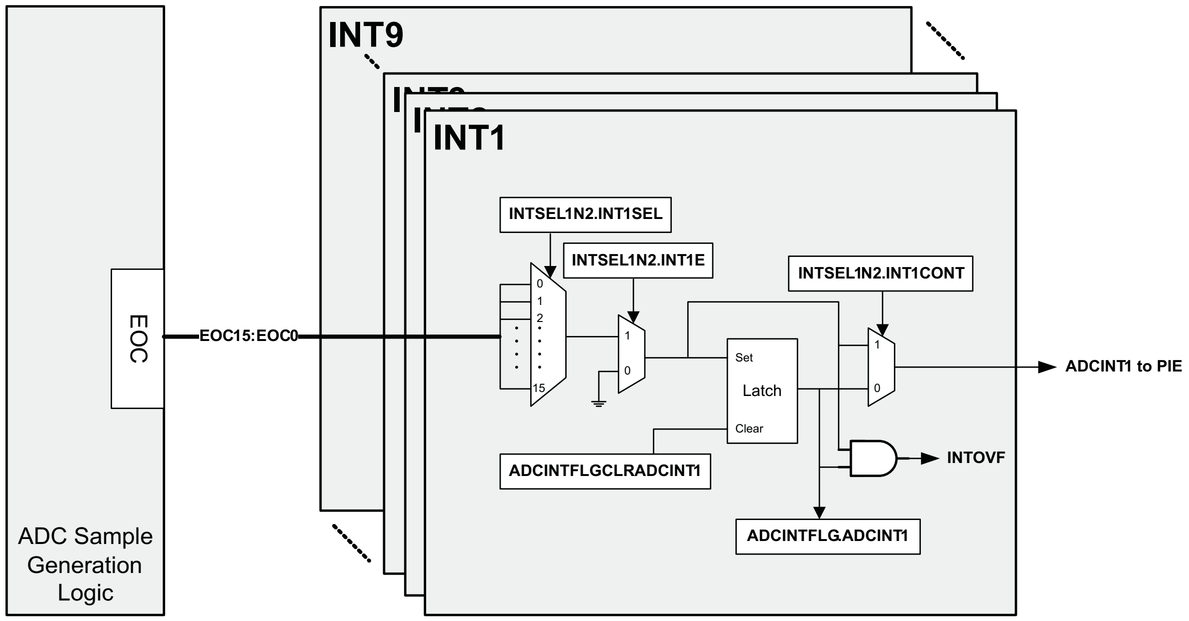
8.1.7 EOC and Interrupt Operation

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 8.1.6. 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 8-7 shows a block diagram of the interrupt structure of the ADC.

Figure 8-7. Interrupt Structure



8.1.8 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 (ADCREFSEL) 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.

8.1.9 ADC Calibration

Inherent in any converter is a zero offset error and a full scale gain error. The ADC is factory calibrated at 25-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.

8.1.9.1 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 datasheet. This can be done either by calling this examples as a part of controlSUITE.unction manually or in the application itself, or by a direct write via CCS. A gel function is provided as part of the *F2806x C/.C++ Header Files and Peripheral Examples* as a part of controlSUITE.

For more information on the Device_cal() function refer to the Boot ROM section in this manual.

Texas Instruments cannot guarantee the parameters specified in the datasheet if a value other than the factory settings contained in the TI reserved OTP memory is written into the ADC trim registers.

8.1.9.2 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 for more details.
3. **Perform multiple conversions on B5 (sample VREFLO) and take an average to account for board noise.** See Section 8.1.3 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 F2806x_Adc.c in the common header files performs these steps.

8.1.9.3 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 ADCREFTRIM register. This register should not be modified after the Device_cal() function is called.

8.1.9.4 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.

8.1.10 Internal/External Reference Voltage Selection

8.1.10.1 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 3.3v range. The equation governing conversions in this mode is:

Digital Value = 0	when Input ≤ 0v
Digital Value = 4096 [(Input – VREFLO)/3.3v]	when 0v < Input < 3.3v
Digital Value = 4095,	when Input ≥ 3.3v

*All fractional values are truncated

**VREFLO must be tied to ground in this mode. This is done internally on some devices.

8.1.10.2 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 3.3v 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 0.5v and VREFHI is 3.0v, a voltage of 1.75v will be converted to the digital result of 2048. See the device datasheet for the allowable ranges of VREFLO and VREFHI. On some devices VREFLO is tied to ground internally, and hence limited to 0v. The equation governing the conversions in this mode is:

Digital Value = 0	when Input ≤ VREFLO
Digital Value = 4096 [(Input – VREFLO)/(VREFHI – VREFLO)]	when VREFLO < Input < VREFHI
Digital Value = 4095,	when Input ≥ VREFHI

*All fractional values are truncated

8.1.11 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 datasheet for specific addresses.

Table 8-2. ADC Configuration and Control Registers (AdcRegs and AdcResult):

Register Name	Address Offset	Size (x16)	Description
ADCCTL1	0x00	1	Control 1 Register[1]
ADCCTL2	0x01	1	Control 2 Register[1]
ADCINTFLG	0x04	1	Interrupt Flag Register
ADCINTFLGCLR	0x05	1	Interrupt Flag Clear Register
ADCINTOVF	0x06	1	Interrupt Overflow Register
ADCINTOVFCLR	0x07	1	Interrupt Overflow Clear Register
INTSEL1N2	0x08	1	Interrupt 1 and 2 Selection Register[1]
INTSEL3N4	0x09	1	Interrupt 3 and 4 Selection Register[1]
INTSEL5N6	0x0A	1	Interrupt 5 and 6 Selection Register[1]
INTSEL7N8	0x0B	1	Interrupt 7 and 8 Selection Register[1]
INTSEL9N10	0x0C	1	Interrupt 9 Selection Register (reserved Interrupt 10 Selection)[1]
SOCPRICTL	0x10	1	SOC Priority Control Register[1]
ADCSAMPLEMODE	0x12	1	Sampling Mode Register[1]
ADCINTSOCSEL1	0x14	1	Interrupt SOC Selection 1 Register (for 8 channels)[1]
ADCINTSOCSEL2	0x15	1	Interrupt SOC Selection 2 Register (for 8 channels)[1]
ADCSOCFLG1	0x18	1	SOC Flag 1 Register (for 16 channels)
ADCSOCFRC1	0x1A	1	SOC Force 1 Register (for 16 channels)
ADCSOCOVF1	0x1C	1	SOC Overflow 1 Register (for 16 channels)
ADCSOCOVFCLR1	0x1E	1	SOC Overflow Clear 1 Register (for 16 channels)
ADCSOC0CTL - ADCSOC15CTL	0x20 - 0x2F	1	SOC0 Control Register to SOC15 Control Register[1]
ADCREFTRIM	0x40	1	Reference Trim Register[1]
ADCOFFTRIM	0x41	1	Offset Trim Register[1]
COMPHYSTCTL	0x4C	1	Comp Hysteresis Control Register[1]
ADCREV – reserved	0x4F	1	Revision Register
ADCRESULT0 - ADCRESULT15	0x00 - 0x0F[2]	1	ADC Result 0 Register to ADC Result 15 Register

[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 AdcRegs.

8.1.11.1 ADC Control Register 1 (ADCCTL1)

NOTE: The following ADC Control Register is EALLOW protected.

Figure 8-8. ADC Control Register 1 (ADCCTL1) (Address Offset 00h)

15	14	13	12-8
RESET	ADCENABLE	ADCBSY	ADCBSYCHN
R-0/W-1	R/W-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 8-3. ADC Control Register 1 (ADCCTL1) Field Descriptions

Bit	Field	Value	Description
15	RESET		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.
		0	no effect
		1	Resets entire ADC module (bit is then set back to 0 by ADC logic)
			Note: 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:
			MOV ADCCTL1, #1xxxxxxxxxxxxxxxb ; Resets the ADC (RESET = 1)
			NOP ; Delay two cycles
			NOP
			MOV ADCCTL1, #0xxxxxxxxxxxxxxxb ; Set to user-desired value
			Note: The second MOV is not required if the default configuration is sufficient. **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 .
14	ADCENABLE		ADC Enable
		0	ADC disabled (**does not** power down ADC)
		1	ADC Enabled. Musts set before an ADC conversion (recommend that it be set directly after setting ADC power-up bits
13	ADCBSY		ADC Busy
			Set when ADC SOC is generated, cleared per below. Used by the ADC state machine to determine if ADC is avaliable to sample.
			Sequential Mode: Cleared 4 ADC clocks after negative edge of S/H pulse
			Simultaneous Mode: Cleared 14 ADC clocks after negative edge of S/H pulse
		0	ADC is available to sample next channel
		1	ADC is busy and cannot sample another channel
12-8	ADCBSYCHN		Set when ADC SOC for current channel is generated
			When ADCBSY = 0: holds the value of the last converted channel
			When ADCBSY = 1: reflects channel currently being processed
		00h	ADCINA0 is currently processing or was last channel converted
		01h	ADCINA1 is currently processing or was last channel converted
		02h	ADCINA2 is currently processing or was last channel converted
		03h	ADCINA3 is currently processing or was last channel converted
		04h	ADCINA4 is currently processing or was last channel converted
		05h	ADCINA5 is currently processing or was last channel converted
		06h	ADCINA6 is currently processing or was last channel converted
		07h	ADCINA7 is currently processing or was last channel converted
		08h	ADCINB0 is currently processing or was last channel converted
		09h	ADCINB1 is currently processing or was last channel converted
		0Ah	ADCINB2 is currently processing or was last channel converted
		0Bh	ADCINB3 is currently processing or was last channel converted
		0Ch	ADCINB4 is currently processing or was last channel converted
		0Dh	ADCINB5 is currently processing or was last channel converted
		0Eh	ADCINB6 is currently processing or was last channel converted
		0Fh	ADCINB7 is currently processing or was last channel converted
		1xh	Invalid value

Table 8-3. ADC Control Register 1 (ADCCTL1) Field Descriptions (continued)

Bit	Field	Value	Description
7	ADCPWDN		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 referencce circuitry is powered down
		1	The analog circuitry inside the core is powered up
6	ADCBGPWD		Bandgap circuit power down (active low)
		0	Bandgap circuitry is powered down
		1	Bandgap buffer's curcuitry inside core is powered up
5	ADCREFPWD		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	ADCREFSEL		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		INT Pulse Generation control
		0	INT pulse generation occurs when ADC begins conversion (neg edge of sample pulse od the sampled signal)
		1	INT pulse generation occurs 1 cycle prior to ADC result latching into its result register
1	VREFLOCONV		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
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 uneffected 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

8.1.11.2 ADC Control Register 2 (ADCCTL2)

NOTE: The following ADC Control Register is EALLOW protected.

Figure 8-9. ADC Control Register 2 (ADCCTL2) (Address Offset 01h)

15–3	2	1	0
Reserved	CLKDIV4EN	ADCNONOVERLAP	CLKDIV2EN
R-0	R/W-0	R/W-0	R/W-0

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

Table 8-4. ADC Control Register 2 (ADCCTL2) Field Descriptions

Bit	Field	Value	Description
15-3	Reserved	0	Reads return a zero; writes have no effect.
2	CLKDIV4EN		When enabled, divides the ADC input clock by 4
		0	ADC clock = CPU clock
		1	ADC clock = CPU clock/4
1	ADCNONOVERLAP		ADCNONOVERLAP control bit
		0	Overlap of sample and conversion is allowed
		1	Overlap of sample is not allowed
0	CLKDIV2EN		When enabled, divides the ADC input clock by 2. When running /2 ADCCLK, scale the minimum sample duration accordingly to meet 116.6ns for better throughput.
		0	ADC clock = CPU clock
		1	ADC clock = CPU clock/2
			Combined with the CLKDIV4EN bit, this determines the ADC Clock to CPU Clock ratio.

CLKDIV2EN	CLKDIV4EN	ADCCLK
0	0	SYSCLK
0	1	SYSCLK
1	0	SYSCLK / 2
1	1	SYSCLK / 4

8.1.11.3 ADC Interrupt Registers

Figure 8-10. ADC Interrupt Flag Register (ADCINTFLG) (Address Offset 04h)

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 8-5. 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	ADCINTx (x = 9 to 1)		ADC Interrupt Flag Bits: Reading this bit indicates if an ADCINT pulse was generated
		0	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. Rather, an ADC interrupt overflow event occurs in the ADCINTOVF register. **Boundary condition for clearing or setting flag bits:** If hardware is trying to set bit while software tries to clear the bit in the same cycle, the following will take place: 1. SW has priority, and will clear the flag 2. HW set will be discarded, no signal will propagate to the PIE form the latch 3. Overflow flag or condition will be generated

Figure 8-11. ADC Interrupt Flag Clear Register (ADCINTFLGCLR) (Address Offset 05h)

15 - 9	8
Reserved	ADCINT9
R-0	R/W-0

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 8-6. 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)		ADC interrupt Flag Clear Bit
		0	No action.
		1	Clears respective flag bit in the ADCINTFLG register. If software tries to set this bit on the same clock cycle that hardware tries to set the flag bit in the ADCINTFLG register, then hardware has priority and the ADCINTFLG bit will be set. In this case the overflow bit in the ADCINTOVF register will not be affected regardless of whether the ADCINTFLG bit was previously set or not.

Figure 8-12. ADC Interrupt Overflow Register (ADCINTOVF) (Address Offset 06h)

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 8-7. ADC Interrupt Overflow Register (ADCINTOVF) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved
8-0	ADCINTx (x = 9 to 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.

Figure 8-13. ADC Interrupt Overflow Clear Register (ADCINTOVFCLR) (Address Offset 07h)

15 - 9	8
Reserved	ADCINT9
R-0	R-0/W-1

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 8-8. 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	ADCINTx (x = 9 to 1)		ADC Interrupt Overflow Clear Bits.
		0	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.

NOTE: The following Interrupt Select Registers are EALLOW protected.

Figure 8-14. Interrupt Select 1 And 2 Register (INTSEL1N2) (Address Offset 08h)

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 8-15. Interrupt Select 3 And 4 Register (INTSEL3N4) (Address Offset 09h)

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 8-16. Interrupt Select 5 And 6 Register (INTSEL5N6) (Address Offset 0Ah)

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 8-17. Interrupt Select 7 And 8 Register (INTSEL7N8) (Address Offset 0Bh)

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 8-18. Interrupt Select 9 And 10 Register (INTSEL9N10) (Address Offset 0Ch)

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 8-9. INTSELxNy Register Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved
14	INTyCONT		ADCINTy Continuous Mode Enable
		0	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		ADCINTy Interrupt Enable
		0	ADCINTy is disabled.
		1	ADCINTy is enabled.
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		ADCINTx Continuous Mode Enable.
		0	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		ADCINTx Interrupt Enable
		0	ADCINTx is disabled.
		1	ADCINTx is enabled .

Table 8-9. INTSELxNy Register Field Descriptions (continued)

Bit	Field	Value	Description
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.

8.1.11.4 ADC Priority Register

NOTE: The following SOC Priority Control Register is EALLOW protected.

Figure 8-19. 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 8-10. SOCPRICTL Register Field Descriptions

Bit	Field	Value	Description
15	ONESHOT	0	One shot mode disabled
		1	One shot mode enabled
14-11	Reserved		Reads return a zero; Writes have no effect.

Table 8-10. SOCPRICTL Register Field Descriptions (continued)

Bit	Field	Value	Description
10-5	RRPOINTER		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.
		00h	SOC0 was last round robin SOC to convert. SOC1 is highest round robin priority.
		01h	SOC1 was last round robin SOC to convert. SOC2 is highest round robin priority.
		02h	SOC2 was last round robin SOC to convert. SOC3 is highest round robin priority.
		03h	SOC3 was last round robin SOC to convert. SOC4 is highest round robin priority.
		04h	SOC4 was last round robin SOC to convert. SOC5 is highest round robin priority.
		05h	SOC5 was last round robin SOC to convert. SOC6 is highest round robin priority.
		06h	SOC6 was last round robin SOC to convert. SOC7 is highest round robin priority.
		07h	SOC7 was last round robin SOC to convert. SOC8 is highest round robin priority.
		08h	SOC8 was last round robin SOC to convert. SOC9 is highest round robin priority.
		09h	SOC9 was last round robin SOC to convert. SOC10 is highest round robin priority.
		0Ah	SOC10 was last round robin SOC to convert. SOC11 is highest round robin priority.
		0Bh	SOC11 was last round robin SOC to convert. SOC12 is highest round robin priority.
		0Ch	SOC12 was last round robin SOC to convert. SOC13 is highest round robin priority.
		0Dh	SOC13 was last round robin SOC to convert. SOC14 is highest round robin priority.
		0Eh	SOC14 was last round robin SOC to convert. SOC15 is highest round robin priority.
		0Fh	SOC15 was last round robin SOC to convert. SOC0 is highest round robin priority.
		1xh	Invalid value
		20h	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.
		Others	Invalid selection.
4-0	SOCPRIORITY		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 SOCs are in high priority mode, arbitrated by SOC number
		Others	Invalid selection.

8.1.11.5 ADC SOC Registers

NOTE: The following ADC Sample Mode Register is EALLOW protected.

Figure 8-20. ADC Sample Mode Register (ADCSAMPLEMODE) (Address Offset 12h)

15–8
Reserved
R-0

7	6	5	4	3	2	1	0
SIMULEN14	SIMULEN12	SIMULEN10	SIMULEN8	SIMULEN6	SIMULEN4	SIMULEN2	SIMULEN0
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 8-11. ADC Sample Mode Register (ADCSAMPLEMODE) Field Descriptions

Bit	Field	Value	Description
15:8	Reserved	0	Reserved
7	SIMULEN14		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.
		0	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		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.
		0	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		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.
		0	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		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.
		0	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 8-11. ADC Sample Mode Register (ADCSAMPLEMODE) Field Descriptions (continued)

Bit	Field	Value	Description
3	SIMULEN6		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.
		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.
		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.
2	SIMULEN4		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.
		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.
		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.
1	SIMULEN2		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.
		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.
		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.
0	SIMULEN0		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.
		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.
		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.

NOTE: The following ADC Interrupt SOC Select Registers are EALLOW protected.

Figure 8-21. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1) (Address Offset 14h)

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 8-12. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1) Register Field Descriptions

Bit	Field	Value	Description
15--0	SOCx (x = 7 to 0)		SOCx ADC Interrupt Trigger Select. Selects which, if any, ADCINT triggers SOCx. This field overrides the TRIGSEL field in the ADCSOCxCTL register.
		00	No ADCINT will trigger SOCx. TRIGSEL field determines SOCx trigger.
		01	ADCINT1 will trigger SOCx. TRIGSEL field is ignored.
		10	ADCINT2 will trigger SOCx. TRIGSEL field is ignored.
		11	Invalid selection.

Figure 8-22. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2) (Address Offset 15h)

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 8-13. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 8)		SOCx ADC Interrupt Trigger Select. Selects which, if any, ADCINT triggers SOCx. This field overrides the TRIGSEL field in the ADCSOCxCTL register.
		00	No ADCINT will trigger SOCx. TRIGSEL field determines SOCx trigger.
		01	ADCINT1 will trigger SOCx. TRIGSEL field is ignored.
		10	ADCINT2 will trigger SOCx. TRIGSEL field is ignored.
		11	Invalid selection.

Figure 8-23. ADC SOC Flag 1 Register (ADCSOCFLG1) (Address Offset 18h)

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 8-14. ADC SOC Flag 1 Register (ADCSOCFLG1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		SOCx Start of Conversion Flag. Indicates the state of individual SOC conversions.
		0	No sample pending for SOCx.
		1	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 **and** 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.

Figure 8-24. ADC SOC Force 1 Register (ADCSOCFRC1) (Address Offset 1Ah)

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 8-15. ADC SOC Force 1 Register (ADCSOCFRC1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		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.
		0	No action.
		1	Force SOCx flag bit to 1. This will cause a conversion to start once priority is given to SOCx.
			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.

Figure 8-25. ADC SOC Overflow 1 Register (ADCSOCOVF1) (Address Offset 1Ch)

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 8-16. ADC SOC Overflow 1 Register (ADCSOCOVF1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		SOCx Start of Conversion Overflow Flag. Indicates an SOCx event was generated while an existing SOCx event was already pending.
		0	No SOCx event overflow
		1	SOCx event overflow
			An overflow condition does not stop SOCx events from being processed. It simply is an indication that a trigger was missed

Figure 8-26. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1) (Address Offset 1Eh)

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 8-17. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		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.
		0	No action.
		1	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.

NOTE: The following ADC SOC0 - SOC15 Control Registers are EALLOW protected.

Figure 8-27. ADC SOC0 - SOC15 Control Registers (ADCSOCxCTL) (Address Offset 20h - 2Fh)

15 - 11	10	9 - 6	5 - 0
TRIGSEL	Reserved	CHSEL	ACQPS
R/W-0	R-0	R/W-0	R/W-0

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

Table 8-18. ADC SOC0 - SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions

Bit	Field	Value	Description
15-11	TRIGSEL		SOCx Trigger Source Select. Configures which trigger will set the respective SOCx flag in the ADCSOCFLG1 register to intiate 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.
		00h	ADCTRIG0 - Software only.
		01h	ADCTRIG1 - CPU Timer 0, TINT0n
		02h	ADCTRIG2 - CPU Timer 1, TINT1n
		03h	ADCTRIG3 - CPU Timer 2, TINT2n
		04h	ADCTRIG4 – XINT2, XINT2SOC
		05h	ADCTRIG5 – ePWM1, ADCSOCA
		06h	ADCTRIG6 – ePWM1, ADCSOCB
		07h	ADCTRIG7 – ePWM2, ADCSOCA
		08h	ADCTRIG8 – ePWM2, ADCSOCB
		09h	ADCTRIG9 – ePWM3, ADCSOCA
		0Ah	ADCTRIG10 – ePWM3, ADCSOCB
		0Bh	ADCTRIG11 – ePWM4, ADCSOCA
		0Ch	ADCTRIG12 – ePWM4, ADCSOCB
		0Dh	ADCTRIG13 – ePWM5, ADCSOCA
		0Eh	ADCTRIG14 – ePWM5, ADCSOCB
		0Fh	ADCTRIG15 – ePWM6, ADCSOCA
		10h	ADCTRIG16 – ePWM6, ADCSOCB
		11h	ADCTRIG17 - ePWM7, ADCSOCA
		12h	ADCTRIG18 - ePWM7, ADCSOCB
		13h	ADCTRIG19 - ePWM8, ADCSOCA
		14h	ADCTRIG20 - ePWM8, ADCSOCB
		Others	Invalid selection.
10	Reserved		Reads return a zero; Writes have no effect.

Table 8-18. ADC SOC0 - SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions (continued)

Bit	Field	Value	Description
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 8-18. ADC SOC0 - SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions (continued)

Bit	Field	Value	Description
5-0	ACQPS		SOCx Acquisition Prescale. Controls the sample and hold window for SOCx. Minimum value allowed is 6.
		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).
Other invalid selections: 10h, 11h, 12h, 13h, 14h, 1Dh, 1Eh, 1Fh, 20h, 21h, 2Ah, 2Bh, 2Ch, 2Dh, 2Eh, 37h, 38h, 39h, 3Ah, 3Bh			

8.1.11.6 ADC Calibration Registers

NOTE: The following ADC Calibration Registers are EALLOW protected.

Figure 8-28. ADC Reference/Gain Trim Register (ADCREFTRIM) (Address Offset 40h)

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 8-19. ADC Reference/Gain Trim Register (ADCREFTRIM) Field Descriptions

Bit	Field	Value	Description
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.

Figure 8-29. ADC Offset Trim Register (ADCOFFTRIM) (Address Offset 41h)

15 - 9	8 - 0
Reserved	OFFTRIM
R-0	R/W-0

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

Table 8-20. ADC Offset Trim Register (ADCOFFTRIM) Field Descriptions

Bit	Field	Value	Description
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.

8.1.11.7 COMP HYSTERESIS CONTROL Register

NOTE: The following COMP HYSTERESIS CONTROL registers are EALLOW protected.

Figure 8-30. COMP HYSTERESIS CONTROL Register

15 12	11	10 7	6	5 2	1	0
Reserved	COMP3_HYST _DISABLE	Reserved	COMP2_HYST _DISABLE	Reserved	COMP1_HYST _DISABLE	Reserved
R-0	R/W--0	R-0	R/W--0	R-0	R/W--0	R-0

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

Table 8-21. COMP HYSTERESIS CONTROL Register Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Reads return a zero; Writes have no effect.
11	COMP3_HYST_DISABLE		.
		0	Hysteresis enable
		1	Hysteresis disable
10-7	Reserved		Reserved
6	COMP2_HYST_DISABLE	0	Hysteresis enable
		1	Hysteresis disable
5-2			
1	COMP1_HYST_DISABLE	0	Hysteresis enable
		1	Hysteresis disable
0	Reserved		Reserved

8.1.11.8 ADC Revision Register

Figure 8-31. ADC Revision Register (ADCREV) (Address Offset 4Fh)

15 8
REV
R-x

7 0
TYPE
R-3h

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

Table 8-22. 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

8.1.11.9 ADC Result Registers

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 8-32. ADC RESULT0 - RESULT15 Registers (ADCRESULTx) (PF1 Block Address Offset 00h - 0Fh)

15 - 12	11 - 0
Reserved	RESULT
R-0	R-0

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

Table 8-23. 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.

8.1.12 ADC Timings

Figure 8-33. Timing Example For Sequential Mode / Late Interrupt Pulse

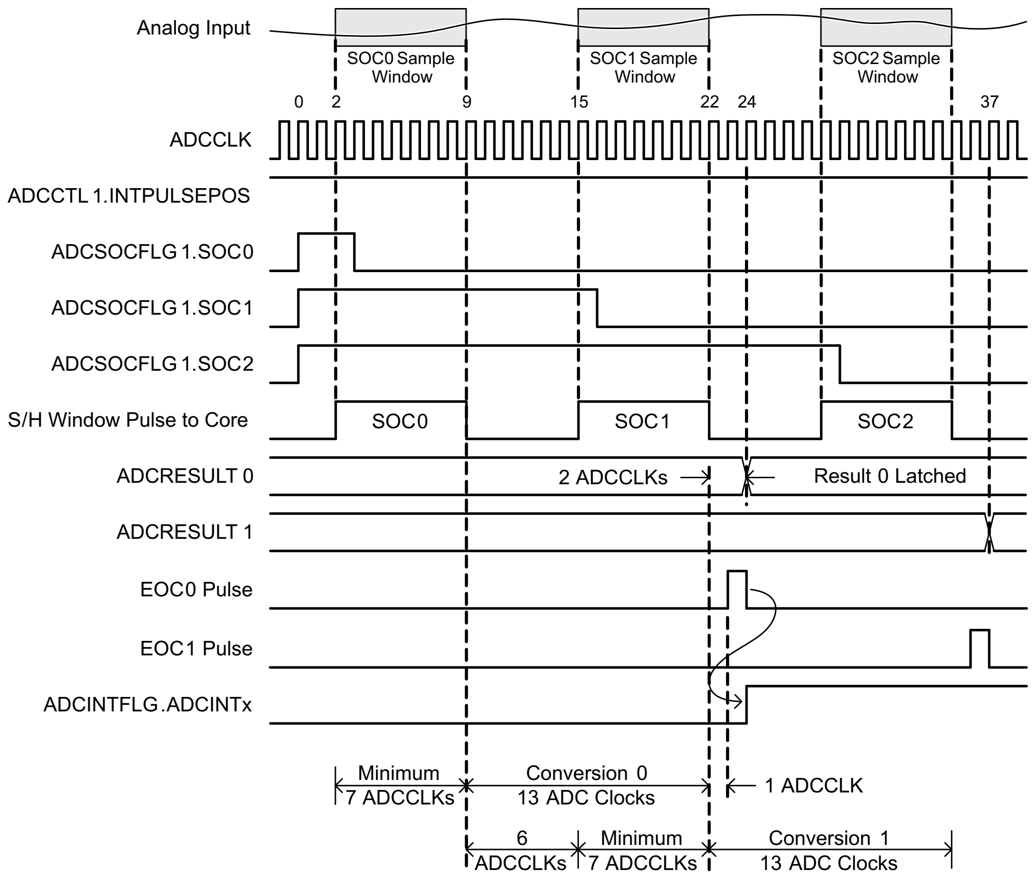


Figure 8-34. Timing Example For Sequential Mode / Early Interrupt Pulse

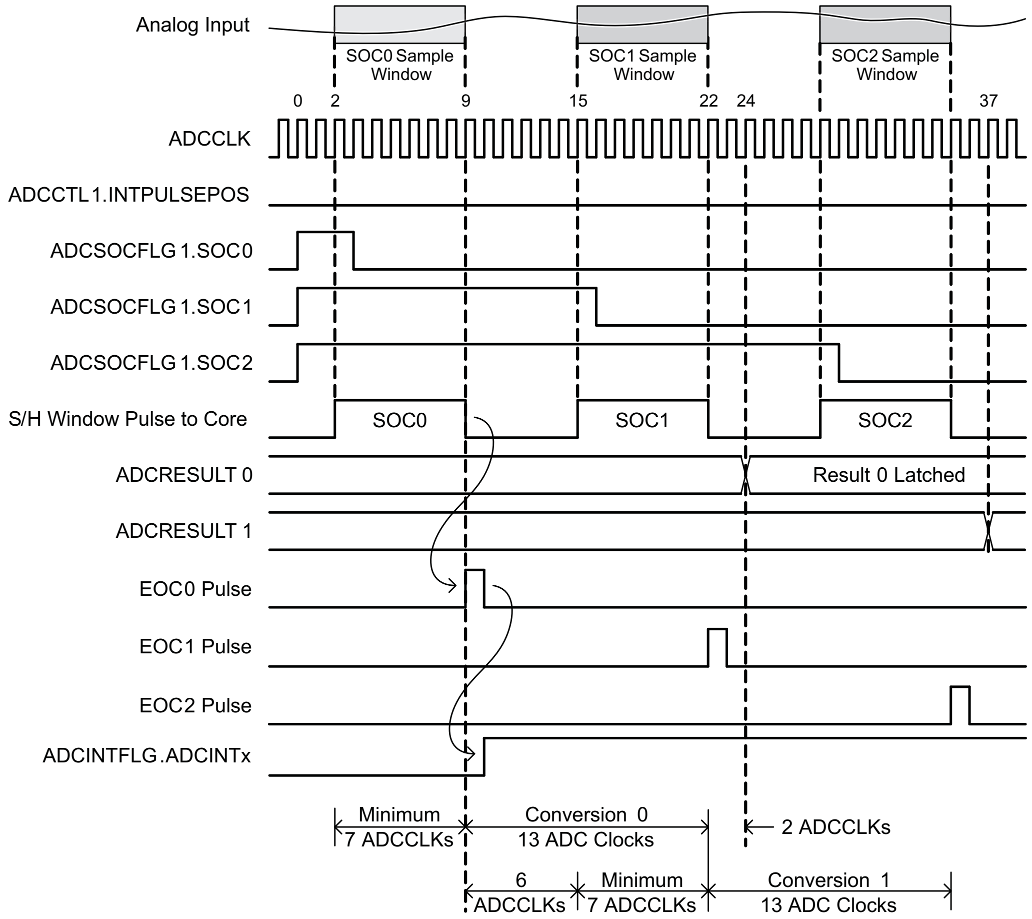


Figure 8-35. Timing Example For Simultaneous Mode / Late Interrupt Pulse

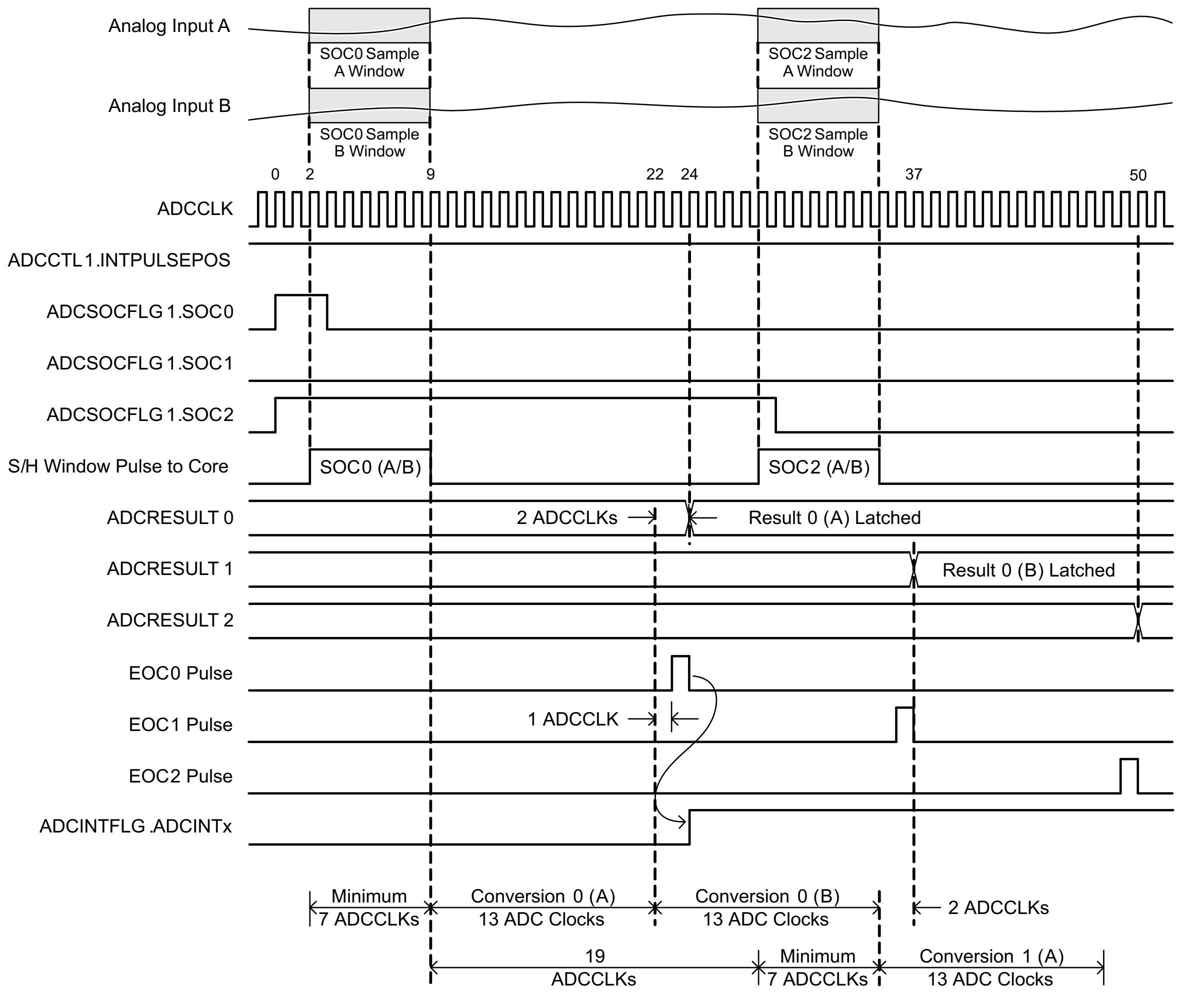


Figure 8-36. Timing Example For Simultaneous Mode / Early Interrupt Pulse

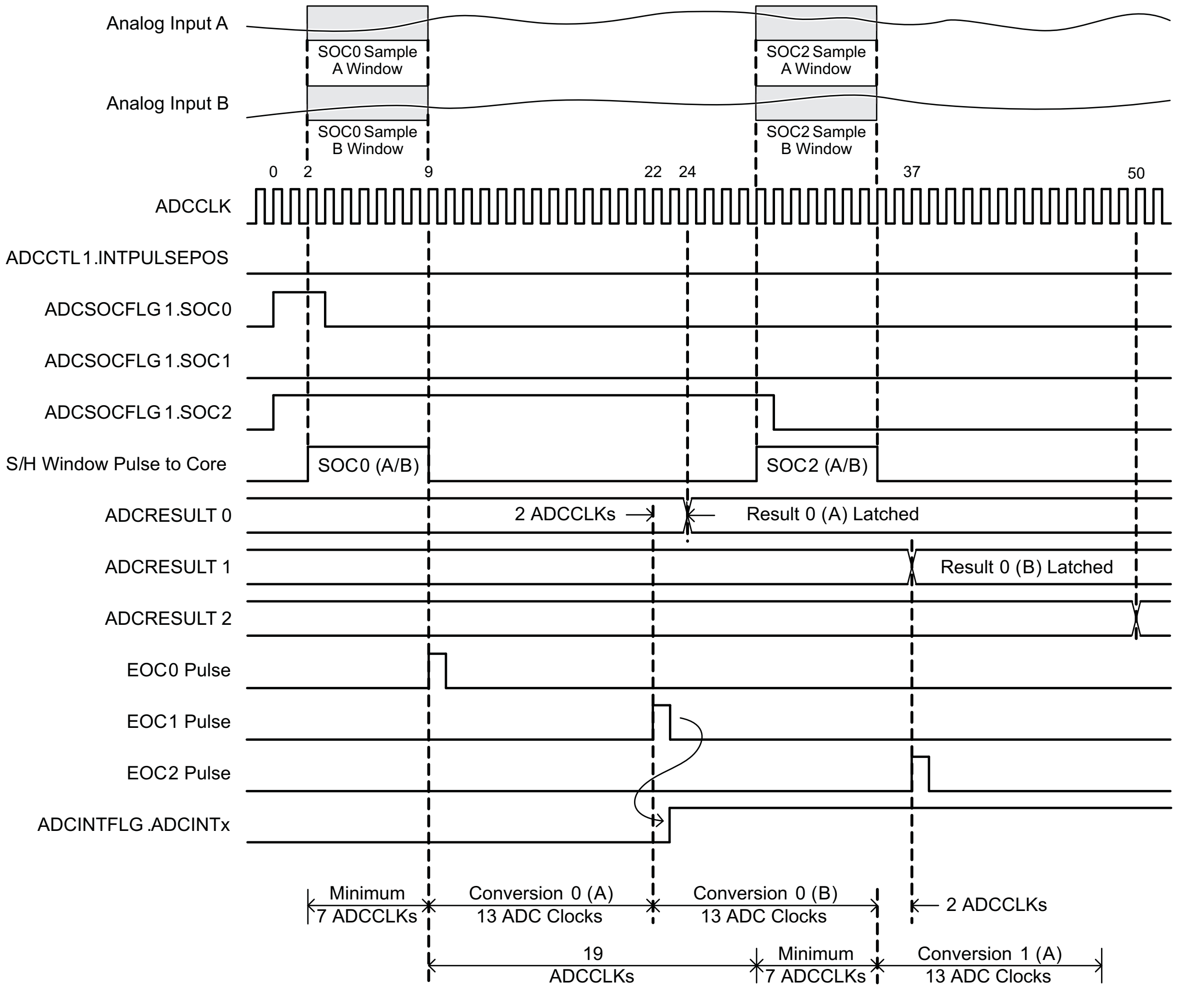
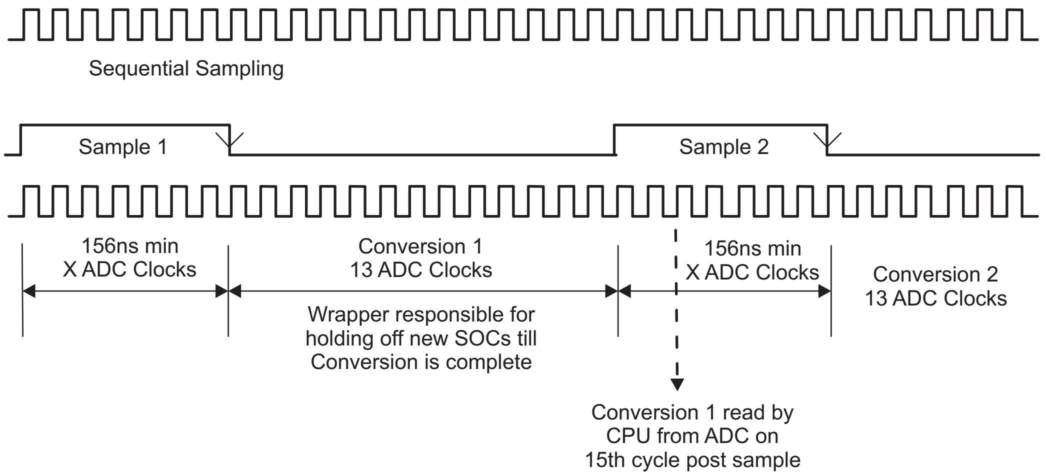


Figure 8-37. Timing Example for NONOVERLAP Mode


Sequential Sampling
Sample 1
Sample 2
156ns min X ADC Clocks
Conversion 1 13 ADC Clocks
156ns min X ADC Clocks
Conversion 2 13 ADC Clocks
Wrapper responsible for holding off new SOCs till Conversion is complete
Conversion 1 read by CPU from ADC on 15th cycle post sample

NOTE: The NONOVERLAP bit in the ADCCTL2 register, when enabled, removes the overlap of sampling and conversion stages.

8.1.13 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 affect on the sample. Refer to Section 8.1.11.1 for information about switching between the external ADCINA5 input pin and the internal temperature sensor.

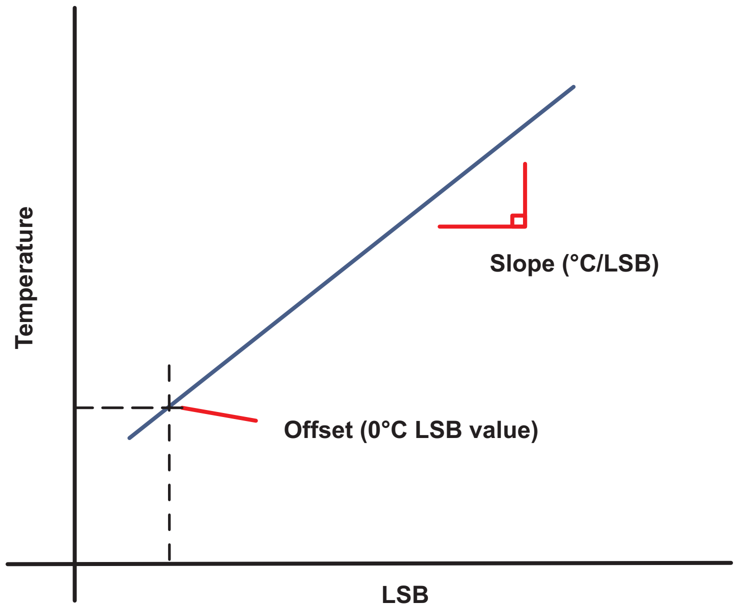
8.1.13.1 Transfer Function

The temperature sensor output and the resulting ADC values increase with increasing junction temperature. The offset is defined as the 0 ºC LSB crossing as illustrated in Figure 8-38. 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:

$$\text{Temperature} = (\textit{sensor} - \text{Offset}) * \text{Slope}$$

Figure 8-38. Temperature Sensor Transfer Function



Refer to the electrical characteristics section in *TMS320F28069, TMS320F28068, TMS320F28067, TMS320F28066, TMS320F28065, TMS320F28064, TMS320F28063, TMS320F28062 Piccolo Microcontrollers Data Manual* (SPRS698) for the slope and offset, or use the stored slope and offset calibrated per device in the factory which can be extract by a function at the following locations.

For F2806x:

- 0x3D7E82 - Slope (ºC / LSB, fixed-point Q15 format)
- 0x3D7E85 - Offset (0 ºC LSB value)

The values listed are assuming a 3.3v 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 threee 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 ºC.

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.ADCSOCFRC1.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)()
```

8.2 Comparator Block

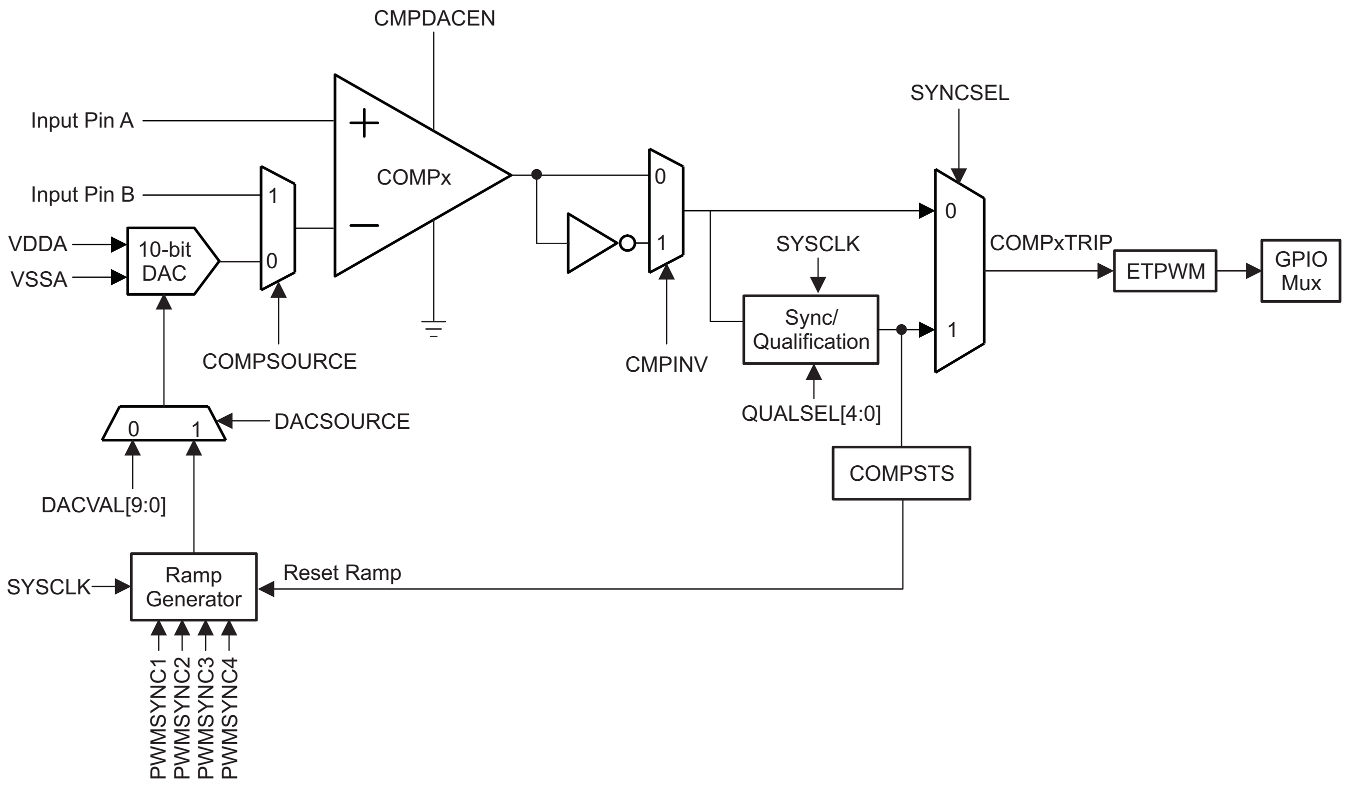
The comparator module described in this reference guide is a true analog voltage comparator in the VDDA domain. The analog portion of the block include the comparator, its inputs and outputs, and the internal DAC reference. The digital circuits, referred to as the wrapper in this document, include the DAC controls, interface to other on-chip logic, output qualification block, and the control signals.

8.2.1 Features

The comparator block can accommodate two external analog inputs or one external analog input using the internal DAC reference for the other input. The output of the comparator can be passed asynchronously or 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.

8.2.2 Block Diagram

Figure 8-39. Comparator Block Diagram



8.2.3 Comparator Function

The comparator in each comparator block is an analog comparator module, and as such its output is asynchronous to the system clock. The truth table for the comparator is shown in Table 8-24.

Figure 8-40. Comparator

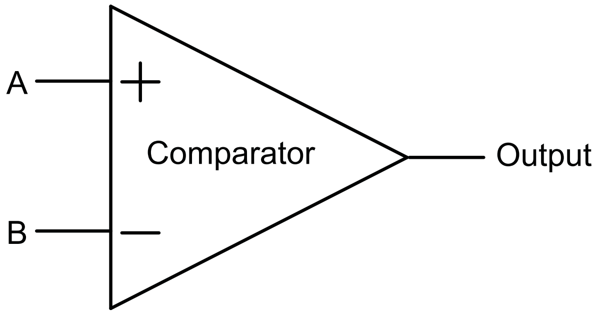


Table 8-24. Comparator Truth Table

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 datasheet 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. Since this bit is part of the wrapper, clocks must be enabled to the comparator block for the COMPSTS bit to actively show the comparator state.

8.2.4 DAC Reference

Each comparator block contains a 10-bit voltage DAC reference that can 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:

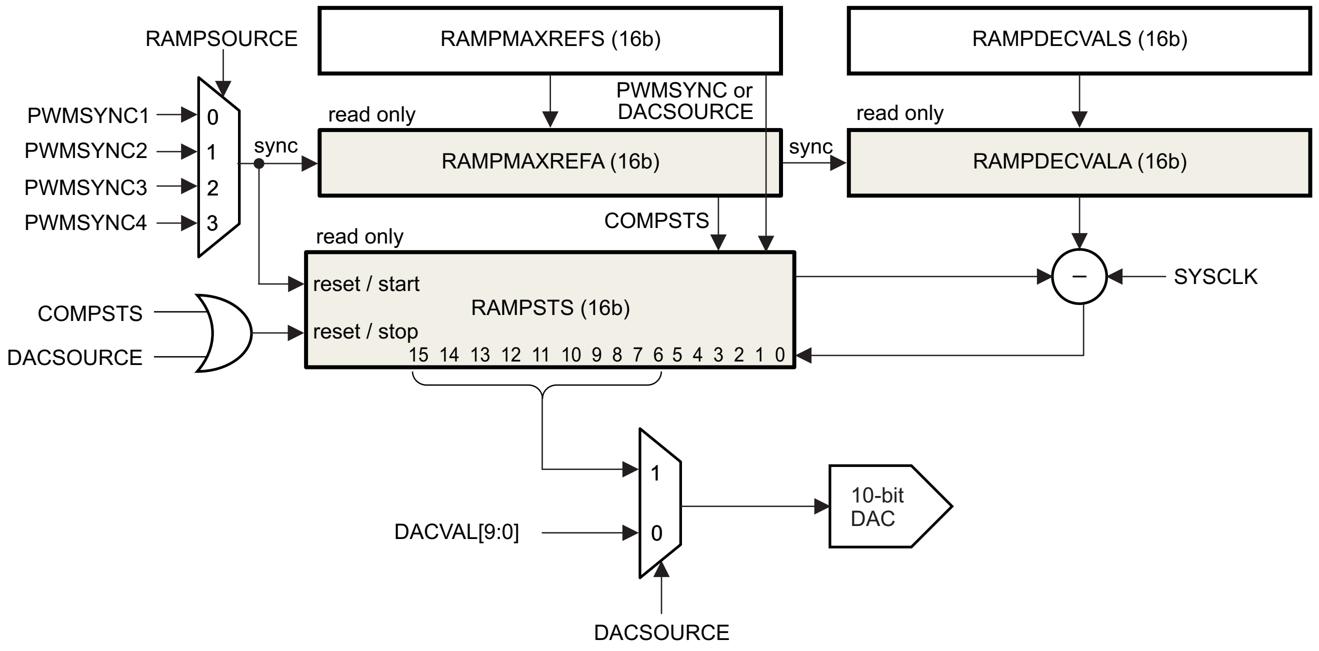
$$V = \frac{\text{DACVAL} * (\text{VDDA-VSSA})}{1023}$$

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.

8.2.5 Ramp Generator Input

When selected, the ramp generator (see Figure 8-41) 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 8-41. Ramp Generator Block Diagram



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.

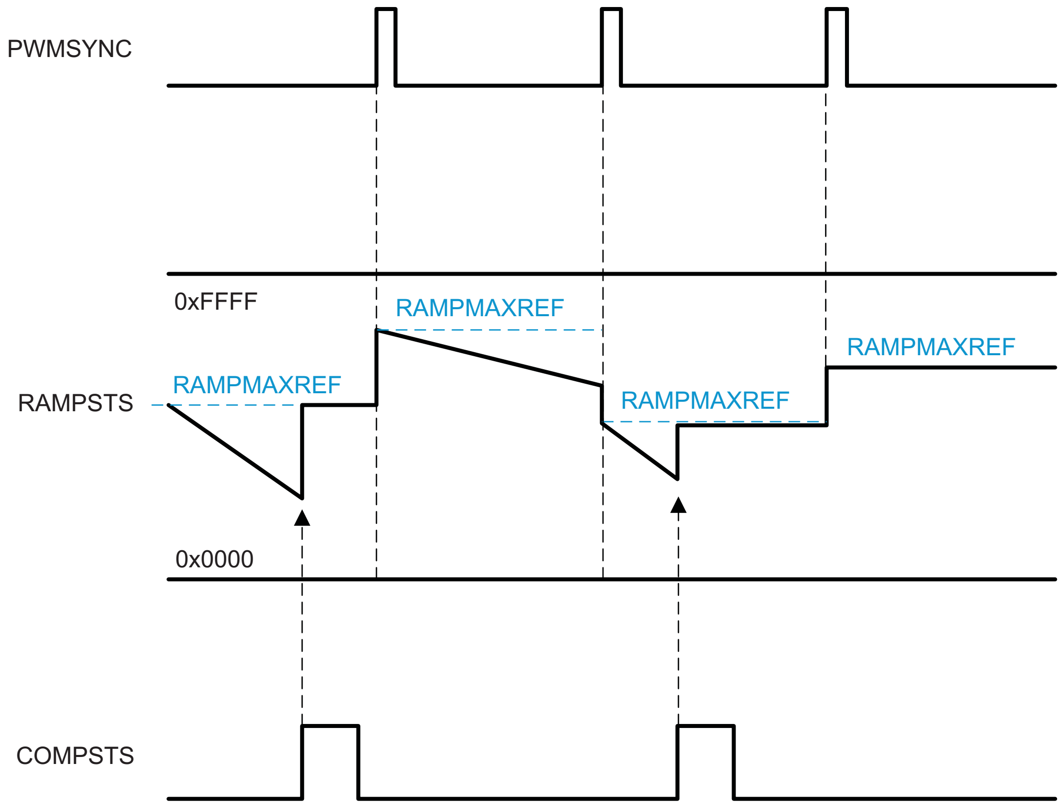
If the COMPSTS bit is set by the comparator while the ramp generator is active, the RAMPSTS register will reset to the value of RAMPMAXREF_ACTIVE and remain static until the next PWMSYNC signal is received. If the value of RAMPSTS reaches zero, the RAMPSTS register will remain static at zero until the 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 8-42

Figure 8-42. Ramp Generator Behavior



8.2.6 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.

8.2.7 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.

8.2.8 Comparator Registers

These devices have three comparators as shown ibelow and described in Table 8-25.

Name	Address Range	Size(x16)	Description
COMP1	6400h – 641Fh	1	Comparator
COMP2	6420h – 643Fh	1	Comparator
COMP3	6440h – 645Fh	1	Comparator

Table 8-25. Comparator Module Registers

Name	Address Range(base)	Size(x16)	Description
COMPCTL	0x00	1	Comparator Control[(1)]
Reserved	0x01	1	Reserved
COMPSTS	0x02	1	Compare Output Status
Reserved	0x03	1	Reserved
DACCTL	0x04	1	DAC Control[(1)]
Reserved	0x05	1	Reserved
DACVAL	0x06	1	10-bit DAC Value
Reserved	0x07	1	Reserved
RAMPMAXREF_ACTIVE	0x08	1	Ramp Generator Maximum Reference (Active)
Reserved	0x09	1	Reserved
RAMPMAXREF_SHDW	0x0A	1	Ramp Generator Maximum Reference (Shadow)
Reserved	0x0B	1	Reserved
RAMPDECVAL_ACTIVE	0x0C	1	Ramp Generator Decrement Value (Active)
Reserved	0x0D	1	Reserved
RAMPDECVAL_SHDW	0x0E	1	Ramp Generator Decrement Value (Shadow)
Reserved	0x0F	1	Reserved
RAMPSTS	0x10	1	Ramp Generator Status
Reserved	0x11 0x1F	15	Reserved

(1) This register is EALLOW protected.

8.2.8.1 Comparator Control (COMPCTL) Register

Figure 8-43. Comparator Control (COMPCTL) Register

15–9	8
Reserved	SYNCSEL
R-0	R/W-0

7–3	2	1	0
QUALSEL	CMPINV	COMPSOURCE	COMPDACE
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 8-26. COMPCTL Register Field Descriptions

Bit	Field	Value	Description
15-9	Reserved		Reads return a 0; Writes have no effect.
8	SYNCSEL		Synchronization select for output of the comparator before being passed to ETPWM/GPIO blocks
		0	Asynchronous version of Comparator output is passed
		1	Synchronous version of comparator output is passed
7-3	QUALSEL		Qualification Period for synchronized output of the comparator
		0h	Synchronized value of comparator is passed through
		1h	Input to the block must be consistent for 2 consecutive clocks before output of Qual block can change
		2h	Input to the block must be consistent for 3 consecutive clocks before output of Qual block can change
		...	...
		Fh	Input to the block must be consistent for 16 consecutive clocks before output of Qual block can change
2	CMPINV		Invert select for Comparator
		0	Output of comparator is passed
		1	Inverted output of comparator is passed
1	COMPSOURCE		Source select for comparator inverting input
		0	Inverting input of comparator connected to internal DAC
		1	Inverting input connected to external pin
0	COMPDACE		Comparator/DAC Enable
		0	Comparator/DAC logic is powered down.
		1	Comparator/DAC logic is powered up.

8.2.8.2 Compare Output Status (COMPSTS) Register

Figure 8-44. Compare Output Status (COMPSTS) Register

15 - 1	0
Reserved	COMPSTS
R-0	R-0

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

Table 8-27. 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

8.2.8.3 DAC Control (DACCTL) Register

Figure 8-45. DAC Control (DACCTL) Register

15 - 14	13 - 8
FREE:SOFT	Reserved
R/W-0	R-0

7 - 5	4 - 1	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 8-28. DACCTL Register Field Descriptions

Bit	Field	Value	Description
15-14	FREE:SOFT		Emulation mode behavior. Selects ramp generator behavior during emulation suspend.
		0h	Stop immediately
		1h	Complete current ramp, and stop on the next PWMSYNC signal
		2h-3h	Run free
13-5	Reserved		Reads return a 0; Writes have no effect.
4-1	RAMPSOURCE		Ramp generator source sync select
		0h	PWMSYNC1 is the source sync
		1h	PWMSYNC2 is the source sync
		2h	PWMSYNC3 is the source sync
		3h	PWMSYNC4 is the source sync
		4h-Fh	Reserved
0	DACSOURCE		DAC source control. Select DACVAL or ramp generator to control the DAC.
		0	DAC controlled by DACVAL
		1	DAC controlled by ramp generator

8.2.8.4 DAC Value (DACVAL) Register

Figure 8-46. DAC Value (DACVAL) Register

15 - 10	9 - 0
Reserved	DACVAL
R-0	R/W-0

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

Table 8-29. 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.

8.2.8.5 Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register

Figure 8-47. Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register

15 - 0
RAMPMAXREFA
R-0

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

Table 8-30. 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.

8.2.8.6 Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register

Figure 8-48. Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register

15 … 0
RAMPMAXREFS
R/W-0

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

Table 8-31. 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

8.2.8.7 Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register

Figure 8-49. Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register

15 … 0
RAMPDECVALA
R-0

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

Table 8-32. 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.

8.2.8.8 Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register

Figure 8-50. Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register

15 … 0
RAMPDECVALS
R/W-0

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

Table 8-33. 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

8.2.8.9 Ramp Generator Status (RAMPSTS) Register

Figure 8-51. Ramp Generator Status (RAMPSTS) Register

15 … 0
RAMPVALUE
R-0

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

Table 8-34. Ramp Generator Status (RAMPSTS) Register Field Descriptions

Bit	Field	Value	Description
15-0	RAMPVALUE	0-FFFFh	16-bit value of down ramp generator

Chapter 9

SPRUH18E – January 2011 – Revised March 2014

Control Law Accelerator (CLA)

The control law accelerator (CLA) is an independent, fully-programmable, 32-bit floating-point math processor that brings concurrent control-loop execution to the C28x family. The low interrupt-latency of the CLA allows it to read ADC samples "just-in-time." This significantly reduces the ADC sample to output delay to enable faster system response and higher MHz control loops. By using the CLA to service time-critical control loops, the main CPU is free to perform other system tasks such as communications and diagnostics. This chapter provides an overview of the architectural structure and components of the control law accelerator.

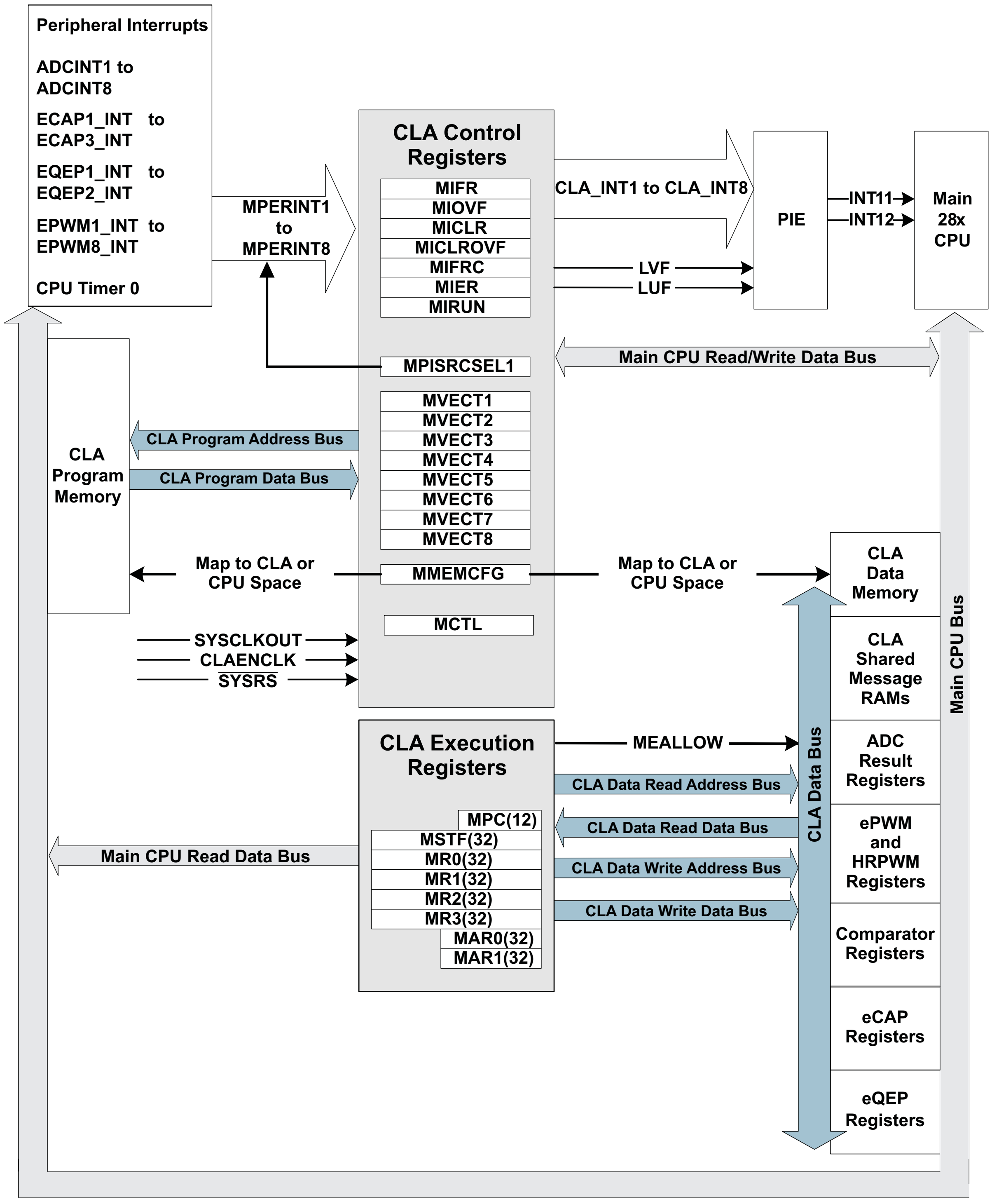
Topic **Page**

9.1 Control Law Accelerator (CLA) Overview

The control law accelerator extends the capabilities of the C28x CPU by adding parallel processing. Time-critical control loops serviced by the CLA can achieve low ADC sample to output delay. Thus, the CLA enables faster system response and higher frequency control loops. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently. The following is a list of major features of the CLA.

- Clocked at the same rate as the main CPU (SYSCLKOUT).
- An independent architecture allowing CLA algorithm execution independent of the main C28x CPU.
  - Complete bus architecture:
    - Program address bus and program data bus
    - Data address bus, data read bus and data write bus
  - Independent eight stage pipeline.
  - 12-bit program counter (MPC)
  - Four 32-bit result registers (MR0-MR3)
  - Two 16-bit auxiliary registers (MAR0, MAR1)
  - Status register (MSTF)
- Instruction set includes:
  - IEEE single-precision (32-bit) floating point math operations
  - Floating-point math with parallel load or store
  - Floating-point multiply with parallel add or subtract
  - 1/X and 1/sqrt(X) estimations
  - Data type conversions.
  - Conditional branch and call
  - Data load/store operations
- The CLA program code can consist of up to eight tasks or interrupt service routines.
  - The start address of each task is specified by the MVECT registers.
  - No limit on task size as long as the tasks fit within the CLA program memory space.
  - One task is serviced at a time through to completion. There is no nesting of tasks.
  - Upon task completion a task-specific interrupt is flagged within the PIE.
  - When a task finishes the next highest-priority pending task is automatically started.
- Task trigger mechanisms:
  - C28x CPU via the IACK instruction
  - Task1 to Task7: the corresponding ADC, ePWM, eQEP, or eCAP module interrupt. For example:
    - Task1: ADCINT1 or EPWM1_INT
    - Task2: ADCINT2 or EPWM2_INT
    - Task 4: ADCINT4 or EPWM4_INT or EQEPx_INT or ECAPx_INT
    - Task 7: ADCINT7 or EPWM7_INT or EQEPx_INT or ECAPx_INT
  - Task8: ADCINT8 or CPU Timer 0 or EQEPx_INT or ECAPx_INT
- Memory and Shared Peripherals:
  - Two dedicated message RAMs for communication between the CLA and the main CPU.
  - The C28x CPU can map CLA program and data memory to the main CPU space or CLA space.
  - The CLA has direct access to the ePWM+HRPWM, Comparator, eCAP, eQEP, and ADC result registers.

Figure 9-1. CLA Block Diagram



9.2 CLA Interface

This chapter describes how the C28x main CPU can interface to the CLA and vice versa.

9.2.1 CLA Memory

The CLA can access three types of memory: program, data and message RAMs. The behavior and arbitration for each type of memory is described in detail in Section 9.7.

- **CLA Program Memory**

  At reset memory designated for CLA program is mapped to the main CPU memory and is treated like any other memory block. While mapped to CPU space, the main CPU can copy the CLA program code into the memory block. During debug the block can also be loaded directly by Code Composer Studio.

  Once the memory is initialized with CLA code, the main CPU maps it to the CLA program space by writing a 1 to the MMEMCFG[PROGE] bit. When mapped to the CLA program space, the block can only be accessed by the CLA for fetching opcodes. The main CPU can only perform debugger accesses when the CLA is either halted or idle. If the CLA is executing code, then all debugger accesses are blocked and the memory reads back all 0x0000.

  CLA program memory is protected by the code security module. All CLA program fetches are performed as 32-bit read operations and all opcodes must be aligned to an even address. Since all CLA opcodes are 32-bits, this alignment naturally occurs.

- **CLA Data Memory**

  There are three CLA data memory blocks on the device. At reset, all blocks are mapped to the main CPU memory space and treated by the CPU like any other memory block. While mapped to CPU space, the main CPU can initialize the memory with data tables and coefficients for the CLA to use.

  Once the memory is initialized with CLA data the main CPU maps it to the CLA space. Each block can be individually mapped via the MMEMCFG[RAM0E], MMEMCFG[RAM1E] and MMEMCFG[RAM2E] bits. When mapped to the CLA data space, the memory can be accessed by either the CLA and/or the CPU for read or write operations by setting the appropriate memory configuration bits, MMEMCFG[RAMxE] and MMEMCFG[RAMxCPUE].

  Each of the CLA data RAMs is protected by the code security module and emulation code security logic.

- **CLA Shared Message RAMs**

  There are two small memory blocks for data sharing and communication between the CLA and the main CPU. The message RAMs are always mapped to both CPU and CLA memory spaces and are protected by the code security module. The message RAMs allow data accesses only; no program fetches can be performed.

  – **CLA to CPU Message RAM**

    The CLA can use this block to pass data to the main CPU. This block is both readable and writable by the CLA. This block is also readable by the main CPU but writes by the main CPU are ignored.

  – **CPU to CLA Message RAM**

    The main CPU can use this block to pass data and messages to the CLA. This message RAM is both readable and writable by the main CPU. The CLA can perform reads but writes by the CLA are ignored.

9.2.2 CLA Memory Bus

The CLA has dedicated bus architecture similar to that of the C28x CPU where there is a program read, data read and data write bus. Thus there can be simultaneous instruction fetch, data read and data write in a single cycle. Like the C28x CPU, the CLA expects memory logic to align any 32-bit read or write to an even address. If the address-generation logic generates an odd address, the CLA will begin reading or writing at the previous even address. This alignment does not affect the address values generated by the address-generation logic.

- **CLA Program Bus**

  The CLA program bus has a access range of 2048 32-bit instructions. Since all CLA instructions are 32-bits, this bus always fetches 32-bits at a time and the opcodes must be even word aligned. The amount of program space available for the CLA is device dependent as described in the device-specific data manual.

- **CLA Data Read Bus**

  The CLA data read bus has a 64K x 16 address range. The bus can perform 16 or 32-bit reads and will automatically stall if there are memory access conflicts. The data read bus has access to both the message RAMs, CLA data memory and the ePWM, HRPWM, Comparator, eCAP, eQEP, and ADC result registers.

- **CLA Data Write Bus**

  The CLA data write bus has a 64K x 16 address range. This bus can perform 16 or 32-bit writes. The bus will automatically stall if there are memory access conflicts. The data write bus has access to the CLA to CPU message RAM, CLA data memory and the ePWM, HRPWM, eCAP, eQEP, and Comparator registers.

9.2.3 Shared Peripherals and EALLOW Protection

The ePWM, HRPWM, Comparator, eCAP, eQEP, and ADC result registers can be accessed by both the CLA and the main CPU. Section 9.5 describes in detail the CLA and CPU arbitration when both access these registers.

Several peripheral control registers are protected from spurious 28x CPU writes by the EALLOW protection mechanism. These same registers are also protected from spurious CLA writes. The EALLOW bit in the main CPU status register 1 (ST1) indicates the state of protection for the main CPU. Likewise the MEALLOW bit in the CLA status register (MSTF) indicates the state of write protection for the CLA. The MEALLOW CLA instruction enables write access by the CLA to EALLOW protected registers. Likewise the MEDIS CLA instruction will disable write access. This way the CLA can enable/disable write access independent of the main CPU.

The 2806x ADC offers the option to generate an early interrupt pulse when the ADC begins conversion. If this option is used to start a ADC triggered CLA task then the 8th instruction can read the result as soon as the conversion completes. The CLA pipeline activity for this scenario is shown in Section 9.5.

9.2.4 CLA Tasks and Interrupt Vectors

The CLA program code is divided up into tasks or interrupt service routines. Tasks do not have a fixed starting location or length. The CLA program memory can be divided up as desired. The CLA knows where a task begins by the content of the associated interrupt vector (MVECT1 to MVECT8) and the end is indicated by the MSTOP instruction.

The CLA supports 8 tasks. Task 1 has the highest priority and task 8 has the lowest priority. A task can be requested by a peripheral interrupt or by software:

- **Peripheral interrupt trigger**

  Each task has specific interrupt sources that can trigger it. Configure the MPISRCSEL1 register to select from the potential sources. For example, task 1 (MVECT1) can be triggered by ADCINT1 or EPWM1_INT as specified in MPISRCSEL1[PERINT1SEL]. You can not, however, trigger task 1 directly using EPWM2_INT. If you need to trigger a task using EPWM2_INT then the best solution is to use task 2 (MVECT2). Another possible solution is to take EPWM2_INT with the main CPU and trigger a task with software.

  To disable the peripheral from sending an interrupt request to the CLA set the PERINT1SEL option to no interrupt.

- **Software trigger**

  Tasks can also be started by the main CPU software writing to the MIFRC register or by the IACK instruction. Using the IACK instruction is more efficient because it does not require you to issue an EALLOW to set MIFR bits. Set the MCTL[IACKE] bit to enable the IACK feature. Each bit in the operand of the IACK instruction corresponds to a task. For example IACK #0x0001 will set bit 0 in the MIFR register to start task 1. Likewise IACK #0x0003 will set bits 0 and 1 in the MIFR register to start task 1 and task 2.

The CLA has its own fetch mechanism and can run and execute a task independent of the main CPU. Only one task is serviced at a time; there is no nesting of tasks. The task currently running is indicated in the MIRUN register. Interrupts that have been received but not yet serviced are indicated in the flag register (MIFR). If an interrupt request from a peripheral is received and that same task is already flagged, then the overflow flag bit is set. Overflow flags will remain set until they are cleared by the main CPU.

If the CLA is idle (no task is currently running) then the highest priority interrupt request that is both flagged (MIFR) and enabled (MIER) will start. The flow is as follows:

1. The associated RUN register bit is set (MIRUN) and the flag bit (MIFR) is cleared.
2. The CLA begins execution at the location indicated by the associated interrupt vector (MVECTx). MVECT is an offset from the first program memory location.
3. The CLA executes instructions until the MSTOP instruction is found. This indicates the end of the task.
4. The MIRUN bit is cleared.
5. The task-specific interrupt to the PIE is issued. This informs the main CPU that the task has completed.
6. The CLA returns to idle.

Once a task completes the next highest-priority pending task is automatically serviced and this sequence repeats.

9.3 CLA Configuration and Debug

This section discusses the steps necessary to configure and debug the CLA.

9.3.1 *Building a CLA Application*

The control law accelerator is programmed in CLA assembly code using the instructions described in Section 9.6. CLA assembly code can, and should, reside in the same project with C28x code. The only restriction is the CLA code must be in its own assembly section. This can be easily done using the .sect assembly directive. This does not prevent CLA and C28x code from being linked into the same memory region in the linker command file.

System and CLA initialization are performed by the main CPU. This would typically be done in C or C++ but can also include C28x assembly code. The main CPU will also copy the CLA code to the program memory and, if needed, initialize the CLA data RAM(s). Once system initialization is complete and the application begins, the CLA will service its interrupts using the CLA assembly code (or tasks). Concurrently the main CPU can perform other tasks.

The C2000 codegen tools V5.2.x and higher support CLA instructions when the following switch is set: --cla_support = cla0.

9.3.2 *Typical CLA Initialization Sequence*

A typical CLA initialization sequence is performed by the main CPU as described in this section.

1. **Copy CLA code into the CLA program RAM**

   The source for the CLA code can initially reside in the flash or a data stream from a communications peripheral or anywhere the main CPU can access it. The debugger can also be used to load code directly to the CLA program RAM during development.

2. **Initialize CLA data RAM if necessary**

   Populate the CLA data RAM with any required data coefficients or constants.

3. **Configure the CLA registers**

   Configure the CLA registers, but keep interrupts disabled until later (leave MIER == 0):

   - **Enable the CLA clock in the PCLKCR3 register.**

     PCLKCR3 register is defined in the device-specific system control and interrupts reference guide.

   - **Populate the CLA task interrupt vectors: MVECT1 to MVECT8.**

     Each vector needs to be initialized with the start address of the task to be executed when the CLA receives the associated interrupt. This address is an offset from the first address in CLA program memory. For example, 0x0000 corresponds to the first CLA program memory address.

   - **Select the task interrupt sources**

     For each task select the interrupt source in the PERINT1SEL register. If a task is going to be generated by software, select no interrupt.

   - **Enable IACK to start a task from software if desired**

     To enable the IACK instruction to start a task set the MCTL[IACKE] bit. Using the IACK instruction avoids having to set and clear the EALLOW bit.

   - **Map CLA data RAM(s) to CLA space if necessary**

     Map any of the data RAMs to the CLA space by writing a 1 to the respective MMEMCFG[RAMxE] bit. After the memory is mapped to CLA space, the main CPU has restricted access to it. Access control to the memory is determined by the combination of the memory configuration bits, MMEMCFG[RAMxE] and MMEMCFG[RAMxCPUE]. Allow two SYSCLKOUT cycles between changing the map configuration of this memory and accessing it.

   - **Map CLA program RAM to CLA space**

     Map the CLA program RAM to CLA space by setting the MMEMCFG[PROGE] bit. After the memory is remapped to CLA space the main CPU will only be able to make debug accesses to the memory block. Access control to the memory is determined by the combination of the memory configuration bits, MMEMCFG[RAMxE] and MMEMCFG[RAMxCPUE]. Allow two SYSCLKOUT cycles between changing the map configuration of these memories and accessing them.

4. **Initialize the PIE vector table and registers**

   When a CLA task completes the associated interrupt in the PIE will be flagged. The CLA overflow and underflow flags also have associated interrupts within the PIE.

5. **Enable CLA tasks/interrupts**

   Set appropriate bits in the interrupt enable register (MIER) to allow the CLA to service interrupts.

6. **Initialize other peripherals**

   Initialize any peripherals (ePWM, ADC etc.) that will generate an interrupt to the CLA and be serviced by a CLA task.

   The CLA is now ready to service interrupts and the message RAMs can be used to pass data between the CPU and the CLA. Typically mapping of the CLA program and data RAMs occurs only during the initialization process. If after some time the you want to re-map these memories back to CPU space then disable interrupts and make sure all tasks have completed by checking the MIRUN register. Always allow two SYSCLKOUT cycles when changing the map configuration of these memories and accessing them.

9.3.3 Debugging CLA Code

Debugging the CLA code is a simple process that occurs independently of the main CPU.

1. **Insert a breakpoint in CLA code**

   Insert a CLA breakpoint (MDEBUGSTOP instruction) into the code where you want the CLA to halt, then rebuild and reload the code. Because the CLA does not flush its pipeline when you single-step, the MDEBUGSTOP instruction must be inserted as part of the code. The debugger cannot insert it as needed.

   If CLA breakpoints are not enabled, then the MDEBUGSTOP will be ignored and is treated as a MNOP. The MDEBUGSTOP instruction can be placed anywhere in the CLA code as long as it is not within three instructions of a MBCNDD, MCCNDD, or MRCNDD instruction.

2. **Enable CLA breakpoints**

   First, enable the CLA breakpoints in the debugger. In Code Composer Studio V3.3, this is done by connecting the CLA debug window (debug->connect). Breakpoints are disabled when this window is disconnected.

3. **Start the task**

   There are three ways to start the task:

   - The peripheral can assert an interrupt
   - The main CPU can execute an IACK instruction, or
   - You can manually write to the MIFRC register in the debugger window

   When the task starts, the CLA will execute instructions until the MDEBUGSTOP is in the D2 phase of the pipeline. At this point, the CLA will halt and the pipeline will be frozen. The MPC register will reflect the address of the MDEBUGSTOP instruction.

4. **Single-step the CLA code**

   Once halted, you can single-step the CLA code one cycle at a time. The behavior of a CLA single-step is different than the main C28x. When issuing a CLA single-step, the pipeline is clocked only one cycle and then again frozen. On the 28x CPU, the pipeline is flushed for each single-step.

   You can also run to the next MDEBUGSTOP or to the end of the task. If another task is pending, it will automatically start when you run to the end of the task.

NOTE: When CLA program memory is mapped to the CLA memory space, a CLA fetch has higher priority than CPU debug reads. For this reason, it is possible for the CLA to permanently block CPU debug accesses if the CLA is executing in a loop. This might occur when initially developing CLA code due to a bug that causes an infinite loop. To avoid locking up the main CPU, the program memory will return all 0x0000 for CPU debug reads when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access to CLA program memory can be performed.

If the CLA gets caught in a infinite loop, you can use a soft or hard reset to exit the condition. A debugger reset will also exit the condition.

There are special cases that can occur when single-stepping a task such that the program counter, MPC, reaches the MSTOP instruction at the end of the task.

- **MPC halts at or after the MSTOP with a task already pending**

  If you are single-stepping or halted in "task A" and "task B" comes in before the MPC reaches the MSTOP, then "task B" will start if you continue to step through the MSTOP instruction. Basically if "task B" is pending before the MPC reaches MSTOP in "task A" then there is no issue in "task B" starting and no special action is required.

- **MPC halts at or after the MSTOP with no task pending**

  In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. If "task B" comes in at this point, it will be flagged in the MIFR register but it may or may not start if you continue to single-step through the MSTOP instruction of "task A."

  It depends on exactly when the new task comes in. To reliably start "task B" perform a soft reset and reconfigure the MIER bits. Once this is done, you can start single-stepping "task B."

This case can be handled slightly differently if there is control over when "task B" comes in (for example using the IACK instruction to start the task). In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. Before forcing "task B," run free to force the CLA out of the debug state. Once this is done you can force "task B" and continue debugging.

5. **If desired, disable CLA breakpoints**

   In CCS V3.3 you can disable the CLA breakpoints by disconnecting the CLA debug window. Make sure to first issue a run or reset; otherwise, the CLA will be halted and no other tasks will start.

9.3.4 CLA Illegal Opcode Behavior

If the CLA fetches an opcode that does not correspond to a legal instruction, it will behave as follows:

- The CLA will halt with the illegal opcode in the D2 phase of the pipeline as if it were a breakpoint. This will occur whether CLA breakpoints are enabled or not.
- The CLA will issue the task-specific interrupt to the PIE.
- The MIRUN bit for the task will remain set.

Further single-stepping ignored once execution halts due to an illegal op-code. To exit this situation, issue either a soft or hard reset of the CLA as described in Section 9.3.5.

9.3.5 Resetting the CLA

There may be times when you need to reset the CLA. For example, during code debug the CLA may enter an infinite loop due to a code bug. The CLA has two types of resets: hard and soft. Both of these resets can be performed by the debugger or by the main CPU.

- **Hard Reset**

  Writing a 1 to the MCTL[HARDRESET] bit will perform a hard reset of the CLA. The behavior of a hard reset is the same as a system reset (via $\overline{\text{XRS}}$ or the debugger). In this case all CLA configuration and execution registers will be set to their default state and CLA execution will halt.

- **Soft Reset**

  Writing a 1 to the MCTL[SOFTRESET] bit performs a soft reset of the CLA. If a task is executing it will halt and the associated MIRUN bit will be cleared. All bits within the interrupt enable (MIER) register will also be cleared so that no new tasks start.

9.4 Register Set

The CLA register set is independent from that of the main CPU. This chapter describes the CLA register set.

9.4.1 Register Memory Mapping

The table below describes the CLA module control and status register set.

Table 9-1. CLA Module Control and Status Register Set

Name	Offset	Size (x16)	EALLOW	CSM Protected	Description
					Task Interrupt Vectors
MVECT1	0x0000	1	Yes	Yes	Task 1 Interrupt Vector
MVECT2	0x0001	1	Yes	Yes	Task 2 Interrupt Vector
MVECT3	0x0002	1	Yes	Yes	Task 3 Interrupt Vector
MVECT4	0x0003	1	Yes	Yes	Task 4 Interrupt Vector
MVECT5	0x0004	1	Yes	Yes	Task 5 Interrupt Vector
MVECT6	0x0005	1	Yes	Yes	Task 6 Interrupt Vector
MVECT7	0x0006	1	Yes	Yes	Task 7 Interrupt Vector
MVECT8	0x0007	1	Yes	Yes	Task 8 Interrupt Vector
					Configuration Registers
MCTL	0x0010	1	Yes	Yes	Control Register
MMEMCFG	0x0011	1	Yes	Yes	Memory Configuration Register
MPISRCSEL1	0x0014	2	Yes	Yes	Peripheral Interrupt Source Select 1 Register
MIFR	0x0020	1	Yes	Yes	Interrupt Flag Register
MIOVF	0x0021	1	Yes	Yes	Interrupt Overflow Flag Register
MIFRC	0x0022	1	Yes	Yes	Interrupt Force Register
MICLR	0x0023	1	Yes	Yes	Interrupt Flag Clear Register
MICLROVF	0x0024	1	Yes	Yes	Interrupt Overflow Flag Clear Register
MIER	0x0025	1	Yes	Yes	Interrupt Enable Register
MIRUN	0x0026	1	Yes	Yes	Interrupt Run Status Register
					Execution Registers [(1)]
MPC	0x0028	1	-	Yes	CLA Program Counter
MAR0	0x0029	1	-	Yes	CLA Auxiliary Register 0
MAR1	0x002A	1	-	Yes	CLA Auxiliary Register 1
MSTF	0x002E	2	-	Yes	CLA Floating-Point Status Register
MR0	0x0030	2	-	Yes	CLA Floating-Point Result Register 0
MR1	0x0034	2	-	Yes	CLA Floating-Point Result Register 1
MR2	0x0038	2	-	Yes	CLA Floating-Point Result Register 2
MR3	0x003C	2	-	Yes	CLA Floating-Point Result Register 3

(1) The main C28x CPU only has read access to the CLA execution registers for debug purposes. The main CPU cannot perform CPU or debugger writes to these registers.

9.4.2 Task Interrupt Vector Registers

Each CLA interrupt has its own interrupt vector (MVECT1 to MVECT8). This interrupt vector points to the first instruction of the associated task. When a task begins, the CLA will start fetching instructions at the location indicated by the appropriate MVECT register .

9.4.2.1 Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Register

The task interrupt vector registers (MVECT1/2/3/4/5/6/7/8) are is shown in Section 9.4.2.1 and described in Figure 9-2.

Figure 9-2. Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Register

15 12	11 0
Reserved	MVECT
R-0	R-0

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

Table 9-2. Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Field Descriptions

Bits	Name	Value	Description [(1)]
15-12	Reserved		Any writes to these bit(s) must always have a value of 0.
11-0	MVECT	0000 - 0FFF	Offset of the first instruction in the associated task from the start of CLA program space. The CLA will begin instruction fetches from this location when the specific task begins. For example: If CLA program memory begins at CPU address 0x009000 and the code for task 5 begins at CPU address 0x009120, then MVECT5 should be initialized with 0x0120. There is one MVECT register per task. Interrupt 1 uses MVECT1, interrupt 2 uses MVECT2 and so forth.

(1) These registers are protected by EALLOW and the code security module.

9.4.3 Configuration Registers

The configuration registers are described here.

9.4.3.1 Control Register (MCTL)

The configuration control register (MCTL) is shown in Figure 9-3 and described in Table 9-3.

Figure 9-3. Control Register (MCTL)

15–8
Reserved
R -0

7–3	2	1	0
Reserved	IACKE	SOFTRESET	HARDRESET
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 9-3. Control Register (MCTL) Field Descriptions

Bits	Name	Value	Description [(1)]
15-3	Reserved		Any writes to these bit(s) must always have a value of 0.
2	IACKE		IACK enable
		0	The CLA ignores the IACK instruction. (default)
		1	Enable the main CPU to use the IACK #16bit instruction to set MIFR bits in the same manner as writing to the MIFRC register. Each bit in the operand, #16bit, corresponds to a bit in the MIFRC register. Using IACK has the advantage of not having to first set the EALLOW bit. This allows the main CPU to efficiently trigger a CLA task through software.
			Examples `IACK #0x0001` Write a 1 to MIFRC bit 0 to force task 1
			`IACK #0x0003` Write a 1 to MIFRC bit 0 and 1 to force task 1 and task 2
1	SOFTRESET		Soft Reset
		0	This bit always reads back 0 and writes of 0 are ignored.
		1	Writing a 1 will cause a soft reset of the CLA. This will stop the current task, clear the MIRUN flag and clear all bits in the MIER register. After a soft reset you must wait at least 1 SYSCLKOUT cycle before reconfiguring the MIER bits. If these two operations are done back-to-back then the MIER bits will not get set.
0	HARDRESET		Hard Reset
		0	This bit always reads back 0 and writes of 0 are ignored.
		1	Writing a 1 will cause a hard reset of the CLA. This will set all CLA registers to their default state.

(1) This register is protected by EALLOW and the code security module.

9.4.3.2 Memory Configuration Register (MMEMCFG)

The MMEMCFG register is used to map the CLA program and data RAMs to either the CPU or the CLA memory space. Typically mapping of the CLA program and data RAMs occurs only during the initialization process. If after some time the you want to re-map these memories back to CPU space, disable interrupts using the (MIER register and make sure all tasks have completed by checking the MIRUN register. Allow two SYSCLKOUT cycles between changing the map configuration of these memories and accessing them. Refer to Section 9.7 for CLA and CPU access arbitration details.

Figure 9-4. Memory Configuration Register (MMEMCFG)

15	11	10	9	8
Reserved		RAM2CPUE	RAM1CPUE	RAM0CPUE
R -0		R/W-0	R/W-0	R/W-0

7	6	5	4	3	1	0
Reserved	RAM2E	RAM1E	RAM0E	Reserved		PROGE
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 9-4. Memory Configuration Register (MMEMCFG) Field Descriptions

Bits	Field	Value	Description
15-11	Reserved		Any writes to these bit(s) must always have a value of 0
10	RAM2CPUE		CLA Data RAM 2 CPU Access Enable Bit: Allows/disallows CPU data accesses to CLA-Data-RAM 2, when the RAM2E bit is set. This bit is used to enable access of the CLA-Data-RAM to the CPU, when the CLA-Data-RAM is mapped to the CLA (RAM2E =1). When the data RAM is not mapped to the CLA (RAM2E =0), this bit has no effect. When RAM2E =1 and RAM2CPUE=0, CPU does not have access to the CLA-Data-RAM. When RAM2E=1 and RAM2CPUE=1, both CPU and CLA have access to the CLA data RAM, the priority of the accesses being determined by the arbitration scheme. Note that CPU data accesses are always allowed when RAM0E bit is cleared. Allow two SYSCLKOUT cycles between changing this bit and accessing the memory.
		0	When RAM2E=0 (RAM not mapped to CLA), CLA accesses are not allowed. Only CPU accesses are allowed. When RAM2E=1 (RAM mapped to CLA), CLA accesses are allowed but CPU accesses are not allowed.
		1	When RAM2E=0 (RAM not mapped to CLA), CLA accesses are not allowed. Only CPU accesses are allowed. When RAM2E=1 (RAM mapped to CLA), both CLA and CPU accesses are allowed.
9	RAM1CPUE		CLA Data RAM 1 CPU Access Enable Bit: Allows/disallows CPU data accesses to CLA-Data-RAM 1, when the RAM1E bit is set. This bit is used to enable access of the CLA-Data-RAM to the CPU, when the CLA-Data-RAM is mapped to the CLA (RAM1E=1). When the data RAM is not mapped to the CLA (RAM1E=0), this bit has no effect. When RAM1E=1 and RAM1CPUE=0, CPU does not have access to the CLA-Data-RAM. When RAM1E=1 and RAM1CPUE=1, both CPU and CLA have access to the CLA data RAM, the priority of the accesses being determined by the arbitration scheme. Note that CPU data accesses are always allowed when RAM1E bit is cleared. Allow two SYSCLKOUT cycles between changing this bit and accessing the memory.
		0	When RAM1E=0 (RAM not mapped to CLA), CLA accesses are not allowed. Only CPU accesses are allowed. When RAM1E=1 (RAM mapped to CLA), CLA accesses are allowed but CPU accesses are not allowed.
		1	When RAM1E=0 (RAM not mapped to CLA), CLA accesses are not allowed. Only CPU accesses are allowed. When RAM1E=1 (RAM mapped to CLA), both CLA and CPU accesses are allowed.

Table 9-4. Memory Configuration Register (MMEMCFG) Field Descriptions (continued)

Bits	Field	Value	Description
8	RAM0CPUE		CLA Data RAM 0 CPU Access Enable Bit:
			Allows/disallows CPU data accesses to CLA-Data-RAM 0, when the RAM0E bit is set. This bit is used to enable access of the CLA-Data-RAM to the CPU, when the CLA-Data-RAM is mapped to the CLA (RAM0E=1).
			When the data RAM is not mapped to the CLA (RAM0E=0), this bit has no effect. When RAM0E=1 and RAM0CPUE=0, CPU does not have access to the CLA-Data-RAM. When RAM0E=1 and RAM0CPUE=1, both CPU and CLA have access to the CLA data RAM, the priority of the accesses being determined by the arbitration scheme.
			Note that CPU data accesses are always allowed when RAM0E bit is cleared. Allow 2 SYSCLKOUT cycles between changing this bit and accessing the memory.
		0	When RAM0E=0 (RAM not mapped to CLA), CLA accesses are not allowed. Only CPU accesses are allowed.
			When RAM0E=1 (RAM mapped to CLA), CLA accesses are allowed but CPU accesses are not allowed.
		1	When RAM0E=0 (RAM not mapped to CLA), CLA accesses are not allowed. Only CPU accesses are allowed.
			When RAM0E=1 (RAM mapped to CLA), both CLA and CPU accesses are allowed.
7	Reserved		Any writes to these bit(s) must always have a value of 0
6	RAM2E		CLA Data RAM 2 Enable
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory
		0	CLA data SARAM block 2 is mapped to the main CPU program and data space. CLA reads will return zero. (default)
		1	CLA data SARAM block 2 is mapped to the CLA space. The RAM2CPUE bit determines the CPU access to this memory
5	RAM1E		CLA Data RAM 1 Enable
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory
		0	CLA data SARAM block 1 is mapped to the main CPU program and data space. CLA reads will return zero. (default)
		1	CLA data SARAM block 1 is mapped to the CLA space. The RAM1CPUE bit determines the CPU access to this memory
4	RAM0E		CLA Data RAM 0 Enable
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory
		0	CLA data SARAM block 0 is mapped to the main CPU program and data space. CLA reads will return zero. (default)
		1	CLA data SARAM block 0 is mapped to the CLA space. The RAM0CPUE bit determines the CPU access to this memory
3 - 1	Reserved		Any writes to these bit(s) must always have a value of 0
0	PROGE		CLA Program Space Enable
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory
		0	CLA program SARAM is mapped to the main CPU program and data space. If the CLA attempts a program fetch the result will be the same as an illegal opcode fetch as described in Section 3.4.(default)
		1	CLA program SARAM is mapped to the CLA program space. The main SPU can only make debug accesses to this block
			In this state the CLA has higher priority than CPU debug reads. It is, therefore, possible for the CLA to permanently block debug accesses if the CLA is executing in a loop. This might occur when if the CLA code has a bug. To avoid this issue, the program memory will reutrn 0x0000 for CPU debug reads (ignore writes) when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access can be performed

9.4.3.3 CLA Peripheral Interrupt Source Select 1 Register (MPISRCSEL1)

Each task has specific peripherals that can start it. For example, Task2 can be started by ADCINT2 or EPWM2_INT. To configure which of the possible peripherals will start a task configure the MPISRCSEL1 register shown in Figure 9-5. Choosing the option "no interrupt source" means that only the main CPU software will be able to start the given task.

Figure 9-5. CLA Peripheral Interrupt Source Select 1 Register (MPISRCSEL1)

31 - 28	27 - 24	23 - 20	19 - 16
PERINT8SEL	PERINT7SEL	PERINT6SEL	PERINT5SEL
R/W-0	R/W-0	R/W-0	R/W-0

15 - 12	11 - 8	7 - 4	3 - 0
PERINT4SEL	PERINT3SEL	PERINT2SEL	PERINT1SEL
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 9-5. Peripheral Interrupt Source Select 1 (MPISRCSEL1) Register Field Descriptions

Bits	Field	Value [(1)]	Description [(2)]
31 - 28	PERINT8SEL		Task 8 Peripheral Interrupt Input Select
		0000	ADCINT8 is the input for interrupt task 8. (default)
		0010	CPU Timer 0 is the input for interrupt task 8. (TINT0)
		0100	eQEP1 is the input for interrupt task 8. (EQEP1_INT)
		0101	eQEP2 is the input for interrupt task 8. (EQEP2_INT)
		1000	eCAP1 is the input for interrupt task 8. (ECAP1_INT)
		1001	eCAP2 is the input for interrupt task 8. (ECAP2_INT)
		1010	eCAP3 is the input for interrupt task 8. (ECAP3_INT)
		Other	No interrupt source for task 8.
27 - 24	PERINT7SEL		Task 7 Peripheral Interrupt Input Select
		0000	ADCINT7 is the input for interrupt task 7. (default)
		0010	ePWM7 is the input for interrupt task 7. (EPWM7_INT)
		0100	eQEP1 is the input for interrupt task 7. (EQEP1_INT)
		0101	eQEP2 is the input for interrupt task 7. (EQEP2_INT)
		1000	eCAP1 is the input for interrupt task 7. (ECAP1_INT)
		1001	eCAP2 is the input for interrupt task 7. (ECAP2_INT)
		1010	eCAP3 is the input for interrupt task 7. (ECAP3_INT)
		Other	No interrupt source for task 7.
23 - 20	PERINT6SEL		Task 6 Peripheral Interrupt Input Select
		0000	ADCINT6 is the input for interrupt task 6. (default)
		0010	ePWM6 is the input for interrupt task 6. (EPWM6_INT)
		0100	eQEP1 is the input for interrupt task 6. (EQEP1_INT)
		0101	eQEP2 is the input for interrupt task 6. (EQEP2_INT)
		1000	eCAP1 is the input for interrupt task 6. (ECAP1_INT)
		1001	eCAP2 is the input for interrupt task 6. (ECAP2_INT)
		1010	eCAP3 is the input for interrupt task 6. (ECAP3_INT)
		Other	No interrupt source for task 6.

(1) All values not shown are reserved.
(2) This register is protected by EALLOW and the code security module.

Table 9-5. Peripheral Interrupt Source Select 1 (MPISRCSEL1) Register Field Descriptions (continued)

Bits	Field	Value [1]	Description [2]
19 - 16	PERINT5SEL		Task 5 Peripheral Interrupt Input Select
		0000	ADCINT5 is the input for interrupt task 5. (default)
		0010	ePWM5 is the input for interrupt task 5. (EPWM5_INT)
		0100	eQEP1 is the input for interrupt task 5. (EQEP1_INT)
		0101	eQEP2 is the input for interrupt task 5. (EQEP2_INT)
		1000	eCAP1 is the input for interrupt task 5. (ECAP1_INT)
		1001	eCAP2 is the input for interrupt task 5. (ECAP2_INT)
		1010	eCAP3 is the input for interrupt task 5. (ECAP3_INT)
		Other	No interrupt source for task 5.
15 - 12	PERINT4SEL		Task 4 Peripheral Interrupt Input Select
		0000	ADCINT4 is the input for interrupt task 4. (default)
		0010	ePWM4 is the input for interrupt task 4. (EPWM4_INT)
		0100	eQEP1 is the input for interrupt task 4. (EQEP1_INT)
		0101	eQEP2 is the input for interrupt task 4. (EQEP2_INT)
		1000	eCAP1 is the input for interrupt task 4. (ECAP1_INT)
		1001	eCAP2 is the input for interrupt task 4. (ECAP2_INT)
		1010	eCAP3 is the input for interrupt task 4. (ECAP3_INT)
		Other	No interrupt source for task 4.
11 - 8	PERINT3SEL		Task 3 Peripheral Interrupt Input Select
		0000	ADCINT3 is the input for interrupt task 3. (default)
		0010	ePWM3 is the input for interrupt task 3. (EPWM3_INT)
		xxx1	No interrupt source for task 3.
7 - 4	PERINT2SEL		Task 2 Peripheral Interrupt Input Select
		0000	ADCINT2 is the input for interrupt task 2. (default)
		0010	ePWM2 is the input for interrupt task 2. (EPWM2_INT)
		xxx1	No interrupt source for task 2.
3 - 0	PERINT1SEL		Task 1Peripheral Interrupt Input Select
		0000	ADCINT1 is the input for interrupt task 1. (default)
		0010	ePWM1 is the input for interrupt task 1. (EPWM1_INT)
		xxx1	No interrupt source

9.4.3.4 Interrupt Enable Register (MIER)

Setting the bits in the interrupt enable register (MIER) allow an incoming interrupt or main CPU software to start the corresponding CLA task. Writing a 0 will block the task, but the interrupt request will still be latched in the flag register (MIFLG). Setting the MIER register bit to 0 while the corresponding task is executing will have no effect on the task. The task will continue to run until it hits the MSTOP instruction.

When a soft reset is issued, the MIER bits are cleared. There should always be at least a 1 SYSCLKOUT delay between issuing the soft reset and reconfiguring the MIER bits.

Figure 9-6. Interrupt Enable Register (MIER)

15-8
Reserved
R -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; R = Read only; -n = value after reset

Table 9-6. Interrupt Enable Register (MIER) Field Descriptions

Bits	Name	Value	Description [(1)]
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Enable
		0	Task 8 interrupt is disabled. (default)
		1	Task 8 interrupt is enabled.
6	INT7		Task 7 Interrupt Enable
		0	Task 7 interrupt is disabled. (default)
		1	Task 7 interrupt is enabled.
5	INT6		Task 6 Interrupt Enable
		0	Task 6 interrupt is disabled. (default)
		1	Task 6 interrupt is enabled.
4	INT5		Task 5 Interrupt Enable
		0	Task 5 interrupt is disabled. (default)
		1	Task 5 interrupt is enabled.
3	INT4		Task 4 Interrupt Enable
		0	Task 4 interrupt is disabled. (default)
		1	Task 4 interrupt is enabled.
2	INT3		Task 3 Interrupt Enable
		0	Task 3 interrupt is disabled. (default)
		1	Task 3 interrupt is enabled.
1	INT2		Task 2 Interrupt Enable
		0	Task 2 interrupt is disabled. (default)
		1	Task 2 interrupt is enabled.
0	INT1		Task 1 Interrupt Enable
		0	Task 1 interrupt is disabled. (default)
		1	Task 1 interrupt is enabled.

(1) This register is protected by EALLOW and the code security module.

9.4.3.5 Interrupt Flag Register (MIFR)

Each bit in the interrupt flag register corresponds to a CLA task. The corresponding bit is automatically set when the task request is received from the peripheral interrupt. The bit can also be set by the main CPU writing to the MIFRC register or using the IACK instruction to start the task. To use the IACK instruction to begin a task first enable this feature in the MCTL register. If the bit is already set when a new peripheral interrupt is received, then the corresponding overflow bit will be set in the MIOVF register.

The corresponding MIFR bit is automatically cleared when the task begins execution. This will occur if the interrupt is enabled in the MIER register and no other higher priority task is pending. The bits can also be cleared manually by writing to the MICLR register. Writes to the MIFR register are ignored.

Figure 9-7. Interrupt Flag Register (MIFR)

15	8
Reserved	
R -0	

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
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 9-7. Interrupt Flag Register (MIFR) Field Descriptions

Bits	Name	Value	Description [(1)]
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Flag
		0	A task 8 interrupt is currently not flagged. (default)
		1	A task 8 interrupt has been received and is pending execution.
6	INT7		Task 7 Interrupt Flag
		0	A task 7 interrupt is currently not flagged. (default)
		1	A task 7 interrupt has been received and is pending execution.
5	INT6		Task 6 Interrupt Flag
		0	A task 6 interrupt is currently not flagged. (default)
		1	A task 6 interrupt has been received and is pending execution.
4	INT5		Task 5 Interrupt Flag
		0	A task 5 interrupt is currently not flagged. (default)
		1	A task 5 interrupt has been received and is pending execution.
3	INT4		Task 4 Interrupt Flag
		0	A task 4 interrupt is currently not flagged. (default)
		1	A task 4 interrupt has been received and is pending execution.
2	INT3		Task 3 Interrupt Flag
		0	A task 3 interrupt is currently not flagged. (default)
		1	A task 3 interrupt has been received and is pending execution.
1	INT2		Task 2 Interrupt Flag
		0	A task 2 interrupt is currently not flagged. (default)
		1	A task 2 interrupt has been received and is pending execution.
0	INT1		Task 1 Interrupt Flag
		0	A task 1 interrupt is currently not flagged. (default)
		1	A task 1 interrupt has been received and is pending execution.

(1) This register is protected by the code security module.

9.4.3.6 Interrupt Overflow Flag Register (MIOVF)

Each bit in the overflow flag register corresponds to a CLA task. The bit is set when an interrupt overflow event has occurred for the specific task. An overflow event occurs when the MIFR register bit is already set when a new interrupt is received from a peripheral source. The MIOVF bits are only affected by peripheral interrupt events. They do not respond to a task request by the main CPU IACK instruction or by directly setting MIFR bits. The overflow flag will remain latched and can only be cleared by writing to the overflow flag clear (MICLROVF) register. Writes to the MIOVF register are ignored.

Figure 9-8. Interrupt Overflow Flag Register (MIOVF)

15-8
Reserved
R -0

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
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 9-8. Interrupt Overflow Flag Register (MIOVF) Field Descriptions

Bits	Name	Value	Description [(1)]
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.

(1) This register is protected by the code security module.

Table 9-8. Interrupt Overflow Flag Register (MIOVF) Field Descriptions (continued)

Bits	Name	Value	Description [1]
7	INT8		Task 8 Interrupt Overflow Flag
		0	A task 8 interrupt overflow has not occurred. (default)
		1	A task 8 interrupt overflow has occurred.
6	INT7		Task 7 Interrupt Overflow Flag
		0	A task 7 interrupt overflow has not occurred. (default)
		1	A task 7 interrupt overflow has occurred.
5	INT6		Task 6 Interrupt Overflow Flag
		0	A task 6 interrupt overflow has not occurred. (default)
		1	A task 6 interrupt overflow has occurred.
4	INT5		Task 5 Interrupt Overflow Flag
		0	A task 5 interrupt overflow has not occurred. (default)
		1	A task 5 interrupt overflow has occurred.
3	INT4		Task 4 Interrupt Overflow Flag
		0	A task 4 interrupt overflow has not occurred. (default)
		1	A task 4 interrupt overflow has occurred.
2	INT3		Task 3 Interrupt Overflow Flag
		0	A task 3 interrupt overflow has not occurred. (default)
		1	A task 3 interrupt overflow has occurred.
1	INT2		Task 2 Interrupt Overflow Flag
		0	A task 2 interrupt overflow has not occurred. (default)
		1	A task 2 interrupt overflow has occurred.
0	INT1		Task 1 Interrupt Overflow Flag
		0	A task 1 interrupt overflow has not occurred. (default)
		1	A task 1 interrupt overflow has occurred.

9.4.3.7 Interrupt Run Status Register (MIRUN)

The interrupt run status register (MIRUN) indicates which task is currently executing. Only one MIRUN bit will ever be set to a 1 at any given time. The bit is automatically cleared when the task competes and the respective interrupt is fed to the peripheral interrupt expansion (PIE) block of the device. This lets the main CPU know when a task has completed. The main CPU can stop a currently running task by writing to the MCTL[SOFTRESET] bit. This will clear the MIRUN flag and stop the task. In this case no interrupt will be generated to the PIE.

Figure 9-9. Interrupt Run Status Register (MIRUN)

15–8
Reserved
R -0

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
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 9-9. Interrupt Run Status Register (MIRUN) Field Descriptions

Bits	Name	Value	Description [1]
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.

[1] This register is protected by the code security module.

Table 9-9. Interrupt Run Status Register (MIRUN) Field Descriptions (continued)

Bits	Name	Value	Description [(1)]
7	INT8		Task 8 Run Status
		0	Task 8 is not executing. (default)
		1	Task 8 is executing.
6	INT7		Task 7 Run Status
		0	Task 7 is not executing. (default)
		1	Task 7 is executing.
5	INT6		Task 6 Run Status
		0	Task 6 is not executing. (default)
		1	Task 6 is executing.
4	INT5		Task 5 Run Status
		0	Task 5 is not executing. (default)
		1	Task 5 is executing.
3	INT4		Task 4 Run Status
		0	Task 4 is not executing. (default)
		1	Task 4 is executing.
2	INT3		Task 3 Run Status
		0	Task 3 is not executing. (default)
		1	Task 3 is executing.
1	INT2		Task 2 Run Status
		0	Task 2 is not executing. (default)
		1	Task 2 is executing.
0	INT1		Task 1 Run Status
		0	Task 1 is not executing. (default)
		1	Task 1 is executing.

9.4.3.8 Interrupt Force Register (MIFRC)

The interrupt force register can be used by the main CPU to start tasks through software. Writing a 1 to a MIFRC bit will set the corresponding bit in the MIFR register. Writes of 0 are ignored and reads always return 0. The IACK #16bit operation can also be used to start tasks and has the same effect as the MIFRC register. To enable IACK to set MIFR bits you must first set the MCTL[IACKE] bit. Using IACK has the advantage of not having to first set the EALLOW bit. This allows the main CPU to efficiently trigger CLA tasks through software.

Figure 9-10. Interrupt Force Register (MIFRC)

15	8
Reserved	
R -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; R = Read only; -n = value after reset

Table 9-10. Interrupt Force Register (MIFRC) Field Descriptions

Bits	Name	Value	Description [(1)]
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 8 interrupt.
6	INT7		Task 7 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 7 interrupt.
5	INT6		Task 6 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 6 interrupt.
4	INT5		Task 5 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 5 interrupt.
3	INT4		Task 4 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 4 interrupt.
2	INT3		Task 3 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 3 interrupt.
1	INT2		Task 2 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 2 interrupt.
0	INT1		Task 1 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 1 interrupt.

(1) This register is protected by EALLOW and the code security module.

9.4.3.9 Interrupt Flag Clear Register (MICLR)

Normally bits in the MIFR register are automatically cleared when a task begins. The interrupt flag clear register can be used to instead manually clear bits in the interrupt flag (MIFR) register. Writing a 1 to a MICLR bit will clear the corresponding bit in the MIFR register. Writes of 0 are ignored and reads always return 0.

Figure 9-11. Interrupt Flag Clear Register (MICLR)

15–8
Reserved
R -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; R = Read only; -n = value after reset

Table 9-11. Interrupt Flag Clear Register (MICLR) Field Descriptions

Bits	Name	Value	Description [(1)]
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 8 interrupt flag.
6	INT7		Task 7 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 7 interrupt flag.
5	INT6		Task 6 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 6 interrupt flag.
4	INT5		Task 5 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 5 interrupt flag.
3	INT4		Task 4 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 4 interrupt flag.
2	INT3		Task 3 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 3 interrupt flag.
1	INT2		Task 2 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 2 interrupt flag.
0	INT1		Task 1 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 1 interrupt flag.

(1) This register is protected by EALLOW and the code security module.

9.4.3.10 Interrupt Overflow Flag Clear Register (MICLROVF)

Overflow flag bits in the MIOVF register are latched until manually cleared using the MICLROVF register. Writing a 1 to a MICLROVF bit will clear the corresponding bit in the MIOVF register. Writes of 0 are ignored and reads always return 0.

Figure 9-12. Interrupt Overflow Flag Clear Register (MICLROVF)

15	8
Reserved	
R -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; R = Read only; -n = value after reset

Table 9-12. Interrupt Overflow Flag Clear Register (MICLROVF) Field Descriptions

Bits	Name	Value	Description [(1)]
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 8 interrupt overflow flag.
6	INT7		Task 7 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 7 interrupt overflow flag.
5	INT6		Task 6 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 6 interrupt overflow flag.
4	INT5		Task 5 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 5 interrupt overflow flag.
3	INT4		Task 4 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 4 interrupt overflow flag.
2	INT3		Task 3 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 3 interrupt overflow flag.
1	INT2		Task 2 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 2 interrupt overflow flag.
0	INT1		Task 1 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 1 interrupt overflow flag.

(1) This register is protected by EALLOW and the code security module.

9.4.4 Execution Registers

The CLA program counter is initialized by the appropriate MVECTx register when an interrupt is received and a task begins execution. The MPC points to the instruction in the decode 2 (D2) stage of the CLA pipeline. After a MSTOP operation, if no other tasks are pending, the MPC will remain pointing to the MSTOP instruction. The MPC register can be read by the main C28x CPU for debug purposes. The main CPU cannot write to MPC.

9.4.4.1 MPC Register

The MPC register is described in Figure 9-13 and described in Table 9-13.

Figure 9-13. Program Counter (MPC)

15 12	11 0
Reserved	MPC
R-0	R-0

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

Table 9-13. Program Counter (MPC) Field Descriptions

Bits	Name	Value	Description [(1)]
15-12	Reserved		Any writes to these bit(s) must always have a value of 0.
11-0	MPC	0000 - 0FFF	Points to the instruction currently in the decode 2 phase of the CLA pipeline. The value is the offset from the first address in the CLA program space.

(1) This register is protected by the code security module. The main CPU can read this register for debug purposes but it can not write to it.

9.4.4.2 MSTF Register

The CLA status register (MSTF) reflects the results of different operations. These are the basic rules for the flags:

- Zero and negative flags are cleared or set based on:
  – floating-point moves to registers
  – the result of compare, minimum, maximum, negative and absolute value operations
  – the integer result of operations such as MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32
- Overflow and underflow flags are set by floating-point math instructions such as multiply, add, subtract and 1/x. These flags may also be connected to the peripheral interrupt expansion (PIE) block on your device. This can be useful for debugging underflow and overflow conditions within an application.

The MSTF register is shown in Figure 10-3 and described in Table 10-5.

Figure 9-14. CLA Status Register (MSTF)

31 24	23 16
Reserved	RPC
R/W-0	R/W-0

15 12	11	10	9	8 7	6	5 4	3	2	1	0
RPC	MEALLOW	Reserved	RND32	Reserved	TF	Reserved	ZF	NF	LUF	LVF
R/W-0	R/W-0	R-0	R/W-0	R-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; *-n* = value after reset

Table 9-14. CLA Status (MSTF) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31 - 24	Reserved	0	Reserved for future use
23 - 12	RPC		Return program counter. The RPC is used to save and restore the MPC address by the MCCNDD and MRCNDD operations.
11	MEALLOW		This bit enables and disables CLA write access to EALLOW protected registers. This is independent of the state of the EALLOW bit in the main CPU status register. This status bit can be saved and restored by the MMOV32 STF, mem32 instruction.
		0	The CLA cannot write to EALLOW protected registers. This bit is cleared by the CLA instruction, MEDIS.
		1	The CLA is allowed to write to EALLOW protected registers. This bit is set by the CLA instruction, MEALLOW.
10	Reserved	0	Any writes to these bit(s) must always have a value of 0.
9	RND32		Round 32-bit Floating-Point Mode Use the MSETFLG and MMOV32 MSTF, mem32 instructions to change the rounding mode.
		0	If this bit is zero, the MMPYF32, MADDF32 and MSUBF32 instructions will round to zero (truncate).
		1	If this bit is one, the MMPYF32, MADDF32 and MSUBF32 instructions will round to the nearest even value.
8 - 7	Reserved	0	Reserved for future use
6	TF		Test Flag The MTESTTF instruction can modify this flag based on the condition tested. The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	The condition tested with the MTESTTF instruction is false.
		1	The condition tested with the MTESTTF instruction is true.
5 - 4	Reserved		These two bits may change based on integer results. These flags are not, however, used by the CLA and therefore marked as reserved.
3	ZF		Zero Flag [(2)] [(3)] • Instructions that modify this flag based on the floating-point value stored in the destination register: MMOV32, MMOVD32, MABSF32, MNEGF32 • Instructions that modify this flag based on the floating-point result of the operation: MCMPF32, MMAXF32, and MMINF32 • Instructions that modify this flag based on the integer result of the operation: MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32 and MLSL32 The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag
		0	The value is not zero.
		1	The value is zero.
2	NF		Negative Flag [(2)] [(3)] • Instructions that modify this flag based on the floating-point value stored in the destination register: MMOV32, MMOVD32, MABSF32, MNEGF32 • Instructions that modify this flag based on the floating-point result of the operation: MCMPF32, MMAXF32, and MMINF32 • Instructions that modify this flag based on the integer result of the operation: MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32 and MLSL32 The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	The value is not negative.
		1	The value is negative.

(1) This register is protected by the code security module. The main CPU can read this register for debug purposes but it can not write to it.
(2) A negative zero floating-point value is treated as a positive zero value when configuring the ZF and NF flags.
(3) A DeNorm floating-point value is treated as a positive zero value when configuring the ZF and NF flags.

Table 9-14. CLA Status (MSTF) Register Field Descriptions (continued)

Bits	Field	Value	Description [(1)]
1	LUF		Latched Underflow Flag
			The following instructions will set this flag to 1 if an underflow occurs: MMPYF32, MADDF32, MSUBF32, MMACF32, MEINVF32, MEISQRTF32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	An underflow condition has not been latched.
		1	An underflow condition has been latched.
0	LVF		Latched Overflow Flag
			The following instructions will set this flag to 1 if an overflow occurs: MMPYF32, MADDF32, MSUBF32, MMACF32, MEINVF32, MEISQRTF32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	An overflow condition has not been latched.
		1	An overflow condition has been latched.

9.5 Pipeline

This section describes the CLA pipeline stages and presents cases where pipeline alignment must be considered.

9.5.1 Pipeline Overview

The CLA pipeline is very similar to the C28x pipeline. The pipeline has eight stages:

- **Fetch 1 (F1)**

  During the F1 stage the program read address is placed on the CLA program address bus.
- **Fetch 2 (F2)**

  During the F2 stage the instruction is read using the CLA program data bus.
- **Decode 1 (D1)**

  During D1 the instruction is decoded.
- **Decode 2 (D2)**

  Generate the data read address. Changes to MAR0 and MAR1 due to post-increment using indirect addressing takes place in the D2 phase. Conditional branch decisions are also made at this stage based on the MSTF register flags.
- **Read 1 (R1)**

  Place the data read address on the CLA data-read address bus. If a memory conflict exists, the R1 stage will be stalled.
- **Read 2 (R2)**

  Read the data value using the CLA data read data bus.
- **Execute (EXE)**

  Execute the operation. Changes to MAR0 and MAR1 due to loading an immediate value or value from memory take place in this stage.
- **Write (W)**

  Place the write address and write data on the CLA write data bus. If a memory conflict exists, the W stage will be stalled.

9.5.2 CLA Pipeline Alignment

The majority of the CLA instructions do not require any special pipeline considerations. This section lists the few operations that do require special consideration.

- **Write Followed by Read**

  In both the C28x and the CLA pipeline the read operation occurs before the write. This means that if a read operation immediately follows a write, then the read will complete first as shown in Table 9-15. In most cases this does not cause a problem since the contents of one memory location does not depend on the state of another. For accesses to peripherals where a write to one location can affect the value in another location the code must wait for the write to complete before issuing the read as shown in Table 9-16.

  This behavior is different for the 28x CPU. For the 28x CPU any write followed by read to the same location is protected by what is called write-followed-by-read protection. This protection automatically stalls the pipeline so that the write will complete before the read. In addition some peripheral frames are protected such that a 28x CPU write to one location within the frame will always complete before a read to the frame. The CLA does not have this protection mechanism. Instead the code must wait to perform the read.

Table 9-15. Write Followed by Read - Read Occurs First

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1 MMOV16 @Reg1, MR3	I1							
I2 MMOV16 MR2, @Reg2	I2	I1						
		I2	I1					
			I2	I1				
				I2	I1			
					I2	I1		
						I2	I1	
							I2	I1

Table 9-16. Write Followed by Read - Write Occurs First

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1 MMOV16 @Reg1, MR3	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
I5 MMOV16 MR2, @Reg2	I5	I4	I3	I2	I1			
		I5	I4	I3	I2	I1		
			I5	I4	I3	I2	I1	
				I5	I4	I3	I2	I1
					I5	I4	I3	
						I5	I4	
							I5	

- **Delayed Conditional instructions: MBCNDD, MCCNDD and MRCNDD**

  Referring to Example 9-1, the following applies to delayed conditional instructions:

  – **I1**

  I1 is the last instruction that can effect the CNDF flags for the branch, call or return instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MBCNDD, MCCNDD or MRCNDD is in the D2 phase.

  – **I2, I3 and I4**

  The three instructions preceding MBCNDD can change MSTF flags but will have no effect on whether the MBCNDD instruction branches or not. This is because the flag modification will occur after the D2 phase of the branch, call or return instruction. These three instructions must not be a MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

  – **I5, I6 and I7**

  The three instructions following a branch, call or return are always executed irrespective of whether the condition is true or not. These instructions must not be MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

  For a more detailed description refer to the functional description for MBCNDD, MCCNDD and MRCNDD.

Example 9-1. Code Fragment For MBCNDD, MCCNDD or MRCNDD

```
   <Instruction 1>    ; I1 Last instruction that can affect flags for
                      ;     the branch, call or return operation

   <Instruction 2>    ; I2 Cannot be stop, branch, call or return
   <Instruction 3>    ; I3 Cannot be stop, branch, call or return
   <Instruction 4>    ; I4 Cannot be stop, branch, call or return

   <branch/call/ret>  ; MBCNDD, MCCNDD or MRCNDD

                      ; I5-I7: Three instructions after are always
                      ; executed whether the branch/call or return is
                      ; taken or not

   <Instruction 5>    ; I5 Cannot be stop, branch, call or return
   <Instruction 6>    ; I6 Cannot be stop, branch, call or return
   <Instruction 7>    ; I7 Cannot be stop, branch, call or return

   <Instruction 8>    ; I8
   <Instruction 9>    ; I9
   ....
```

- **Stop or Halting a Task: MSTOP and MDEBUGSTOP**

  The MSTOP and MDEBUGSTOP instructions cannot be placed three instructions before or after a conditional branch, call or return instruction (MBCNDD, MCCNDD or MRCNDD). Refer to Example 9-1. To single-step through a branch/call or return, insert the MDEBUGSTOP at least four instructions back and step from there.

- **Loading MAR0 or MAR1**

  A load of auxiliary register MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Referring to Example 9-2, the following applies when loading the auxiliary registers:

  – **I1 and I2**

    The two instructions following the load instruction will use the value in MAR0 or MAR1 before the update occurs.

  – **I3**

    Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win and the auxiliary register will not be updated with #_X.

  – **I4**

    Starting with the 4th instruction MAR0 or MAR1 will have the new value.

Example 9-2. Code Fragment for Loading MAR0 or MAR1

```
; Assume MAR0 is 50 and #_X is 20

   MMOVI16 MAR0, #_X ; Load MAR0 with address of X (20)
   <Instruction 1>   ; I1 Will use the old value of MAR0 (50)
   <Instruction 2>   ; I2 Will use the old value of MAR0 (50)
   <Instruction 3>   ; I3 Cannot use MAR0
   <Instruction 4>   ; I4 Will use the new value of MAR0 (20)
   <Instruction 5>   ; I5 Will use the new value of MAR0 (20
   ....
```

9.5.2.1 ADC Early Interrupt to CLA Response

The 2806x ADC offers the option to generate an early interrupt pulse when the ADC begins conversion. This option is selected by setting the ADCCTL1[INTPULSEPOS] bit as documented in the Analog-to-Digital Converter and Comparator section in this manual. If this option is used to start a CLA task then the CLA will be able to read the result as soon as the conversion completes and the ADC result register updates. This just-in-time sampling along with the low interrupt response of the CLA enable faster system response and higher frequency control loops.

The timing for the ADC conversion is shown in the ADC Reference Guide timing diagrams. From a CLA perspective, the pipeline activity is shown in Table 9-17. The 8th instruction is in the R2 phase just in time to read the result register. While the first seven (7) instructions in the task (I1 to I7) will enter the R2 phase of the pipeline too soon to read the conversion, they can be efficiently used for pre-processing calculations needed by the task.

Table 9-17. ADC to CLA Early Interrupt Response

ADC Activity	CLA Activity	F1	F2	D1	D2	R1	R2	E	W
Sample									
Sample									
...									
Sample									
Conversion (1)	Interrupt Received								
Conversion (2)	Task Startup								
Conversion (3)	Task Startup								
Conversion (4)	I1	I1							
Conversion (5)	I2	I2	I1						
Conversion (6)	I3	I3	I2	I1					
Conversion (7)	I4	I4	I3	I2	I1				
Conversion (8)	I5	I5	I4	I3	I2	I1			
Conversion (9)	I6	I6	I5	I4	I3	I2	I1		
Conversion (10)	I7	I7	I6	I5	I4	I3	I2		
Conversion (11)	I8 Read ADC RESULT	I8	I7	I6	I5	I4	I3		
Conversion (12)			I8	I7	I6	I5	I4		
Conversion (13)				I8	I7	I6	I5		
Conversion Complete					I8	I7	I6		
RESULT Latched						I8	I7		
RESULT Available							I8		

9.5.3 Parallel Instructions

Parallel instructions are single opcodes that perform two operations in parallel. The following types of parallel instructions are available: math operation in parallel with a move operation, or two math operations in parallel. Both operations complete in a single cycle and there are no special pipeline alignment requirements.

Example 9-3. Math Operation with Parallel Load

```
;  MADDF32 || MMOV32 instruction: 32-bit floating-point add with parallel move
;  MADDF32 is a 1 cycle operation
;  MMOV32 is a 1 cycle operation
         MADDF32    MR0,  MR1, #2    ; MR0 = MR1 + 2,
      || MMOV32   MR1,  @Val         ; MR1 gets the contents of Val
                                     ; <-- MMOV32 completes here (MR1 is valid)
                                     ; <-- DDF32 completes here (MR0 is valid)
         MMPYF32 MR0, MR0, MR1       ; Any instruction, can use MR1 and/or MR0
```

Example 9-4. Multiply with Parallel Add

```
;  MMPYF32 || MADDF32 instruction: 32-bit floating-point multiply with parallel add
;  MMPYF32 is a 1 cycle operation
;  MADDF32 is a 1 cycle operation
     MMPYF32 MR0, MR1, MR3        ; MR0 = MR1 * MR3
 ||  MADDF32 MR1, MR2, MR0        ; MR1 = MR2 + MR0 (Uses value of MR0 before MMPYF32)
                                  ; <-- MMPYF32 and MADDF32 complete here (MR0 and MR1 are valid)
     MMPYF32 MR1, MR1, MR0        ; Any instruction, can use MR1 and/or MR0
```

9.6 Instruction Set

This section describes the assembly language instructions of the control law accelerator. Also described are parallel operations, conditional operations, resource constraints, and addressing modes. The instructions listed here are independent from C28x and C28x+FPU instruction sets.

9.6.1 *Instruction Descriptions*

This section gives detailed information on the instruction set. Each instruction may present the following information:

- Operands
- Opcode
- Description
- Exceptions
- Pipeline
- Examples
- See also

The example INSTRUCTION is shown to familiarize you with the way each instruction is described. The example describes the kind of information you will find in each part of the individual instruction description and where to obtain more information. CLA instructions follow the same format as the C28x; the source operand(s) are always on the right and the destination operand(s) are on the left.

The explanations for the syntax of the operands used in the instruction descriptions for the C28x CLA are given in Table 9-18.

Table 9-18. Operand Nomenclature

Symbol	Description
#16FHi	16-bit immediate (hex or float) value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FHiHex	16-bit immediate hex value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FLoHex	A 16-bit immediate hex value that represents the lower 16-bits of an IEEE 32-bit floating-point value
#32Fhex	32-bit immediate value that represents an IEEE 32-bit floating-point value
#32F	Immediate float value represented in floating-point representation
#0.0	Immediate zero
#SHIFT	Immediate value of 1 to 32 used for arithmetic and logical shifts.
addr	Opcode field indicating the addressing mode
CNDF	Condition to test the flags in the MSTF register
FLAG	Selected flags from MSTF register (OR) 8 bit mask indicating which floating-point status flags to change
MAR0	auxiliary register 0
MAR1	auxiliary register 1
MARx	Either MAR0 or MAR1
mem16	16-bit memory location accessed using direct or indirect addressing modes
mem32	32-bit memory location accessed using direct or indirect addressing modes
MRa	MR0 to MR3 registers
MRb	MR0 to MR3 registers
MRc	MR0 to MR3 registers
MRd	MR0 to MR3 registers
MRe	MR0 to MR3 registers
MRf	MR0 to MR3 registers
MSTF	CLA Floating-point Status Register
shift	Opcode field indicating the number of bits to shift.
VALUE	Flag value of 0 or 1 for selected flag (OR) 8 bit mask indicating the flag value; 0 or 1

Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

Table 9-19. INSTRUCTION dest, source1, source2 Short Description

	Description
dest1	Description for the 1st operand for the instruction
source1	Description for the 2nd operand for the instruction
source2	Description for the 3rd operand for the instruction
Opcode	This section shows the opcode for the instruction
Description	Detailed description of the instruction execution is described. Any constraints on the operands imposed by the processor or the assembler are discussed.
Restrictions	Any constraints on the operands or use of the instruction imposed by the processor are discussed.
Pipeline	This section describes the instruction in terms of pipeline cycles as described in Section 9.5
Example	Examples of instruction execution. If applicable, register and memory values are given before and after instruction execution. Some examples are code fragments while other examples are full tasks that assume the CLA is correctly configured and the main CPU has passed it data.
Operands	Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

9.6.2 Addressing Modes and Encoding

The CLA uses the same address to access data and registers as the main CPU. For example if the main CPU accesses an ePWM register at address 0x00 6800, then the CLA will access it using address 0x6800. Since all CLA accessible memory and registers are within the low 64k x 16 of memory, only the low 16-bits of the address are used by the CLA.

To address the CLA data memory, message RAMs and shared peripherals, the CLA supports two addressing modes:

- Direct addressing mode: Uses the address of the variable or register directly.
- Indirect addressing with 16-bit post increment. This mode uses either XAR0 or XAR1.

The CLA does not use a data page pointer or a stack pointer. The two addressing modes are encoded as shown Table 9-20.

Table 9-20. Addressing Modes

Addressing Mode	'addr' Opcode Field Encode [(1)]	Description	
@dir	**0000**	**Direct Addressing Mode**	
		Example 1: MMOV32 MR1, @_VarA	
		Example 2: MMOV32 MR1, @_EPwm1Regs.CMPA.all	
		In this case the 'mmmm mmmm mmmm mmmm' opcode field will be populated with the 16-bit address of the variable. This is the low 16-bits of the address that you would use to access the variable using the main CPU.	
		For example @_VarA will populate the address of the variable VarA. and @_EPwm1Regs.CMPA.all will populate the address of the CMPA register.	
***MAR0[#imm16]++**	**0001**	**MAR0 Indirect Addressing with 16-bit Immediate Post Increment**	
***MAR1[#imm16]++**	**0010**	**MAR1 Indirect Addressing with 16-bit Immediate Post Increment**	
		addr = MAR0 (or MAR1) MAR0 (or MAR1) += #imm16	Access memory using the address stored in MAR0 (or MAR1). Then post increment MAR0 (or MAR1) by #imm16.
		Example 1: MMOV32 MR0, *MAR0[2]++	
		Example 2: MMOV32 MR1, *MAR1[-2]++	
		For a post increment of 0 the assembler will accept both *MAR0 and *MAR0[0]++.	
		The 'mmmm mmmm mmmm mmmm' opcode field will be populated with the signed 16-bit pointer offset. For example if #imm16 is 2, then the opcode field will be 0x0002. Likewise if #imm16 is -2, then the opcode field will be 0xFFFE.	
		If addition of the 16-bit immediate causes overflow, then the value will wrap around on a 16-bit boundary.	

(1) Values not shown are reserved.

Encoding for the shift fields in the MASR32, MLSR32 and MLSL32 instructions is shown in Table 9-21

Table 9-21. Shift Field Encoding

Shift Value	'shift' Opcode Field Encode
1	0000
2	0001
3	0010
....	....
32	1111

Table 9-22 shows the condition field encoding for conditional instructions such as MNEGF, MSWAPF, MBCNDD, MCCNDD and MRCNDD

Table 9-22. Condition Field Encoding

Encode [(1)]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [(2)]	Unconditional with flag modification	None

(1) Values not shown are reserved.

(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

9.6.3 Instructions

The instructions are listed alphabetically, preceded by a summary.

Table 9-23. General Instructions

Table 9-23. General Instructions (continued)

MMABSF32 MRa, MRb *32-Bit Floating-Point Absolute Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0010 0000
```

Description

The absolute value of MRb is loaded into MRa. Only the sign bit of the operand is modified by the MMABSF32 instruction.

```
if (MRb < 0) {MRa = -MRb};
        else {MRa =  MRb};
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
NF = 0;
ZF = 0;
if ( MRa(30:23) == 0) ZF = 1;
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ MR0, #-2.0 ; MR0 = -2.0 (0xC0000000)
MMABSF32 MR0, MR0  ; MR0 = 2.0 (0x40000000), ZF = NF = 0

MMOVIZ MR0, #5.0  ; MR0 = 5.0 (0x40A00000)
MMABSF32 MR0, MR0  ; MR0 = 5.0 (0x40A00000), ZF = NF = 0

MMOVIZ MR0, #0.0  ; MR0 = 0.0
MMABSF32 MR0, MR0  ; MR0 = 0.0 ZF = 1, NF = 0
```

See also

MNEGF32 MRa, MRb {, CNDF}

MADD32 MRa, MRb, MRc *32-Bit Integer Add*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point destination register (MR0 to MR3)
MRc	CLA floating-point destination register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 000cc bbaa
MSW: 0111 1110 1100 0000
```

Description

32-bit integer addition of MRb and MRc.

```
MRa(31:0) = MRb(31:0) + MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; };
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
; Calculate Y2 = A + B + C
;
_Cla1Task1:
       MMOV32 MR0, @_A      ; MR0 = 1 (0x00000001)
       MMOV32 MR1, @_B      ; MR1 = 2 (0x00000002)
       MMOV32 MR2, @_C      ; MR2 = -7 (0xFFFFFFF9)
       MADD32 MR3, MR0, MR1 ; A + B
       MADD32 MR3, MR2, MR3 ; A + B + C = -4 (0xFFFFFFFC)
       MMOV32 @_y2, MR3     ; Store y2
       MSTOP                ; end of task
```

See also

MAND32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MADDF32 MRa, #16FHi, MRb *32-Bit Floating-Point Addition*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1100 bbaa
```

Description

Add MRb to the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb + #16FHi:0;
```

This instruction can also be written as MADDF32 MRa, MRb, #16FHi.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Add to MR1 the value 2.0 in 32-bit floating-point format
; Store the result in MR0
  MADDF32 MR0, #2.0, MR1    ; MR0 = 2.0 + MR1

; Add to MR3 the value -2.5 in 32-bit floating-point format
; Store the result in MR2
  MADDF32 MR2, #-2.5, MR3   ; MR2 = -2.5 + MR3

; Add to MR3 the value 0x3FC00000 (1.5)
; Store the result in MR3
  MADDF32 MR3, #0x3FC0, MR3 ; MR3 = 1.5 + MR3
```

See also

MADDF32 MRa, MRb, #16FHi
MADDF32 MRa, MRb, MRc
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MADDF32 MRa, MRb, #16FHi *32-Bit Floating-Point Addition*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1100 bbaa
```

Description

Add MRb to the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb + #16FHi:0;
```

This instruction can also be written as MADDF32 MRa, #16FHi, MRb.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example 1

```
; X is an array of 32-bit floating-point values
; Find the maximum value in an array X
; and store it in Result
;
_Cla1Task1:
    MMOVI16     MAR1,#_X          ; Start address
    MUI16TOF32 MR0, @_len         ; Length of the array
    MNOP                          ; delay for MAR1 load
    MNOP                          ; delay for MAR1 load
    MMOV32      MR1, *MAR1[2]++   ; MR1 = X0
LOOP
    MMOV32      MR2, *MAR1[2]++   ; MR2 = next element
    MMAXF32     MR1,  MR2         ; MR1 = MAX(MR1, MR2)
    MADDF32     MR0, MR0, #-1.0   ; Decrement the counter
    MCMPF32     MR0 #0.0          ; Set/clear flags for MBCNDD
    MNOP
    MNOP
    MNOP
    MBCNDD LOOP, NEQ              ; Branch if not equal to zero
    MMOV32 @_Result, MR1          ; Always executed
    MNOP                          ; Always executed
    MNOP                          ; Always executed
    MSTOP                         ; End of task
```

Example 2

```
; Show the basic operation of MADDF32
;
; Add to MR1 the value 2.0 in 32-bit floating-point format
; Store the result in MR0
    MADDF32 MR0, MR1, #2.0    ; MR0 = MR1 + 2.0

; Add to MR3 the value -2.5 in 32-bit floating-point format
; Store the result in MR2
    MADDF32 MR2, MR3, #-2.5   ; MR2 = MR3 + (-2.5)

; Add to MR0 the value 0x3FC00000 (1.5)
; Store the result in MR0
   MADDF32 MR0, MR0, #0x3FC0  ; MR0 = MR0 + 1.5
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, MRc
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MADDF32 MRa, MRb, MRc *32-Bit Floating-Point Addition*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 000 0000 00cc bbaa
MSW: 0111 1100 0010 0000
```

Description

Add the contents of MRc to the contents of MRb and load the result into MRa.

```
MRa = MRb + MRc;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Given M1, X1 and B1 are 32-bit floating point numbers
; Calculate Y1 = M1*X1+B1
;
_Cla1Task1:
   MMOV32 MR0,@M1      ; Load MR0 with M1
   MMOV32 MR1,@X1      ; Load MR1 with X1
   MMPYF32 MR1,MR1,MR0 ; Multiply M1*X1
|| MMOV32 MR0,@B1      ; and in parallel load MR0 with B1
   MADDF32 MR1,MR1,MR0 ; Add M*X1 to B1 and store in MR1
   MMOV32 @Y1,MR1      ; Store the result
   MSTOP               ; end of task
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, #16FHi
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MADDF32 MRd, MRe, MRf||MMOV32 mem32, MRa *32-Bit Floating-Point Addition with Parallel Move*

Operands

MRd	CLA floating-point destination register for the MADDF32 (MR0 to MR3)
MRe	CLA floating-point source register for the MADDF32 (MR0 to MR3)
MRf	CLA floating-point source register for the MADDF32 (MR0 to MR3)
mem32	32-bit memory location accessed using direct or indirect addressing. This will be the destination of the MMOV32.
MRa	CLA floating-point source register for the MMOV32 (MR0 to MR3)

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0101 ffee ddaa addr
```

Description

Perform an MADDF32 and a MMOV32 in parallel. Add MRf to the contents of MRe and store the result in MRd. In parallel move the contents of MRa to the 32-bit location mem32.

```
MRd = MRe + MRf;
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

Both MADDF32 and MMOV32 complete in a single cycle.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A * B) + C
;
_Cla1Task2:
   MMOV32   MR0, @_A        ; Load MR0 with A
   MMOV32   MR1, @_B        ; Load MR1 with B
   MMPYF32  MR1, MR1, MR0   ; Multiply A*B
|| MMOV32   MR0, @_C        ;     and in parallel load MR0 with C
   MADDF32  MR1, MR1, MR0   ; Add (A*B) to C
|| MMOV32   @_Y2, MR1       ;     and in parallel store A*B
   MMOV32   @_Y3, MR1       ; Store the A*B + C
   MSTOP                    ; end of task
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, #16FHi
MADDF32 MRa, MRb, MRc
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32

MADDF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 *32-Bit Floating-Point Addition with Parallel Move*

Operands

MRd	CLA floating-point destination register for the MADDF32 (MR0 to MR3). MRd cannot be the same register as MRa.
MRe	CLA floating-point source register for the MADDF32 (MR0 to MR3)
MRf	CLA floating-point source register for the MADDF32 (MR0 to MR3)
MRa	CLA floating-point destination register for the MMOV32 (MR0 to MR3). MRa cannot be the same register as MRd.
mem32	32-bit memory location accessed using direct or indirect addressing. This is the source for the MMOV32.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0001 ffee ddaa addr
```

Description

Perform an MADDF32 and a MMOV32 operation in parallel. Add MRf to the contents of MRe and store the result in MRd. In parallel move the contents of the 32-bit location mem32 to MRa.

```
MRd = MRe + MRf;
MRa = [mem32];
```

Restrictions

The destination register for the MADDF32 and the MMOV32 must be unique. That is, MRa and MRd cannot be the same register.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

The MMOV32 Instruction will set the NF and ZF flags as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; };
```

Pipeline

The MADDF32 and the MMOV32 both complete in a single cycle.

Example 1

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y1 = A + 4B
;           Y2 = A + C
;
_Cla1Task1:
    MMOV32 MR0, @A         ; Load MR0 with A
    MMOV32 MR1, @B         ; Load MR1 with B
    MMPYF32 MR1, MR1, #4.0 ; Multiply 4 * B
||  MMOV32 MR2, @C           and in parallel load C
    MADDF32 MR3, MR0, MR1  ; Add A + 4B
    MADDF32 MR3, MR0, MR2  ; Add A + C
||  MMOV32 @Y1, MR3        ; and in parallel store A+4B
    MMOV32 @Y2, MR3        ; store A + C MSTOP
                           ; end of task
```

Example 2

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y3 = (A + B)
;           Y4 = (A + B) * C
;
_Cla1Task2:
     MMOV32 MR0, @A         ; Load MR0 with A
     MMOV32 MR1, @B         ; Load MR1 with B
     MADDF32 MR1, MR1, MR0  ; Add A+B
||   MMOV32 MR0, @C         ; and in parallel load MR0 with C
     MMPYF32 MR1, MR1, MR0  ; Multiply (A+B) by C
||   MMOV32 @Y3, MR1        ; and in parallel store A+B
     MMOV32 @Y4, MR1        ; Store the (A+B) * C
     MSTOP                  ; end of task
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, #16FHi
MADDF32 MRa, MRb, MRc
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MAND32 MRa, MRb, MRc *Bitwise AND*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 0110 0000
```

Description

Bitwise AND of MRb with MRc.

```
MRa(31:0) = MRb(31:0) AND MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ   MR0,    #0x5555  ; MR0 = 0x5555AAAA
MMOVXI   MR0,    #0xAAAA

MMOVIZ   MR1,    #0x5432  ; MR1 = 0x5432FEDC
MMOVXI   MR1,    #0xFEDC

; 0101 AND 0101 = 0101 (5)
; 0101 AND 0100 = 0100 (4)
; 0101 AND 0011 = 0001 (1)
; 0101 AND 0010 = 0000 (0)
; 1010 AND 1111 = 1010 (A)
; 1010 AND 1110 = 1010 (A)
; 1010 AND 1101 = 1000 (8)
; 1010 AND 1100 = 1000 (8)

MAND32 MR2, MR1, MR0      ; MR3 = 0x5410AA88
```

See also

MADD32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MASR32 MRa, #SHIFT *Arithmetic Shift Right*

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#SHIFT	Number of bits to shift (1 to 32)

Opcode

```
LSW: 0000 0000 0shi ftaa
MSW: 0111 1011 0100 0000
```

Description

Arithmetic shift right of MRa by the number of bits indicated. The number of bits can be 1 to 32.

```
MARa(31:0) = Arithmetic Shift(MARa(31:0) by #SHIFT bits);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given m2 = (int32)32
;       x2 = (int32)64
;       b2 = (int32)-128
;
; Calculate
;       m2 = m2/2
;       x2 = x2/4
;       b2 = b2/8
;
_Cla1Task2:
   MMOV32 MR0, @_m2 ; MR0 = 32 (0x00000020)
   MMOV32 MR1, @_x2 ; MR1 = 64 (0x00000040)
   MMOV32 MR2, @_b2 ; MR2 = -128 (0xFFFFFF80)
   MASR32 MR0, #1   ; MR0 = 16 (0x00000010)
   MASR32 MR1, #2   ; MR1 = 16 (0x00000010)
   MASR32 MR2, #3   ; MR2 = -16 (0xFFFFFFF0)
   MMOV32 @_m2, MR0 ; store results
   MMOV32 @_x2, MR1
   MMOV32 @_b2, MR2
   MSTOP ; end of task
```

See also

MADD32 MRa, MRb, MRc
MAND32 MRa, MRb, MRc
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MBCNDD 16BitDest {, CNDF} *Branch Conditional Delayed*

Operands

16BitDest	16-bit destination if condition is true
CNDF	Optional condition tested

Opcode

```
LSW: dest dest dest dest
MSW: 0111 1001 1000 cndf
```

Description

If the specified condition is true, then branch by adding the signed 16BitDest value to the MPC value. Otherwise, continue without branching. If the address overflows, it wraps around. Therefore a value of "0xFFFE" will put the MPC back to the MBCNDD instruction. Since the MPC is only 12-bits, unused bits the upper 4 bits of the destination address are ignored.

Please refer to the pipeline section for important information regarding this instruction.

```
if (CNDF == TRUE) MPC += 16BitDest;
```

CNDF is one of the following conditions:

Encode [(1)]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [(2)]	Unconditional with flag modification	None

(1) Values not shown are reserved.

(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Restrictions

The MBCNDD instruction is not allowed three instructions before or after a MBCNDD, MCCNDD or MRCNDD instruction. Refer to the pipeline section for more information.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

The MBCNDD instruction by itself is a single-cycle instruction. As shown in Table 9-24 for each branch 6 instruction slots are executed; three before the branch instruction (I2-I4) and three after the branch instruction (I5-I7). The total number of cycles for a branch taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a branch can, therefore, range from 1 to 7 cycles. The number of cycles for a branch taken may not be the same as for a branch not taken.

Referring to Table 9-24 and Table 9-25, the instructions before and after MBCNDD have the following properties:

- **I1**
  - I1 is the last instruction that can effect the CNDF flags for the MBCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MBCNDD is in the D2 phase.
  - There are no restrictions on the type of instruction for I1.
- **I2, I3 and I4**
  - The three instructions proceeding MBCNDD can change MSTF flags but will have no effect on whether the MBCNDD instruction branches or not. This is because the flag modification will occur after the D2 phase of the MBCNDD instruction.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.
- **I5, I6 and I7**
  - The three instructions following MBCNDD are always executed irrespective of whether the branch is taken or not.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

```
<Instruction 1>   ; I1 Last instruction that can affect flags for
                  ; the MBCNDD operation
<Instruction 2>   ; I2 Cannot be stop, branch, call or return
<Instruction 3>   ; I3 Cannot be stop, branch, call or return
<Instruction 4>   ; I4 Cannot be stop, branch, call or return
MBCNDD _Skip, NEQ ; Branch to Skip if not eqal to zero
                  ; Three instructions after MBCNDD are always
                  ; executed whether the branch is taken or not
<Instruction 5>   ; I5 Cannot be stop, branch, call or return
<Instruction 6>   ; I6 Cannot be stop, branch, call or return
<Instruction 7>   ; I7 Cannot be stop, branch, call or return
<Instruction 8>   ; I8
<Instruction 9>   ; I9
....
_Skip:
 <Destination 1>  ; d1 Can be any instruction
 <Destination 2>  ; d2
 <Destination 3>  ; d3
....
....
MSTOP
....
```

Table 9-24. Pipeline Activity For MBCNDD, Branch Not Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MBCNDD	MBCNDD	I4	I3	I2	I1			
I5	I5	MBCNDD	I4	I3	I2	I1		
I6	I6	I5	MBCNDD	I4	I3	I2	I1	
I7	I7	I6	I5	MBCNDD	I4	I3	I2	
I8	I8	I7	I6	I5	-	I4	I3	
I9	I9	I8	I7	I6	I5	-	I4	
I10	I10	I9	I8	I7	I6	I5	-	
		I10	I9	I8	I7	I6	I5	
			I10	I9	I8	I7	I6	
				I10	I9	I8	I7	
					I10	I9	I8	
						I10	I9	
							I10	

Table 9-25. Pipeline Activity For MBCNDD, Branch Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MBCNDD	MBCNDD	I4	I3	I2	I1			
I5	I5	MBCNDD	I4	I3	I2	I1		
I6	I6	I5	MBCNDD	I4	I3	I2	I1	
I7	I7	I6	I5	MBCNDD	I4	I3	I2	
d1	d1	I7	I6	I5	-	I4	I3	
d2	d2	d1	I7	I6	I5	-	I4	
d3	d3	d2	d1	I7	I6	I5	-	
		d3	d2	d1	I7	I6	I5	
			d3	d2	d1	I7	I6	
				d3	d2	d1	I7	
					d3	d2	d1	
						d3	d2	
							d3	

Example 1

```
; if (State == 0.1)
; RampState = RampState || RAMPMASK
; else if (State == 0.01)
; CoastState = CoastState || COASTMASK
; else
; SteadyState = SteadyState || STEADYMASK
;
_Cla1Task1:
 MMOV32 MR0, @State
 MCMPF32 MR0, #0.1        ; Affects flags for 1st MBCNDD (A)
 MNOP
 MNOP
 MNOP
 MBCNDD Skip1, NEQ        ; (A) If State != 0.1, go to Skip1
 MNOP ; Always executed
 MNOP ; Always executed
 MNOP ; Always executed
 MMOV32 MR1, @RampState   ; Execute if (A) branch not taken
 MMOVXI MR2, #RAMPMASK    ; Execute if (A) branch not taken
 MOR32 MR1, MR2           ; Execute if (A) branch not taken
 MMOV32 @RampState, MR1   ; Execute if (A) branch not taken
 MSTOP                    ; end of task if (A) branch not taken
Skip1:
 MCMPF32 MR0,#0.01        ; Affects flags for 2nd MBCNDD (B)
 MNOP
 MNOP
 MNOP
 MBCNDD Skip2,NEQ         ; (B) If State != 0.01, go to Skip2
 MNOP ; Always executed
 MNOP ; Always executed
 MNOP ; Always executed
 MMOV32 MR1, @CoastState  ; Execute if (B) branch not taken
 MMOVXI MR2, #COASTMASK   ; Execute if (B) branch not taken
 MOR32 MR1, MR2           ; Execute if (B) branch not taken
 MMOV32 @CoastState, MR1  ; Execute if (B) branch not taken
 MSTOP
Skip2:
 MMOV32 MR3, @SteadyState ; Executed if (B) branch taken
 MMOVXI MR2, #STEADYMASK  ; Executed if (B) branch taken
 MOR32 MR3, MR2           ; Executed if (B) branch taken
 MMOV32 @SteadyState, MR3 ; Executed if (B) branch taken
 MSTOP
```

Example 2

```
; This example is the same as Example 1, except
; the code is optimized to take advantage of delay slots
;
; if (State == 0.1)
; RampState = RampState || RAMPMASK
; else if (State == 0.01)
; CoastState = CoastState || COASTMASK
; else
; SteadyState = SteadyState || STEADYMASK
;
_Cla1Task2:
 MMOV32 MR0, @State
 MCMPF32 MR0, #0.1         ; Affects flags for 1st MBCNDD (A)
 MCMPF32 MR0, #0.01        ; Check used by 2nd MBCNDD (B)
 MMTESTTF EQ                ; Store EQ flag in TF for 2nd MBCNDD (B)
 MNOP
 MBCNDD Skip1, NEQ         ; (A) If State != 0.1, go to Skip1
 MMOV32 MR1, @RampState    ; Always executed
 MMOVXI MR2, #RAMPMASK     ; Always executed
 MOR32 MR1, MR2            ; Always executed
 MMOV32 @RampState, MR1    ; Execute if (A) branch not taken
 MSTOP                     ; end of task if (A) branch not taken

Skip1:
 MMOV32 MR3, @SteadyState
 MMOVXI MR2, #STEADYMASK
 MOR32 MR3, MR2
 MBCNDD Skip2, NTF         ; (B) if State != .01, go to Skip2
 MMOV32 MR1, @CoastState   ; Always executed
 MMOVXI MR2, #COASTMASK    ; Always executed
 MOR32 MR1, MR2            ; Always executed
 MMOV32 @CoastState, MR1   ; Execute if (B) branch not taken
 MSTOP                     ; end of task if (B) branch not taken

Skip2:
 MMOV32 @SteadyState, MR3  ; Executed if (B) branch taken
 MSTOP
```

See also MCCNDD 16BitDest, CNDF
MRCNDD CNDF

MCCNDD 16BitDest {, CNDF} *Call Conditional Delayed*

Operands

16BitDest	16-bit destination if condition is true
CNDF	Optional condition to be tested

Opcode

```
LSW: dest dest dest dest
MSW: 0111 1001 1001 cndf
```

Description

If the specified condition is true, then store the return address in the RPC field of MSTF and make the call by adding the signed 16BitDest value to the MPC value. Otherwise, continue code execution without making the call. If the address overflows, it wraps around. Therefore a value of "0xFFFE" will put the MPC back to the MCCNDD instruction. Since the MPC is only 12 bits, unused bits the upper 4 bits of the destination address are ignored.

Please refer to the pipeline section for important information regarding this instruction.

```
if (CNDF == TRUE)
{
    RPC = return address;
    MPC += 16BitDest;
};
```

CNDF is one of the following conditions:

Encode [(3)]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [(4)]	Unconditional with flag modification	None

(3) Values not shown are reserved.
(4) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Restrictions

The MCCNDD instruction is not allowed three instructions before or after a MBCNDD, MCCNDD, or MRCNDD instruction. Refer to the Pipeline section for more details.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

The MCCNDD instruction by itself is a single-cycle instruction. As shown in Table 9-26, for each call 6 instruction slots are executed; three before the call instruction (I2-I4) and three after the call instruction (I5-I7). The total number of cycles for a call taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a call can, therefore, range from 1 to 7 cycles. The number of cycles for a call taken may not be the same as for a call not taken.

Referring to the following code fragment and the pipeline diagrams in Table 9-26 and Table 9-27, the instructions before and after MCCNDD have the following properties:

- **I1**
  - I1 is the last instruction that can effect the CNDF flags for the MCCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MCCNDD is in the D2 phase.
  - There are no restrictions on the type of instruction for I1.
- **I2, I3 and I4**
  - The three instructions proceeding MCCNDD can change MSTF flags but will have no effect on whether the MCCNDD instruction makes the call or not. This is because the flag modification will occur after the D2 phase of the MCCNDD instruction.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.
- **I5, I6 and I7**
  - The three instructions following MBCNDD are always executed irrespective of whether the branch is taken or not.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

```
<Instruction 1>   ; I1 Last instruction that can affect flags for
                  ;    the MCCNDD operation
<Instruction 2>   ; I2 Cannot be stop, branch, call or return
<Instruction 3>   ; I3 Cannot be stop, branch, call or return
<Instruction 4>   ; I4 Cannot be stop, branch, call or return

MCCNDD _func, NEQ ; Call to func if not eqal to zero

                  ; Three instructions after MCCNDD are always
                  ; executed whether the call is taken or not

<Instruction 5>   ; I5 Cannot be stop, branch, call or return
<Instruction 6>   ; I6 Cannot be stop, branch, call or return
<Instruction 7>   ; I7 Cannot be stop, branch, call or return
<Instruction 8>   ; I8 The address of this instruction is saved
                  ;    in the RPC field of the MSTF register.
                  ;    Upon return this value is loaded into MPC
                  ;    and fetching continues from this point.
<Instruction 9>   ; I9
....
_func:
<Destination 1>   ; d1 Can be any instruction
<Destination 2>   ; d2
<Destination 3>   ; d3
<Destination 4>   ; d4 Last instruction that can affect flags for
                  ;    the MRCNDD operation

<Destination 5>   ; d5 Cannot be stop, branch, call or return
<Destination 6>   ; d6 Cannot be stop, branch, call or return
<Destination 7>   ; d7 Cannot be stop, branch, call or return

MRCNDD  UNC       ; Return to <Instruction 8>, unconditional

                  ; Three instructions after MRCNDD are always
                  ; executed whether the return is taken or not

<Destination 8>   ; d8 Cannot be stop, branch, call or return
<Destination 9>   ; d9 Cannot be stop, branch, call or return
<Destination 10>  ; d10 Cannot be stop, branch, call or return
<Destination 11>  ; d11
....
MSTOP
```

Table 9-26. Pipeline Activity For MCCNDD, Call Not Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MCCNDD	MCCNDD	I4	I3	I2	I1			
I5	I5	MCCNDD	I4	I3	I2	I1		
I6	I6	I5	MCCNDD	I4	I3	I2	I1	
I7	I7	I6	I5	MCCNDD	I4	I3	I2	
I8	I8	I7	I6	I5	-	I4	I3	
I9	I9	I8	I7	I6	I5	-	I4	
I10	I10	I9	I8	I7	I6	I5	-	
etc		I10	I9	I8	I7	I6	I5	
....			I10	I9	I8	I7	I6	
....				I10	I9	I8	I7	
....					I10	I9	I8	
						I10	I9	
							I10	

Table 9-27. Pipeline Activity For MCCNDD, Call Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MCCNDD	MCCNDD	I4	I3	I2	I1			
I5	I5	MCCNDD	I4	I3	I2	I1		
I6	I6	I5	MCCNDD	I4	I3	I2	I1	
I7 [(1)]	I7	I6	I5	MCCNDD	I4	I3	I2	
d1	d1	I7	I6	I5	-	I4	I3	
d2	d2	d1	I7	I6	I5	-	I4	
d3	d3	d2	d1	I7	I6	I5	-	
etc		d3	d2	d1	I7	I6	I5	
....			d3	d2	d1	I7	I6	
....				d3	d2	d1	I7	
....					d3	d2	d1	
						d3	d2	
							d3	

(1) The RPC value in the MSTF register will point to the instruction following I7 (instruction I8).

Example ;

See also MBCNDD #16BitDest, CNDF
MMOV32 mem32, MSTF
MMOV32 MSTF, mem32
MRCNDD CNDF

MCMP32 MRa, MRb *32-Bit Integer Compare for Equal, Less Than or Greater Than*

Operands

MRa	CLA floating-point source register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1111 0010 0000
```

Description

Set ZF and NF flags on the result of MRa - MRb where MRa and MRb are 32-bit integers. For a floating point compare refer to MCMPF32.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
If(MRa ==    MRb) {ZF=1; NF=0;}
If(MRa >  MRb) {ZF=0; NF=0;}
If(MRa <  MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
; Behavior of ZF and NF flags for different comparisons
;
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
    MMOV32 MR0, @_A ; MR0 = 1 (0x00000001)
    MMOV32 MR1, @_B ; MR1 = 2 (0x00000002)
    MMOV32 MR2, @_C ; MR2 = -7 (0xFFFFFFF9)
    MCMP32 MR2, MR2 ; NF = 0, ZF = 1
    MCMP32 MR0, MR1 ; NF = 1, ZF = 0
    MCMP32 MR1, MR0 ; NF = 0, ZF = 0
```

See also

MADD32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MCMPF32 MRa, MRb *32-Bit Floating-Point Compare for Equal, Less Than or Greater Than*

Operands

MRa	CLA floating-point source register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0000 0000
```

Description

Set ZF and NF flags on the result of MRa - MRb. The MCMPF32 instruction is performed as a logical compare operation. This is possible because of the IEEE format offsetting the exponent. Basically the bigger the binary number, the bigger the floating-point value.

Special cases for inputs:

- Negative zero will be treated as positive zero.
- A denormalized value will be treated as positive zero.
- Not-a-Number (NaN) will be treated as infinity.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
If(MRa == MRb)   {ZF=1; NF=0;}
If(MRa > MRb) {ZF=0; NF=0;}
If(MRa < MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
; Behavior of ZF and NF flags for different comparisons

    MMOVIZ    MR1, #-2.0  ; MR1 = -2.0 (0xC0000000)
    MMOVIZ    MR0, #5.0   ; MR0 = 5.0 (0x40A00000)
    MCMPF32   MR1, MR0    ; ZF = 0, NF = 1
    MCMPF32   MR0, MR1    ; ZF = 0, NF = 0
    MCMPF32   MR0, MR0    ; ZF = 1, NF = 0
```

See also

MCMPF32 MRa, #16FHi
MMAXF32 MRa, #16FHi
MMAXF32 MRa, MRb
MMINF32 MRa, #16FHi
MMINF32 MRa, MRb

MCMPF32 MRa, #16FHi *32-Bit Floating-Point Compare for Equal, Less Than or Greater Than*

Operands

MRa	CLA floating-point source register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 1100 00aa
```

Description

Compare the value in MRa with the floating-point value represented by the immediate operand. Set the ZF and NF flags on (MRa - #16FHi:0).

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

The MCMPF32 instruction is performed as a logical compare operation. This is possible because of the IEEE floating-point format offsets the exponent. Basically the bigger the binary number, the bigger the floating-point value.

Special cases for inputs:

- Negative zero will be treated as positive zero.
- Denormalized value will be treated as positive zero.
- Not-a-Number (NaN) will be treated as infinity.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
If(MRa == #16FHi:0)   {ZF=1, NF=0;}
If(MRa > #16FHi:0) {ZF=0, NF=0;}
If(MRa < #16FHi:0) {ZF=0, NF=1;}
```

Pipeline

This is a single-cycle instruction

Example 1

```
; Behavior of ZF and NF flags for different comparisons

  MMOVIZ      MR1, #-2.0  ; MR1 = -2.0 (0xC0000000)
  MMOVIZ      MR0, #5.0   ; MR0 = 5.0 (0x40A00000)
  MCMPF32     MR1, #-2.2  ; ZF = 0, NF = 0
  MCMPF32     MR0, #6.5   ; ZF = 0, NF = 1
  MCMPF32     MR0, #5.0   ; ZF = 1, NF = 0
```

Example 2

```
; X is an array of 32-bit floating-point values
; and has len elements. Find the maximum value in
; the array and store it in Result
;
; Note: MCMPF32 and MSWAPF can be replaced with MMAXF32
;
_Cla1Task1:
  MMOVI16 MAR1,#_X        ; Start address
  MUI16TOF32 MR0, @_len   ; Length of the array
  MNOP                    ; delay for MAR1 load
  MNOP                    ; delay for MAR1 load
  MMOV32 MR1, *MAR1[2]++  ; MR1 = X0

LOOP
  MMOV32 MR2, *MAR1[2]++  ; MR2 = next element
  MCMPF32 MR2, MR1        ; Compare MR2 with MR1
  MSWAPF MR1, MR2, GT     ; MR1 = MAX(MR1, MR2)
  MADDF32 MR0, MR0, #-1.0 ; Decrememt the counter
  MCMPF32 MR0 #0.0        ; Set/clear flags for MBCNDD
  MNOP
  MNOP
  MNOP
  MBCNDD LOOP, NEQ        ; Branch if not equal to zero
  MMOV32 @_Result, MR1    ; Always executed
  MNOP                    ; Always executed
  MNOP                    ; Always executed
  MSTOP                   ; End of task
```

See also

MCMPF32 MRa, MRb
MMAXF32 MRa, #16FHi
MMAXF32 MRa, MRb
MMINF32 MRa, #16FHi
MMINF32 MRa, MRb

MDEBUGSTOP *Debug Stop Task*

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 0110 0000
```

Description

When CLA breakpoints are enabled, the MDEBUGSTOP instruction is used to halt a task so that it can be debugged. That is, MDEBUGSTOP is the CLA breakpoint. If CLA breakpoints are not enabled, the MDEBUGSTOP instruction behaves like a MNOP. Unlike the MSTOP, the MIRUN flag is not cleared and an interrupt is not issued. A single-step or run operation will continue execution of the task.

Restrictions

The MDEBUGSTOP instruction cannot be placed 3 instructions before or after a MBCNDD, MCCNDD or MRCNDD instruction.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

;

See also

MSTOP

MEALLOW — *Enable CLA Write Access to EALLOW Protected Registers*

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1001 0000
```

Description

This instruction sets the MEALLOW bit in the CLA status register MSTF. When this bit is set, the CLA is allowed write access to EALLOW protected registers. To again protect against CLA writes to protected registers, use the MEDIS instruction.

MEALLOW and MEDIS only control CLA write access; reads are allowed even if MEALLOW has not been executed. MEALLOW and MEDIS are also independant from the main CPU's EALLOW/EDIS. This instruction does not modify the EALLOW bit in the main CPU's status register. The MEALLOW bit in MSTF only controls access for the CLA while the EALLOW bit in the ST1 register only controls access for the main CPU.

As with EALLOW, the MEALLOW bit is overridden via the JTAG port, allowing full control of register accesses during debug from Code Composer Studio.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; C header file including definition of
; the EPwm1Regs structure
;
; The ePWM TZSEL register is EALLOW protected
;
   .cdecls C,LIST,"CLAShared.h"
   ...
_Cla1Task1:
   ...
   MEALLOW                            ; Allow CLA write access
   MMOV16 @_EPwm1Regs.TZSEL.all, MR3 ; Write to TZSEL
   MEDIS                              ; Disallow CLA write access
   ...
   ...
   MSTOP
```

See also

MEDIS

MEDIS *Disable CLA Write Access to EALLOW Protected Registers*

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1011 0000
```

Description

This instruction clears the MEALLOW bit in the CLA status register MSTF. When this bit is clear, the CLA is not allowed write access to EALLOW protected registers. To enable CLA writes to protected registers, use the MEALLOW instruction.

MEALLOW and MEDIS only control CLA write access; reads are allowed even if MEALLOW has not been executed. MEALLOW and MEDIS are also independant from the main CPU's EALLOW/EDIS. This instruction does not modify the EALLOW bit in the main CPU's status register. The MEALLOW bit in MSTF only controls access for the CLA while the EALLOW bit in the ST1 register only controls access for the main CPU.

As with EALLOW, the MEALLOW bit is overridden via the JTAG port, allowing full control of register accesses during debug from Code Composer Studio.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; C header file including definition of
; the EPwm1Regs structure
;
; The ePWM TZSEL register is EALLOW protected
;
   .cdecls C,LIST,"CLAShared.h"
   ...
_Cla1Task1:
   ...
   MEALLOW                          ; Allow CLA write access
   MMOV16 @_EPwm1Regs.TZSEL.all, MR3 ; Write to TZSEL
   MEDIS                            ; Disallow CLA write access
   ...
   ...
   MSTOP
```

See also

MEALLOW

MEINVF32 MRa, MRb *32-Bit Floating-Point Reciprocal Approximation*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1111 0000 0000
```

Description

This operation generates an estimate of 1/X in 32-bit floating-point format accurate to approximately 8 bits. This value can be used in a Newton-Raphson algorithm to get a more accurate answer. That is:

```
Ye = Estimate(1/X);
Ye = Ye*(2.0 - Ye*X);
Ye = Ye*(2.0 - Ye*X);
```

After two iterations of the Newton-Raphson algorithm, you will get an exact answer accurate to the 32-bit floating-point format. On each iteration the mantissa bit accuracy approximately doubles. The MEINVF32 operation will not generate a negative zero, DeNorm or NaN value.

```
MRa = Estimate of 1/MRb;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MEINVF32 generates an underflow condition.
- LVF = 1 if MEINVF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Calculate Num/Den using a Newton-Raphson algorithum for 1/Den
; Ye = Estimate(1/X)
; Ye = Ye*(2.0 - Ye*X)
; Ye = Ye*(2.0 - Ye*X)
;
_Cla1Task1:
    MMOV32 MR1, @_Den      ; MR1 = Den
    MEINVF32 MR2, MR1      ; MR2 = Ye = Estimate(1/Den)
    MMPYF32 MR3, MR2, MR1  ; MR3 = Ye*Den
    MSUBF32 MR3, #2.0, MR3 ; MR3 = 2.0 - Ye*Den
    MMPYF32 MR2, MR2, MR3  ; MR2 = Ye = Ye*(2.0 - Ye*Den)
    MMPYF32 MR3, MR2, MR1  ; MR3 = Ye*Den
 || MMOV32 MR0, @_Num      ; MR0 = Num
    MSUBF32 MR3, #2.0, MR3 ; MR3 = 2.0 - Ye*Den
    MMPYF32 MR2, MR2, MR3  ; MR2 = Ye = Ye*(2.0 - Ye*Den)
 || MMOV32 MR1, @_Den      ; Reload Den To Set Sign
    MNEGF32 MR0, MR0, EQ   ; if(Den == 0.0) Change Sign Of Num
    MMPYF32 MR0, MR2, MR0  ; MR0 = Y = Ye*Num
    MMOV32 @_Dest, MR0     ; Store result
    MSTOP                  ; end of task
```

See also

MEISQRTF32 MRa, MRb

MEISQRTF32 MRa, MRb *32-Bit Floating-Point Square-Root Reciprocal Approximation*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0100 0000
```

Description

This operation generates an estimate of 1/sqrt(X) in 32-bit floating-point format accurate to approximately 8 bits. This value can be used in a Newton-Raphson algorithm to get a more accurate answer. That is:

```
Ye = Estimate(1/sqrt(X));
Ye = Ye*(1.5 - Ye*Ye*X/2.0);
Ye = Ye*(1.5 - Ye*Ye*X/2.0);
```

After 2 iterations of the Newton-Raphson algorithm, you will get an exact answer accurate to the 32-bit floating-point format. On each iteration the mantissa bit accuracy approximately doubles. The MEISQRTF32 operation will not generate a negative zero, DeNorm or NaN value.

```
MRa = Estimate of 1/sqrt (MRb);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MEISQRTF32 generates an underflow condition.
- LVF = 1 if MEISQRTF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Y = sqrt(X)
; Ye = Estimate(1/sqrt(X));
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Y = X*Ye
;
_Cla1Task3:
    MMOV32 MR0, @_x          ; MR0 = X
    MEISQRTF32 MR1, MR0      ; MR1 = Ye = Estimate(1/sqrt(X))
    MMOV32 MR1, @_x, EQ      ; if(X == 0.0) Ye = 0.0
    MMPYF32 MR3, MR0, #0.5   ; MR3 = X*0.5
    MMPYF32 MR2, MR1, MR3    ; MR2 = Ye*X*0.5
    MMPYF32 MR2, MR1, MR2    ; MR2 = Ye*Ye*X*0.5
    MSUBF32 MR2, #1.5, MR2   ; MR2 = 1.5 - Ye*Ye*X*0.5
    MMPYF32 MR1, MR1, MR2    ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
    MMPYF32 MR2, MR1, MR3    ; MR2 = Ye*X*0.5
    MMPYF32 MR2, MR1, MR2    ; MR2 = Ye*Ye*X*0.5
    MSUBF32 MR2, #1.5, MR2   ; MR2 = 1.5 - Ye*Ye*X*0.5
    MMPYF32 MR1, MR1, MR2    ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
    MMPYF32 MR0, MR1, MR0    ; MR0 = Y = Ye*X
    MMOV32 @_y, MR0          ; Store Y = sqrt(X)
    MSTOP                    ; end of task
```

See also

MEINVF32 MRa, MRb

MF32TOI16 MRa, MRb *Convert 32-Bit Floating-Point Value to 16-Bit Integer*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1110 0000
```

Description

Convert a 32-bit floating point value in MRb to a 16-bit integer and truncate. The result will be stored in MRa.

```
MRa(15:0) = F32TOI16(MRb);
MRa(31:16) = sign extension of MRa(15);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #5.0   ; MR0        = 5.0 (0x40A00000)
MF32TOI16   MR1, MR0    ; MR1(15:0)  = MF32TOI16(MR0) = 0x0005
                        ; MR1(31:16) = Sign extension of MR1(15) = 0x0000
MMOVIZ      MR2, #-5.0  ; MR2        = -5.0 (0xC0A00000)
MF32TOI16   MR3, MR2    ; MR3(15:0)  = MF32TOI16(MR2) = -5 (0xFFFB)
                        ; MR3(31:16) = Sign extension of MR3(15) = 0xFFFF
```

See also

MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOI16R MRa, MRb *Convert 32-Bit Floating-Point Value to 16-Bit Integer and Round*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0110 0000
```

Description

Convert the 32-bit floating point value in MRb to a 16-bit integer and round to the nearest even value. The result is stored in MRa.

```
MRa(15:0) = F32TOI16round(MRb);
MRa(31:16) = sign extension of MRa(15);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ MR0, #0x3FD9    ; MR0(31:16) = 0x3FD9
MMOVXI MR0, #0x999A    ; MR0(15:0) = 0x999A
                       ; MR0 = 1.7 (0x3FD9999A)
MF32TOI16R MR1, MR0    ; MR1(15:0) = MF32TOI16round (MR0) = 2 (0x0002)
                       ; MR1(31:16) = Sign extension of MR1(15) = 0x0000
MMOVF32 MR2, #-1.7     ; MR2 = -1.7 (0xBFD9999A)
MF32TOI16R MR3, MR2    ; MR3(15:0) = MF32TOI16round (MR2) = -2 (0xFFFE)
                       ; MR3(31:16) = Sign extension of MR2(15) = 0xFFFF
```

See also

MF32TOI16 MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOI32 MRa, MRb *Convert 32-Bit Floating-Point Value to 32-Bit Integer*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0110 0000
```

Description

Convert the 32-bit floating-point value in MRb to a 32-bit integer value and truncate. Store the result in MRa.

```
MRa = F32TOI32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example 1

```
MMOVF32    MR2, #11204005.0   ; MR2 = 11204005.0 (0x4B2AF5A5)
MF32TOI32  MR3, MR2           ; MR3 = MF32TOI32(MR2) = 11204005 (0x00AAF5A5)
MMOVF32    MR0, #-11204005.0  ; MR0 = -11204005.0 (0xCB2AF5A5)
MF32TOI32  MR1, MR0           ; MR1 = MF32TOI32(MR0) = -11204005 (0xFF550A5B)
```

Example 2

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
; Convert M, X and B from IQ24 to float
;
_Cla1Task2:
   MI32TOF32 MR0, @_M          ; MR0 = 0x4BC00000
   MI32TOF32 MR1, @_X          ; MR1 = 0x4C200000
   MI32TOF32 MR2, @_B          ; MR2 = 0xCB000000
   MMPYF32   MR0, MR0, #0x3380 ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
   MMPYF32   MR1, MR1, #0x3380 ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
   MMPYF32   MR2, MR2, #0x3380 ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
   MMPYF32   MR3, MR0, MR1     ; M*X
   MADDF32   MR2, MR2, MR3     ; Y=MX+B = 3.25 (0x40500000)

; Convert Y from float32 to IQ24
   MMPYF32 MR2, MR2, #0x4B80   ; Y * 1*2^24
   MF32TOI32 MR2, MR2          ; IQ24(Y) = 0x03400000
   MMOV32 @_Y, MR2             ; store result
   MSTOP                       ; end of task
```

See also

MF32TOUI32 MRa, MRb
MI32TOF32 MRa, MRb
MI32TOF32 MRa, mem32
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MF32TOUI16 MRa, MRb *Convert 32-Bit Floating-Point Value to 16-bit Unsigned Integer*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1010 0000
```

Description

Convert the 32-bit floating point value in MRb to an unsigned 16-bit integer value and truncate to zero. The result will be stored in MRa. To instead round the integer to the nearest even value use the MF32TOUI16R instruction.

```
MRa(15:0) = F32TOUI16(MRb);
MRa(31:16) = 0x0000;
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #9.0    ; MR0 = 9.0 (0x41100000)
MF32TOUI16  MR1, MR0     ; MR1(15:0) = MF32TOUI16(MR0) = 9 (0x0009)
                         ; MR1(31:16) = 0x0000
MMOVIZ      MR2, #-9.0   ; MR2 = -9.0 (0xC1100000)
MF32TOUI16  MR3, MR2     ; MR3(15:0) = MF32TOUI16(MR2) = 0 (0x0000)
                         ; MR3(31:16) = 0x0000
```

See also

MF32TOI16 MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOUI16R MRa, MRb *Convert 32-Bit Floating-Point Value to 16-bit Unsigned Integer and Round*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1100 0000
```

Description

Convert the 32-bit floating-point value in MRb to an unsigned 16-bit integer and round to the closest even value. The result will be stored in MRa. To instead truncate the converted value, use the MF32TOUI16 instruction.

```
MRa(15:0)  = MF32TOUI16round(MRb);
MRa(31:16) = 0x0000;
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ       MR0, #0x412C    ; MR0 = 0x412C
MMOVXI       MR0, #0xCCCD    ; MR0 = 0xCCCD ; MR0 = 10.8 (0x412CCCCD)
MF32TOUI16R  MR1, MR0        ; MR1(15:0) = MF32TOUI16round(MR0) = 11 (0x000B)
                             ; MR1(31:16) = 0x0000
MMOVF32      MR2, #-10.8     ; MR2 = -10.8 (0x0xC12CCCCD)
MF32TOUI16R  MR3, MR2        ; MR3(15:0) = MF32TOUI16round(MR2) = 0 (0x0000)
                             ; MR3(31:16) = 0x0000
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOUI32 MRa, MRb *Convert 32-Bit Floating-Point Value to 32-Bit Unsigned Integer*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1010 0000
```

Description

Convert the 32-bit floating-point value in MRb to an unsigned 32-bit integer and store the result in MRa.

```
MRa = F32TOUI32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #12.5   ; MR0 = 12.5 (0x41480000)
MF32TOUI32  MR0, MR0     ; MR0 = MF32TOUI32 (MR0) = 12 (0x0000000C)
MMOVIZ      MR1, #-6.5   ; MR1 = -6.5 (0xC0D00000)
MF32TOUI32  MR2, MR1     ; MR2 = MF32TOUI32 (MR1) = 0.0 (0x00000000)
```

See also

MF32TOI32 MRa, MRb
MI32TOF32 MRa, MRb
MI32TOF32 MRa, mem32
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MFRACF32 MRa, MRb *Fractional Portion of a 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0000 0000
```

Description

Returns in MRa the fractional portion of the 32-bit floating-point value in MRb

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ    MR2, #19.625 ; MR2 = 19.625 (0x419D0000)
MFRACF32  MR3, MR2     ; MR3 = MFRACF32(MR2) = 0.625 (0x3F200000)0)
```

See also

MI16TOF32 MRa, MRb *Convert 16-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1000 0000
```

Description

Convert the 16-bit signed integer in MRb to a 32-bit floating point value and store the result in MRa.

```
MRa = MI16TOF32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #0x0000    ; MR0(31:16) = 0.0 (0x0000)
MMOVXI      MR0, #0x0004    ; MR0(15:0) = 4.0 (0x0004)
MI16TOF32   MR1, MR0        ; MR1 = MI16TOF32 (MR0) = 4.0 (0x40800000)

MMOVIZ      MR2, #0x0000    ; MR2(31:16) = 0.0 (0x0000)
MMOVXI      MR2, #0xFFFC    ; MR2(15:0) = -4.0 (0xFFFC)
MI16TOF32   MR3, MR2        ; MR3 = MI16TOF32 (MR2) = -4.0 (0xC0800000)
MSTOP
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MI16TOF32 MRa, mem16 *Convert 16-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem16	16-bit source memory location to be converted

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 00aa addr
```

Description

Convert the 16-bit signed integer indicated by the mem16 pointer to a 32-bit floating-point value and store the result in MRa.

```
MRa = MI16TOF32[mem16];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction:

Example

```
; Assume A = 4 (0x0004)
;        B = -4 (0xFFFC)

   MI16TOF32 MR0, @_A ; MR0 = MI16TOF32(A) = 4.0 (0x40800000)
   MI16TOF32 MR1, @_B ; MR1 = MI16TOF32(B) = -4.0 (0xC0800000
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MI32TOF32 MRa, mem32 *Convert 32-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem32	32-bit memory source for the MMOV32 operation.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 01aa addr
```

Description

Convert the 32-bit signed integer indicated by mem32 to a 32-bit floating point value and store the result in MRa.

```
MRa = MI32TOF32[mem32];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
; Convert M, X and B from IQ24 to float
;

_Cla1Task3:
   MI32TOF32 MR0, @_M         ; MR0 = 0x4BC00000
   MI32TOF32 MR1, @_X         ; MR1 = 0x4C200000
   MI32TOF32 MR2, @_B         ; MR2 = 0xCB000000
   MMPYF32 MR0, MR0, #0x3380  ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
   MMPYF32 MR1, MR1, #0x3380  ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
   MMPYF32 MR2, MR2, #0x3380  ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
   MMPYF32 MR3, MR0, MR1      ; M*X
   MADDF32 MR2, MR2, MR3      ; Y=MX+B = 3.25 (0x40500000)

; Convert Y from float32 to IQ24
   MMPYF32 MR2, MR2, #0x4B80  ; Y * 1*2^24
   MF32TOI32 MR2, MR2         ; IQ24(Y) = 0x03400000
   MMOV32 @_Y, MR2            ; store result
   MSTOP                      ; end of task
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, MRb
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MI32TOF32 MRa, MRb *Convert 32-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1000 0000
```

Description

Convert the signed 32-bit integer in MRb to a 32-bit floating-point value and store the result in MRa.

```
MRa = MI32TOF32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Example1:
;
   MMOVIZ    MR2, #0x1111 ; MR2(31:16) = 4369 (0x1111)
   MMOVXI    MR2, #0x1111 ; MR2(15:0) = 4369 (0x1111)
                          ; MR2 = +286331153 (0x11111111)
   MI32TOF32 MR3, MR2     ; MR3 = MI32TOF32 (MR2) = 286331153.0 (0x4D888888)
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, mem32
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MLSL32 MRa, #SHIFT *Logical Shift Left*

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#SHIFT	Number of bits to shift (1 to 32)

Opcode

```
LSW: 0000 0000 0shi ftaa
MSW: 0111 1011 1100 0000
```

Description

Logical shift left of MRa by the number of bits indicated. The number of bits can be 1 to 32.

```
MARa(31:0) = Logical Shift Left(MARa(31:0) by #SHIFT bits);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given m2 = (int32)32
;       x2 = (int32)64
;       b2 = (int32)-128
;
; Calculate:
;       m2 = m2*2
;       x2 = x2*4
;       b2 = b2*8
;
_Cla1Task3:
   MMOV32 MR0, @_m2   ; MR0 = 32 (0x00000020)
   MMOV32 MR1, @_x2   ; MR1 = 64 (0x00000040)
   MMOV32 MR2, @_b2   ; MR2 = -128 (0xFFFFFF80)
   MLSL32 MR0, #1     ; MR0 = 64 (0x00000040)
   MLSL32 MR1, #2     ; MR1 = 256 (0x00000100)
   MLSL32 MR2, #3     ; MR2 = -1024 (0xFFFFFC00)
   MMOV32 @_m2, MR0   ; Store results
   MMOV32 @_x2, MR1
   MMOV32 @_b2, MR2
   MSTOP              ; end of task
```

See also

MADD32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MAND32 MRa, MRb, MRc
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MLSR32 MRa, #SHIFT *Logical Shift Right*

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#SHIFT	Number of bits to shift (1 to 32)

Opcode

```
LSW: 0000 0000 0shi ftaa
MSW: 0111 1011 1000 0000
```

Description

Logical shift right of MRa by the number of bits indicated. The number of bits can be 1 to 32. Unlike the arithmetic shift (MASR32), the logical shift does not preserve the number's sign bit. Every bit in the operand is moved the specified number of bit positions, and the vacant bit-positions are filled in with zeros

```
MARa(31:0) = Logical Shift Right(MARa(31:0) by #SHIFT bits);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
; Illustrate the difference between MASR32 and MLSR32

MMOVIZ MR0, #0xAAAA   ; MR0 = 0xAAAA5555
MMOVXI MR0, #0x5555

MMOV32 MR1, MR0       ; MR1 = 0xAAAA5555
MMOV32 MR2, MR0       ; MR2 = 0xAAAA5555

MASR32 MR1, #1        ; MR1 = 0xD5552AAA
MLSR32 MR2, #1        ; MR2 = 0x55552AAA

MASR32 MR1, #1        ; MR1 = 0xEAAA9555
MLSR32 MR2, #1        ; MR2 = 0x2AAA9555

MASR32 MR1, #6        ; MR1 = 0xFFAAAA55
MLSR32 MR2, #6        ; MR2 = 0x00AAAA55
```

See also

MADD32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MAND32 MRa, MRb, MRc
MLSL32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MMACF32 MR3, MR2, MRd, MRe, MRf ||MMOV32 MRa, mem32 *32-Bit Floating-Point Multiply and Accumulate with Parallel Move*

Operands

MR3	floating-point destination/source register MR3 for the add operation
MR2	CLA floating-point source register MR2 for the add operation
MRd	CLA floating-point destination register (MR0 to MR3) for the multiply operation MRd cannot be the same register as MRa
MRe	CLA floating-point source register (MR0 to MR3) for the multiply operation
MRf	CLA floating-point source register (MR0 to MR3) for the multiply operation
MRa	CLA floating-point destination register for the MMOV32 operation (MR0 to MR3). MRa cannot be MR3 or the same register as MRd.
mem32	32-bit source for the MMOV32 operation

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0011 ffee ddaa addr
```

Description

Multiply and accumulate the contents of floating-point registers and move from register to memory. The destination register for the MMOV32 cannot be the same as the destination registers for the MMACF32.

```
MR3 = MR3 + MR2;
MRd = MRe * MRf;
MRa = [mem32];
```

Restrictions

The destination registers for the MMACF32 and the MMOV32 must be unique. That is, MRa cannot be MR3 and MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMACF32 (add or multiply) generates an underflow condition.
- LVF = 1 if MMACF32 (add or multiply) generates an overflow condition.

MMOV32 sets the NF and ZF flags as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
```

Pipeline

MMACF32 and MMOV32 complete in a single cycle.

Example 1

```
; Perform 5 multiply and accumulate operations:
;
; X and Y are 32-bit floating point arrays
;
; 1st multiply: A = X0 * Y0
; 2nd multiply: B = X1 * Y1
; 3rd multiply: C = X2 * Y2
; 4th multiply: D = X3 * Y3
; 5th multiply: E = X3 * Y3
;
; Result = A + B + C + D + E
;
_Cla1Task1:
   MMOVI16 MAR0, #_X                ; MAR0 points to X array
   MMOVI16 MAR1, #_Y                ; MAR1 points to Y array
   MNOP                             ; Delay for MAR0, MAR1 load
   MNOP                             ; Delay for MAR0, MAR1 load
                                    ; <-- MAR0 valid
   MMOV32 MR0, *MAR0[2]++           ; MR0 = X0, MAR0 += 2
                                    ; <-- MAR1 valid
   MMOV32 MR1, *MAR1[2]++           ; MR1 = Y0, MAR1 += 2

   MMPYF32 MR2, MR0, MR1            ; MR2 = A = X0 * Y0
|| MMOV32 MR0, *MAR0[2]++           ; In parallel MR0 = X1, MAR0 += 2
   MMOV32 MR1, *MAR1[2]++           ; MR1 = Y1, MAR1 += 2

   MMPYF32 MR3, MR0, MR1            ; MR3 = B = X1 * Y1
|| MMOV32 MR0, *MAR0[2]++           ; In parallel MR0 = X2, MAR0 += 2
   MMOV32 MR1, *MAR1[2]++           ; MR1 = Y2, MAR2 += 2

   MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2
|| MMOV32 MR0, *MAR0[2]++           ; In parallel MR0 = X3
   MMOV32 MR1, *MAR1[2]++           ; MR1 = Y3 M

   MACF32 MR3, MR2, MR2, MR0, MR1  ; MR3 = (A + B) + C, MR2 = D = X3 * Y3
|| MMOV32 MR0, *MAR0                ; In parallel MR0 = X4
   MMOV32 MR1, *MAR1                ; MR1 = Y4

   MMPYF32 MR2, MR0, MR1            ; MR2 = E = X4 * Y4
|| MADDF32 MR3, MR3, MR2            ; in parallel MR3 = (A + B + C) + D

   MADDF32 MR3, MR3, MR2            ; MR3 = (A + B + C + D) + E
   MMOV32 @_Result, MR3             ; Store the result
   MSTOP                            ; end of task
```

Example 2

```
; sum = X0*B0 + X1*B1 + X2*B2 + Y1*A1 + Y2*B2
;
;       X2 = X1
;       X1 = X0
;       Y2 = Y1 ; Y1 = sum
;
_ClaTask2:
     MMOV32     MR0, @_B2      ; MR0 = B2
     MMOV32     MR1, @_X2      ; MR1 = X2
     MMPYF32    MR2, MR1, MR0  ; MR2 = X2*B2
  || MMOV32     MR0, @_B1      ; MR0 = B1
     MMOVD32    MR1, @_X1      ; MR1 = X1, X2 = X1
     MMPYF32    MR3, MR1, MR0  ; MR3 = X1*B1
  || MMOV32     MR0, @_B0      ; MR0 = B0
     MMOVD32    MR1, @_X0      ; MR1 = X0, X1 = X0

;  MR3 = X1*B1 + X2*B2, MR2 = X0*B0
;  MR0 = A2
     MMACF32 MR3, MR2, MR2, MR1, MR0
  || MMOV32 MR0, @_A2 M

     MOV32 MR1, @_Y2           ; MR1 = Y2

;  MR3 = X0*B0 + X1*B1 + X2*B2, MR2 = Y2*A2
;  MR0 = A1
     MMACF32 MR3, MR2, MR2, MR1, MR0
  || MMOV32 MR0, @_A1

     MMOVD32 MR1,@_Y1          ; MR1 = Y1, Y2 = Y1
     MADDF32 MR3, MR3, MR2     ; MR3 = Y2*A2 + X0*B0 + X1*B1 + X2*B2
  || MMPYF32 MR2, MR1, MR0     ; MR2 = Y1*A1
     MADDF32 MR3, MR3, MR2     ; MR3 = Y1*A1 + Y2*A2 + X0*B0 + X1*B1 + X2*B2
     MMOV32 @_Y1, MR3          ; Y1 = MR3
     MSTOP                     ; end of task
```

See also MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MMAXF32 MRa, MRb *32-Bit Floating-Point Maximum*

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0010 0000
```

Description

```
if(MRa < MRb) MRa = MRb;
```

Special cases for the output from the MMAXF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == MRb)  {ZF=1; NF=0;}
if(MRa > MRb) {ZF=0; NF=0;}
if(MRa < MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example 1

```
MMOVIZ   MR0, #5.0  ; MR0 =  5.0  (0x40A00000)
MMOVIZ   MR1, #-2.0 ; MR1 = -2.0  (0xC0000000)
MMOVIZ   MR2, #-1.5 ; MR2 = -1.5  (0xBFC00000)
MMAXF32   MR2, MR1  ; MR2 = -1.5, ZF = NF = 0
MMAXF32   MR1, MR2  ; MR1 = -1.5, ZF = 0, NF = 1
MMAXF32   MR2, MR0  ; MR2 =  5.0, ZF = 0, NF = 1
MAXF32   MR0, MR2   ; MR2 =  5.0, ZF = 1, NF = 0
```

Example 2

```
; X is an array of 32-bit floating-point values
; Find the maximum value in an array X
; and store it in Result
;
_Cla1Task1:
  MMOVI16    MAR1,#_X          ; Start address
  MUI16TOF32 MR0, @_len        ; Length of the array
  MNOP                         ; delay for MAR1 load
  MNOP                         ; delay for MAR1 load
  MMOV32     MR1, *MAR1[2]++   ; MR1 = X0
LOOP
  MMOV32     MR2, *MAR1[2]++   ; MR2 = next element
  MMAXF32    MR1, MR2          ; MR1 = MAX(MR1, MR2)
  MADDF32    MR0, MR0, #-1.0   ; Decrememt the counter
  MCMPF32    MR0 #0.0          ; Set/clear flags for MBCNDD
  MNOP
  MNOP
  MNOP
  MBCNDD     LOOP, NEQ         ; Branch if not equal to zero
  MMOV32     @_Result, MR1     ; Always executed
  MNOP                         ; Always executed
  MNOP                         ; Always executed
  MSTOP                        ; End of task
```

See also

MCMPF32 MRa, MRb
MCMPF32 MRa, #16FHi
MMAXF32 MRa, #16FHi
MMINF32 MRa, MRb

MMINF32 MRa, #16FHi

MMAXF32 MRa, #16FHi *32-Bit Floating-Point Maximum*

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1001 0000 00aa
```

Description

Compare MRa with the floating-point value represented by the immediate operand. If the immediate value is larger, then load it into MRa.

```
if(MRa < #16FHi:0) MRa = #16FHi:0;
```

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

Special cases for the output from the MMAXF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == #16FHi:0)  {ZF=1; NF=0;}
if(MRa > #16FHi:0) {ZF=0; NF=0;}
if(MRa < #16FHi:0) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ    MR0, #5.0  ; MR0 =  5.0  (0x40A00000)
MMOVIZ    MR1, #4.0  ; MR1 =  4.0  (0x40800000)
MMOVIZ    MR2, #-1.5 ; MR2 = -1.5  (0xBFC00000)
MMAXF32   MR0, #5.5  ; MR0 =  5.5, ZF = 0, NF = 1
MMAXF32   MR1, #2.5  ; MR1 =  4.0, ZF = 0, NF = 0
MMAXF32   MR2, #-1.0 ; MR2 = -1.0, ZF = 0, NF = 1
MMAXF32   MR2, #-1.0 ; MR2 = -1.5, ZF = 1, NF = 0
```

See also

MMAXF32 MRa, MRb
MMINF32 MRa, MRb
MMINF32 MRa, #16FHi

MMINF32 MRa, MRb *32-Bit Floating-Point Minimum*

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0100 0000
```

Description

```
if(MRa > MRb) MRa = MRb;
```

Special cases for the output from the MMINF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == MRb)  {ZF=1; NF=0;}
if(MRa > MRb) {ZF=0; NF=0;}
if(MRa < MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example 1

```
MMOVIZ MR0, #5.0  ; MR0 = 5.0 (0x40A00000)
MMOVIZ MR1, #4.0  ; MR1 = 4.0 (0x40800000)
MMOVIZ MR2, #-1.5 ; MR2 = -1.5 (0xBFC00000)
MMINF32 MR0, MR1  ; MR0 = 4.0, ZF = 0, NF = 0
MMINF32 MR1, MR2  ; MR1 = -1.5, ZF = 0, NF = 0
MMINF32 MR2, MR1  ; MR2 = -1.5, ZF = 1, NF = 0
MMINF32 MR1, MR0  ; MR2 = -1.5, ZF = 0, NF = 1
```

Example 2

```
;
; X is an array of 32-bit floating-point values
; Find the minimum value in an array X
; and store it in Result
;

_Cla1Task1:
MMOVI16    MAR1,#_X         ; Start address
MUI16TOF32 MR0, @_len       ; Length of the array
MNOP                        ; delay for MAR1 load
MNOP                        ; delay for MAR1 load
MMOV32     MR1, *MAR1[2]++  ; MR1 = X0
LOOP
MMOV32     MR2, *MAR1[2]++  ; MR2 = next element
MMINF32    MR1, MR2         ; MR1 = MAX(MR1, MR2)
MADDF32    MR0, MR0, #-1.0  ; Decrememt the counter
MCMPF32    MR0 #0.0         ; Set/clear flags for MBCNDD
MNOP
MNOP
MNOP
MBCNDD     LOOP, NEQ        ; Branch if not equal to zero
MMOV32     @_Result, MR1    ; Always executed
MNOP                        ; Always executed
MNOP                        ; Always executed
MSTOP                       ; End of task
```

See also

MMAXF32 MRa, MRb
MMAXF32 MRa, #16FHi

MMINF32 MRa, #16FHi

MMINF32 MRa, #16FHi *32-Bit Floating-Point Minimum*

Operands

MRa	floating-point source/destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1001 0100 00aa
```

Description

Compare MRa with the floating-point value represented by the immediate operand. If the immidate value is smaller, then load it into MRa.

```
if(MRa > #16FHi:0) MRa = #16FHi:0;
```

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

Special cases for the output from the MMINF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == #16FHi:0)   {ZF=1; NF=0;}
if(MRa > #16FHi:0) {ZF=0; NF=0;}
if(MRa < #16FHi:0) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ   MR0, #5.0   ; MR0 =  5.0  (0x40A00000)
MMOVIZ   MR1, #4.0   ; MR1 =  4.0  (0x40800000)
MMOVIZ   MR2, #-1.5  ; MR2 = -1.5  (0xBFC00000)
MMINF32  MR0, #5.5   ; MR0 =  5.0, ZF = 0, NF = 1
MMINF32  MR1, #2.5   ; MR1 =  2.5, ZF = 0, NF = 0
MMINF32  MR2, #-1.0  ; MR2 = -1.5, ZF = 0, NF = 1
MMINF32  MR2, #-1.5  ; MR2 = -1.5, ZF = 1, NF = 0
```

See also

MMAXF32 MRa, #16FHi
MMAXF32 MRa, MRb
MMINF32 MRa, MRb

MMOV16 MARx, MRa, #16I *Load the Auxiliary Register with MRa + 16-bit Immediate Value*

Operands

MARx	Auxiliary register MAR0 or MAR1
MRa	CLA Floating-point register (MR0 to MR3)
#16I	16-bit immediate value

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOV16 MAR0, MRa, #16I)
MSW: 0111 1111 1101 00AA

LSW: IIII IIII IIII IIII (opcode of MMOV16 MAR1, MRa, #16I)
MSW: 0111 1111 1111 00AA
```

Description

Load the auxiliary register, MAR0 or MAR1, with MRa(15:0) + 16-bit immediate value. Refer to the pipeline section for important information regarding this instruction.

```
MARx = MRa(15:0) + #16I;
```

Flags

This instruction does not modify flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction. The load of MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Therefore the following applies when loading the auxiliary registers:

- **I1 and I2**

  The two instructions following MMOV16 will use MAR0/MAR1 before the update occurs. Thus these two instructions will use the old value of MAR0 or MAR1.
- **I3**

  Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win and the auxiliary register will not be updated with #_X.
- **I4**

  Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOVI16.

```
; Assume MAR0 is 50, MR0 is 10, and #_X is 20

  MMOV16 MAR0, MR0, #_X        ; Load MAR0 with address of X (20) + MR0 (10)
  <Instruction 1>       ; I1 Will use the old value of MAR0 (50)
  <Instruction 2>       ; I2 Will use the old value of MAR0 (50)
  <Instruction 3>       ; I3 Cannot use MAR0
  <Instruction 4>       ; I4 Will use the new value of MAR0 (30)
  <Instruction 5>       ; I5
```

Table 9-28. Pipeline Activity For MMOV16 MARx, MRa , #16I

Instruction	F1	F2	D1	D2	R1	R2	E	W
MMOV16 MAR0, MR0, #_X	MMOV16							
I1	I1	MMOV16						
I2	I2	I1	MMOV16					
I3	I3	I2	I1	MMOV16				
I4	I4	I3	I2	I1	MMOV16			
I5	I5	I4	I3	I2	I1	MMOV16		
I6	I6	I5	I4	I3	I2	I1	MMOV16	

Example 1

```
; Calculate an offset into a sin/cos table
;
_Cla1Task1:
    MMOV32 MR0,@_rad                 ; MR0 = rad
    MMOV32 MR1,@_TABLE_SIZEDivTwoPi ; MR1 = TABLE_SIZE/(2*Pi)
    MMPYF32 MR1,MR0,MR1              ; MR1 = rad* TABLE_SIZE/(2*Pi)
||  MMOV32 MR2,@_TABLE_MASK          ; MR2 = TABLE_MASK
    MF32TOI32 MR3,MR1                ; MR3 = K=int(rad*TABLE_SIZE/(2*Pi))
    MAND32 MR3,MR3,MR2               ; MR3 = K & TABLE_MASK
    MLSL32 MR3,#1                    ; MR3 = K * 2

    MMOV16 MAR0,MR3,#_Cos0           ; MAR0 K*2+addr of table.Cos0
    MFRACF32 MR1,MR1                 ; I1
    MMOV32 MR0,@_TwoPiDivTABLE_SIZE ; I2
    MMPYF32 MR1,MR1,MR0              ; I3
||  MMOV32 MR0,@_Coef3

    MMOV32 MR2,*MAR0[#-64]++         ; MR2 = *MAR0, MAR0 += (-64)
    ...
    ...
    MSTOP ; end of task
```

Example 2

```
; This task logs the last NUM_DATA_POINTS
; ADCRESULT1 values in the array VoltageCLA
;
; When the last element in the array has been
; filled, the task will go back to the
; the first element.
;
; Before starting the ADC conversions, force
; Task 8 to initialize the ConversionCount to zero
;
_Cla1Task2:
   MMOVZ16      MR0, @_ConversionCount      ;I1 Current Conversion
   MMOV16 MAR1, MR0, #_VoltageCLA           ;I2 Next array location
   MUI16TOF32   MR0, MR0                    ;I3 Convert count to float32
   MADDF32      MR0, MR0, #1.0              ;I4 Add 1 to conversion count
   MCMPF32      MR0, #NUM_DATA_POINTS.0     ;I5 Compare count to max
   MF32TOUI16   MR0, MR0                    ;I6 Convert count to Uint16
   MNOP                                     ;I7 Wait till I8 to read result
   MMOVZ16      MR2, @_AdcResult.ADCRESULT1 ;I8 Read ADCRESULT1
   MMOV16       *MAR1, MR2                  ; Store ADCRESULT1
   MBCNDD       _RestartCount, GEQ          ; If count >= NUM_DATA_POINTS
   MMOVIZ       MR1, #0.0                   ; Always executed: MR1=0
   MNOP
   MNOP
   MMOV16       @_ConversionCount, MR0      ; If branch not taken
   MSTOP                                    ; store current count
_RestartCount
   MMOV16       @_ConversionCount, MR1      ; If branch taken, restart count
   MSTOP                                    ; end of task

; This task initializes the ConversionCount
; to zero
;
_Cla1Task8:
   MMOVIZ MR0, #0.0
   MMOV16 @_ConversionCount, MR0
   MSTOP
_ClaT8End:
```

See also

MMOV16 MARx, mem16 *Load MAR1 with 16-bit Value*

Operands

MARx	CLA auxiliary register MAR0 or MAR1
mem16	16-bit destination memory accessed using indirect or direct addressing modes

Opcode

```
LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 MAR0, mem16)
MSW: 0111 0110 0000 addr

LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 MAR1, mem16)
MSW: 0111 0110 0100 addr
```

Description

Load MAR0 or MAR1 with the 16-bit value pointed to by mem16. Refer to the pipeline section for important information regarding this instruction.

```
MAR1 = [mem16];
```

Flags

No flags MSTF flags are affected.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction. The load of MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Therefore the following applies when loading the auxiliary registers:

- **I1 and I2**

  The two instructions following MMOV16 will use MAR0/MAR1 before the update occurs. Thus these two instructions will use the old value of MAR0 or MAR1.
- **I3**

  Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win snd the auxiliary register will not be updated with #_X.
- **I4**

  Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOV16.

```
; Assume MAR0 is 50 and @_X is 20

MMOV16 MAR0, @_X          ; Load MAR0 with the contents of X (20)
<Instruction 1>     ; I1 Will use the old value of MAR0 (50)
<Instruction 2>     ; I2 Will use the old value of MAR0 (50)
<Instruction 3>     ; I3 Cannot use MAR0
<Instruction 4>     ; I4 Will use the new value of MAR0 (20)
<Instruction 5>     ; I5
....
```

Table 9-29. Pipeline Activity For MMOV16 MAR0/MAR1, mem16

Instruction	F1	F2	D1	D2	R1	R2	E	W
MMOV16 MAR0, @_X	MMOV16							
I1	I1	MMOV16						
I2	I2	I1	MMOV16					
I3	I3	I2	I1	MMOV16				
I4	I4	I3	I2	I1	MMOV16			
I5	I5	I4	I3	I2	I1	MMOV16		
I6	I6	I5	I4	I3	I2	I1	MMOV16	

Example

```
; This task logs the last NUM_DATA_POINTS
; ADCRESULT1 values in the array VoltageCLA
;
; When the last element in the array has been
; filled, the task will go back to the
; the first element.
;
; Before starting the ADC conversions, force
; Task 8 to initialize the ConversionCount to zero
;
_Cla1Task2:
   MMOVZ16      MR0, @_ConversionCount      ;I1 Current Conversion
   MMOV16       MAR1, MR0, #_VoltageCLA     ;I2 Next array location
   MUI16TOF32   MR0, MR0                    ;I3 Convert count to float32
   MADDF32      MR0, MR0, #1.0              ;I4 Add 1 to conversion count
   MCMPF32      MR0, #NUM_DATA_POINTS.0     ;I5 Compare count to max
   MF32TOUI16   MR0, MR0                    ;I6 Convert count to Uint16
   MNOP                                     ;I7 Wait till I8 to read result
   MMOVZ16      MR2, @_AdcResult.ADCRESULT1 ;I8 Read ADCRESULT1
   MMOV16       *MAR1, MR2                  ; Store ADCRESULT1
   MBCNDD       _RestartCount, GEQ          ; If count >= NUM_DATA_POINTS
   MMOVIZ       MR1, #0.0                   ; Always executed: MR1=0
   MNOP
   MNOP
   MMOV16       @_ConversionCount, MR0      ; If branch not taken MSTOP
                                            ; store current count
_RestartCount
   MMOV16       @_ConversionCount, MR1      ; If branch taken, restart count
   MSTOP                                    ; end of task

; This task initializes the ConversionCount
; to zero
;
_Cla1Task8:
   MMOVIZ       MR0, #0.0
   MMOV16       @_ConversionCount, MR0
   MSTOP
_ClaT8End:
```

See also

MMOV16 mem16, MARx *Move 16-Bit Auxiliary Register Contents to Memory*

Operands

mem16	16-bit destination memory accessed using indirect or direct addressing modes
MARx	CLA auxiliary register MAR0 or MAR1

Opcode

```
LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 mem16, MAR0)
MSW: 0111 0110 1000 addr

LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 mem16, MAR1)
MSW: 0111 0110 1100 addr
```

Description

Store the contents of MAR0 or MAR1 in the 16-bit memory location pointed to by mem16.

```
[mem16] = MAR0;
```

Flags

No flags MSTF flags are affected.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

See also

MMOV16 mem16, MRa *Move 16-Bit Floating-Point Register Contents to Memory*

Operands

mem16	16-bit destination memory accessed using indirect or direct addressing modes
MRa	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 11aa addr
```

Description

Move 16-bit value from the lower 16-bits of the floating-point register (MRa(15:0)) to the location pointed to by mem16.

```
[mem16] = MRa(15:0);
```

Flags

No flags MSTF flags are affected.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; This task logs the last NUM_DATA_POINTS
; ADCRESULT1 values in the array VoltageCLA
;
; When the last element in the array has been
; filled, the task will go back to the
; the first element.
;
; Before starting the ADC conversions, force
; Task 8 to initialize the ConversionCount to zero
;
 _Cla1Task2:
   MMOVZ16       MR0, @_ConversionCount      ;I1 Current Conversion
   MMOV16        MAR1, MR0, #_VoltageCLA     ;I2 Next array location
   MUI16TOF32    MR0, MR0                    ;I3 Convert count to float32
   MADDF32       MR0, MR0, #1.0              ;I4 Add 1 to conversion count
   MCMPF32       MR0, #NUM_DATA_POINTS.0     ;I5 Compare count to max
   MF32TOUI16    MR0, MR0                    ;I6 Convert count to Uint16
   MNOP                                      ;I7 Wait till I8 to read result
   MMOVZ16       MR2, @_AdcResult.ADCRESULT1 ;I8 Read ADCRESULT1
   MMOV16        *MAR1, MR2                  ; Store ADCRESULT1
   MBCNDD        _RestartCount, GEQ          ; If count >= NUM_DATA_POINTS
   MMOVIZ        MR1, #0.0                   ; Always executed: MR1=0
   MNOP
   MNOP
   MMOV16        @_ConversionCount, MR0      ; If branch not taken MSTOP
                                             ; store current count
_RestartCount
   MMOV16        @_ConversionCount, MR1      ; If branch taken, restart count
   MSTOP                                     ; end of task

; This task initializes the ConversionCount
; to zero
;
_Cla1Task8:
   MMOVIZ MR0, #0.0
   MMOV16 @_ConversionCount, MR0
   MSTOP
_ClaT8End:
```

See also

MMOVIZ MRa, #16FHi
MMOVXI MRa, #16FLoHex

MMOV32 mem32, MRa *Move 32-Bit Floating-Point Register Contents to Memory*

Operands

MRa	floating-point register (MR0 to MR3)
mem32	32-bit destination memory accessed using indirect or direct addressing modes

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 11aa addr
```

Description

Move from MRa to 32-bit memory location indicated by mem32.

```
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

No flags affected.

Pipeline

This is a single-cycle instruction.

Example

```
; Perform 5 multiply and accumulate operations:
;
; X and Y are 32-bit floating point arrays;
; 1st multiply: A = X0 * Y0
; 2nd multiply: B = X1 * Y1
; 3rd multiply: C = X2 * Y2
; 4th multiply: D = X3 * Y3
; 5th multiply: E = X3 * Y3;
; Result = A + B + C + D + E
;
_Cla1Task1:
   MMOVI16       MAR0, #_X                ; MAR0 points to X array
   MMOVI16       MAR1, #_Y                ; MAR1 points to Y array
   MNOP                                   ; Delay for MAR0, MAR1 load
   MNOP                                   ; Delay for MAR0, MAR1 load
                                          ; <-- MAR0 valid
   MMOV32        MR0, *MAR0[2]++          ; MR0 = X0, MAR0 += 2
                                          ; <-- MAR1 valid
   MMOV32        MR1, *MAR1[2]++          ; MR1 = Y0, MAR1 += 2
   MMPYF32       MR2,  MR0, MR1           ; MR2 = A = X0 * Y0
|| MMOV32        MR0, *MAR0[2]++          ; In parallel MR0 = X1, MAR0 += 2
   MMOV32        MR1, *MAR1[2]++          ; MR1 = Y1, MAR1 += 2
   MMPYF32       MR3, MR0, MR1            ; MR3 = B = X1 * Y1
|| MMOV32        MR0, *MAR0[2]++          ; In parallel MR0 = X2, MAR0 += 2
   MMOV32        MR1, *MAR1[2]++          ; MR1 = Y2, MAR2 += 2

   MMACF32       MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2
|| MMOV32        MR0, *MAR0[2]++          ; In parallel MR0 = X3
   MMOV32        MR1, *MAR1[2]++          ; MR1 = Y3

   MMACF32       MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3
|| MMOV32        MR0, *MAR0               ; In parallel MR0 = X4
   MMOV32        MR1, *MAR1               ; MR1 = Y4
   MMPYF32       MR2, MR0, MR1            ; MR2 = E = X4 * Y4
|| MADDF32       MR3, MR3, MR2            ; in parallel MR3 = (A + B + C) + D
   MADDF32       MR3, MR3, MR2            ; MR3 = (A + B + C + D) + E
   MMOV32        @_Result, MR3            ; Store the result MSTOP ; end of task
```

See also

MMOV32 mem32, MSTF

MMOV32 mem32, MSTF *Move 32-Bit MSTF Register to Memory*

Operands

MSTF	floating-point status register
mem32	32-bit destination memory

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0111 0100 addr
```

Description

Copy the CLA's floating-point status register, MSTF, to memory.

```
[mem32] = MSTF;
```

Flags

This instruction does not modify flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

See also

MMOV32 mem32, MRa

MMOV32 MRa, mem32 {, CNDF} *Conditional 32-Bit Move*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem32	32-bit memory location accessed using direct or indirect addressing
CNDF	optional condition.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 00cn dfaa addr
```

Description

If the condition is true, then move the 32-bit value referenced by mem32 to the floating-point register indicated by MRa.

```
if (CNDF == TRUE) MRa = [mem32];
```

CNDF is one of the following conditions:

Encode [(1)]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [(2)]	Unconditional with flag modification	None

(1) Values not shown are reserved.

(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

```
if(CNDF == UNCF)
{
  NF = MRa(31);
  ZF = 0;
  if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
}
  else No flags modified;
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given A, B, X, M1 and M2 are 32-bit floating-point
; numbers
;
; if(A > B) calculate Y = X*M1
; if(A < B) calculate Y = X*M2
;
_Cla1Task5:
    MMOV32    MR0, @_A
    MMOV32    MR1, @_B
    MCMPF32   MR0, MRB
    MMOV32    MR2, @_M1, EQ  ; if A > B, MR2 = M1
                             ;     Y = M1*X
    MMOV32    MR2, @_M2, NEQ ; if A < B, MR2 = M2
                             ;     Y = M2*X
    MMOV32    MR3, @_X
    MMPYF32   MR3, MR2, MR3  ; Calculate Y
    MMOV32    @_Y, MR3       ; Store Y
    MSTOP                    ; end of task
```

See also

MMOV32 MRa, MRb {, CNDF}
MMOVD32 MRa, mem32

MMOV32 MRa, MRb {, CNDF} *Conditional 32-Bit Move*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
CNDF	optional condition.

Opcode

```
LSW: 0000 0000 cndf bbaa
MSW: 0111 1010 1100 0000
```

Description

If the condition is true, then move the 32-bit value in MRb to the floating-point register indicated by MRa.

```
if (CNDF == TRUE) MRa = MRb;
```

CNDF is one of the following conditions:

Encode [3]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [4]	Unconditional with flag modification	None

[3] Values not shown are reserved.
[4] This is the default operation if no CNDF field is specified. This condition will allow the ZF, and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

```
if(CNDF == UNCF)
 {
   NF = MRa(31); ZF = 0;
   if(MRa(30:23) == 0) {ZF = 1; NF = 0;}
}
else No flags modified;
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given: X = 8.0
;        Y = 7.0
;        A = 2.0
;        B = 5.0
; _ClaTask1
    MMOV32   MR3, @_X      ; MR3 = X = 8.0
    MMOV32   MR0, @_Y      ; MR0 = Y = 7.0
    MMAXF32  MR3, MR0      ; ZF = 0, NF = 0, MR3 = 8.0
    MMOV32   MR1, @_A, GT  ; true, MR1 = A = 2.0
    MMOV32   MR1, @_B, LT  ; false, does not load MR1
    MMOV32   MR2, MR1, GT  ; true, MR2 = MR1 = 2.0
    MMOV32   MR2, MR0, LT  ; false, does not load MR2
    MSTOP
```

See also MMOV32 MRa, mem32 {,CNDF}

MMOV32 MSTF, mem32 *Move 32-Bit Value from Memory to the MSTF Register*

Operands

MSTF	CLA status register
mem32	32-bit source memory location

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0111 0000 addr
```

Description

Move from memory to the CLA's status register MSTF. This instruction is most useful when nesting function calls (via MCCNDD).

```
MSTF = [mem32];
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	Yes	Yes	Yes	Yes	Yes

Loading the status register will overwrite all flags and the RPC field. The MEALLOW field is not affected.

Pipeline

This is a single-cycle instruction.

Example

See also

MMOV32 mem32, MSTF

MMOVD32 MRa, mem32 *Move 32-Bit Value from Memory with Data Copy*

Operands

MRa	CLA floating-point register (MR0 to MR3)
mem32	32-bit memory location accessed using direct or indirect addressing

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 00aa addr
```

Description

Move the 32-bit value referenced by mem32 to the floating-point register indicated by MRa.

```
MRa = [mem32];
[mem32+2] = [mem32];
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0){ ZF = 1; NF = 0; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; sum = X0*B0 + X1*B1 + X2*B2 + Y1*A1 + Y2*B2
;
;       X2 = X1
;       X1 = X0
;       Y2 = Y1
;       Y1 = sum
;
_Cla1Task2:
    MMOV32 MR0, @_B2       ; MR0 = B2
    MMOV32 MR1, @_X2       ; MR1 = X2
    MMPYF32 MR2, MR1, MR0  ; MR2 = X2*B2
||  MMOV32 MR0, @_B1       ; MR0 = B1
    MMOVD32 MR1, @_X1      ; MR1 = X1, X2 = X1
    MMPYF32 MR3, MR1, MR0  ; MR3 = X1*B1
||  MMOV32 MR0, @_B0       ; MR0 = B0
    MMOVD32 MR1, @_X0      ; MR1 = X0, X1 = X0

; MR3 = X1*B1 + X2*B2, MR2 = X0*B0
; MR0 = A2
    MMACF32 MR3, MR2, MR2, MR1, MR0
||  MMOV32 MR0, @_A2

    MMOV32 MR1, @_Y2       ; MR1 = Y2

; MR3 = X0*B0 + X1*B1 + X2*B2, MR2 = Y2*A2
; MR0 = A1
    MMACF32 MR3, MR2, MR2, MR1, MR0
||  MMOV32 MR0, @_A1

    MMOVD32 MR1,@_Y1       ; MR1 = Y1, Y2 = Y1
    MADDF32 MR3, MR3, MR2  ; MR3 = Y2*A2 + X0*B0 + X1*B1 + X2*B2
||  MMPYF32 MR2, MR1, MR0  ; MR2 = Y1*A1
    MADDF32 MR3, MR3, MR2  ; MR3 = Y1*A1 + Y2*A2 + X0*B0 + X1*B1 + X2*B2
    MMOV32 @_Y1, MR3       ; Y1 = MR3
    MSTOP                  ; end of task
```

See also

MMOV32 MRa, mem32 {,CNDF}

MMOVF32 MRa, #32F *Load the 32-Bits of a 32-Bit Floating-Point Register*

Operands

This instruction is an alias for MMOVIZ and MMOVXI instructions. The second operand is translated by the assembler such that the instruction becomes:

`MMOVIZ MRa, #16FHiHex MMOVXI MRa, #16FLoHex`

MRa	CLA floating-point destination register (MR0 to MR3)
#32F	immediate float value represented in floating-point representation

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOVIZ MRa, #16FHiHex)
MSW: 0111 1000 0100 00aa
LSW: IIII IIII IIII IIII (opcode of MMOVXI MRa, #16FLoHex)
MSW: 0111 1000 1000 00aa
```

Description

Note: This instruction accepts the immediate operand only in floating-point representation. To specify the immediate value as a hex value (IEEE 32-bit floating-point format) use the MOVI32 MRa, #32FHex instruction.

Load the 32-bits of MRa with the immediate float value represented by #32F.

#32F is a float value represented in floating-point representation. The assembler will only accept a float value represented in floating-point representation. That is, 3.0 can only be represented as #3.0. #0x40400000 will result in an error.

```
MRa = #32F;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

Depending on #32FH, this instruction takes one or two cycles. If all of the lower 16-bits of the IEEE 32-bit floating-point format of #32F are zeros, then the assembler will convert MMOVF32 into only MMOVIZ instruction. If the lower 16-bits of the IEEE 32-bit floating-point format of #32F are not zeros, then the assembler will convert MMOVF32 into MMOVIZ and MMOVXI instructions.

Example

```
MMOVF32 MR1, #3.0    ; MR1 = 3.0 (0x40400000)
                     ; Assembler converts this instruction as
                     ; MMOVIZ MR1, #0x4040

MMOVF32 MR2, #0.0    ; MR2 = 0.0 (0x00000000)
                     ; Assembler converts this instruction as
                     ; MMOVIZ MR2, #0x0

MMOVF32 MR3, #12.265 ; MR3 = 12.625 (0x41443D71)
                     ; Assembler converts this instruction as
                     ; MMOVIZ MR3, #0x4144
                     ; MMOVXI MR3, #0x3D71
```

See also

MMOVIZ MRa, #16FHi
MMOVXI MRa, #16FLoHex
MMOVI32 MRa, #32FHex

MMOVI16 MARx, #16I *Load the Auxiliary Register with the 16-Bit Immediate Value*

Operands

MARx	Auxiliary register MAR0 or MAR1
#16I	16-bit immediate value

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOVI16 MAR0, #16I)
MSW: 0111 1111 1100 0000

LSW: IIII IIII IIII IIII (opcode of MMOVI16 MAR1, #16I)
MSW: 0111 1111 1110 0000
```

Description

Load the auxiliary register, MAR0 or MAR1, with a 16-bit immediate value. Refer to the pipeline section for important information regarding this instruction.

```
MARx = #16I;
```

Flags

This instruction does not modify flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction. The immediate load of MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Therefore the following applies when loading the auxiliary registers:

- **I1 and I2**

  The two instructions following MMOVI16 will use MAR0/MAR1 before the update occurs. Thus these two instructions will use the old value of MAR0 or MAR1.

- **I3**

  Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win snd the auxiliary register will not be updated with #_X.

- **I4**

  Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOVI16.

```
;  Assume MAR0 is 50 and #_X is 20

MMOVI16 MAR0, #_X            ; Load MAR0 with address of X (20)
<Instruction 1>         ; I1 Will use the old value of MAR0 (50)
<Instruction 2>         ; I2 Will use the old value of MAR0 (50)
<Instruction 3>         ; I3 Cannot use MAR0
<Instruction 4>         ; I4 Will use the new value of MAR0 (20)
<Instruction 5>         ; I5
....
```

Table 9-30. Pipeline Activity For MMOVI16 MAR0/MAR1, #16I

Instruction	F1	F2	D1	D2	R1	R2	E	W
MMOVI16 MAR0, #_X	MMOVI16							
I1	I1	MMOVI16						
I2	I2	I1	MMOVI16					
I3	I3	I2	I1	MMOVI16				
I4	I4	I3	I2	I1	MMOVI16			
I5	I5	I4	I3	I2	I1	MMOVI16		
I6	I6	I5	I4	I3	I2	I1	MMOVI 16	

MMOVI32 MRa, #32FHex *Load the 32-Bits of a 32-Bit Floating-Point Register with the Immediate*

Operands

MRa	floating-point register (MR0 to MR3)
#32FHex	A 32-bit immediate value that represents an IEEE 32-bit floating-point value.

This instruction is an alias for MMOVIZ and MMOVXI instructions. The second operand is translated by the assembler such that the instruction becomes:

```
MMOVIZ MRa, #16FHiHex
MMOVXI MRa, #16FLoHex
```

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOVIZ MRa, #16FHiHex)
MSW: 0111 1000 0100 00aa

LSW: IIII IIII IIII IIII (opcode of MMOVXI MRa, #16FLoHex)
MSW: 0111 1000 1000 00aa
```

Description

Note: This instruction only accepts a hex value as the immediate operand. To specify the immediate value with a floating-point representation use the MMOVF32 MRa, #32F instruction.

Load the 32-bits of MRa with the immediate 32-bit hex value represented by #32Fhex.

#32Fhex is a 32-bit immediate hex value that represents the IEEE 32-bit floating-point value of a floating-point number. The assembler will only accept a hex immediate value. That is, 3.0 can only be represented as #0x40400000. #3.0 will result in an error.

```
MRa = #32FHex;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

Depending on #32FHex, this instruction takes one or two cycles. If all of the lower 16-bits of #32FHex are zeros, then assembler will convert MOVI32 to the MMOVIZ instruction. If the lower 16-bits of #32FHex are not zeros, then assembler will convert MOVI32 to a MMOVIZ and a MMOVXI instruction.

Example

```
MOVI32   MR1, #0x40400000 ; MR1 = 0x40400000
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR1, #0x4040

MOVI32  MR2, #0x00000000  ; MR2 = 0x00000000
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR2, #0x0

MOVI32  MR3, #0x40004001  ; MR3 = 0x40004001
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR3, #0x4000
                          ; MMOVXI MR3, #0x4001

MOVI32  MR0, #0x00004040  ; MR0 = 0x00004040
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR0, #0x0000
                          ; MMOVXI MR0, #0x4040
```

See also

MMOVIZ MRa, #16FHi
MMOVXI MRa, #16FLoHex
MMOVF32 MRa, #32F

MMOVIZ MRa, #16FHi *Load the Upper 16-Bits of a 32-Bit Floating-Point Register*

Operands

MRa	floating-point register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 0100 00aa
```

Description

Load the upper 16-bits of MRa with the immediate value #16FHi and clear the low 16-bits of MRa.

#16FHiHex is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. The assembler will only accept a decimal or hex immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

By itself, MMOVIZ is useful for loading a floating-point register with a constant in which the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). If a constant requires all 32-bits of a floating-point register to be iniitalized, then use MMOVIZ along with the MMOVXI instruction.

```
MRa(31:16) = #16FHi;
MRa(15:0)  = 0;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Load MR0 and MR1 with -1.5 (0xBFC00000)
     MMOVIZ    MR0, #0xBFC0     ; MR0 = 0xBFC00000 (1.5)
     MMOVIZ    MR1, #-1.5       ; MR1 = -1.5 (0xBFC00000)

; Load MR2 with pi = 3.141593 (0x40490FDB)
     MMOVIZ    MR2, #0x4049     ; MR2 = 0x40490000
     MMOVXI    MR2, #0x0FDB     ; MR2 = 0x40490FDB
```

See also

MMOVF32 MRa, #32F
MMOVI32 MRa, #32FHex
MMOVXI MRa, #16FLoHex

MMOVZ16 MRa, mem16 *Load MRx With 16-bit Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem16	16-bit source memory location

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 10aa addr
```

Description

Move the 16-bit value referenced by mem16 to the floating-point register indicated by MRa.

```
MRa(31:16) = 0;
MRa(15:0) = [mem16];
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = 0;
if (MRa(31:0)== 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

MMOVXI MRa, #16FLoHex *Move Immediate to the Low 16-Bits of a Floating-Point Register*

Operands

MRa	CLA floating-point register (MR0 to MR3)
#16FLoHex	A 16-bit immediate hex value that represents the lower 16-bits of an IEEE 32-bit floating-point value. The upper 16-bits will not be modified.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 1000 00aa
```

Description

Load the low 16-bits of MRa with the immediate value #16FLoHex. #16FLoHex represents the lower 16-bits of an IEEE 32-bit floating-point value. The upper 16-bits of MRa will not be modified. MMOVXI can be combined with the MMOVIZ instruction to initialize all 32-bits of a MRa register.

```
MRa(15:0) = #16FLoHex;
MRa(31:16) = Unchanged;
```

Flags

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Load MR0 with pi = 3.141593 (0x40490FDB)
    MMOVIZ      MR0,#0x4049   ; MR0 = 0x40490000
    MMOVXI      MR0,#0x0FDB   ; MR0 = 0x40490FDB
```

See also

MMOVIZ MRa, #16FHi

MMPYF32 MRa, MRb, MRc *32-Bit Floating-Point Multiply*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 0000 0000
```

Description

Multiply the contents of two floating-point registers.

```
MRa = MRb * MRc;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Calculate Num/Den using a Newton-Raphson algorithum for 1/Den
; Ye = Estimate(1/X)
; Ye = Ye*(2.0 - Ye*X)
; Ye = Ye*(2.0 - Ye*X)
;
_Cla1Task1:
    MMOV32     MR1, @_Den      ; MR1 = Den
    MEINVF32   MR2, MR1        ; MR2 = Ye = Estimate(1/Den)
    MMPYF32    MR3, MR2, MR1   ; MR3 = Ye*Den
    MSUBF32    MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32    MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
    MMPYF32    MR3, MR2, MR1   ; MR3 = Ye*Den
||  MMOV32     MR0, @_Num      ; MR0 = Num
    MSUBF32    MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32    MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
||  MMOV32     MR1, @_Den      ; Reload Den To Set Sign
    MNEGF32    MR0, MR0, EQ    ; if(Den == 0.0) Change Sign Of Num
    MMPYF32    MR0, MR2, MR0   ; MR0 = Y = Ye*Num
    MMOV32     @_Dest, MR0     ; Store result
    MSTOP                      ; end of task
```

See also

MMPYF32 MRa, #16FHi, MRb
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf
MMPYF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MMPYF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf
MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRa, #16FHi, MRb *32-Bit Floating-Point Multiply*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1000 baaa
```

Description

Multiply MRb with the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb * #16FHi:0;
```

This instruction can also be written as MMPYF32 MRa, MRb, #16FHi.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example 1

```
; Same as example 2 but #16FHi is represented in float
    MMOVIZ    MR3, #2.0      ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, #3.0, MR3 ; MR0 = 3.0 * MR3 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0       ; Save the result in variable X
```

Example 2

```
; Same as example 1 but #16FHi is represented in Hex
    MMOVIZ    MR3, #2.0         ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, #0x4040, MR3 ; MR0 = 0x4040 * MR3 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0          ; Save the result in variable X
```

Example 3

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
;
_Cla1Task2:
;
; Convert M, X and B from IQ24 to float
     MI32TOF32   MR0, @_M          ; MR0 = 0x4BC00000
     MI32TOF32   MR1, @_X          ; MR1 = 0x4C200000
     MI32TOF32   MR2, @_B          ; MR2 = 0xCB000000
     MMPYF32     MR0, MR0, #0x3380 ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
     MMPYF32     MR1, MR1, #0x3380 ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
     MMPYF32     MR2, MR2, #0x3380 ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
     MMPYF32     MR3, MR0, MR1     ; M*X
     MADDF32     MR2, MR2, MR3     ; Y=MX+B = 3.25 (0x40500000)
; Convert Y from float32 to IQ24
     MMPYF32     MR2, MR2, #0x4B80 ; Y * 1*2^24
     MF32TOI32   MR2, MR2          ; IQ24(Y) = 0x03400000
     MMOV32 @_Y, MR2               ; store result
     MSTOP                         ; end of task
```

See also

MMPYF32 MRa, MRb, #16FHi
MMPYF32 MRa, MRb, MRc
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MMPYF32 MRa, MRb, #16FHi *32-Bit Floating-Point Multiply*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1000 baaa
```

Description

Multiply MRb with the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb * #16FHi:0;
```

This instruction can also be writen as MMPYF32 MRa, #16FHi, MRb.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example 1

```
;Same as example 2 but #16FHi is represented in float
    MMOVIZ    MR3, #2.0       ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, MR3, #3.0  ; MR0 = MR3 * 3.0 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0        ; Save the result in variable X
```

Example 2

```
;Same as above example but #16FHi is represented in Hex
    MMOVIZ    MR3, #2.0          ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, MR3, #0x4040  ; MR0 = MR3 * 0x4040 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0           ; Save the result in variable X
```

Example 3

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
_Cla1Task2:
;
; Convert M, X and B from IQ24 to float
     MI32TOF32   MR0, @_M          ; MR0 = 0x4BC00000
     MI32TOF32   MR1, @_X          ; MR1 = 0x4C200000
     MI32TOF32   MR2, @_B          ; MR2 = 0xCB000000
     MMPYF32     MR0, #0x3380, MR0 ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
     MMPYF32     MR1, #0x3380, MR1 ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
     MMPYF32     MR2, #0x3380, MR2 ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
     MMPYF32     MR3, MR0, MR1     ; M*X
     MADDF32     MR2, MR2, MR3     ; Y=MX+B = 3.25 (0x40500000)

; Convert Y from float32 to IQ24
     MMPYF32     MR2, #0x4B80, MR2 ; Y * 1*2^24
     MF32TOI32   MR2, MR2          ; IQ24(Y) = 0x03400000
     MMOV32      @_Y, MR2          ; store result
     MSTOP                         ; end of task
```

See also

MMPYF32 MRa, #16FHi, MRb
MMPYF32 MRa, MRb, MRc

MMPYF32 MRa, MRb, MRc||MADDF32 MRd, MRe, MRf *32-Bit Floating-Point Multiply with Parallel Add*

Operands

MRa	CLA floating-point destination register for MMPYF32 (MR0 to MR3) MRa cannot be the same register as MRd
MRb	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRc	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRd	CLA floating-point destination register for MADDF32 (MR0 to MR3) MRd cannot be the same register as MRa
MRe	CLA floating-point source register for MADDF32 (MR0 to MR3)
MRf	CLA floating-point source register for MADDF32 (MR0 to MR3)

Opcode

```
LSW: 0000 ffee ddcc bbaa
MSW: 0111 1010 0000 0000
```

Description

Multiply the contents of two floating-point registers with parallel addition of two registers.

```
MRa = MRb * MRc;
MRd = MRe + MRf;
```

Restrictions

The destination register for the MMPYF32 and the MADDF32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 or MADDF32 generates an underflow condition.
- LVF = 1 if MMPYF32 or MADDF32 generates an overflow condition.

Pipeline

Both MMPYF32 and MADDF32 complete in a single cycle.

Example

```
; Perform 5 multiply and accumulate operations:
;
; X and Y are 32-bit floating point arrays
;
; 1st multiply: A = X0 * Y0
; 2nd multiply: B = X1 * Y1
; 3rd multiply: C = X2 * Y2
; 4th multiply: D = X3 * Y3
; 5th multiply: E = X3 * Y3
;
; Result = A + B + C + D + E
;
_Cla1Task1:
    MMOVI16    MAR0, #_X                ; MAR0 points to X array
    MMOVI16    MAR1, #_Y                ; MAR1 points to Y array
    MNOP                                ; Delay for MAR0, MAR1 load
    MNOP                                ; Delay for MAR0, MAR1 load
                                        ; <-- MAR0 valid
    MMOV32     MR0, *MAR0[2]++          ; MR0 = X0, MAR0 += 2
                                        ; <-- MAR1 valid
    MMOV32     MR1, *MAR1[2]++          ; MR1 = Y0, MAR1 += 2

    MMPYF32    MR2, MR0, MR1            ; MR2 = A = X0 * Y0
||  MMOV32     MR0, *MAR0[2]++          ; In parallel MR0 = X1, MAR0 += 2
    MMOV32     MR1, *MAR1[2]++          ; MR1 = Y1, MAR1 += 2

    MMPYF32    MR3, MR0, MR1            ; MR3 = B = X1 * Y1
||  MMOV32     MR0, *MAR0[2]++          ; In parallel MR0 = X2, MAR0 += 2
    MMOV32     MR1, *MAR1[2]++          ; MR1 = Y2, MAR2 += 2

    MMACF32    MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2
||  MMOV32     MR0, *MAR0[2]++          ; In parallel MR0 = X3
    MMOV32     MR1, *MAR1[2]++          ; MR1 = Y3

    MMACF32    MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3
||  MMOV32     MR0, *MAR0               ; In parallel MR0 = X4
    MMOV32     MR1, *MAR1               ; MR1 = Y4

    MMPYF32    MR2, MR0, MR1            ; MR2 = E = X4 * Y4
||  MADDF32    MR3, MR3, MR2            ; in parallel MR3 = (A + B + C) + D

    MADDF32    MR3, MR3, MR2            ; MR3 = (A + B + C + D) + E
    MMOV32     @_Result, MR3            ; Store the result
    MSTOP                               ; end of task
```

See also MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 *32-Bit Floating-Point Multiply with Parallel Move*

Operands

MRd	CLA floating-point destination register for the MMPYF32 (MR0 to MR3) MRd cannot be the same register as MRa
MRe	CLA floating-point source register for the MMPYF32 (MR0 to MR3)
MRf	CLA floating-point source register for the MMPYF32 (MR0 to MR3)
MRa	CLA floating-point destination register for the MMOV32 (MR0 to MR3) MRa cannot be the same register as MRd
mem32	32-bit memory location accessed using direct or indirect addressing. This will be the source of the MMOV32.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0000 ffee ddaa addr
```

Description

Multiply the contents of two floating-point registers and load another.

```
MRd = MRe * MRf;
MRa = [mem32];
```

Restrictions

The destination register for the MMPYF32 and the MMOV32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

The MMOV32 Instruction will set the NF and ZF flags as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
```

Pipeline

Both MMPYF32 and MMOV32 complete in a single cycle.

Example 1

```
; Given M1, X1 and B1 are 32-bit floating point
; Calculate Y1 = M1*X1+B1
;
_Cla1Task1:
    MMOV32     MR0, @M1        ; Load MR0 with M1
    MMOV32     MR1, @X1        ; Load MR1 with X1
    MMPYF32    MR1, MR1, MR0   ; Multiply M1*X1
||  MMOV32     MR0, @B1        ; and in parallel load MR0 with B1
    MADDF32    MR1, MR1, MR0   ; Add M*X1 to B1 and store in MR1
    MMOV32     @Y1, MR1        ; Store the result
    MSTOP                      ; end of task
```

Example 2

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A * B) * C
;
_Cla1Task2:
    MMOV32    MR0, @A        ; Load MR0 with A
    MMOV32    MR1, @B        ; Load MR1 with B
    MMPYF32   MR1, MR1, MR0  ; Multiply A*B
||  MMOV32    MR0, @C        ; and in parallel load MR0 with C
    MMPYF32   MR1, MR1, MR0  ; Multiply (A*B) by C
||  MMOV32    @Y2, MR1       ; and in parallel store A*B
    MMOV32    @Y3, MR1       ; Store the result
    MSTOP                    ; end of task
```

See also

MMPYF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRd, MRe, MRf ||MMOV32 mem32, MRa *32-Bit Floating-Point Multiply with Parallel Move*

Operands

MRd	CLA floating-point destination register for the MMPYF32 (MR0 to MR3)
MRe	CLA floating-point source register for the MMPYF32 (MR0 to MR3)
MRf	CLA floating-point source register for the MMPYF32 (MR0 to MR3)
mem32	32-bit memory location accessed using direct or indirect addressing. This will be the destination of the MMOV32.
MRa	CLA floating-point source register for the MMOV32 (MR0 to MR3)

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0100 ffee ddaa addr
```

Description

Multiply the contents of two floating-point registers and move from memory to register.

```
MRd = MRe * MRf;
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

MMPYF32 and MMOV32 both complete in a single cycle.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A * B) * C
;
_Cla1Task2:
     MMOV32    MR0, @A        ; Load MR0 with A
     MMOV32    MR1, @B        ; Load MR1 with B
     MMPYF32   MR1, MR1, MR0  ; Multiply A*B
||   MMOV32    MR0, @C        ; and in parallel load MR0 with C
     MMPYF32   MR1, MR1, MR0  ; Multiply (A*B) by C
||   MMOV32    @Y2, MR1       ; and in parallel store A*B
     MMOV32    @Y3, MR1       ; Store the result
     MSTOP                    ; end of task
```

See also

MMPYF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRa, MRb, MRc ||MSUBF32 MRd, MRe, MRf *32-Bit Floating-Point Multiply with Parallel Subtract*

Operands

MRa	CLA floating-point destination register for MMPYF32 (MR0 to MR3) MRa cannot be the same register as MRd
MRb	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRc	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRd	CLA floating-point destination register for MSUBF32 (MR0 to MR3) MRd cannot be the same register as MRa
MRe	CLA floating-point source register for MSUBF32 (MR0 to MR3)
MRf	CLA floating-point source register for MSUBF32 (MR0 to MR3)

Opcode

```
LSW: 0000 ffee ddcc bbaa
MSW: 0111 1010 0100 0000
```

Description

Multiply the contents of two floating-point registers with parallel subtraction of two registers.

```
MRa = MRb * MRc;
MRd = MRe - MRf;
```

Restrictions

The destination register for the MMPYF32 and the MSUBF32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 or MSUBF32 generates an underflow condition.
- LVF = 1 if MMPYF32 or MSUBF32 generates an overflow condition.

Pipeline

MMPYF32 and MSUBF32 both complete in a single cycle.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A - B)
;
_Cla1Task2:
     MMOV32   MR0, @A         ; Load MR0 with A
     MMOV32   MR1, @B         ; Load MR1 with B
     MMPYF32  MR2, MR0, MR1   ; Multiply (A*B)
||   MSUBF32  MR3, MR0, MR1   ; and in parallel Sub (A-B)
     MMOV32   @Y2, MR2        ; Store A*B
     MMOV32   @Y3, MR3        ; Store A-B
     MSTOP                    ; end of task
```

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MSUBF32 MRd, MRe, MRf || MMOV32 mem32, MRa

MNEGF32 MRa, MRb{, CNDF} *Conditional Negation*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
CNDF	condition tested

Opcode

```
LSW: 0000 0000 cndf bbaa
MSW: 0111 1010 1000 0000
```

Description

```
if (CNDF == true) {MRa = - MRb; }
else {MRa = MRb; }
```

CNDF is one of the following conditions:

Encode [5]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [6]	Unconditional with flag modification	None

[5] Values not shown are reserved.
[6] This is the default operation if no CNDF field is specified. This condition will allow the ZF, and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

Pipeline

This is a single-cycle instruction.

Example 1

```
; Show the basic operation of MNEGF32
;
     MMOVIZ    MR0, #5.0      ; MR0 = 5.0 (0x40A00000)
     MMOVIZ    MR1, #4.0      ; MR1 = 4.0 (0x40800000)
     MMOVIZ    MR2, #-1.5     ; MR2 = -1.5 (0xBFC00000)
     MMPYF32   MR3, MR1, MR2  ; MR3 = -6.0
     MMPYF32   MR0, MR0, MR1  ; MR0 = 20.0
     MMOVIZ    MR1, #0.0
     MCMPF32   MR3, MR1       ; NF = 1
     MNEGF32   MR3, MR3, LT   ; if NF = 1, MR3 = 6.0
     MCMPF32   MR0, MR1       ; NF = 0
     MNEGF32   MR0, MR0, GEQ  ; if NF = 0, MR0 = -20.0
```

Example 2

```
; Calculate Num/Den using a Newton-Raphson algorithum for 1/Den
; Ye = Estimate(1/X)
; Ye = Ye*(2.0 - Ye*X)
; Ye = Ye*(2.0 - Ye*X)
;
_Cla1Task1:
    MMOV32    MR1, @_Den      ; MR1 = Den
    MEINVF32  MR2, MR1        ; MR2 = Ye = Estimate(1/Den)
    MMPYF32   MR3, MR2, MR1   ; MR3 = Ye*Den
    MSUBF32   MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32   MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
    MMPYF32   MR3, MR2, MR1   ; MR3 = Ye*Den
||  MMOV32    MR0, @_Num      ; MR0 = Num
    MSUBF32   MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32   MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
||  MMOV32    MR1, @_Den      ; Reload Den To Set Sign
    MNEGF32   MR0, MR0, EQ    ; if(Den == 0.0) Change Sign Of Num
    MMPYF32   MR0, MR2, MR0   ; MR0 = Y = Ye*Num
    MMOV32    @_Dest, MR0     ; Store result
    MSTOP                     ; end of task
```

See also MABSF32 MRa, MRb

MNOP *No Operation*

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1010 0000
```

Description

Do nothing. This instruction is used to fill required pipeline delay slots when other instructions are not available to fill the slots.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; X is an array of 32-bit floating-point values
; Find the maximum value in an array X
; and store it in Result
;
_Cla1Task1:
    MMOVI16    MAR1,#_X         ; Start address
    MUI16TOF32 MR0, @_len       ; Length of the array
    MNOP                        ; delay for MAR1 load
    MNOP                        ; delay for MAR1 load
    MMOV32     MR1, *MAR1[2]++  ; MR1 = X0
LOOP
    MMOV32     MR2, *MAR1[2]++  ; MR2 = next element
    MMAXF32    MR1, MR2         ; MR1 = MAX(MR1, MR2)
    MADDF32    MR0, MR0, #-1.0  ; Decrememt the counter
    MCMPF32    MR0 #0.0         ; Set/clear flags for MBCNDD
    MNOP                        ; Too late to affect MBCNDD
    MNOP                        ; Too late to affect MBCNDD
    MNOP                        ; Too late to affect MBCNDD
    MBCNDD    LOOP, NEQ         ; Branch if not equal to zero
    MMOV32    @_Result, MR1     ; Always executed
    MNOP                        ; Pad to seperate MBCNDD and MSTOP
    MNOP                        ; Pad to seperate MBCNDD and MSTOP
    MSTOP                       ; End of task
```

See also

MOR32 MRa, MRb, MRc *Bitwise OR*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 1000 0000
```

Description

Bitwise OR of MRb with MRc.

```
MARa(31:0) = MARb(31:0) OR MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ    MR0, #0x5555 ; MR0 = 0x5555AAAA
MMOVXI    MR0, #0xAAAA

MMOVIZ    MR1, #0x5432 ; MR1 = 0x5432FEDC
MMOVXI    MR1, #0xFEDC

                       ; 0101 OR 0101 = 0101 (5)
                       ; 0101 OR 0100 = 0101 (5)
                       ; 0101 OR 0011 = 0111 (7)
                       ; 0101 OR 0010 = 0111 (7)
                       ; 1010 OR 1111 = 1111 (F)
                       ; 1010 OR 1110 = 1110 (E)
                       ; 1010 OR 1101 = 1111 (F)
                       ; 1010 OR 1100 = 1110 (E)
MOR32 MR2, MR1, MR0    ; MR3 = 0x5555FEFE
```

See also

MAND32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc

MRCNDD {CNDF} *Return Conditional Delayed*

Operands

CNDF	optional condition.

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1001 1010 cndf
```

Description

If the specified condition is true, then the RPC field of MSTF is loaded into MPC and fetching continues from that location. Otherwise program fetches will continue without the return.

Please refer to the pipeline section for important information regarding this instruction.

```
if (CNDF == TRUE) MPC = RPC;
```

CNDF is one of the following conditions:

Encode [7]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [8]	Unconditional with flag modification	None

(7) Values not shown are reserved.

(8) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

The MRCNDD instruction by itself is a single-cycle instruction. As shown in Table 9-31, for each return 6 instruction slots are executed; three before the return instruction (d5-d7) and three after the return instruction (d8-d10). The total number of cycles for a return taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a return can, therefore, range from 1 to 7 cycles. The number of cycles for a return taken may not be the same as for a return not taken.

Referring to the following code fragment and the pipeline diagrams in Table 9-31 and Table 9-32, the instructions before and after MRCNDD have the following properties:

```
;
;
<Instruction 1>    ; I1 Last instruction that can affect flags for
                   ; the MCCNDD operation
<Instruction 2>    ; I2 Cannot be stop, branch, call or return
<Instruction 3>    ; I3 Cannot be stop, branch, call or return
<Instruction 4>    ; I4 Cannot be stop, branch, call or return

MCCNDD _func, NEQ  ; Call to func if not eqal to zero
                   ; Three instructions after MCCNDD are always
                   ; executed whether the call is taken or not
<Instruction 5>    ; I5 Cannot be stop, branch, call or return
<Instruction 6>    ; I6 Cannot be stop, branch, call or return
<Instruction 7>    ; I7 Cannot be stop, branch, call or return
<Instruction 8>    ; I8 The address of this instruction is saved
                   ; in the RPC field of the MSTF register.
                   ; Upon return this value is loaded into MPC
                   ; and fetching continues from this point.
<Instruction 9>    ; I9
<Instruction 10>   ; I10
....
....
_func:
<Destination 1>    ; d1 Can be any instruction
<Destination 2>    ; d2
<Destination 3>    ; d3
<Destination 4>    ; d4 Last instruction that can affect flags for
                   ; the MRCNDD operation
<Destination 5>    ; d5 Cannot be stop, branch, call or return
<Destination 6>    ; d6 Cannot be stop, branch, call or return
<Destination 7>    ; d7 Cannot be stop, branch, call or return

MRCNDD  NEQ        ; Return to <Instruction 8> if not equal to zero
                   ; Three instructions after MRCNDD are always
                   ; executed whether the return is taken or not
<Destination 8>    ; d8 Cannot be stop, branch, call or return
<Destination 9>    ; d9 Cannot be stop, branch, call or return
<Destination 10>   ; d10 Cannot be stop, branch, call or return
<Destination 11>   ; d11
<Destination 12>   ; d12
....
....
MSTOP
....
```

- **d4**
  - d4 is the last instruction that can effect the CNDF flags for the MRCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to return or not when MRCNDD is in the D2 phase.
  - There are no restrictions on the type of instruction for d4.
- **d5, d6 and d7**
  - The three instructions proceeding MRCNDD can change MSTF flags but will have no effect on whether the MRCNDD instruction makes the return or not. This is

because the flag modification will occur after the D2 phase of the MRCNDD instruction.

- These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

- **d8, d9 and d10**
  - The three instructions following MRCNDD are always executed irrespective of whether the return is taken or not.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

Table 9-31. Pipeline Activity For MRCNDD, Return Not Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
d4	d4	d3	d2	d1	I7	I6	I5	
d5	d5	d4	d3	d2	d1	I7	I6	
d6	d6	d5	d4	d3	d2	d1	i7	
d7	d7	d6	d5	d4	d3	d2	d1	
MRCNDD	MRCNDD	d7	d6	d5	d4	d3	d2	
d8	d8	MRCNDD	d7	d6	d5	d4	d3	
d9	d9	d8	MRCNDD	d7	d6	d5	d4	
d10	d10	d9	d8	MRCNDD	d7	d6	d5	
d11	d11	d10	d9	d8	-	d7	d6	
d12	d12	d11	d10	d9	d8	-	d7	
etc....	....	d12	d11	d10	d9	d8	-	
....	....	....	d12	d11	d10	d9	d8	
....	....	....	....	d12	d11	d10	d9	
					d12	d11	d10	
						d12	d11	
							d12	

Table 9-32. Pipeline Activity For MRCNDD, Return Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
d4	d4	d3	d2	d1	I7	I6	I5	
d5	d5	d4	d3	d2	d1	I7	I6	
d6	d6	d5	d4	d3	d2	d1	i7	
d7	d7	d6	d5	d4	d3	d2	d1	
MRCNDD	MRCNDD	d7	d6	d5	d4	d3	d2	
d8	d8	MRCNDD	d7	d6	d5	d4	d3	
d9	d9	d8	MRCNDD	d7	d6	d5	d4	
d10	d10	d9	d8	MRCNDD	d7	d6	d5	
I8	I8	d10	d9	d8	-	d7	d6	
I9	I9	I8	d10	d9	d8	-	d7	
I10	I10	I9	I8	d10	d9	d8	-	
etc....	....	I10	I9	I8	d10	d9	d8	
....	....		I10	I9	I8	d10	d9	
....	....			I10	I9	I8	d10	
					I10	I9	I8	
						I10	I9	
							I10	

Example ;

See also MBCNDD #16BitDest, CNDF
MCCNDD 16BitDest, CNDF
MMOV32 mem32, MSTF
MMOV32 MSTF, mem32

MSETFLG FLAG, VALUE *Set or Clear Selected Floating-Point Status Flags*

Operands

FLAG	8 bit mask indicating which floating-point status flags to change.
VALUE	8 bit mask indicating the flag value; 0 or 1.

Opcode

```
LSW: FFFF FFFF VVVV VVVV
MSW: 0111 1001 1100 0000
```

Description

The MSETFLG instruction is used to set or clear selected floating-point status flags in the MSTF register. The FLAG field is an 11-bit value that indicates which flags will be changed. That is, if a FLAG bit is set to 1 it indicates that flag will be changed; all other flags will not be modified. The bit mapping of the FLAG field is shown below:

reserved	RNDF32	TF	reserved	reserved	ZF	NF	LUF	LVF
8	7	6	5	4	3	2	1	0

The VALUE field indicates the value the flag should be set to; 0 or 1.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	Yes	Yes	Yes	Yes	Yes

Any flag can be modified by this instruction. The MEALLOW and RPC fields cannot be modified with this instruction.

Pipeline

This is a single-cycle instruction.

Example

To make it easier and legible, the assembler accepts a FLAG=VALUE syntax for the MSTFLG operation as shown below:

```
MSETFLG RNDF32=0, TF=0, NF=1; FLAG = 11000100; VALUE = 00XXX1XX;
```

See also

MMOV32 mem32, MSTF
MMOV32 MSTF, mem32

MSTOP *Stop Task*

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1000 0000
```

Description

The MSTOP instruction must be placed to indicate the end of each task. In addition, placing MSTOP in unused memory locations within the CLA program RAM can be useful for debugging and preventing run away CLA code. When MSTOP enters the D2 phase of the pipeline, the MIRUN flag for the task is cleared and the associated interrupt is flagged in the PIE vector table.

There are three special cases that can occur when single-stepping a task such that the MPC reaches the MSTOP instruction.

1. If you are single-stepping or halted in "task A" and "task B" comes in before the MPC reaches the MSTOP, then "task B" will start if you continue to step through the MSTOP instruction. Basically if "task B" is pending before the MPC reaches MSTOP in "task A" then there is no issue in "task B" starting and no special action is required.
2. In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. If "task B" comes in at this point, it will be flagged in the MIFR register but it may or may not start if you continue to single-step through the MSTOP instruction of "task A". It depends on exactly when the new task comes in. To reliably start "task B" perform a soft reset and reconfigure the MIER bits. Once this is done, you can start single-stepping "task B".
3. Case 2 can be handled slightly differently if there is control over when "task B" comes in (for example using the IACK instruction to start the task). In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. Before forcing "task B", run free to force the CLA out of the debug state. Once this is done you can force "task B" and continue debugging.

Restrictions

The MSTOP instruction cannot be placed 3 instructions before or after a MBCNDD, MCCNDD or MRCNDD instruction.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction. Table 9-33 shows the pipeline behavior of the MSTOP instruction. The MSTOP instruction cannot be placed with 3 instructions of a MBCNDD, MCCNDD or MRCNDD instruction.

Table 9-33. Pipeline Activity For MSTOP

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
MSTOP	MSTOP	I3	I2	I1				
I4	I4	MSTOP	I3	I2	I1			
I5	I5	I4	MSTOP	I3	I2	I1		
I6	I6	I5	I4	MSTOP	I3	I2	I1	
New Task Arbitrated and Piroitized	-	-	-	-	-	I3	I2	
New Task Arbitrated and Piroitized	-	-	-	-	-	-	I3	
I1	I1	-	-	-	-	-	-	
I2	I2	I1	-	-	-	-	-	
I3	I3	I2	I1	-	-	-	-	
I4	I4	I3	I2	I1	-	-	-	
I5	I5	I4	I3	I2	I1	-	-	
I6	I6	I5	I4	I3	I2	I1	-	
I7	I7	I6	I5	I4	I3	I2	I1	
etc								

Example

```
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
; Calculate Y2 = A - B - C
_Cla1Task3:
   MMOV32   MR0, @_A      ; MR0 = 1 (0x00000001)
   MMOV32   MR1, @_B      ; MR1 = 2 (0x00000002)
   MMOV32   MR2, @_C      ; MR2 = -7 (0xFFFFFFF9)
   MSUB32   MR3, MR0, MR1 ; A + B
   MSUB32   MR3, MR3, MR2 ; A + B + C = 6 (0x0000006)
   MMOV32   @_y2, MR3     ; Store y2
   MSTOP                  ; End of task
```

See also MDEBUGSTOP

MSUB32 MRa, MRb, MRc *32-Bit Integer Subtraction*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point destination register (MR0 to MR3)
MRc	CLA floating-point destination register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 1110 0000
```

Description

32-bit integer addition of MRb and MRc.

```
MARa(31:0) = MARb(31:0) - MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
;
Calculate Y2 = A - B - C
;
_Cla1Task3:
   MMOV32   MR0, @_A          ; MR0 = 1 (0x00000001)
   MMOV32   MR1, @_B          ; MR1 = 2 (0x00000002)
   MMOV32   MR2, @_C          ; MR2 = -7 (0xFFFFFFF9)
   MSUB32   MR3, MR0, MR1     ; A + B
   MSUB32   MR3, MR3, MR2     ; A + B + C = 6 (0x0000006)
   MMOV32   @_y2, MR3         ; Store y2
   MSTOP                      ; End of task
```

See also

MADD32 MRa, MRb, MRc
MAND32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc

MSUBF32 MRa, MRb, MRc *32-Bit Floating-Point Subtraction*

Operands

MRa	CLA floating-point destination register (MR0 to R1)
MRb	CLA floating-point source register (MR0 to R1)
MRc	CLA floating-point source register (MR0 to R1)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 0100 0000
```

Description

Subtract the contents of two floating-point registers

```
MRa = MRb - MRc;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = A + B - C
;
_Cla1Task5:
   MMOV32    MR0, @_A      ; Load MR0 with A
   MMOV32    MR1, @_B      ; Load MR1 with B
   MADDF32   MR0, MR1, MR0 ; Add A + B
|| MMOV32    MR1, @_C      ;   and in parallel load C
   MSUBF32   MR0, MR0, MR1 ; Subtract C from (A + B)
   MMOV32    @Y,  MR0      ; (A+B) - C
   MSTOP                   ; end of task
```

See also

MSUBF32 MRa, #16FHi, MRb
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MSUBF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSUBF32 MRa, #16FHi, MRb *32-Bit Floating-Point Subtraction*

Operands

MRa	CLA floating-point destination register (MR0 to R1)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.
MRb	CLA floating-point source register (MR0 to R1)

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 0000 baaa
```

Description

Subtract MRb from the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = #16FHi:0 - MRb;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Y = sqrt(X)
; Ye = Estimate(1/sqrt(X));
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Y = X*Ye
;
_Cla1Task3:
   MMOV32     MR0, @_x          ; MR0 = X
   MEISQRTF32 MR1, MR0          ; MR1 = Ye = Estimate(1/sqrt(X))
   MMOV32     MR1, @_x, EQ      ; if(X == 0.0) Ye = 0.0
   MMPYF32    MR3, MR0, #0.5    ; MR3 = X*0.5
   MMPYF32    MR2, MR1, MR3     ; MR2 = Ye*X*0.5
   MMPYF32    MR2, MR1, MR2     ; MR2 = Ye*Ye*X*0.5
   MSUBF32    MR2, #1.5, MR2    ; MR2 = 1.5 - Ye*Ye*X*0.5
   MMPYF32    MR1, MR1, MR2     ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
   MMPYF32    MR2, MR1, MR3     ; MR2 = Ye*X*0.5
   MMPYF32    MR2, MR1, MR2     ; MR2 = Ye*Ye*X*0.5
   MSUBF32    MR2, #1.5, MR2    ; MR2 = 1.5 - Ye*Ye*X*0.5
   MMPYF32    MR1, MR1, MR2     ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
   MMPYF32    MR0, MR1, MR0     ; MR0 = Y = Ye*X
   MMOV32     @_y, MR0          ; Store Y = sqrt(X)
   MSTOP                        ; end of task
```

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MSUBF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSUBF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 *32-Bit Floating-Point Subtraction with Parallel Move*

Operands

MRd	CLA floating-point destination register (MR0 to MR3) for the MSUBF32 operation MRd cannot be the same register as MRa
MRe	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
MRf	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
MRa	CLA floating-point destination register (MR0 to MR3) for the MMOV32 operation MRa cannot be the same register as MRd
mem32	32-bit memory location accessed using direct or indirect addressing. Source for the MMOV32 operation.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0010 ffee ddaa addr
```

Description

Subtract the contents of two floating-point registers and move from memory to a floating-point register.

```
MRd = MRe - MRf;
MRa = [mem32];
```

Restrictions

The destination register for the MSUBF32 and the MMOV32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

The MMOV32 Instruction will set the NF and ZF flags as follows:

Pipeline

Both MSUBF32 and MMOV32 complete in a single cycle.

Example

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
```

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRa, #16FHi, MRb
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSUBF32 MRd, MRe, MRf ||MMOV32 mem32, MRa *32-Bit Floating-Point Subtraction with Parallel Move*

Operands

MRd	CLA floating-point destination register (MR0 to MR3) for the MSUBF32 operation
MRe	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
MRf	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
mem32	32-bit destination memory location for the MMOV32 operation
MRa	CLA floating-point source register (MR0 to MR3) for the MMOV32 operation

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0110 ffee ddaa addr
```

Description

Subtract the contents of two floating-point registers and move from a floating-point register to memory.

```
MRd = MRe - MRf;
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

Pipeline

Both MSUBF32 and MMOV32 complete in a single cycle.

Example

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRa, #16FHi, MRb
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSWAPF MRa, MRb {, CNDF} *Conditional Swap*

Operands

MRa	CLA floating-point register (MR0 to MR3)
MRb	CLA floating-point register (MR0 to MR3)
CNDF	Optional condition tested based on the MSTF flags

Opcode

```
LSW: 0000 0000 CNDF bbaa
MSW: 0111 1011 0000 0000
```

Description

Conditional swap of MRa and MRb.

```
if (CNDF == true) swap MRa and MRb;
```

CNDF is one of the following conditions:

Encode [1]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [2]	Unconditional with flag modification	None

[1] Values not shown are reserved.

[2] This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

No flags affected

Pipeline

This is a single-cycle instruction.

Example

```
; X is an array of 32-bit floating-point values
; and has len elements. Find the maximum value in
; the array and store it in Result
;
; Note: MCMPF32 and MSWAPF can be replaced by MMAXF32
;
_Cla1Task1:
    MMOVI16    MAR1,#_X         ; Start address
    MUI16TOF32 MR0, @_len       ; Length of the array
    MNOP                        ; delay for MAR1 load
    MNOP                        ; delay for MAR1 load
    MMOV32     MR1, *MAR1[2]++  ; MR1 = X0
LOOP
    MMOV32     MR2, *MAR1[2]++  ; MR2 = next element
    MCMPF32    MR2, MR1         ; Compare MR2 with MR1
    MSWAPF     MR1, MR2, GT     ; MR1 = MAX(MR1, MR2)
    MADDF32    MR0, MR0, #-1.0  ; Decrememt the counter
    MCMPF32    MR0 #0.0         ; Set/clear flags for MBCNDD
    MNOP
    MNOP
    MNOP
    MBCNDD     LOOP, NEQ        ; Branch if not equal to zero
    MMOV32     @_Result, MR1    ; Always executed
    MNOP                        ; Always executed
    MNOP                        ; Always executed
    MSTOP                       ; End of task
```

See also

MMTESTTF CNDF *Test MSTF Register Flag Condition*

Operands

CNDF	condition to test based on MSTF flags

Opcode

```
LSW: 0000 0000 0000 cndf
MSW: 0111 1111 0100 0000
```

Description

Test the CLA floating-point condition and if true, set the MSTF[TF] flag. If the condition is false, clear the MSTF[TF] flag. This is useful for temporarily storing a condition for later use.

```
if (CNDF == true) TF = 1;
else TF = 0;
```

CNDF is one of the following conditions:

Encode [3]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF [4]	Unconditional with flag modification	None

[3] Values not shown are reserved.
[4] This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	Yes	No	No	No	No

```
TF = 0;
if (CNDF == true) TF = 1;
```

Note: If (CNDF == UNC or UNCF), the TF flag will be set to 1.

Pipeline

This is a single-cycle instruction.

Example

```
; if (State == 0.1)
;    RampState = RampState || RAMPMASK
; else if (State == 0.01)
;    CoastState = CoastState || COASTMASK
; else
;    SteadyState = SteadyState || STEADYMASK
;
_Cla1Task2:
   MMOV32    MR0, @_State
   MCMPF32   MR0, #0.1        ; Affects flags for 1st MBCNDD (A)
   MCMPF32   MR0, #0.01       ; Check used by 2nd MBCNDD (B)
   MMTESTTF   EQ               ; Store EQ flag in TF for 2nd MBCNDD (B)
   MNOP
   MBCNDD    _Skip1, NEQ      ; (A) If State != 0.1, go to Skip1
   MMOV32    MR1, @_RampState ; Always executed
   MMOVXI    MR2, #RAMPMASK   ; Always executed
   MOR32     MR1, MR2         ; Always executed
   MMOV32    @_RampState, MR1 ; Execute if (A) branch not taken
   MSTOP                      ; end of task if (A) branch not taken

_Skip1:
   MMOV32    MR3, @_SteadyState
   MMOVXI    MR2, #STEADYMASK
   MOR32     MR3, MR2
   MBCNDD    _Skip2, NTF       ; (B) if State != .01, go to Skip2
   MMOV32    MR1, @_CoastState ; Always executed
   MMOVXI    MR2, #COASTMASK   ; Always executed
   MOR32     MR1, MR2          ; Always executed
   MMOV32    @_CoastState, MR1 ; Execute if (B) branch not taken
   MSTOP                       ; end of task if (B) branch not taken

_Skip2:
   MMOV32 @_SteadyState, MR3   ; Executed if (B) branch taken
   MSTOP
```

See also

MUI16TOF32 MRa, mem16 *Convert Unsigned 16-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem16	16-bit source memory location

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 01aa addr
```

Description

When converting F32 to I16/UI16 data format, the MF32TOI16/UI16 operation truncates to zero while the MF32TOI16R/UI16R operation will round to nearest (even) value.

```
MRa = UI16TOF32[mem16];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MUI16TOF32 MRa, MRb *Convert Unsigned 16-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1110 0000
```

Description

Convert an unsigned 16-bit integer to a 32-bit floating-point value. When converting float32 to I16/UI16 data format, the MF32TOI16/UI16 operation truncates to zero while the MF32TOI16R/UI16R operation will round to nearest (even) value.

```
MRa = UI16TOF32[MRb];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVXI MR1, #0x800F ; MR1(15:0) = 32783 (0x800F)
MUI16TOF32 MR0, MR1 ; MR0 = UI16TOF32 (MR1(15:0))
                    ; = 32783.0 (0x47000F00)
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16

MUI32TOF32 MRa, mem32 *Convert Unsigned 32-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem32	32-bit memory location accessed using direct or indirect addressing

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 10aa addr
```

Description

```
MRa = UI32TOF32[mem32];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Given x2, m2 and b2 are Uint32 numbers:
;
; x2 = Uint32(2) = 0x00000002
; m2 = Uint32(1) = 0x00000001
; b2 = Uint32(3) = 0x00000003
;
; Calculate y2 = x2 * m2 + b2
;
_Cla1Task1:
   MUI32TOF32  MR0, @_m2      ; MR0 = 1.0 (0x3F800000)
   MUI32TOF32  MR1, @_x2      ; MR1 = 2.0 (0x40000000)
   MUI32TOF32  MR2, @_b2      ; MR2 = 3.0 (0x40400000)
   MMPYF32     MR3, MR0, MR1  ; M*X
   MADDF32     MR3, MR2, MR3  ; Y=MX+B = 5.0 (0x40A00000)
   MF32TOUI32  MR3, MR3       ; Y = Uint32(5.0) = 0x00000005
   MMOV32      @_y2, MR3      ; store result
   MSTOP                      ; end of task
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, mem32
MI32TOF32 MRa, MRb
MUI32TOF32 MRa, MRb

MUI32TOF32 MRa, MRb *Convert Unsigned 32-Bit Integer to 32-Bit Floating-Point Value*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1100 0000
```

Description

```
MRa = UI32TOF32 [MRb];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ     MR3, #0x8000 ; MR3(31:16) = 0x8000
MMOVXI     MR3, #0x1111 ; MR3(15:0) = 0x1111
                        ; MR3 = 2147488017
MUI32TOF32 MR3, MR3     ; MR3 = MUI32TOF32 (MR3) = 2147488017.0 (0x4F000011)
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, mem32
MI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MXOR32 MRa, MRb, MRc *Bitwise Exclusive Or*

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 1010 0000
```

Description

Bitwise XOR of MRb with MRc.

```
MARa(31:0) = MARb(31:0) XOR MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ MR0, #0x5555   ; MR0 = 0x5555AAAA
  MMOVXI MR0, #0xAAAA

  MMOVIZ MR1, #0x5432   ; MR1 = 0x5432FEDC
  MMOVXI MR1, #0xFEDC

  ; 0101 XOR 0101 = 0000 (0)
  ; 0101 XOR 0100 = 0001 (1)
  ; 0101 XOR 0011 = 0110 (6)
  ; 0101 XOR 0010 = 0111 (7)
  ; 1010 XOR 1111 = 0101 (5)
  ; 1010 XOR 1110 = 0100 (4)
  ; 1010 XOR 1101 = 0111 (7)
  ; 1010 XOR 1100 = 0110 (6)

  MXOR32 MR2, MR1, MR0  ; MR3 = 0x01675476
```

See also

MAND32 MRa, MRb, MRc
MOR32 MRa, MRb, MRc

9.7 Appendix A: CLA and CPU Arbitration

Typically, CLA activity is independent of the CPU activity. Under the circumstance where both the CLA and the CPU are attempting to access memory or a peripheral register within the same interface concurrently, an arbitration procedure will occur. This appendix describes this arbitration.

9.7.1 CLA and CPU Arbitration

Typically, CLA activity is independent of the CPU activity. Under the circumstance where both the CLA and the CPU are attempting to access memory or a peripheral register within the same interface concurrently, an arbitration procedure will occur. The one exception is the ADC result registers which do not create a conflict when read by both the CPU and the CLA simultaneously even if different addresses are accessed. Any combined accesses between the different interfaces, or where the CPU access is outside of the interface that the CLA is accessing do not create a conflict.

The interfaces that can have conflict arbitration are:

- CLA Message RAMs
- CLA Program Memory
- CLA Data RAMs

9.7.1.1 CLA Message RAMs

Message RAMs consist of two blocks. These blocks are for passing data between the main CPU and the CLA. No opcode fetches are allowed from the message RAMs. The two message RAMs have the following characteristics:

- **CLA to CPU Message RAM:**

  The following accesses are allowed:
  - CPU reads
  - CLA reads and writes
  - CPU debug reads and writes

  The following accesses are ignored
  - CPU writes

  Priority of accesses are (highest priority first):
  1. CLA write
  2. CPU debug write
  3. CPU data read, program read, CPU debug read
  4. CLA data read

- **CPU to CLA Message RAM:**

  The following accesses are allowed:
  - CPU reads and writes
  - CLA reads
  - CPU debug reads and writes

  The following accesses are ignored
  - CLA writes

  Priority of accesses are (highest priority first):
  1. CLA read
  2. CPU data write, program write, CPU debug write
  3. CPU data read, CPU debug read
  4. CPU program read

9.7.1.2 CLA Program Memory

The behavior of the program memory depends on the state of the MMEMCFG[PROGE] bit. This bit controls whether the memory is mapped to CLA space or CPU space.

- **MMEMCFG[PROGE] == 0**

  In this case the memory is mapped to the CPU. The CLA will be halted and no tasks shoud be incoming.
  - Any CLA fetch will be treated as an illegal opcode condition as described in Section 9.3.4. This condition will not occur if the proper procedure is followed to map the program memory.
  - CLA reads and writes cannot occur
  - The memory block behaves as any normal SARAM block mapped to CPU memory space.

  Priroty of accesses are (highest priority first):
  1. CPU data write, program write, debug write
  2. CPU data read, program read, debug read
  3. CPU fetch, program read

- **MMEMCFG[PROGE] == 1**

  In this case the memory block is mapped to CLA space. The CPU can only make debug accesses.
  - CLA reads and writes cannot occur
  - CLA fetches are allowed
  - CPU fetches return 0 which is an illegal opcode and will cause an ITRAP interrupt.
  - CPU data reads and program reads return 0
  - CPU data writes and program writes are ignored

  Priroty of accesses are (highest priority first):
  1. CLA fetch
  2. CPU debug write
  3. CPU debug read

NOTE: Because the CLA fetch has higher priority than CPU debug reads, it is possible for the CLA to permanently block debug accesses if the CLA is executing in a loop. This might occur when initially developing CLA code due to a bug. To avoid this issue, the program memory will return all 0x0000 for CPU debug reads (ignore writes) when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access can be performed.

9.7.1.3 CLA Data Memory

There are three independent data memory blocks. The behavior of the data memory depends on the state of the MMEMCFG[RAM0E], MMEMCFG[RAM1E] and MMEMCFG[RAM2E] bits. These bits determine whether the memory blocks are mapped to CLA space or CPU space.

- **MMEMCFG[RAMxE] == 0, MMEMCFG[RAMxCPUE] = 0/1**

  In this case the memory block is mapped to the CPU.
  - CLA fetches cannot occur to this block.
  - CLA reads return 0.
  - CLA writes are ignored.
  - The memory block behaves as any normal SARAM block mapped to the CPU memory space.

  Priroty of accesses are (highest priority first):
  1. CPU data write/program write/debug access write
  2. CPU data read/debug access read
  3. CPU fetch/program read
- **MMEMCFG[RAMxE] == 1, MMEMCFG[RAMxCPUE] = 0**

  In this case the memory block is mapped to CLA space. The CPU can make only debug accesses.
  - CLA fetches cannot occur to this block.
  - CLA read and CLA writes are allowed.
  - CPU fetches return 0
  - CPU data reads and program reads return 0.
  - CPU data writes and program writes are ignored.

  Priority of accesses are (highest priority first):
  1. CLA data write
  2. CPU debug write
  3. CPU debug read
  4. CLA read
- **MMEMCFG[RAMxE] == 1, MMEMCFG[RAMxCPUE] = 1**

  In this case the memory block is mapped to CLA space. The CPU has read and write access to the memory in addition to debug accesses.
  - CLA fetches cannot occur to this block.
  - CLA read and CLA writes are allowed.
  - CPU fetches return 0
  - CPU data reads and writes are allowed.
  - CPU program reads return 0 while program writes are ignored.

  Priority of accesses are (highest priority first):
  1. CLA data write
  2. CPU debug access write/CPU data write
  3. CPU debug access read/ CPU data read
  4. CLA read

9.7.1.4 Peripheral Registers (ePWM, HRPWM, Comparator, eCAP, eQEP)

Accesses to the registers follow these rules:

- If both the CPU and CLA request access at the same time, then the CLA will have priority and the main CPU is stalled.
- If a CPU access is in progress and another CPU access is pending, then the CLA will have priority over the pending CPU access. In this case the CLA access will begin when the current CPU access completes.

- While a CPU access is in progress any incoming CLA access will be stalled.
- While a CLA access is in progress any incoming CPU access will be stalled.
- A CPU write operation has priority over a CPU read operation.
- A CLA write operation has priority over a CLA read operation.
- If the CPU is performing a read-modify-write operation and the CLA performs a write to the same location, the CLA write may be lost if the operation occurs in-between the CPU read and write. For this reason, you should not mix CPU and CLA accesses to same location.

Chapter 10

SPRUH18E – January 2011 – Revised March 2014

Viterbi, Complex Math and CRC Unit (VCU)

The C28x Viterbi, Complex Math and CRC Unit (VCU) is a fully programmable block which accelerates the performance of communications-based algorithms by up to a factor of 8X over C28x alone. In addition to eliminating the need for a second processor to manage the communications link, the performance gains of the VCU provides headroom for future system growth and higher bit rates or, conversely, enables devices to operate at a lower MHz to reduce system cost and power consumption. This document provides an overview of the architectural structure and instruction set of the C28x VCU.

The VCU module described in this reference guide is a Type 0 VCU. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566) for a list of all devices with a VCU module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type. This document describes the architecture, pipeline, instruction set, and interrupts of the C28x+VCU.

Topic **Page**

10.1 Overview

The C28x with VCU (C28x+VCU) processor extends the capabilities of the C28x fixed-point or floating-point CPU by adding registers and instructions to support the following algorithm types:

- **Viterbi decoding**

  Viterbi decoding is commonly used in baseband communications applications. The viterbi decode algorithm consists of 3 main parts: branch metric calculations, compare-select (viterbi butterfly) and a traceback operation. Table 10-1 shows a summary of the VCU performance for each of these operations.

Table 10-1. Viterbi Decode Performance

Viterbi Operation	VCU Cycles
Branch Metric Calculation (code rate = 1/2)	1
Branch Metric Calculation (code rate = 1/3)	2p
Viterbi Butterfly (add-compare-select)	2 [(1)]
Traceback per Stage	3 [(2)]

(1) C28x CPU takes 15 cycles per butterfly.
(2) C28x CPU takes 22 cycles per stage.

- **Cyclic redundancy check (CRC)**

  CRC algorithms provide a straightforward method for verifying data integrity over large data blocks, communication packets, or code sections. The C28x+VCU can perform 8-, 16-, and 32-bit CRCs. For example, the VCU can compute the CRC for a block length of 10 bytes in 10 cycles. A CRC result register contains the current CRC which is updated whenever a CRC instruction is executed.

- **Complex math**

  Complex math is used in many applications. The VCU A few of which are:

  – Fast fourier transform (FFT)

  The complex FFT is used in spread spectrum communications, as well in many signal processing algorithms.

  – Complex filters

  Complex filters improve data reliability, transmission distance, and power efficiency. The C28x+VCU can perform a complex I and Q multiple with coefficients (four multiplies) in a single cycle. In addition, the C28x+VCU can read/write the real and imaginary parts of 16-bit complex data to memory in a single cycle.

  Table 10-2 shows a summary of the VCU operations enabled by the VCU:

Table 10-2. Complex Math Performance

Complex Math Operation	VCU Cycles	Notes
Add Or Subtract	1	32 +/- 32 = 32-bit (Useful for filters)
Add or Subtract	1	16 +/- 32 = 15-bit (Useful for FFT)
Multiply	2p	16 x 16 = 32-bit
Multiply & Accumulate (MAC)	2p	32 + 32 = 32-bit, 16 x 16 = 32-bit
RPT MAC	2p+N	Repeat MAC. Single cycle after the first operation.

This C28x+VCU draws from the best features of digital signal processing; reduced instruction set computing (RISC); and microcontroller architectures, firmware, and tool sets. The C2000 features include a modified Harvard architecture and circular addressing. The RISC features are single-cycle instruction execution, register-to-register operations, and modified Harvard architecture (usable in Von Neumann mode). The microcontroller features include ease of use through an intuitive instruction set, byte packing and unpacking, and bit manipulation. The modified Harvard architecture of the CPU enables instruction and data fetches to be performed in parallel. The CPU can read instructions and data while it writes data simultaneously to maintain the single-cycle instruction operation across the pipeline. The CPU does this over six separate address/data buses.

Throughout this document the following notations are used:

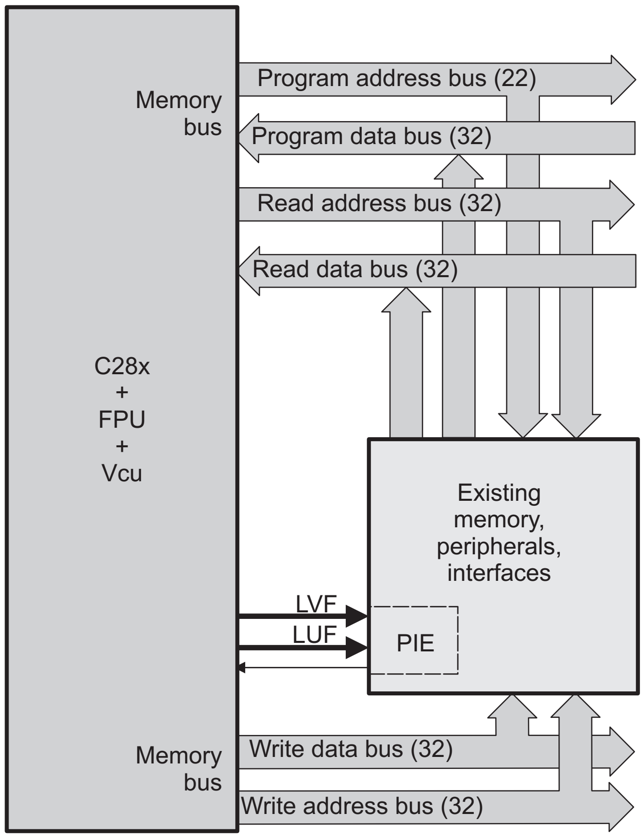
- C28x refers to the C28x fixed-point CPU.
- C28x plus Floating-Point and C28x+FPU both refer to the C28x CPU with enhancements to support IEEE single-precision floating-point operations.
- C28x plus VCU and C28x+VCU both refer to the C28x CPU with enhancements to support viterbi decode, complex math and CRC.
- Some devices have both the FPU and the VCU. These are refered to as C28x+FPU+VCU.

10.2 Components of the C28x plus VCU

The VCU extends the capabilities of the C28x CPU and C28x+FPU processors by adding additional instructions. No changes have been made to existing instructions, pipeline, or memory bus architecture. Therefore, programs written for the C28x are completely compatible with the C28x+VCU. All of the features of the C28x documented in TMS320C28x DSP CPU and Instruction Set Reference Guide (literature number SPRU430) apply to the C28x+VCU. All features documented in the TMS320C28x Floating Point Unit and Instruction Set Reference Guide (SPRUE02) apply to the C28x+FPU+VCU.

Figure 10-1shows the block diagram of the VCU.

Figure 10-1. C28x + VCU Block Diagram



The C28x+VCU contains the same features as the C28x fixed-point CPU:

- A central processing unit for generating data and program-memory addresses; decoding and executing instructions; performing arithmetic, logical, and shift operations; and controlling data transfers among CPU registers, data memory, and program memory.
- Emulation logic for monitoring and controlling various parts and functions of the device and for testing device operation. This logic is identical to that on the C28x fixed-point CPU.
- Signals for interfacing with memory and peripherals, clocking and controlling the CPU and the emulation logic, showing the status of the CPU and the emulation logic, and using interrupts. This logic is identical to the C28x fixed-point CPU.
- Arithmetic logic unit (ALU). The 32-bit ALU performs 2s-complement arithmetic and Boolean logic

operations.
- Address register arithmetic unit (ARAU). The ARAU generates data memory addresses and increments or decrements pointers in parallel with ALU operations.
- Fixed-Point instructions are pipeline protected. This pipeline for fixed-point instructions is identical to that on the C28x fixed-point CPU. The CPU implements an 8-phase pipeline that prevents a write to and a read from the same location from occurring out of order.
- Barrel shifter. This shifter performs all left and right shifts of fixed-point data. It can shift data to the left by up to 16 bits and to the right by up to 16 bits.
- Fixed-Point Multiplier. The multiplier performs 32-bit × 32-bit 2s-complement multiplication with a 64-bit result. The multiplication can be performed with two signed numbers, two unsigned numbers, or one signed number and one unsigned number.

The VCU adds the following features:
- Instructions to support Cyclic Redundancy Check (CRC) or a polynomial code checksum
  – CRC8
  – CRC16
  – CRC32
- Clocked at the same rate as the main CPU (SYSCLKOUT).
- Instructions to support a software implementation of a Viterbi Decoder
  – Branch metrics calculations
  – Add-Compare Select or Viterbi Butterfly
  – Traceback
- Complex Math Arithmetic Unit
  – Add or Subtract
  – Multiply
  – Multiply and Accumulate (MAC)
  – Repeat MAC (RPT || MAC).
- Independent register space. These registers function as source and destination registers for VCU instructions.
- Some VCU instructions require pipeline alignment. This alignment is done through software to allow the user to improve performance by taking advantage of required delay slots. See Section 10.4 for more information.

Devices with the floatint-point unit also include:
- Floating point unit (FPU). The 32-bit FPU performs IEEE single-precision floating-point operations.
- Dedicated floating-point registers.

10.2.1 Emulation Logic

The emulation logic is identical to that on the C28x fixed-point CPU. This logic includes the following features. For more details about these features, refer to the TMS320C28x DSP CPU and Instruction Set Reference Guide (literature number SPRU430):
- Debug-and-test direct memory access (DT-DMA). A debug host can gain direct access to the content of registers and memory by taking control of the memory interface during unused cycles of the instruction pipeline
- A counter for performance benchmarking.
- Multiple debug events. Any of the following debug events can cause a break in program execution:
  – A breakpoint initiated by the ESTOP0 or ESTOP1 instruction.
  – An access to a specified program-space or data-space location. When a debug event causes the C28x to enter the debug-halt state, the event is called a break event.
- Real-time mode of operation.

10.2.2 Memory Map

Like the C28x, the C28x+VCU uses 32-bit data addresses and 22-bit program addresses. This allows for a total address reach of 4G words (1 word = 16 bits) in data space and 4M words in program space. Memory blocks on all C28x+VCU designs are uniformly mapped to both program and data space. For specific details about each of the map segments, see the data manual for a particular device device.

10.2.3 CPU Interrupt Vectors

The C28x+VCU interrupt vectors are identical to those on the C28x CPU. Sixty-four addresses in program space are set aside for a table of 32 CPU interrupt vectors. For more information about the CPU vectors, see TMS320C28x CPU and Instruction Set Reference Guide (literature number SPRU430). Typically the CPU interrupt vectors are only used during the boot up of the device by the boot ROM. Once an application has taken control it should initalize and enable the peripheral interrupt expansion block (PIE).

10.2.4 Memory Interface

The C28x+VCU memory interface is identical to that on the C28x. The C28x+VCU memory map is accessible outside the CPU by the memory interface, which connects the CPU logic to memories, peripherals, or other interfaces. The memory interface includes separate buses for program space and data space. This means an instruction can be fetched from program memory while data memory is being accessed. The interface also includes signals that indicate the type of read or write being requested by the CPU. These signals can select a specified memory block or peripheral for a given bus transaction. In addition to 16-bit and 32-bit accesses, the CPU supports special byte-access instructions that can access the least significant byte (LSByte) or most significant byte (MSByte) of an addressed word. Strobe signals indicate when such an access is occurring on a data bus.

10.2.5 Address and Data Buses

Like the C28x, the memory interface has three address buses:

- PAB: Program address bus: The 22-bit PAB carries addresses for reads and writes from program space.
- DRAB: Data-read address bus: The 32-bit DRAB carries addresses for reads from data space.
- DWAB: Data-write address bus: The 32-bit DWAB carries addresses for writes to data space.

The memory interface also has three data buses:

- PRDB: Program-read data bus: The 32-bit PRDB carries instructions during reads from program space.
- DRDB: Data-read data bus: The 32-bit DRDB carries data during reads from data space.
- DWDB: Data-/Program-write data bus: The 32-bit DWDB carries data during writes to data space or program space.

A program-space read and a program-space write cannot happen simultaneously because both use the PAB. Similarly, a program-space write and a data-space write cannot happen simultaneously because both use the DWDB. Transactions that use different buses can happen simultaneously. For example, the CPU can read from program space (using PAB and PRDB), read from data space (using DRAB and DRDB), and write to data space (using DWAB and DWDB) at the same time. This behavior is identical to the C28x CPU.

10.2.6 Alignment of 32-Bit Accesses to Even Addresses

The C28x+VPU expects memory wrappers or peripheral-interface logic to align any 32-bit read or write to an even address. If the address-generation logic generates an odd address, the CPU will begin reading or writing at the previous even address. This alignment does not affect the address values generated by the address-generation logic.

Most instruction fetches from program space are performed as 32-bit read operations and are aligned accordingly. However, alignment of instruction fetches are effectively invisible to a programmer. When instructions are stored to program space, they do not have to be aligned to even addresses. Instruction boundaries are decoded within the CPU.

You need to be concerned with alignment when using instructions that perform 32-bit reads from or writes to data space.

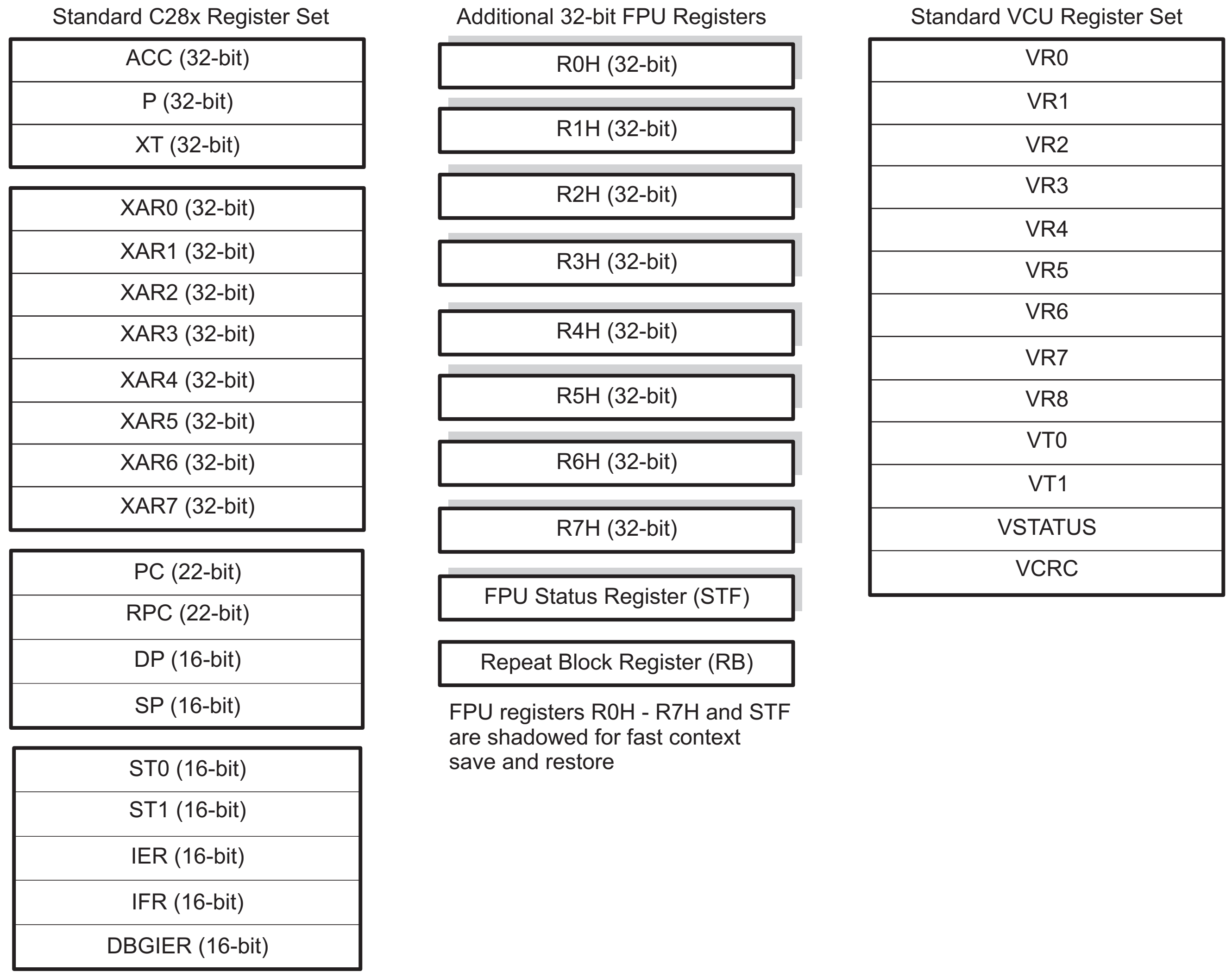
10.3 Register Set

Devices with the C28x+VCU include the standard C28x register set plus an additional set of VCU specific registers. The additional VCU registers are the following:

- Result registers: VR0, VR1... VR8
- Traceback registers: VT0, VT1
- Configuraiton and status register: VSTATUS
- CRC result register: VCRC
- Repeat block register: RB

Figure 10-2 shows the register sets for the 28x CPU, the FPU and the VCU. The following section discusses the VCU register set in detail.

Figure 10-2. C28x + FPU + VCU Registers



10.3.1 VCU Register Set

The table below describes the VCU module register set. The last three columns indicate whether the particular module within the VCU can make use of the register.

Table 10-3. VCU Register Set

Register Name	Size	Description	Viterbi	Complex Math	CRC
VR0	32-bits	General purpose register 0	Yes	Yes	No
VR1	32-bits	General purpose register 1	Yes	Yes	No
VR2	32-bits	General purpose register 2	Yes	Yes	No
VR3	32-bits	General purpose register 3	Yes	Yes	No
VR4	32-bits	General purpose register 4	Yes	Yes	No
VR5	32-bits	General purpose register 5	Yes	Yes	No
VR6	32-bits	General purpose register 6	Yes	Yes	No
VR7	32-bits	General purpose register 7	Yes	Yes	No
VR8	32-bits	General purpose register 8	Yes	No	No
VT0	32-bits	32-bit transition bit register 0	Yes	No	No
VT1	32-bits	32-bit transition bit register 1	Yes	No	No
VSTATUS	32-bits	VCU status and configuration register [1]	Yes	Yes	No
VCRC	32-bits	Cyclic redundancy check (CRC) result register	No	No	Yes

[1] Debugger writes are not allowed to the VSTATUS register.

Table 10-4 lists the CPU registers available on devices with the C28x, the C28x+FPU, the C28x+VCU and the C28x+FPU+VCU.

Table 10-4. 28x CPU Register Summary

Register	C28x CPU	C28x+FPU	C28x+VCU	C28x+FPU+VCU	Description
ACC	Yes	Yes	Yes	Yes	Fixed-point accumulator
AH	Yes	Yes	Yes	Yes	High half of ACC
AL	Yes	Yes	Yes	Yes	Low half of ACC
XAR0 - XAR7	Yes	Yes	Yes	Yes	Auxiliary register 0 - 7
AR0 - AR7	Yes	Yes	Yes	Yes	Low half of XAR0 - XAR7
DP	Yes	Yes	Yes	Yes	Data-page pointer
IFR	Yes	Yes	Yes	Yes	Interrupt flag register
IER	Yes	Yes	Yes	Yes	Interrupt enable register
DBGIER	Yes	Yes	Yes	Yes	Debug interrupt enable register
P	Yes	Yes	Yes	Yes	Fixed-point product register
PH	Yes	Yes	Yes	Yes	High half of P
PL	Yes	Yes	Yes	Yes	Low half of P
PC	Yes	Yes	Yes	Yes	Program counter
RPC	Yes	Yes	Yes	Yes	Return program counter
SP	Yes	Yes	Yes	Yes	Stack pointer
ST0	Yes	Yes	Yes	Yes	Status register 0
ST1	Yes	Yes	Yes	Yes	Status register 1
XT	Yes	Yes	Yes	Yes	Fixed-point multiplicand register
T	Yes	Yes	Yes	Yes	High half of XT
TL	Yes	Yes	Yes	Yes	Low half of XT
ROH - R7H	No	Yes	No	Yes	Floating-point Unit result registers
STF	No	Yes	No	Yes	Floating-point Uint status register
RB	No	Yes	Yes	Yes	Repeat block register
VR0 - VR8	No	No	Yes	Yes	VCU general purpose registers
VT0, VT1	No	No	Yes	Yes	VCU transition bit register 0 and 1
VSTATUS	No	No	Yes	Yes	VCU status and configuration
VCRC	No	No	Yes	Yes	CRC result register

10.3.2 VCU Status Register (VSTATUS)

The VCU status register (VSTATUS) register is described in Figure 10-3. There is no single instruction to directly transfer the VSTATUS register to a C28x register. To transfer the contents:

1. Store VSTATUS into memory using VMOV32 mem32, VSTATUS instruction
2. Load the value from memory into a main C28x CPU register.

Configuration bits within the VSTATUS registers are set or cleared using VCU instructions.

Figure 10-3. VCU Status Register (VSTATUS)

31-16
Reserved
R-0

R/W-0

15	14	13	12	11	10	9-5	4-0
Reserved		OVRI	OVFR	RND	SAT	SHIFTL	SHIFTR
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 10-5. VCU Status (VSTATUS) Register Field Descriptions

Bits	Field	Value	Description
31 - 14	Reserved	0	Reserved for future use
13	OVFI		Overflow or Underflow Flag: Imaginary Part
		0	No overflow or underflow has been detected.
		1	Indiates an overflow or underflow has occurred during the computation of the imaginary part of operations shown in Table 10-6. This bit will be set regardless of the value of the VSTATUS[SAT] bit. OVRI bit will remain set until it is cleared by executing the VCLROVFI instruction.
12	OVFR		Overflow or Underflow Flag: Real Part
		0	No overflow or underflow has been detected.
		1	Indicates overflow or underflow has occurred during a real number calculation for operations shown in Table 10-6. This bit will be set regardless of the value of the VSTATUS[SAT] bit. This bit will remain set until it is cleared by executing the VCLROVFR instruction.
11	RND		Rounding When a right-shift operation is performed the lower bits of the value will be lost. The RND bit determines if the shifted value is rounded or if the shifted-out bits are simply truncated. This is described in . Operations which use right-shift and rounding are shown in Table 10-6. The RND bit is set by the VRNDON instruction and cleared by the VRNDOFF instruction.
		0	Rounding is not performed. Bits shifted out right are truncated.
		1	Rounding is performed. Refer to the instruction descriptions for information on how the operation is affected by the RND bit.
10	SAT		Saturation This bit determines whether saturation will be performed for operations shown in Table 10-6. The SAT bit is set by the VSATON instruction and is cleared by the VSATOFF instruction.
		0	No saturation is performed.
		1	Saturation is performed.
9-5	SHIFTL		Left Shift Operations which use left-shift are shown in Table 10-6 The shift SHIFTL field can be set or cleared by the VSETSHL instruction.
		0	No left shift.
		0x01 - 0x1F	Refer to the instruction description for information on how the operation is affected by the shift value. During the left-shift, the lower bits are filled with 0's.
4-0	SHIFTR		Right Shift Operations which use right-shift and rounding are shown in Table 10-6. The shift SHIFTR field can be set or cleared by the VSETSHR instruction.
		0	No right shift.
		0x01 - 0x1F	Refer to the instruction descriptions for information on how the operation is affected by the shift value. During the right-shift, the lower bits are lost, and the shifted value is sign extended. If rounding is enabled (VSTATUS[RND] == 1) , then the value will be rounded instead of truncated.

Table 10-6 shows a summary of the operations that are affected by or modify bits in the VSTATUS register.

Table 10-6. Operation Interaction with VSTATUS Bits

Operation [(1)]	Description	OVFI	OVFR	RND	SAT	SHIFT L	SHIFT R
VITDLADDSUB	Viterbi Add and Subtract Low	-	Y	-	Y	-	-
VITDHADDSUB	Viterbi Add and Subtract High	-	Y	-	Y	-	-
VITDLSUBADD	Viterbi Subtract and Add Low	-	Y	-	Y	-	-
VITDHSUBADD	Viterbi Subtract and Add High	-	Y	-	Y	-	-
VITBM2	Viterbi Branch Metric CR 1/2	-	Y	-	Y	-	-
VITBM3	Viterbi Branch Metric CR 1/3	-	Y	-	Y	-	-
VCADD	Complex 32 + 32 = 32	Y	Y	Y	Y	-	Y

(1) Some parallel instructions also include these operations. In this case, the operation will also modify, or be affected by, VSTATUS bits as when used as part of a parallel instruction.

Table 10-6. Operation Interaction with VSTATUS Bits (continued)

Operation [(1)]	Description	OVFI	OVFR	RND	SAT	SHIFT L	SHIFT R
VCDADD16	Complex 16 + 32 = 32	Y	Y	Y	Y	Y	Y
VCDSUB16	Complex 16 - 32 = 32	Y	Y	Y	Y	Y	Y
VCMAC	Complex 32 + 32 = 32, 16 x 16 = 32	Y	Y	Y	Y	-	Y
VCMPY	Complex 16 x 16 = 32	Y	Y		Y	-	-
VCSUB	Complex 32 -32 = 32	Y	Y	Y	Y	-	Y

10.3.3 Repeat Block Register (RB)

The repeat block instruction (RPTB) applies to devices with the C28x+FPU and the C28x+VCU. This instruction allows you to repeat a block of code as shown in Example 10-1.

Example 10-1. The Repeat Block (RPTB) Instruction uses the RB Register

```
; find the largest element and put its address in XAR6
;
; This example makes use of floating-point (C28x + FPU) instructions
;
;
    MOV32 R0H, *XAR0++;
    .align 2                 ; Aligns the next instruction to an even address
    NOP                      ; Makes RPTB odd aligned - required for a block size of 8
    RPTB VECTOR_MAX_END, AR7 ; RA is set to 1
    MOVL ACC,XAR0
    MOV32 R1H,*XAR0++        ; RSIZE reflects the size of the RPTB block
    MAXF32 R0H,R1H           ; in this case the block size is 8
    MOVST0 NF,ZF
    MOVL XAR6,ACC,LT
VECTOR_MAX_END:              ; RE indicates the end address. RA is cleared
```

The C28x FPU or VCU automatically populates the RB register based on the execution of a RPTB instruction. This register is not normally read by the application and does not accept debugger writes.

Figure 10-4. Repeat Block Register (RB)

31	30	29-23	22-16
RAS	RA	RSIZE	RE
R-0	R-0	R-0	R-0

15-0
RC
R-0

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

Table 10-7. Repeat Block (RB) Register Field Descriptions

Bits	Field	Value	Description
31	RAS		Repeat Block Active Shadow Bit
			When an interrupt occurs the repeat active, RA, bit is copied to the RAS bit and the RA bit is cleared. When an interrupt return instruction occurs, the RAS bit is copied to the RA bit and RAS is cleared.
		0	A repeat block was not active when the interrupt was taken.
		1	A repeat block was active when the interrupt was taken.
30	RA		Repeat Block Active Bit
		0	This bit is cleared when the repeat counter, RC, reaches zero.
			When an interrupt occurs the RA bit is copied to the repeat active shadow, RAS, bit and RA is cleared. When an interrupt return, IRET, instruction is executed, the RAS bit is copied to the RA bit and RAS is cleared.
		1	This bit is set when the RPTB instruction is executed to indicate that a RPTB is currently active.
29-23	RSIZE		Repeat Block Size
			This 7-bit value specifies the number of 16-bit words within the repeat block. This field is initialized when the RPTB instruction is executed. The value is calculated by the assembler and inserted into the RPTB instruction's RSIZE opcode field.
		0-7	Illegal block size.
		8/9-0x7F	A RPTB block that starts at an even address must include at least 9 16-bit words and a block that starts at an odd address must include at least 8 16-bit words. The maximum block size is 127 16-bit words. The codegen assembler will check for proper block size and alignment.

Table 10-7. Repeat Block (RB) Register Field Descriptions (continued)

Bits	Field	Value	Description
22-16	RE		Repeat Block End Address This 7-bit value specifies the end address location of the repeat block. The RE value is calculated by hardware based on the RSIZE field and the PC value when the RPTB instruction is executed. RE = lower 7 bits of (PC + 1 + RSIZE)
15-0	RC		Repeat Count
		0	The block will not be repeated; it will be executed only once. In this case the repeat active, RA, bit will not be set.
		1-0xFFFF	This 16-bit value determines how many times the block will repeat. The counter is initialized when the RPTB instruction is executed and is decremented when the PC reaches the end of the block. When the counter reaches zero, the repeat active bit is cleared and the block will be executed one more time. Therefore the total number of times the block is executed is RC+1.

10.4 Pipeline

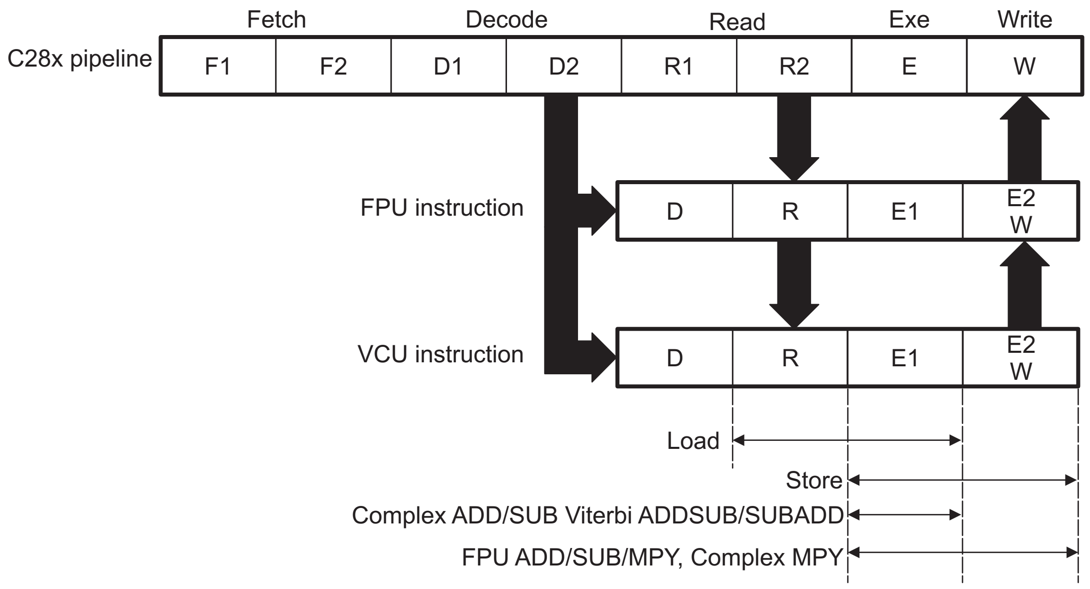
This section describes the VCU pipeline stages and presents cases where pipeline alignment must be considered.

10.4.1 Pipeline Overview

The C28x VCU pipeline is identical to the C28x pipeline for all standard C28x instructions. In the decode2 stage (D2), it is determined if an instruction is a C28x instruction, a FPU instruction, or a VCU instruction. The pipeline flow is shown in Figure 10-5.

Notice that stalls due to normal C28x pipeline stalls (D2) and memory waitstates (R2 and W) will also stall any C28x VCU instruction. Most C28x VCU instructions are single cycle and will complete in the VCU E1 or W stage which aligns to the C28x pipeline. Some instructions will take an additional execute cycle (E2). For these instructions you must wait a cycle for the result from the instruction to be available. The rest of this section will describe when delay cycles are required. Keep in mind that the assembly tools for the C28x+VCU will issue an error if a delay slot has not been handled correctly.

Figure 10-5. C28x + FCU + VCU Pipeline



10.4.2 General Guidelines for Floating-Point Pipeline Alignment

The majority of the VCU instructions do not require any special pipeline considerations. This section lists the few operations that do require special consideration.

While the C28x+VCU assembler will issue errors for pipeline conflicts, you may still find it useful to understand when software delays are required. This section describes three guidelines you can follow when writing C28x+VCU assembly code.

VCU instructions that require delay slots have a 'p' after their cycle count. For example '2p' stands for 2 pipelined cycles. This means that an instruction can be started every cycle, but the result of the instruction will only be valid one instruction later.

There are three general guidelines to determine if an instruction needs a delay slot:

1. Branch metric calculation for a code rate of 1/3 takes 2p cycles.
2. Complex multiply and MAC take 2p cycles.
3. Everything else does not require a delay slot.

An example of the complex multiply instruction is shown in Example 10-2. VCMPY is a 2p instruction and therefore requires one delay slot. The destination registers for the operation, VR2 and VR3, will be updated one cycle after the instruction for a total of 2 cycles. Therefore, a NOP or instruction that does not use VR2 or VR3 must follow this instruction.

Any memory stall or pipeline stall will also stall the VCU. This keeps the VCU aligned with the C28x pipeline and there is no need to change the code based on the waitstates of a memory block.

Example 10-2. 2p Instruction Pipeline Alignment

```
   VCMPY VR3, VR2, VR1, VR0     ; 2 pipeline cycles (2p)
   NOP                          ; 1 cycle delay or non-conflicting instruction
                                ; <-- VCMPY completes, VR2 and VR3 updated
   NOP                          ; Any instruction
```

10.4.3 Parallel Instructions

Parallel instructions are single opcodes that perform two operations in parallel. The guidelines provided in Section 10.4.2 apply to parallel instructions as well. In this case the cycle count will be given for both operations. For example, a branch metric calculation for code rate of 1/3 with a parallel load takes 2p/1 cycles. This means the branch metric portion of the operation takes 2 pipelined cycles while the move portion of the operation is single cycle. NOPs or other non conflicting instructions must be inserted to align the branch metric calculation portion of the operation as shown in Example 10-4 .

Example 10-3. Branch Metric CR 1/2 Calculation with Parallel Load

```
;  VITBM2 || VMOV32 instruction: branch metrics calculation with parallel load
;  VBITM2 is a 1 cycle operation  (code rate = 1/2)
;  VMOV32 is a 1 cycle operation
;
   VITBM2   VR0                ; Load VR0 with the 2 branch metrics
|| VMOV32   VR2,  @Val         ; VR2 gets the contents of Val
                               ; <-- VMOV32 completes here (VR2 is valid)
                               ; <-- VITBM2 completes here (VR0 is valid)
   <instruction 2>             ; Any instruction, can use VR2 and/or VR0
```

Example 10-4. Branch Metric CR 1/3 Calculation with Parallel Load

```
;  VITBM3 || VMOV32 instruction: branch metrics calculation with parallel load
;  VBITM3 is a 2p cycle operation  (code rate = 1/3)
;  VMOV32 is a 1 cycle operation
;
   VITBM3   VR0, VR1, VR2      ; Load VR0 and VR1 with the 4 branch metrics
|| VMOV32   VR2,  @Val         ; VR2 gets the contents of Val
                               ; <-- VMOV32 completes here (VR2 is valid)
   <instruciton 2>             ; Must not use VR0 or VR1.  Can use VR2.
                               ; <-- VITBM3 completes here (VR0, VR1 are valid)
   <instruction 3>             ; Any instruction, can use VR2 and/or VR0
```

10.4.4 Invalid Delay Instructions

All VCU, FPU and fixed-point instructions can be used in VCU instruction delay slots as long as source and destination register conflicts are avoided. The C28x+VCU assembler will issue an error anytime you use an conflicting instruction within a delay slot. The following guidelines can be used to avoid these conflicts.

NOTE: ***Destination register conflicts in delay slots:***

Any operation used for pipeline alignment delay must not use the same destination register as the instruction requiring the delay. See Example 10-5.

In Example 10-5 the VCMPY instruction uses VR2 and VR3 as its destination registers. The next instruction should not use VR2 or VR3 as a destination. Since the VMOV32 instruction uses the VR3 register a pipeline conflict will be issued by the assembler. This conflict can be resolved by using a register other than VR2 for the VMOV32 instruction as shown in Example 10-6.

Example 10-5. Destination Register Conflict

```
; Invalid delay instruction.
; Both instructions use the same destination register (VR3)
;
   VCMPY  VR3, VR2, VR1, VR0    ; 2p instruction
   VMOV32 VR3, mem32            ; Invalid delay instruction
                                ; <-- VCMPY completes, VR3, VR2 are valid
```

Example 10-6. Destination Register Conflict Resolved

```
; Valid delay instruction
;
   VCMPY  VR3, VR2, VR1, VR0    ; 2p instruction
   VMOV32 VR7, mem32            ; Valid delay instruction
```

NOTE: ***Instructions in delay slots cannot use the instruction's destination register as a source register.***

Any operation used for pipeline alignment delay must not use the destination register of the instruction requiring the delay as a source register as shown in Example 10-7. For parallel instructions, the current value of a register can be used in the parallel operation before it is overwritten as shown in Example 10-9.

In Example 10-7 the VCMPY instruction again uses VR3 and VR2 as its destination registers. The next instruction should not use VR3 or VR2 as its source since the VCMPY will take an additional cycle to complete. Since the VCADD instruction uses the VR2 as a source register a pipeline conflict will be issued by the assembler. The use of VR3 will also cause a pipeline conflict. This conflict can be resolved by using a register other than VR2 or VR3 or by inserting a non-conflicting instruction between the VCMPY and VCADD instructions. Since the VNEG does not use VR2 or VR3 this instruction can be moved before the VCADD as shown in Example 10-8.

Example 10-7. Destination/Source Register Conflict

```
; Invalid delay instruction.
; VCADD should not use VR2 or VR3 as a source operand
;
   VCMPY VR3, VR2, VR1, VR0     ; 2p instruction
   VCADD VR5, VR4, VR3, VR2     ; Invalid delay instruction
   VNEG  VR0                    ; <- VCMPY completes, VR3, VR2 valid
```

Example 10-8. Destination/Source Register Conflict Resolved

```
; Valid delay instruction.
;
   VCMPY VR3, VR2, VR1, VR0     ; 2p instruction
   VNEG  VR0                    ; Non conflicting instruction or NOP
   VCADD VR5, VR4, VR3, VR2     ; <- VCMPY completes, VR3, VR2 valid
```

It should be noted that a source register for the 2nd operation within a parallel instruction can be the same as the destination register of the first operation. This is because the two operations are started at the same time. The 2nd operation is not in the delay slot of the first operation. Consider Example 10-9 where the VCMPY uses VR3 and VR2 as its destination registers. The VMOV32 is the 2nd operation in the instruction and can freely use VR3 or VR2 as a source register. In the example, the contents of VR3 before the multiply will be used by MOV32.

Example 10-9. Parallel Instruction Destination/Source Exception

```
; Valid parallel operation.
;
   VCMPY  VR3,   VR2, VR1, VR0  ; 2p/1 instruction
|| VMOV32 mem32, VR3            ; <-- Uses VR3 before the VCMPY update
                                ; <-- mem32 updated
   NOP                          ; <-- Delay for VCMPY
                                ; <-- VR2, VR3 updated
```

Likewise, the source register for the 2nd operation within a parallel instruction can be the same as one of the source registers of the first operation. The VCMPY operation in Example 10-10 uses the VR0 register as one of its sources. This register is also updated by the VMOV32 instruction. The multiplication operation will use the value in VR0 before the VMOV32 updates it.

Example 10-10. Parallel Instruction Destination/Source Exception

```
; Valid parallel operation.
   VCMPY  VR3, VR2, VR1, VR0  ; 2p/1 instruction
|| VMOV32 VR0, mem32            ; <-- Uses VR3 before the VCMPY update
                                ; <-- mem32 updated
   NOP                          ; <-- Delay for VCMPY
                                ; <-- VR2, VR3 updated
```

NOTE: ***Operations within parallel instructions cannot use the same destination register.***

When two parallel operations have the same destination register, the result is invalid.

For example, see Example 10-11.

If both operations within a parallel instruction try to update the same destination register as shown in Example 10-11 the assembler will issue an error.

Example 10-11. Invalid Destination Within a Parallel Instruction

```
; Invalid parallel instruction. Both operations use VR3 as a destination register
;
   VCMPY  VR3, VR2, VR1, VR0    ; 2p/1 instruction
|| VMOV32 VR3, mem32            ; <-- Invalid
```

10.5 Instruction Set

This section describes the assembly language instructions of the VCU. Also described are parallel operations, conditional operations, resource constraints, and addressing modes. The instructions listed here are independant from C28x and C28x+FPU instruction sets.

10.5.1 Instruction Descriptions

This section gives detailed information on the instruction set. Each instruction may present the following information:

- Operands
- Opcode
- Description
- Exceptions
- Pipeline
- Examples
- See also

The example INSTRUCTION is shown to familiarize you with the way each instruction is described. The example describes the kind of information you will find in each part of the individual instruction description and where to obtain more information. VCU instructions follow the same format as the C28x; the source operand(s) are always on the right and the destination operand(s) are on the left.

The explanations for the syntax of the operands used in the instruction descriptions for the C28x VCU are given in Table 10-8.

Table 10-8. Operand Nomenclature

Symbol	Description
#16FHi	16-bit immediate (hex or float) value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FHiHex	16-bit immediate hex value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FLoHex	A 16-bit immediate hex value that represents the lower 16-bits of an IEEE 32-bit floating-point value
#32Fhex	32-bit immediate value that represents an IEEE 32-bit floating-point value
#32F	Immediate float value represented in floating-point representation
#0.0	Immediate zero
#5-bit	5-bit immediate unsigned value
addr	Opcode field indicating the addressing mode
Im(X), Im(Y)	Imaginary part of the input X or input Y
Im(Z)	Imaginary part of the output Z
Re(X), Re(Y)	Real part of the input X or input Y
Re(Z)	Real part of the output Z
mem16	Pointer (using any of the direct or indirect addressing modes) to a 16-bit memory location
mem32	Pointer (using any of the direct or indirect addressing modes) to a 32-bit memory location
VRa	VR0 - VR8 registers. Some instructions exclude VR8. Refer to the instruction description for details.
VR0H, VR1H...VR7H	VR0 - VR7 registers, high half.
VR0L, VR1L....VR7L	VR0 - VR7 registers, low half.
VT0, VT1	Transition bit register VT0 or VT1.

Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

Table 10-9. INSTRUCTION dest, source1, source2 Short Description

	Description
dest1	Description for the 1st operand for the instruction
source1	Description for the 2nd operand for the instruction
source2	Description for the 3rd operand for the instruction
Opcode	This section shows the opcode for the instruction
Description	Detailed description of the instruction execution is described. Any constraints on the operands imposed by the processor or the assembler are discussed.
Restrictions	Any constraints on the operands or use of the instruction imposed by the processor are discussed.
Pipeline	This section describes the instruction in terms of pipeline cycles as described in Section 10.4
Example	Examples of instruction execution. If applicable, register and memory values are given before and after instruction execution. Some examples are code fragments while other examples are full tasks that assume the VCU is correctly configured and the main CPU has passed it data.
Operands	Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

10.5.2 General Instructions

The instructions are listed alphabetically, preceded by a summary.

Table 10-10. General Instructions

Title	Page
POP RB —Pop the RB Register from the Stack	702
PUSH RB —Push the RB Register onto the Stack	704
RPTB label, loc16 —Repeat A Block of Code	706
RPTB label, #RC —Repeat a Block of Code	708
VCLEAR VRa —Clear General Purpose Register	710
VCLEARALL —Clear All General Purpose and Transition Bit Registers	711
VCLROVFI —Clear Imaginary Overflow Flag	712
VCLROVFR —Clear Real Overflow Flag	713
VMOV16 mem16, VRaL —Store General Purpose Register, Low Half	714
VMOV16 VRaL, mem16 —Load General Purpose Register, Low Half	715
VMOV32 mem32, VRa —Store General Purpose Register	716
VMOV32 mem32, VSTATUS —Store VCU Status Register	717
VMOV32 mem32, VTa —Store Transition Bit Register	718
VMOV32 VRa, mem32 —Load 32-bit General Purpose Register	719
VMOV32 VSTATUS, mem32 —Load VCU Status Register	720
VMOV32 VTa, mem32 —Load 32-bit Transition Bit Register	721
VMOVD32 VRa, mem32 —Load Register with Data Move	722
VMOVIX VRa, #16I —Load Upper Half of a General Purpose Register with I6-bit Immediate	723
VMOVZI VRa, #16I —Load General Purpose Register with Immediate	724
VMOVXI VRa, #16I —Load Low Half of a General Purpose Register with Immediate	725
VRNDOFF —Disable Rounding	726
VRNDON —Enable Rounding	727
VSATOFF —Disable Saturation	728
VSATON —Enable Saturation	729
VSETSHL #5-bit —Initialize the Left Shift Value	730
VSETSHR #5-bit —Initialize the Left Shift Value	731

POP RB *Pop the RB Register from the Stack*

Operands

RB	repeat block register

Opcode

```
LSW: 1111 1111 1111 0001
```

Description

Restore the RB register from stack. If a high-priority interrupt contains a RPTB instruction, then the RB register must be stored on the stack before the RPTB block and restored after the RTPB block. In a low-priority interrupt RB must always be saved and restored. This save and restore must occur when interrupts are disabled.

Flags

This instruction does not affect any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not have to save the RB register.

```
; Repeat Block within a High-Priority Interrupt (Non-Interruptible)
;
; Interrupt:              ; RAS = RA, RA = 0
   ...
   PUSH RB                ; Save RB register only if a RPTB block is used in the ISR
   ...
   ...
   RPTB _BlockEnd, AL   ; Execute the block AL+1 times
   ...
   ...
   ...
_BlockEnd                 ; End of block to be repeated
   ...
   ...
   POP RB                 ; Restore RB register ...
   IRET                   ; RA = RAS, RAS = 0
```

A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled.

```
; Repeat Block within a Low-Priority Interrupt (Interruptible)
;
; Interrupt:
                          ; RAS = RA, RA = 0
   ...
   PUSH RB                ; Always save RB register
   ...
   CLRC INTM              ; Enable interrupts only after saving RB
   ...
   ...
   ...
; ISR may or may not include a RPTB block
   ...
   ...
   SETC INTM              ; Disable interrupts before restoring RB
   ...
   POP RB                 ; Always restore RB register
   ...
   IRET                   ; RA = RAS, RAS = 0
```

See also PUSH RB

RPTB label, loc16
RPTB label, #RC

PUSH RB *Push the RB Register onto the Stack*

Operands

RB	repeat block register

Opcode `LSW: 1111 1111 1111 0000`

Description Save the RB register on the stack. If a high-priority interrupt contains a RPTB instruction, then the RB register must be stored on the stack before the RPTB block and restored after the RTPB block. In a low-priority interrupt RB must always be saved and restored. This save and restore must occur when interrupts are disabled.

Flags This instruction does not affect any flags in the VSTATUS register.

Pipeline This is a single-cycle instruction.

Example A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not have to save the RB register.

```
; Repeat Block within a High-Priority Interrupt (Non-Interruptible)
;
; Interrupt:            ; RAS = RA, RA = 0
   ...
   PUSH RB              ; Save RB register only if a RPTB block is used in the ISR
   ...
   ...
   RPTB _BlockEnd, AL  ; Execute the block AL+1 times
   ...
   ...
   ...
_BlockEnd               ; End of block to be repeated
   ...
   ...
   POP RB               ; Restore RB register ...
   IRET                 ; RA = RAS, RAS = 0
```

A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled.

```
; Repeat Block within a Low-Priority Interrupt (Interruptible)
;
; Interrupt:
                        ; RAS = RA, RA = 0
   ...
   PUSH RB              ; Always save RB register
   ...
   CLRC INTM            ; Enable interrupts only after saving RB
   ...
   ...
   ...
; ISR may or may not include a RPTB block
   ...
   ...
   SETC INTM            ; Disable interrupts before restoring RB
   ...
   POP RB               ; Always restore RB register
   ...
   IRET                 ; RA = RAS, RAS = 0
```

See also POP RB

RPTB label, loc16
RPTB label, #RC

RPTB label, loc16 *Repeat A Block of Code*

Operands

label	This label is used by the assembler to determine the end of the repeat block and to calculate RSIZE. This label should be placed immediately after the last instruction included in the repeat block.
loc16	16-bit location for the repeat count value.

Opcode

```
LSW: 1011 0101 0bbb bbbb
MSW: 0000 0000   loc16
```

Description

Initialize repeat block loop, repeat count from [loc16]

Restrictions

- The maximum block size is ≤127 16-bit words.
- An even aligned block must be ≥ 9 16-bit words.
- An odd aligned block must be ≥ 8 16-bit words.
- Interrupts must be disabled when saving or restoring the RB register.
- Repeat blocks cannot be nested.
- Any discontinuity type operation is not allowed inside a repeat block. This includes all call, branch or TRAP instructions. Interrupts are allowed.
- Conditional execution operations are allowed.

Flags

This instruction does not affect any flags in the VSTATUS register.

Pipeline

This instruction takes four cycles on the first iteration and zero cycles thereafter. No special pipeline alignment is required.

Example

The minimum size for the repeat block is 8 words if the block is even aligned and 9 words if the block is odd aligned. If you have a block of 8 words, as in the following example, you can make sure the block is odd aligned by proceeding it by a .align 2 directive and a NOP instruction. The .align 2 directive will make sure the NOP is even aligned. Since a NOP is a 16-bit instruction the RPTB will be odd aligned. For blocks of 9 or more words, this is not required.

```
; Repeat Block of 8 Words (Interruptible)
;
; Note: This example makes use of floating-point (C28x+FPU) instructions
;
;
; find the largest element and put its address in XAR6
   .align 2
   NOP
   RPTB _VECTOR_MAX_END, AR7
; Execute the block AR7+1 times
   MOVL ACC,XAR0 MOV32 R1H,*XAR0++    ; min size = 8, 9 words
   MAXF32 R0H,R1H                     ; max size = 127 words
   MOVST0 NF,ZF
   MOVL XAR6,ACC,LT
_VECTOR_MAX_END:                      ; label indicates the end
                                      ; RA is cleared
```

When an interrupt is taken the repeat active (RA) bit in the RB register is automatically copied to the repeat active shadow (RAS) bit. When the interrupt exits, the RAS bit is automatically copied back to the RA bit. This allows the hardware to keep track if a repeat loop was active whenever an interrupt is taken and restore that state automatically.

A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not have to save the RB register.

```
; Repeat Block within a High-Priority Interrupt (Non-Interruptible)
;
; Interrupt:              ; RAS = RA, RA = 0
   ...
   PUSH RB                ; Save RB register only if a RPTB block is used in the ISR
   ...
   ...
   RPTB _BlockEnd, AL    ; Execute the block AL+1 times
   ...
   ...
   ...
_BlockEnd                 ; End of block to be repeated
   ...
   ...
   POP RB                 ; Restore RB register ...
   IRET                   ; RA = RAS, RAS = 0
```

A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled.

```
; Repeat Block within a Low-Priority Interrupt (Interruptible)
;
; Interrupt:
                          ; RAS = RA, RA = 0
   ...
   PUSH RB                ; Always save RB register
   ...
   CLRC INTM              ; Enable interrupts only after saving RB
   ...
   ...
   ...
; ISR may or may not include a RPTB block
   ...
   ...
   SETC INTM              ; Disable interrupts before restoring RB
   ...
   POP RB                 ; Always restore RB register
   ...
   IRET                   ; RA = RAS, RAS = 0
```

See also

POP RB
PUSH RB
RPTB label, #RC

RPTB label, #RC *Repeat a Block of Code*

Operands

label	This label is used by the assembler to determine the end of the repeat block and to calculate RSIZE. This label should be placed immediately after the last instruction included in the repeat block.
#RC	16-bit immediate value for the repeat count.

Opcode

```
LSW: 1011 0101 1bbb bbbb
MSW: cccc cccc cccc cccc
```

Description Repeat a block of code. The repeat count is specified as a immediate value.

Restrictions

- The maximum block size is ≤127 16-bit words.
- An even aligned block must be ≥ 9 16-bit words.
- An odd aligned block must be ≥ 8 16-bit words.
- Interrupts must be disabled when saving or restoring the RB register.
- Repeat blocks cannot be nested.
- Any discontinuity type operation is not allowed inside a repeat block. This includes all call, branch or TRAP instructions. Interrupts are allowed.
- Conditional execution operations are allowed.

Flags This instruction does not affect any flags in the VSTATUS register.

Pipeline This instruction takes one cycle on the first iteration and zero cycles thereafter. No special pipeline alignment is required.

Example The minimum size for the repeat block is 8 words if the block is even aligned and 9 words if the block is odd aligned. If you have a block of 8 words, as in the following example, you can make sure the block is odd aligned by proceeding it by a .align 2 directive and a NOP instruction. The .align 2 directive will make sure the NOP is even aligned. Since a NOP is a 16-bit instruction the RPTB will be odd aligned. For blocks of 9 or more words, this is not required.

```
; Repeat Block of 8 Words (Interruptible)
;
; Note: This example makes use of floating-point (C28x+FPU) instructions
;
; find the largest element and put its address in XAR6
;
   .align 2
   NOP
   RPTB _VECTOR_MAX_END, AR7
; Execute the block AR7+1 times
   MOVL ACC,XAR0 MOV32 R1H,*XAR0++   ; min size = 8, 9 words
   MAXF32 R0H,R1H                    ; max size = 127 words
   MOVST0 NF,ZF
   MOVL XAR6,ACC,LT
_VECTOR_MAX_END:                     ; label indicates the end
                                     ; RA is cleared
```

When an interrupt is taken the repeat active (RA) bit in the RB register is automatically copied to the repeat active shadow (RAS) bit. When the interrupt exits, the RAS bit is automatically copied back to the RA bit. This allows the hardware to keep track if a repeat loop was active whenever an interrupt is taken and restore that state automatically.

A high priority interrupt is defined as an interrupt that cannot itself be interrupted. In a high priority interrupt, the RB register must be saved if a RPTB block is used within the interrupt. If the interrupt service routine does not include a RPTB block, then you do not have to save the RB register.

```
; Repeat Block within a High-Priority Interrupt (Non-Interruptible)
;
; Interrupt:            ; RAS = RA, RA = 0
   ...
   PUSH RB              ; Save RB register only if a RPTB block is used in the ISR
   ...
   ...
   RPTB #_BlockEnd, #5  ; Execute the block AL+1 times
   ...
   ...
   ...
_BlockEnd               ; End of block to be repeated
   ...
   ...
   POP RB               ; Restore RB register ...
   IRET                 ; RA = RAS, RAS = 0
```

A low-priority interrupt is defined as an interrupt that allows itself to be interrupted. The RB register must always be saved and restored in a low-priority interrupt. The RB register must stored before interrupts are enabled. Likewise before restoring the RB register interrupts must first be disabled.

```
; Repeat Block within a Low-Priority Interrupt (Interruptible)
;
; Interrupt:
                        ; RAS = RA, RA = 0
   ...
   PUSH RB              ; Always save RB register
   ...
   CLRC INTM            ; Enable interrupts only after saving RB
   ...
   ...
   ...
; ISR may or may not include a RPTB block
   ...
   ...
   SETC INTM            ; Disable interrupts before restoring RB
   ...
   POP RB               ; Always restore RB register
   ...
   IRET                 ; RA = RAS, RAS = 0
```

See also

POP RB
PUSH RB
RPTB label, loc16

VCLEAR VRa *Clear General Purpose Register*

Operands

VRa	General purpose register: VR0, VR1... VR8

Opcode

```
LSW: 1110 0110 1111 1000
MSW: 0000 0000 0000 aaaa
```

Description

Clear the specified general purpose register.

```
VRa = 0x00000000;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
;
; Code fragment from a viterbi traceback
; For the first iteration the previous state metric must be
; initalized to zero (VR0).
;
   VCLEAR VR0                   ; Clear the VR0 register
   MOVL XAR5,*+XAR4[0]          ; Point XAR5 to an array
;
; For first stage
;
   VMOV32 VT0, *--XAR3
   VMOV32 VT1, *--XAR3
   VTRACE *XAR5++,VR0,VT0,VT1   ; Uses VR0 (which is zero)
;
; etc...
;
```

See also

VCLEARALL
VTCLEAR

VCLEARALL *Clear All General Purpose and Transition Bit Registers*

Operands

none

Opcode

```
LSW: 1110 0110 1111 1001
MSW: 0000 0000 0000 0000
```

Description

Clear all of the general purpose registers (VR0, VR1... VR8) and the transition bit registers (VT0 and VT1).

```
VR0 = 0x00000000;
VR0 = 0x00000000;
VR2 = 0x00000000;
VR3 = 0x00000000;
VR4 = 0x00000000;
VR5 = 0x00000000;
VR6 = 0x00000000;
VR7 = 0x00000000;
VR8 = 0x00000000;
VT0 = 0x00000000;
VT1 = 0x00000000;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
;
;  Context save all VCU VRa and VTa registers
;
   VMOV32  *SP++, VR0
   VMOV32  *SP++, VR1
   VMOV32  *SP++, VR2
   VMOV32  *SP++, VR3
   VMOV32  *SP++, VR4
   VMOV32  *SP++, VR5
   VMOV32  *SP++, VR6
   VMOV32  *SP++, VR7
   VMOV32  *SP++, VR8
   VMOV32  *SP++, VT0
   VMOV32  *SP++, VT1
;
; Clear VR0 - VR8, VT0 and VT1
;
   VCLEARALL
;
; etc...
```

See also

VCLEAR VRa
VTCLEAR

VCLROVFI *Clear Imaginary Overflow Flag*

Operands

none

Opcode

```
LSW: 1110 0101 0000 1011
```

Description

Clear the imaginary overflow flag in the VSTATUS register. To clear the real flag, use the VCLROVFR instruction. The imaginary flag bit can be set by instructions shown in Table 10-6. Refer to invidual instruction descriptions for details.

```
VSTATUS[OVFI] = 0;
```

Flags

This instruction clears the OVFI flag.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLROVFR
VRNDON
VSATFOFF
VSATON

VCLROVFR *Clear Real Overflow Flag*

Operands

none

Opcode

```
LSW: 1110 0101 0000 1010
```

Description

Clear the real overflow flag in the VSTATUS register. To clear the imaginary flag, use the VCLROVFI instruction. The imaginary flag bit can be set by instructions shown in Table 10-6. Refer to invidual instruction descriptions for details.

```
VSTATUS[OVFR] = 0;
```

Flags

This instruction clears the OVFR flag.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLROVFI
VRNDON
VSATFOFF
VSATON

VMOV16 mem16, VRaL *Store General Purpose Register, Low Half*

Operands

mem16	Pointer to a 16-bit memory location. This will be the destination of the VMOV16.
VRaL	Low word of a general purpose register: VR0L, VR1L...VR8L.

Opcode

```
LSW: 1110 0010 0001 1000
MSW: 0000 aaaa  mem16
```

Description

Store the low 16-bits of the specified general purpose register into the 16-bit memory location.

```
[mem16] = VRa[15:0];
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOV16 VRaL, mem16

VMOV16 VRaL, mem16 *Load General Purpose Register, Low Half*

Operands

VRaL	Low word of a general purpose register: VR0L, VR1L....VR8L
mem16	Pointer to a 16-bit memory location. This will be the source for the VMOV16.

Opcode

```
LSW: 1110 0010 1100 1001
MSW: 0000 aaaa  mem16
```

Description

Load the lower 16 bits of the specified general purpose register with the contents of memory pointed to by mem16.

```
VRa[15:0] = [mem16];
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
;
; Loop will run 106 times for 212 inputs to decoder
;
; Code fragment from viterbi decoder
;
_LOOP:
;
;
; Calculate the branch metrics for code rate = 1/3
; Load VR0L, VR1L and VR2L with inputs
; to the decoder from the array pointed to by XAR5
;
;
   VMOV16 VR0L, *XAR5++
   VMOV16 VR1L, *XAR5++
   VMOV16 VR2L, *XAR5++
;
; VR0L = BM0
; VR0H = BM1
; VR1L = BM2
; VR1H = BM3
; VR2L = pt_old[0]
; VR2H = pt_old[1]
;
  VITBM3 VR0, VR1, VR2
  VMOV32 VR2, *XAR1++
; etc...
```

See also

VMOV16 mem16, VRaL

VMOV32 mem32, VRa *Store General Purpose Register*

Operands

mem32	Pointer to a 32-bit memory location. This will be the destination of the VMOV32.
VRa	General purpose reigster VR0, VR1... VR8

Opcode

```
LSW: 1110 0010 0000 0100
MSW: 0000 aaaa  mem32
```

Description

Store the 32-bit contents of the specified general purpose register into the memory location pointed to by mem32.

```
[mem32] = VRa;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOV32 mem32, VSTATUS
VMOV32 mem32, VTa
VMOV32 VRa, mem32
VMOV32 VTa, mem32

VMOV32 mem32, VSTATUS *Store VCU Status Register*

Operands

mem32	Pointer to a 32-bit memory location. This will be the destination of the VMOV32.
VSTATUS	VCU status register.

Opcode

```
LSW: 1110 0010 0000 1101
MSW: 0000 0000  mem32
```

Description Store the VSTATUS register into the memory location pointed to by mem32.

```
[mem32] = VSTATUS;
```

Flags This instruction does not modify any flags in the VSTATUS register.

Pipeline This is a single-cycle instruction.

Example

See also

VMOV32 mem32, VRa
VMOV32 mem32, VTa
VMOV32 VRa, mem32
VMOV32 VSTATUS, mem32
VMOV32 VTa, mem32

VMOV32 mem32, VTa *Store Transition Bit Register*

Operands

mem32	pointer to a 32-bit memory location. This will be the destination of the VMOV32.
VTa	Transition bits register VT0 or VT1

Opcode

```
LSW: 1110 0010 0000 0101
MSW: 0000 00tt  mem32
```

Description

Store the 32-bits of the specified transition bits register into the memory location pointed to by mem32.

```
[mem32] = VTa;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOV32 mem32, VRa
VMOV32 mem32, VSTATUS
VMOV32 VRa, mem32
VMOV32 VSTATUS, mem32
VMOV32 VTa, mem32

VMOV32 VRa, mem32 *Load 32-bit General Purpose Register*

Operands

VRa	General purpose register VR0, VR1....VR8
mem32	Pointer to a 32-bit memory location. This will be the source of the VMOV32.

Opcode

```
LSW: 1110 0011 1111 0000
MSW: 0000 aaaa  mem32
```

Description

Load the specified general purpose register with the 32-bit value in memory pointed to by mem32.

```
VRa = [mem32];
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOV32 mem32, VRa
VMOV32 mem32, VSTATUS
VMOV32 mem32, VTa
VMOV32 VSTATUS, mem32
VMOV32 VTa, mem32

VMOV32 VSTATUS, mem32 *Load VCU Status Register*

Operands

VSTATUS	VCU status register
mem32	Pointer to a 32-bit memory location. This will be the source of the VMOV32.

Opcode

```
LSW: 1110 0010 1011 0000
MSW: 0000 0000  mem32
```

Description

Load the VSTATUS register with the 32-bit value in memory pointed to by mem32.

```
VSTATUS = [mem32];
```

Flags

This instruction modifies all bits within the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOV32 mem32, VSTATUS
VMOV32 mem32, VTa
VMOV32 VRa, mem32
VMOV32 VTa, mem32

VMOV32 VTa, mem32 *Load 32-bit Transition Bit Register*

Operands

VTa	Transition bit register: VT0, VT1
mem32	Pointer to a 32-bit memory location. This will be the source of the VMOV32.

Opcode

```
LSW: 1110 0011 1111 0001
MSW: 0000 00tt  mem32
```

Description

Load the specified transition bit register with the 32-bit value in memory pointed to by mem32 .

```
VTa = [mem32];
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOV32 mem32, VSTATUS
VMOV32 mem32, VTa
VMOV32 VRa, mem32
VMOV32 VSTATUS, mem32

VMOVD32 VRa, mem32 *Load Register with Data Move*

Operands

VRa	General purpose registger, VR0, VR1.... VR8
mem32	Pointer to a 32-bit memory location. This will be the source of the VMOV32.

Opcode

```
LSW: 1110 0010 0010 0100
MSW: 0000 aaaa  mem32
```

Description

Load the specified general purpose register with the 32-bit value in memory pointed to by mem32. In addition, copy the next 32-bit value in memory to the location pointed to by mem32.

```
VRa = [mem32];
[mem32 + 2] = [mem32];
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOVIX VRa, #16I *Load Upper Half of a General Purpose Register with I6-bit Immediate*

Operands

VRa	General purpose registger, VR0, VR1... VR8
#16I	16-bit immediate value

Opcode

```
LSW: 1110 0111 1110 IIII
MSW: IIII IIII IIII aaaa
```

Description

Load the upper 16-bits of the specified general purpose register with an immediate value. Leave the lower 16-bits of the register unchanged.

```
VRa[15:0] = unchanged;
VRa[31:16] = #16I;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOVZI VRa, #16I
VMOVXI VRa, #16I

VMOVZI VRa, #16I *Load General Purpose Register with Immediate*

Operands

VRa	General purpose registger, VR0, VR1...VR8
#16I	16-bit immediate value

Opcode

```
LSW: 1110 0111 1111 IIII
MSW: IIII IIII IIII aaaa
```

Description

Load the lower 16-bits of the specified general purpose register with an immediate value. Clear the upper 16-bits of the register.

```
VRa[15:0] = #16I;
VRa[31:16] = 0x0000;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOVIX VRa, #16I
VMOVXI VRa, #16I

VMOVXI VRa, #16I *Load Low Half of a General Purpose Register with Immediate*

Operands

VRa	General purpose registger, VR0 - VR8
#16I	16-bit immediate value

Opcode

```
LSW: 1110 0111 0111 IIII
MSW: IIII IIII IIII aaaa
```

Description

Load the lower 16-bits of the specified general purpose register with an immediate value. Leave the upper 16 bits unchanged.

```
VRa[15:0] = #16I;
VRa[31:16] = unchanged;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VMOVIX VRa, #16I
VMOVZI VRa, #16I

VRNDOFF *Disable Rounding*

Operands

none

Opcode

```
LSW: 1110 0101 0000 1001
```

Description

This instruction disables the rounding mode by clearning the RND bit in the VSTATUS register. When rounding is disabled, the result of the shift right operation for addition and subtraction operations will be truncated instead of rounded. The operations affected by rounding are shown in Table 10-6. Refer to the individual instruction descriptions for information on how rounding effects the operation. To enable rounding use the VRNDON instruction.

For more information on rounding, refer to .

```
VSTATUS[RND] = 0;
```

Flags

This instruction clears the RND bit in the VSTATUS register. It does not change any flags.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLROVFI
VCLROVFR
VRNDON
VSATFOFF
VSATON

VRNDON *Enable Rounding*

Operands

none

Opcode

```
LSW: 1110 0101 0000 1000
```

Description

This instruction enables the rounding mode by setting the RND bit in the VSTATUS register. When rounding is enabled, the result of the shift right operation for addition and subtraction operations will be rounded instead of being truncated. The operations affected by rounding are shown in Table 10-6. Refer to the individual instruction descriptions for information on how rounding effects the operation. To disable rounding use the VRNDOFF instruction.

For more information on rounding, refer to .

```
VSTATUS[RND] = 1;
```

Flags

This instruction sets the RND bit in the VSTATUS register. It does not change any flags.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLROVFI
VCLROVFR
VRNDOFF
VSATFOFF
VSATON

VSATOFF *Disable Saturation*

Operands

none

Opcode

```
LSW: 1110 0101 0000 0111
```

Description

This instruction disables the satuartion mode by clearing the SAT bit in the VSTATUS register. When saturation is disabled, results of addition and subtraction are allowed to overflow or underflow. When saturation is enabled, results will instead be set to a maximum or minimum value instead of being allowed to overflow or underflow. To enable saturation use the VSATON instruction.

```
VSTATUS[SAT] = 0
```

Flags

This instruction clears the the SAT bit in the VSTATUS register. It does not change any flags.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLROVFI
VCLROVFR
VRNDOFF
VRNDON
VSATON

VSATON *Enable Saturation*

Operands

none

Opcode

```
LSW: 1110 0101 0000 0110
```

Description

This instruction enables the satuartion mode by setting the SAT bit in the VSTATUS register. When saturation is enables, results of addition and subtraction are not allowed to overflow or underflow. Results will, instead, be set to a maximum or minimum value. To disable saturation use the VSATOFF instruction..

```
VSTATUS[SAT] = 1
```

Flags

This instruction sets the SAT bit in the VSTATUS register. It does not change any flags.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLROVFI
VCLROVFR
VRNDOFF
VRNDON
VSATOFF

VSETSHL #5-bit *Initialize the Left Shift Value*

Operands

#5-bit	5-bit, unsigned, immediate value

Opcode

```
LSW: 1110 0101 110s ssss
```

Description

Load VSTATUS[SHIFTL] with an unsigned, 5-bit, immediate value. The left shift value specifies the number of bits an operand is shifed by. A value of zero indicates no shift will be performed. The left shift is used by the and VCDSUB16 and VCDADD16 operations. Refer to the description of these instructions for more information. To load the right shift value use the VSETSHR #5-bit instruction.

```
VSTATUS[VSHIFTL] = #5-bit
```

Flags

This instruction changes the VSHIFTL value in the VSTATUS register. It does not change any flags.

Pipeline

This is a single-cycle instruction.

Example

See also

VSETSHR #5-bit

VSETSHR #5-bit *Initialize the Left Shift Value*

Operands

#5-bit	5-bit, unsigned, immediate value

Opcode

```
LSW: 1110 0101 010s ssss
```

Description

Load VSTATUS[SHIFTR] with an unsigned, 5-bit, immediate value. The right shift value specifies the number of bits an operand is shifed by. A value of zero indicates no shift will be performed. The right shift is used by the VCADD, VCSUB, VCDADD16 and VCDSUB16 operations. It is also used by the addition portion of the VCMAC. Refer to the description of these instructions for more information.

```
VSTATUS[VSHIFTR] = #5-bit
```

Flags

This instruction changes the VSHIFTR value in the VSTATUS register. It does not change any flags.

Pipeline

This is a single-cycle instruction.

Example

See also

VSETSHL #5-bit

10.5.3 Complex Math Instructions

The instructions are listed alphabetically, preceded by a summary.

Table 10-11. Complex Math Instructions

VCADD VR5, VR4, VR3, VR2 *Complex 32 + 32 = 32 Addition*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Each operand for this instruction includes a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR5	32-bit integer representing the real part of the first input: Re(X)
VR4	32-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)

The result is also a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR5 and VR4 as shown below:

Output Register	Value
VR5	32-bit integer representing the real part of the result: Re(Z) = Re(X) + (Re(Y) >> SHIFTR)
VR4	32-bit integer representing the imaginary part of the result: Im(Z) = Im(X) + (Im(Y) >> SHIFTR)

Opcode

```
LSW: 1110 0101 0000 0010
```

Description

Complex 32 + 32 = 32-bit addition operation.

The second input operand (stored in VR3 and VR2) is shifted right by VSTATUS[SHIFR] bits before the addition. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of an overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
//
// X:  VR5 = Re(X)    VR4 = Im(X)
// Y:  VR3 = Re(Y)    VR2 = Im(Y)
//
// Calculate Z = X + Y
//
   if (RND == 1)
   {
      VR5 = VR5 + round(VR3 >> SHIFTR);  // Re(Z)
      VR4 = VR4 + round(VR2 >> SHIFTR);  // Im(Z)
   }
   else
   {
      VR5 = VR5 + (VR3 >> SHIFTR);       // Re(Z)
      VR4 = VR4 + (VR2 >> SHIFTR);       // Im(Z)
   }
   if (SAT == 1)
   {
      sat32(VR5);
      sat32(VR4);
   }
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR5 computation (real part) overflows or underflows.
- OVFI is set if the VR4 computation (imaginary part) overflows or underflows.

Pipeline

This is a single-cycle instruction.

Example

See also

VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
VCADD VR7, VR6, VR5, VR4
VCLROVFI
VCLROVFR
VRNDOFF
VRNDON
VSATON
VSATOFF
VSETSHR #5-bit

VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32 *Complex 32+32 = 32 Add with Parallel Load*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Each complex number includes a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR5	32-bit integer representing the real part of the first input: Re(X)
VR4	32-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)
mem32	pointer to a 32-bit memory location

The result is also a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR5 and VR4 as shown below:

Output Register	Value
VR5	32-bit integer representing the real part of the result: Re(Z) = Re(X) + (Re(Y) >> SHIFTR)
VR4	32-bit integer representing the imaginary part of the result: Im(Z) = Im(X) + (Im(Y) >> SHIFTR)
VRa	contents of the memory pointed to by [mem32]. VRa can not be VR5, VR4 or VR8.

Opcode

```
LSW: 1110 0011 1111 1000
MSW: 0000 aaaa  mem32
```

Description

Complex 32 + 32 = 32-bit addition operation with parallel register load.

The second input operand (stored in VR3 and VR2) is shifted right by VSTATUS[SHIFR] bits before the addition. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of an overflow or underflow.

In parallel with the addition, VRa is loaded with the contents of memory pointed to by mem32.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
//
// VR5 = Re(X)    VR4 = Im(X)
// VR3 = Re(Y)    VR2 = Im(Y)
//
// Z = X + Y
//
   if (RND == 1)
   {
      VR5 = VR5 + round(VR3 >> SHIFTR);  // Re(Z)
      VR4 = VR4 + round(VR2 >> SHIFTR);  // Im(Z)
   }
   else
   {
      VR5 = VR5 + (VR3 >> SHIFTR);       // Re(Z)
      VR4 = VR4 + (VR2 >> SHIFTR);       // Im(Z)
   }
   if (SAT == 1)
   {
      sat32(VR5);
      sat32(VR4);
   }
   VRa = [mem32];
```

Flags This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR5 computation (real part) overflows.
- OVFI is set if the VR4 computation (imaginary part) overflows.

Pipeline Both operations complete in a single cycle (1/1 cycles).

Example

See also VCADD VR7, VR6, VR5, VR4
VCLROVFI
VCLROVFR
VRNDOFF
VRNDON
VSATON
VSATOFF
VSETSHR #5-bit

VCADD VR7, VR6, VR5, VR4 *Complex 32 + 32 = 32- Addition*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Each complex number includes a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR7	32-bit integer representing the real part of the first input: Re(X)
VR6	32-bit integer representing the imaginary part of the first input: Im(X)
VR5	32-bit integer representing the real part of the 2nd input: Re(Y)
VR4	32-bit integer representing the imaginary part of the 2nd input: Im(Y)

The result is also a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR7 and VR6 as shown below:

Output Register	Value
VR6	32-bit integer representing the real part of the result: Re(Z) = Re(X) + (Re(Y) >> SHIFTR)
VR7	32-bit integer representing the imaginary part of the result: Im(Z) = Im(X) + (Im(Y) >> SHIFTR)

Opcode

```
LSW: 1110 0101 0010 1010
```

Description

Complex 32 + 32 = 32-bit addition operation.

The second input operand (stored in VR5 and VR4) is shifted right by VSTATUS[SHIFR] bits before the addition. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of an overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
//
// VR5 = Re(X)    VR4 = Im(X)
// VR3 = Re(Y)    VR2 = Im(Y)
//
// Z = X + Y
//
   if (RND == 1)
   {
      VR7 = VR7 + round(VR5 >> SHIFTR);  // Re(Z)
      VR6 = VR6 + round(VR4 >> SHIFTR);  // Im(Z)
   }
   else
   {
      VR7 = VR5 + (VR5 >> SHIFTR);       // Re(Z)
      VR6 = VR4 + (VR4 >> SHIFTR);       // Im(Z)
   }
   if (SAT == 1)
   {
      sat32(VR7);
      sat32(VR6);
   }
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR7 computation (real part) overflows.
- OVFI is set if the VR6 computation (imaginary part) overflows.

Pipeline

This is a single-cycle instruction.

See also	VCADD VR5, VR4, VR3, VR2
	VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
	VCLROVFI
	VCLROVFR
	VRNDOFF
	VRNDON
	VSATON
	VSATOFF
	VSETSHR #5-bit

VCDADD16 VR5, VR4, VR3, VR2 *Complex 16 + 32 = 16 Addition*

Operands

Before the operation, the inputs should be loaded into registers as shown below. The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR4H	16-bit integer representing the real part of the first input: Re(X)
VR4L	16-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)

The result is a complex number with a 16-bit real and a 16-bit imaginary part. The result is stored in VR5 as shown below:

Output Register	Value
VR5H	16-bit integer representing the real part of the result: Re(Z) = (Re(X) << SHIFTL) + (Re(Y)) >> SHIFTR
VR5L	16-bit integer representing the imaginary part of the result: Im(Z) = (Im(X) << SHIFTL) + (Im(Y)) >> SHIFTR

Opcode

```
LSW: 1110 0101 0000 0100
```

Description

Complex 16 + 32 = 16-bit operation. This operation is useful for algorithms similar to a complex FFT. The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Before the addition, the first input is sign extended to 32-bits and shifted left by VSTATUS[VSHIFTL] bits. The result of the addition is left shifted by VSTATUS[VSHIFTR] before it is stored in VR5H and VR5L. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of a 16-bit overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
// SHIFTL is VSTATUS[SHIFTL]
//
// VR4H = Re(X)    16-bit
// VR4L = Im(X)    16-bit
// VR3  = Re(Y)    32-bit
// VR2  = Im(Y)    32-bit
//
// Calculate Z = X + Y
//

   temp1 = sign_extend(VR4H);         // 32-bit extended Re(X)
   temp2 = sign_extend(VR4L);         // 32-bit extended Im(X)

   temp1 = (temp1 << SHIFTL) + VR3;   // Re(Z) intermediate
   temp2 = (temp2 << SHIFTL) + VR2;   // Im(Z) intermediate

   if (RND == 1)
   {
      temp1 = round(temp1 >> SHIFTR);
      temp2 = round(temp2 >> SHIFTR);
   }
   else
   {
      temp1 = truncate(temp1 >> SHIFTR);
      temp2 = truncate(temp2 >> SHIFTR);
   }
```

```
if (SAT == 1)
{
   VR5H = sat16(temp1);
   VR5L = sat16(temp2);
}
else
{
   VR5H = temp1[15:0];
   VR5L = temp2[15:0];
}
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the real-part computation (VR5H) overflows or underflows.
- OVFI is set if the imaginary-part computation (VR5L) overflows or underflows.

Pipeline

This is a single-cycle instruction.

Example

```
;
;Example: Z = X + Y
;
; X =  4 +  3j    (16-bit real + 16-bit imaginary)
; Y = 13 + 12j    (32-bit real + 32-bit imaginary)
;
; Real:
;   temp1 = 0x00000004 + 0x0000000D = 0x00000011
;   VR5H  = temp1[15:0] = 0x0011 = 17
; Imaginary:
;   temp2 = 0x00000003 + 0x0000000C = 0x0000000F
;   VR5L  = temp2[15:0] = 0x000F = 15
;
    VSATOFF                        ; VSTATUS[SAT] = 0
    VRNDOFF                        ; VSTATUS[RND] = 0
    VSETSHR    #0                  ; VSTATUS[SHIFTR] = 0
    VSETSHL    #0                  ; VSTATUS[SHIFTL] = 0
    VCLEARALL                      ; VR0, VR1...VR8 == 0
    VMOVXI     VR3, #13            ; VR3 = Re(Y) =  13
    VMOVXI     VR2, #12            ; VR2 = Im(Y) =  12
    VMOVXI     VR4, #3
    VMOVIX     VR4, #4             ; VR4 = X = 0x00040003 =  4 +  3j
    VCDADD16   VR5, VR4, VR3, VR2  ; VR5 = Z = 0x0011000F = 17 + 15j
```

The next example illustrates the operation with a right shift value defined.

```
;
; Example: Z = X + Y with Right Shift
;
; X =  4 +  3j    (16-bit real + 16-bit imaginary)
; Y = 13 + 12j    (32-bit real + 32-bit imaginary)
;
; Real:
;   temp1 = (0x00000004  + 0x0000000D ) >> 1
;   temp1 = (0x00000011) >> 1 = 0x0000008.8
;   VR5H  = temp1[15:0] = 0x0008 = 8
; Imaginary:
;   temp2 = (0x00000003  + 0x0000000C ) >> 1
;   temp2 = (0x0000000F) >> 1 = 0x0000007.8
;   VR5L  = temp2[15:0] = 0x0007 = 7
;
    VSATOFF                        ; VSTATUS[SAT] = 0
    VRNDOFF                        ; VSTATUS[RND] = 0
    VSETSHR    #1                  ; VSTATUS[SHIFTR] = 1
    VSETSHL    #0                  ; VSTATUS[SHIFTL] = 0
    VCLEARALL                      ; VR0, VR1...VR8 == 0
    VMOVXI     VR3, #13            ; VR3 = Re(Y) =  13
    VMOVXI     VR2, #12            ; VR2 = Im(Y) =  12
    VMOVXI    VR4, #3
    VMOVIX    VR4, #4             ; VR4 = X = 0x00040003 =  4 +  3j
    VCDADD16  VR5, VR4, VR3, VR2  ; VR5 = Z = 0x00080007 =  8 +  7j
```

The next example illustrates the operation with a right shift value defined as well as rounding.

```
;
; Example: Z = X + Y with Right Shift and Rounding
;
; X =  4 +  3j    (16-bit real + 16-bit imaginary)
; Y = 13 + 12j    (32-bit real + 32-bit imaginary)
;
; Real:
;   temp1 = round((0x00000004  + 0x0000000D ) >> 1)
;   temp1 = round(0x00000011 >> 1)
;   temp1 = round(0x0000008.8) = 0x00000009
;   VR5H  = temp1[15:0] = 0x0011 = 8
; Imaginary:
;   temp2 = round(0x00000003  + 0x0000000C ) >> 1)
;   temp2 = round(0x0000000F >> 1)
;   temp2 = round(0x0000007.8) = 0x00000008
;   VR5L  = temp2[15:0] = 0x0008 = 8
;
    VSATOFF                       ; VSTATUS[SAT] = 0
    VRNDON                        ; VSTATUS[RND] = 1
    VSETSHR   #1                  ; VSTATUS[SHIFTR] = 1
    VSETSHL   #0                  ; VSTATUS[SHIFTL] = 0
    VCLEARALL                     ; VR0, VR1...VR8 == 0
    VMOVXI    VR3, #13            ; VR3 = Re(Y) =  13
    VMOVXI    VR2, #12            ; VR2 = Im(Y) =  12
    VMOVXI    VR4, #3
    VMOVIX    VR4, #4             ; VR4 = X = 0x00040003 =  4 +  3j
    VCDADD16  VR5, VR4, VR3, VR2  ; VR5 = Z = 0x00090008 =  9 +  8j
```

The next example illustrates the operation with both a right and left shift value defined along with rounding.

```
;
; Example: Z = X + Y with Right Shift, Left Shift and Rounding
;
; X = -4 + 3j    (16-bit real + 16-bit imaginary)
; Y = 13 - 9j    (32-bit real + 32-bit imaginary)
;
; Real:
;   temp1 = 0xFFFFFFFC << 2 + 0x0000000D
;   temp1 = 0xFFFFFFF0      + 0x0000000D = 0xFFFFFFFD
;   temp1 = 0xFFFFFFFD >> 1 = 0xFFFFFFFE.8
;   temp1 = round(0xFFFFFFFE.8) = 0xFFFFFFFF
;   VR5H  = temp1[15:0] 0xFFFF = -1;
; Imaginary:
;   temp2 = 0x00000003 << 2 + 0xFFFFFFF7
;   temp2 = 0x0000000C      + 0xFFFFFFF7 = 0x00000003
;   temp2 = 0x00000003 >> 1 = 0x00000001.8
;   temp1 = round(0x000000001.8 = 0x000000002
;   VR5L  = temp2[15:0] 0x0002 = 2
;
    VSATOFF                       ; VSTATUS[SAT] = 0
    VRNDON                        ; VSTATUS[RND] = 1
    VSETSHR   #1                  ; VSTATUS[SHIFTR] = 1
    VSETSHL   #2                  ; VSTATUS[SHIFTL] = 2
    VCLEARALL                     ; VR0, VR1...VR8 == 0
    VMOVXI    VR3, #13            ; VR3 = Re(Y) = 13 = 0x0000000D
    VMOVXI    VR2, #-9            ; VR2 = Im(Y) = -9
    VMOVIX    VR2, #0xFFFF        ; sign extend VR2 = 0xFFFFFFF7
    VMOVXI    VR4, #3
    VMOVIX    VR4, #-4            ; VR4 = X = 0xFFFC0003 = -4 +  3j
    VCDADD16  VR5, VR4, VR3, VR2  ; VR5 = Z = 0xFFFF0002 = -1 +  2j
```

See also	VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
	VCADD VR7, VR6, VR5, VR4
	VCDADD16 VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
	VRNDOFF
	VRNDON
	VSATON
	VSATOFF
	VSETSHL #5-bit
	VSETSHR #5-bit

VCDADD16 VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32 *Complex Double Add with Parallel Load*

Operands

Before the operation, the inputs should be loaded into registers as shown below. The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR4H	16-bit integer representing the real part of the first input: Re(X)
VR4L	16-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)
mem32	pointer to a 32-bit memory location.

The result is a complex number with a 16-bit real and a 16-bit imaginary part. The result is stored in VR5 as shown below:

Output Register	Value
VR5H	16-bit integer representing the real part of the result: Re(Z) = (Re(X) << SHIFTL) + (Re(Y)) >> SHIFTR
VR5L	16-bit integer representing the imaginary part of the result: Im(Z) = (Im(X) << SHIFTL) + (Im(Y)) >> SHIFTR
VRa	Contents of the memory pointed to by [mem32]. VRa can not be VR5 or VR8.

Opcode

```
LSW: 1110 0011 1111 1010
MSW: 0000 aaaa  mem32
```

Description

Complex 16 + 32 = 16-bit operation with parallel register load. This operation is useful for algorithms similar to a complex FFT.

The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Before the addition, the first input is sign extended to 32-bits and shifted left by VSTATUS[VSHIFTL] bits. The result of the addition is left shifted by VSTATUS[VSHIFTR] before it is stored in VR5H and VR5L. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of a 16-bit overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
// SHIFTL is VSTATUS[SHIFTL]
//
// VR4H = Re(X)    16-bit
// VR4L = Im(X)    16-bit
// VR3  = Re(Y)    32-bit
// VR2  = Im(Y)    32-bit

   temp1 = sign_extend(VR4H);        // 32-bit extended Re(X)
   temp2 = sign_extend(VR4L);        // 32-bit extended Im(X)

   temp1 = (temp1 << SHIFTL) + VR3;  // Re(Z) intermediate
   temp2 = (temp2 << SHIFTL) + VR2;  // Im(Z) intermediate

   if (RND == 1)
   {
      temp1 = round(temp1 >> SHIFTR);
      temp2 = round(temp2 >> SHIFTR);
   }
   else
   {
      temp1 = truncate(temp1 >> SHIFTR);
```

```
        temp2 = truncate(temp2 >> SHIFTR);
     }
     if (SAT == 1)
     {
        VR5H = sat16(temp1);
        VR5L = sat16(temp2);
     }
     else
     {
        VR5H = temp1[15:0];
        VR5L = temp2[15:0];
     }
     VRa = [mem32];
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the real-part (VR5H) computation overflows or underflows.
- OVFI is set if the imaginary-part (VR5L) computation overflows or underflows.

Pipeline

Both operations complete in a single cycle.

Example

For more information regarding the addition operation, please refer to the examples for the VCDADD16 VR5, VR4, VR3, VR2 instruction.

```
;
;Example: Right Shift, Left Shift and Rounding
;
; X = -4 + 3j    (16-bit real + 16-bit imaginary)
; Y = 13 - 9j    (32-bit real + 32-bit imaginary)
;
;
; Real:
;   temp1 = 0xFFFFFFFC << 2 + 0x0000000D
;   temp1 = 0xFFFFFFF0      + 0x0000000D = 0xFFFFFFFD
;   temp1 = 0xFFFFFFFD >> 1 = 0xFFFFFFFE.8
;   temp1 = round(0xFFFFFFFE.8) = 0xFFFFFFFF
;   VR5H  = temp1[15:0] 0xFFFF = -1;
; Imaginary:
;   temp2 = 0x00000003 << 2 + 0xFFFFFFF7
;   temp2 = 0x0000000C      + 0xFFFFFFF7 = 0x00000003
;   temp2 = 0x00000003 >> 1 = 0x00000001.8
;   temp1 = round(0x00000001.8 = 0x00000002
;   VR5L  = temp2[15:0] 0x0002 = 2
;
    VSATOFF                        ; VSTATUS[SAT] = 0
    VRNDON                         ; VSTATUS[RND] = 1
    VSETSHR   #1                   ; VSTATUS[SHIFTR] = 1
    VSETSHL   #2                   ; VSTATUS[SHIFTL] = 2
    VCLEARALL                      ; VR0, VR1...VR8 == 0
    VMOVXI    VR3, #13             ; VR3 = Re(Y) = 13 = 0x0000000D
    VMOVXI    VR2, #-9             ; VR2 = Im(Y) = -9
    VMOVIX    VR2, #0xFFFF         ; sign extend VR2 = 0xFFFFFFF7
    VMOVXI    VR4, #3
    VMOVIX    VR4, #-4             ; VR4 = X = 0xFFFC0003 = -4 +  3j
    VCDADD16  VR5, VR4, VR3, VR2   ; VR5 = Z = 0xFFFF0002 = -1 +  2j
||  VCMOV32   VR2, *XAR7           ; VR2 = value pointed to by XAR7
```

See also

VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
VCADD VR7, VR6, VR5, VR4
VRNDOFF
VRNDON
VSATON
VSATOFF
VSETSHL #5-bit
VSETSHR #5-bit

VCDSUB16 VR6, VR4, VR3, VR2 *Complex 16-32 = 16 Subtract*

Operands

Before the operation, the inputs should be loaded into registers as shown below. The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR4H	16-bit integer representing the real part of the first input: Re(X)
VR4L	16-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)

The result is a complex number with a 16-bit real and a 16-bit imaginary part. The result is stored in VR6 as shown below:

Output Register	Value
VR6H	16-bit integer representing the real part of the result: Re(Z) = (Re(X) << SHIFTL) - (Re(Y)) >> SHIFTR
VR6L	16-bit integer representing the imaginary part of the result: Im(Z) = (Im(X) << SHIFTL) - (Im(Y)) >> SHIFTR

Opcode

```
LSW: 1110 0101 0000 0101
```

Description

Complex 16 - 32 = 16-bit operation. This operation is useful for algorithms similar to a complex FFT.

The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Before the addition, the first input is sign extended to 32-bits and shifted left by VSTATUS[VSHIFTL] bits. The result of the subtraction is left shifted by VSTATUS[VSHIFTR] before it is stored in VR5H and VR5L. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of a 16-bit overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
// SHIFTL is VSTATUS[SHIFTL]
//
// VR4H = Re(X)    16-bit
// VR4L = Im(X)    16-bit
// VR3  = Re(Y)    32-bit
// VR2  = Im(Y)    32-bit

   temp1 = sign_extend(VR4H);        // 32-bit extended Re(X)
   temp2 = sign_extend(VR4L);        // 32-bit extended Im(X)

   temp1 = (temp1 << SHIFTL) - VR3;  // Re(Z) intermediate
   temp2 = (temp2 << SHIFTL) - VR2;  // Im(Z) intermediate

   if (RND == 1)
   {
      temp1 = round(temp1 >> SHIFTR);
      temp2 = round(temp2 >> SHIFTR);
   }
   else
   {
      temp1 = truncate(temp1 >> SHIFTR);
      temp2 = truncate(temp2 >> SHIFTR);
   }
   if (SAT == 1)
   {
      VR5H = sat16(temp1);
      VR5L = sat16(temp2);
   }
   else
   {
      VR5H = temp1[15:0];
      VR5L = temp2[15:0];
   }
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the real-part (VR6H) computation overflows or underflows.
- OVFI is set if the imaginary-part (VR6L) computation overflows or underflows.

Pipeline

This is a single-cycle instruction.

Example

```
;
; Example: Z = X - Y
;
; X =  4 +  6j   (16-bit real + 16-bit imaginary)
; Y = 13 + 22j   (32-bit real + 32-bit imaginary)
;
; Z = (4 - 13) + (6 - 22)j = -9 - 16j
;
    VSATOFF                       ; VSTATUS[SAT] = 0
    VRNDOFF                       ; VSTATUS[RND] = 0
    VSETSHR   #0                  ; VSTATUS[SHIFTR] = 0
    VSETSHL   #0                  ; VSTATUS[SHIFTL] = 0
    VCLEARALL                     ; VR0, VR1...VR8 = 0
    VMOVXI    VR3, #13            ; VR3 = Re(Y) =  13 = 0x0000000D
    VMOVXI    VR2, #22            ; VR2 = Im(Y) = 22j = 0x00000016
    VMOVXI    VR4, #6
    VMOVIX    VR4, #4             ; VR4 = X = 0x00040006 =  4 +   6j
    VCDSUB16  VR6, VR4, VR3, VR2  ; VR5 = Z = 0xFFF7FFF0 = -9 + -16j
```

The next example illustrates the operation with a right shift value defined.

```
;
; Example: Z = X - Y with Right Shift

; Y =  4 +  6j   (16-bit real + 16-bit imaginary)
; X = 13 + 22j   (32-bit real + 32-bit imaginary)
;
; Real:
;    temp1 = (0x00000004 - 0x0000000D) >> 1
;    temp1 = (0xFFFFFFF7) >> 1
;    temp1 = 0xFFFFFFFFB
;    VR5H  = temp1[15:0] = 0xFFFB = -5
; Imaginary:
;    temp2 = (0x00000006 - 0x00000016) >> 1
;    temp2 = (0xFFFFFFF0) >> 1
;    temp2 = 0xFFFFFFF8
;    VR5L  = temp2[15:0] = 0xFFF8 = -8
;
    VSATOFF                      ; VSTATUS[SAT] = 0
    VRNDOFF                      ; VSTATUS[RND] = 0
    VSETSHR   #1                 ; VSTATUS[SHIFTR] = 1
    VSETSHL   #0                 ; VSTATUS[SHIFTL] = 0
    VCLEARALL                    ; VR0, VR1...VR8 == 0
    VMOVXI    VR3, #13           ; VR3 = Re(Y) =  13 = 0x0000000D
    VMOVXI    VR2, #22           ; VR2 = Im(Y) = 22j = 0x00000016
    VMOVXI    VR4, #6
    VMOVIX    VR4, #4            ; VR4 = X = 0x00040006 =  4 +  6j
    VCDSUB16  VR6, VR4, VR3, VR2 ; VR5 = Z = 0xFFFBFFF8 = -5 + -8j
```

The next example illustrates rounding with a right shift value defined.

```
;
; Example: Z = X-Y with Rounding and Right Shift
;
; X =   4 +  6j           (16-bit real + 16-bit imaginary)
; Y = -13 + 22j           (32-bit real + 32-bit imaginary)
;
; Real:
;    temp1 = round((0x00000004 - 0xFFFFFFF3) >> 1)
;    temp1 = round(0x00000011) >> 1)
;    temp1 = round(0x000000008.8) = 0x000000009
;    VR5H  = temp1[15:0] = 0x0009 = 9
; Imaginary:
;    temp2 = round((0x00000006 - 0x00000016) >> 1)
;    temp2 = round(0xFFFFFFF0) >> 1)
;    temp2 = round(0xFFFFFFF8.0) = 0xFFFFFFF8
;    VR5L  = temp2[15:0] = 0xFFF8 = -8
;
    VSATOFF                       ; VSTATUS[SAT] = 0
    VRNDON                        ; VSTATUS[RND] = 1
    VSETSHR   #1                  ; VSTATUS[SHIFTR] = 1
    VSETSHL   #0                  ; VSTATUS[SHIFTL] = 0
    VCLEARALL                     ; VR0, VR1...VR8 == 0
    VMOVXI    VR3, #-13           ; VR3 = Re(Y)
    VMOVIX    VR3, #0xFFFF        ; sign extend VR3 = -13 = 0xFFFFFFF3
    VMOVXI    VR2, #22            ; VR2 = Im(Y) = 22j = 0x00000016
    VMOVXI    VR4, #6
    VMOVIX    VR4, #4             ; VR4 = X = 0x00040006 =  4 +  6j
    VCDSUB16  VR6, VR4, VR3, VR2  ; VR5 = Z = 0x0009FFF8 =  9 + -8j
```

The next example illustrates rounding with both a left and a right shift value defined.

```
;
; Example: Z = X-Y with Rounding and both Left and Right Shift
;
; X =   4 +  6j     (16-bit real + 16-bit imaginary)
; Y = -13 + 22j     (32-bit real + 32-bit imaginary)
;
; Real:
;    temp1 = round((0x00000004 << 2 - 0xFFFFFFF3) >> 1)
;    temp1 = round((0x00000010      - 0xFFFFFFF3) >> 1)
;    temp1 = round( 0x0000001D >> 1)
;    temp1 = round( 0x0000000E.8) = 0x0000000F
;    VR5H  = temp1[15:0] = 0x000F = 15
; Imaginary:
;    temp2 = round((0x00000006 << 2 - 0x00000016) >> 1)
;    temp2 = round((0x00000018      - 0x00000016) >> 1)
;    temp2 = round( 0x00000002 >> 1)
;    temp1 = round( 0x00000001.0) = 0x00000001
;    VR5L  = temp2[15:0] = 0x0001 = 1
;
    VSATOFF                       ; VSTATUS[SAT] = 0
    VRNDON                        ; VSTATUS[RND] = 1
    VSETSHR   #1                  ; VSTATUS[SHIFTR] = 1
    VSETSHL   #2                  ; VSTATUS[SHIFTL] = 2
    VCLEARALL                     ; VR0, VR1...VR8 == 0
    VMOVXI    VR3, #-13           ; VR3 = Re(Y)
    VMOVIX    VR3, #0xFFFF        ; sign extend VR3 = -13 = 0xFFFFFFF3
    VMOVXI    VR2, #22            ; VR2 = Im(Y) = 22j = 0x00000016
    VMOVXI    VR4, #6
    VMOVIX    VR4, #4             ; VR4 = X = 0x00040006 =  4 +  6j
    VCDSUB16  VR6, VR4, VR3, VR2  ; VR5 = Z = 0x000F0001 = 15 +  1j
```

See also

VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
VCADD VR7, VR6, VR5, VR4
VRNDOFF

VRNDON
VSATON
VSATOFF
VSETSHL #5-bit
VSETSHR #5-bit

VCDSUB16 VR6, VR4, VR3, VR2 || VMOV32 VRa, mem32 *Complex 16+32 = 16 Add with Parallel Load*

Operands

Before the operation, the inputs should be loaded into registers as shown below. The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR4H	16-bit integer representing the real part of the first input: Re(X)
VR4L	16-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)
mem32	pointer to a 32-bit memory location.

The result is a complex number with a 16-bit real and a 16-bit imaginary part. The result is stored in VR6 as shown below:

Output Register	Value
VR6H	16-bit integer representing the real part of the result: Re(Z) = (Re(X) << SHIFTL) + (Re(Y)) >> SHIFTR
VR6L	16-bit integer representing the imaginary part of the result: Im(Z) = (Im(X) << SHIFTL) + (Im(Y)) >> SHIFTR
VRa	Contents of the memory pointed to by [mem32]. VRa can not be VR6 or VR8.

Opcode

```
LSW: 1110 0010 1100 1010
MSW: 0000 0000  mem16
```

Description

Complex 16 - 32 = 16-bit operation with parallel load. This operation is useful for algorithms similar to a complex FFT.

The first operand is a complex number with a 16-bit real and 16-bit imaginary part. The second operand has a 32-bit real and a 32-bit imaginary part.

Before the addition, the first input is sign extended to 32-bits and shifted left by VSTATUS[VSHIFTL] bits. The result of the subtraction is left shifted by VSTATUS[VSHIFTR] before it is stored in VR5H and VR5L. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of a 16-bit overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
// SHIFTL is VSTATUS[SHIFTL]
//
// VR4H = Re(X)    16-bit
// VR4L = Im(X)    16-bit
// VR3  = Re(Y)    32-bit
// VR2  = Im(Y)    32-bit

   temp1 = sign_extend(VR4H);        // 32-bit extended Re(X)
   temp2 = sign_extend(VR4L);        // 32-bit extended Im(X)

   if (RND == 1)
   {
      temp1 = round(temp1 >> SHIFTR);
      temp2 = round(temp2 >> SHIFTR);
   }
   else
   {
      temp1 = truncate(temp1 >> SHIFTR);
      temp2 = truncate(temp2 >> SHIFTR);
   }
   if (SAT == 1)
```

```
{
   VR5H = sat16(temp1);
   VR5L = sat16(temp2);
}
else
{
   VR5H = temp1[15:0];
   VR5L = temp2[15:0];
}
VRa = [mem32];
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the real-part (VR6H) computation overflows or underflows.
- OVFI is set if the imaginary-part (VR6I) computation overflows or underflows.

Pipeline

Both operations complete in a single cycle.

Example

For more information regarding the subtraction operation, please refer to VCDSUB16 VR6, VR4, VR3, VR2.

```
;
; Example: Z = X-Y with Rounding and both Left and Right Shift
;
; X =   4 +  6j     (16-bit real + 16-bit imaginary)
; Y = -13 + 22j     (32-bit real + 32-bit imaginary)
;
; Real:
;    temp1 = round((0x00000004 << 2 - 0xFFFFFFF3) >> 1)
;    temp1 = round((0x00000010      - 0xFFFFFFF3) >> 1)
;    temp1 = round( 0x0000001D >> 1)
;    temp1 = round( 0x0000000E.8) = 0x0000000F
;    VR5H  = temp1[15:0] = 0x000F = 15
; Imaginary:
;    temp2 = round((0x00000006 << 2 - 0x00000016) >> 1)
;    temp2 = round((0x00000018      - 0x00000016) >> 1)
;    temp2 = round( 0x00000002 >> 1)
;    temp1 = round( 0x00000001.0) = 0x00000001
;    VR5L  = temp2[15:0] = 0x0001 = 1
;
    VSATOFF                        ; VSTATUS[SAT] = 0
    VRNDON                         ; VSTATUS[RND] = 1
    VSETSHR   #1                   ; VSTATUS[SHIFTR] = 1
    VSETSHL   #2                   ; VSTATUS[SHIFTL] = 2
    VCLEARALL                      ; VR0, VR1...VR8 == 0
    VMOVXI    VR3, #-13            ; VR3 = Re(Y)
    VMOVIX    VR3, #0xFFFF         ; sign extend VR3 = -13 = 0xFFFFFFF3
    VMOVXI    VR2, #22             ; VR2 = Im(Y) = 22j = 0x00000016
    VMOVXI    VR4, #6
    VMOVIX    VR4, #4              ; VR4 = X = 0x00040006 =  4 +  6j
    VCDSUB16  VR6, VR4, VR3, VR2   ; VR5 = Z = 0x000F0001 = 15 +  1j
 || VCMOV32   VR2, *XAR7           ; VR2 = contents pointed to by XAR7
```

See also

VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
VCADD VR7, VR6, VR5, VR4
VRNDOFF
VRNDON
VSATON
VSATOFF
VSETSHL #5-bit
VSETSHR #5-bit

VCMAC VR5, VR4, VR3, VR2, VR1, VR0 *Complex Multiply and Accumulate*

Operands

Before the operation, the inputs should be loaded into registers as shown below.

Input Register	Value
VR5	32-bit integer, previous real-part accumulation
VR4	32-bit integer, previous imaginary-part accumulation
VR3	32-bit integer, real result from the previous multiply
VR2	32-bit integer, imaginary result from the previous multiply
VR0H	16-bit integer representing the real part of the first input: Re(X)
VR0L	16-bit integer representing the imaginary part of the first input: Im(X)
VR1H	16-bit integer representing the real part of the second input: Re(Y)
VR1L	16-bit integer representing the imaginary part of the second input: Im(Y)

Note: The user will need to do one final addition to accumulate the final multiplications (Real-VR3 and Imaginary-VR2) into the result registers.

The result is stored as shown below:

Output Register	Value
VR5	32-bit real part of the total accumulation Re(sum) = Re(sum) + Re(mpy)
VR4	32-bit imaginary part of the total accumulation Im(sum) = Im(sum) + Im(mpy

Opcode

```
LSW: 1110 0101 0011 0001
```

Description

Complex multiply operation.

```
// VR5 = Accumulation of the real part
// VR4 = Accumulation of the imaginary part
//
// VR0 = X + jX:   VR0[31:16] = X,   VR0[15:0] = jX
// VR1 = Y + jY:   VR1[31:16] = Y,   VR1[15:0] = jY
//
// Perform add
//
   if (RND == 1)
   {
      VR5 = VR5 + round(VR3 >> SHIFTR);
      VR4 = VR4 + round(VR2 >> SHIFTR);
   }
   else
   {
      VR5 = VR5 + (VR3 >> SHIFTR);
      VR4 = VR4 + (VR2 >> SHIFTR);
   }
//
// Perform multiply (X + jX) * (Y * jY)
//
   VR3 = VR0H * VR1H - VR0L * VR1L;   Real result
   VR2 = VR0H * VR1L + VR0L * VR1H;   Imaginary result
   if(SAT == 1)
   {
      sat32(VR3);
      sat32(VR2);
   }
   VRa = [mem32];
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR3 computation (real part) overflows or underflows.
- OVFI is set if the VR2 computation (imaginary part) overflows or underflows.

Pipeline

This is a 2p-cycle instruction.

Example

See also

VCLROVFI
VCLROVFR
VCMAC VR5, VR4, VR3, VR2, VR1, VR0 || VMOV32 VRa, mem32
VSATON
VSATOFF

VCMAC VR5, VR4, VR3, VR2, VR1, VR0 || VMOV32 VRa, mem32 *Complex Multiply and Accumulate with Parallel Load*

Operands

Before the operation, the inputs should be loaded into registers as shown below.

Input Register	Value
VR5	Previous real-part accumulation
VR4	Previous imaginary-part accumulation
VR3	32-bit real result from the previous multiply
VR2	32-bit imaginary result from the previous multiply
VR0H	16-bit integer representing the real part of the first input: Re(X)
VR0L	16-bit integer representing the imaginary part of the first input: Im(X)
VR1H	16-bit integer representing the real part of the second input: Re(Y)
VR1L	16-bit integer representing the imaginary part of the second input: Im(Y)
mem32	Pointer to 32-bit memory location.

Note: The user will need to do one final addition to accumulate the final multiplications (Real-VR3 and Imaginary-VR2) into the result registers.

The result is stored as shown below:

Output Register	Value
VR5	32-bit real part of the total accumulation Re(sum) = Re(sum) + Re(mpy)
VR4	32-bit imaginary part of the total accumulation Im(sum) = Im(sum) + Im(mpy)
VRa	Contents of the memory pointed to by [mem32]. VRa cannot be VR5, VR4 or VR8

Note:

Opcode

```
LSW: 1110 0010 1100 1010
MSW: 0000 0000  mem32
```

Description

Complex multiply operation.

```
// VR5 = Accumulation of the real part
// VR4 = Accumulation of the imaginary part
//
// VR0 = X + Xj:   VR0[31:16] = Re(X),   VR0[15:0] = Im(X)
// VR1 = Y + Yj:   VR1[31:16] = Re(Y),   VR1[15:0] = Im(Y)
//
// Perform add
//
   if (RND == 1)
   {
      VR5 = VR5 + round(VR3 >> SHIFTR);
      VR4 = VR4 + round(VR2 >> SHIFTR);
   }
   else
   {
      VR5 = VR5 + (VR3 >> SHIFTR);
      VR4 = VR4 + (VR2 >> SHIFTR);
   }
//
// Perform multiply Z =  (X + Xj) * (Y * Yj)
//
   VR3 = VR0H * VR1H - VR0L * VR1L;   // Re(Z)
   VR2 = VR0H * VR1L + VR0L * VR1H;   // Im(Z)
   if(SAT == 1)
   {
      sat32(VR3);
      sat32(VR2);
   }
   VRa = [mem32];
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR3 computation (real part) overflows or underflows.
- OVFI is set if the VR2 computation (imaginary part) overflows or underflows.

Pipeline

This is a 2p/1-cycle instruction. The multiply and accumulate is a 2p-cycle operation and the VMOV32 is a single-cycle operation.

Example

See also

VCLROVFI
VCLROVFR
VCMAC VR5, VR4, VR3, VR2, VR1, VR0
VSATON
VSATOFF

VCMAC VR7, VR6, VR5, VR4, mem32, *XAR7++ *Complex Multiply and Accumulate*

Operands

The VMAC alternates which registers are used between each cycle. For odd cycles (1, 3, 5, etc) the following registers are used:

Odd Cycle Input	Value
VR5	Previous real-part total accumulation: Re(odd_sum)
VR4	Previous imaginary-part total accumulation: Im(odd-sum)
VR1	Previous real result from the multiply: Re(odd-mpy)
VR0	Previous imaginary result from the multiply Im(odd-mpy)
[mem32]	Pointer to a 32-bit memory location representing the first input to the multiply [mem32][31:16] = Re(X) [mem32][15:0] = Im(X)
XAR7	Pointer to a 32-bit memory location representing the second input to the multiply *XAR7[31:16] = Re(Y) *XAR7[15:0] = Im(Y)

The result from odd cycle is stored as shown below:

Odd Cycle Output	Value
VR5	32-bit real part of the total accumulation Re(odd_sum) = Re(odd_sum) + Re(odd_mpy)
VR4	32-bit imaginary part of the total accumulation Im(sum) = Im(odd_sum) + Im(odd_mpy)
VR1	32-bit real result from the multiplication: Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
VR0	32-bit imaginary result from the multiplication: Im(Z) = Re(X)*Im(Y) + Re(Y)*Im(X)

For even cycles (2, 4, 6, etc) the following registers are used:

Even Cycle Input	Value
VR7	Previous real-part total accumulation: Re(even_sum)
VR6	Previous imaginary-part total accumulation: Im(even-sum)
VR3	Previous real result from the multiply: Re(even-mpy)
VR2	Previous imaginary result from the multiply Im(even-mpy)
[mem32]	Pointer to a 32-bit memory location representing the first input to the multiply [mem32][31:16] = Re(X); (a) [mem32][15:0] = Im(X); (b)
XAR7	Pointer to a 32-bit memory location representing the second input to the multiply: *XAR7[31:16] = Re(Y); (c) *XAR7[15:0] = Im(Y); (d)

The result from even cycles is stored as shown below:

Even Cycle Output	Value
VR7	32-bit real part of the total accumulation Re(even_sum) = Re(even_sum) + Re(even_mpy)
VR6	32-bit imaginary part of the total accumulation Im(even_sum) = Im(even_sum) + Im(even_mpy)
VR3	32-bit real result from the multiplication: Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
VR2	32-bit imaginary result from the multiplication: Im(Z) = Re(X)*Im(Y) + Re(Y)*Im(X)

Opcode

```
LSW: 1110 0010 0101 0000
MSW: 00bb baaa mem32
```

Description

Perform a repeated multiply and accumulate operation. This instruction is the only VCU instruction that can be repeated using the single repeat instruction (RPT ||). When repeated, the destination of the accumulate will alternate between VR7/VR6 and VR5/VR4 on each cycle.

```
// Cycle 1:
//
// Perform accumulate
//
   if(RND == 1)
   {
      VR5 = VR5 + round(VR1 >> SHIFTR)
      VR4 = VR4 + round(VR0 >> SHIFTR)
    }
    else
    {
      VR5 = VR5 + (VR1 >> SHIFTR)
      VR4 = VR4 + (VR0 >> SHIFTR)
    }
    //
    // X and Y array element 0
    //
    VR1 = Re(X)*Re(Y) - Im(X)*Im(Y)
    VR0 = Re(X)*Im(Y) + Re(Y)*Im(X)
//
// Cycle 2:
//
// Perform accumulate
//
   if(RND == 1)
   {
      VR7 = VR7 + round(VR3 >> SHIFTR)
      VR6 = VR6 + round(VR2 >> SHIFTR)
    }
    else
    {
      VR7 = VR7 + (VR3 >> SHIFTR)
      VR6 = VR6 + (VR2 >> SHIFTR)
    }
    //
    // X and Y array element 1
    //
    VR3 = Re(X)*Re(Y) - Im(X)*Im(Y)
    VR2 = Re(X)*Im(Y) + Re(Y)*Im(X)
//
// Cycle 3:
//
// Perform accumulate
//
   if(RND == 1)
   {
      VR5 = VR5 + round(VR1 >> SHIFTR)
      VR4 = VR4 + round(VR0 >> SHIFTR)
    }
    else
    {
      VR5 = VR5 + (VR1 >> SHIFTR)
      VR4 = VR4 + (VR0 >> SHIFTR)
    }
    //
    // X and Y array element 2
    //
    VR1 = Re(X)*Re(Y) - Im(X)*Im(Y)
    VR0 = Re(X)*Im(Y) + Re(Y)*Im(X)
  etc...
```

Restrictions VR0, VR1, VR2, and VR3 will be used as temporary storage by this instruction.

Flags The VSTATUS register flags are modified as follows:

- OVFR is set in the case of an overflow or underflow of the addition or subtraction operations.

- OVFI is set in the case an overflow or underflow of the imaginary part of the addition or subtraction operations.

Pipeline

When repeated the VMAC takes 2p + N cycles where N is the number of times the instruction is repeated. When repeated, this instruction has the following pipeline restrictions:

```
   <instruction1>                        ; No restriction
   <instruction2>                        ; Cannot be a 2p instruction that writes
                                         ; to VR0, VR1...VR7 registers
   RPT #(N-1)                            ; Execute N times, where N is even
|| VCMAC VR7, VR6, VR5, VR4, *XAR6++, *XAR7++
   <instruction3>                        ; No restrictions.
                                         ; Can read VR0, VR1... VR8
```

MACF32 can also be used standalone. In this case, the insruction takes 2 cycles and the following pipeline restrictions apply:

```
 <instruction1> ; No restriction <instruction2> ; Cannot be a 2p instruction that
writes ; to R2H, R3H, R6H or R7H MACF32 R7H, R3H, *XAR6, *XAR7 ; R3H = R3H + R2H,
R2H = [mem32] * [XAR7++] ; <--
 R2H and R3H are valid (note: no delay required) NOP
```

Example

Cascading of RPT || VMAC is allowed as long as the first and subsequent counts are even. Cascading is useful for creating interruptible windows so that interrupts are not delayed too long by the RPT instruction. For example:

```
;
; Example of cascaded VMAC instructions
;
   VCLEARALL        ; Zero the accumulation registers
;
; Execute MACF32 N+1 (4) times
;
   RPT #3
|| VCMAC VR7, VR6, VR5, VR4, *XAR6++, *XAR7++
;
; Execute MACF32 N+1 (6) times
;
   RPT #5
|| VCMAC VR7, VR6, VR5, VR4, *XAR6++, *XAR7++
;
; Repeat MACF32 N+1 times where N+1 is even
;
   RPT #N
|| MACF32 R7H, R3H, *XAR6++, *XAR7++
   ADDF32 VR7, VR6, VR5, VR4
```

See also

VCMPY VR3, VR2, VR1, VR0 *Complex Multiply*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Both inputs are complex numbers with a 16-bit real and 16-bit imaginary part.

Input Register	Value
VR0H	16-bit integer representing the real part of the first input: Re(X)
VR0L	16-bit integer representing the imaginary part of the first input: Im(X)
VR1H	16-bit integer representing the real part of the 2nd input: Re(Y)
VR1L	16-bit integer representing the imaginary part of the 2nd input: Im(Y)

The result is a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR2 and VR3 as shown below:

Output Register	Value
VR3	16-bit integer representing the real part of the result: Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
VR2	16-bit integer representing the imaginary part of the result: Im(Z) = Re(X)*Im(Y) + Im(X)*Re(Y)

Opcode

```
LSW: 1110 0101 0000 0000
```

Description

Complex 16 x 16 = 32-bit multiply operation.

If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of a 32-bit overflow or underflow.

```
// VR0 = X + Xj:   VR0[31:16] = Re(X),   VR0[15:0] = Im(X)
// VR1 = Y + Yj:   VR1[31:16] = Re(Y),   VR1[15:0] = Im(Y)
//
// Calculate: Z = (X + jX) * (Y + jY)
//
   VR3 = VR0H * VR1H - VR0L * VR1L;     // Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
   VR2 = VR0H * VR1L + VR0L * VR1H;     // Im(Z) = Re(X)*Im(Y) + Im(X)*Re(Y)
   if(SAT == 1)
   {
      sat32(VR3);
      sat32(VR2);
   }
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR3 computation (real part) overflows or underflows.
- OVFI is set if the VR2 computation (imaginary part) overflows or underflows.

Pipeline

This is a 2p-cycle instruction. The instruciton following this one should not use VR3 or VR2.

Example

```
; Example 1
; X =  4 + 6j
; Y = 12 + 9j
;
; Z = X * Y
; Re(Z) = 4*12 - 6*9  = -6
; Im(Z) = 4*9  + 6*12 = 108
;
    VSATOFF                         ; VSTATUS[SAT] = 0
    VCLEARALL                       ; VR0, VR1...VR8 == 0
    VMOVXI    VR0, #6
    VMOVIX    VR0, #4               ; VR0 = X = 0x00040006 =  4 +  6j
    VMOVXI    VR1, #9
    VMOVIX    VR1, #12              ; VR1 = Y = 0x000C0009 = 12 +  9j
    VCMPY     VR3, VR2, VR1, VR0    ; VR3 = Re(Z) = 0xFFFFFFFA = -6
                                    ; VR2 = Im(Z) = 0x0000006C = 108
```

```
<instruction 1>              ; <- Must not use VR2, VR3
                             ; <- VCMPY completes, VR2, VR3 valid
<instruciton 2>              ; Can use VR2, VR3
```

See also

VCLROVFI
VCLROVFR
VCMAC VR5, VR4, VR3, VR2, VR1, VR0
VCMAC VR5, VR4, VR3, VR2, VR1, VR0 || VMOV32 VRa, mem32
VSATON
VSATOFF

VCMPY VR3, VR2, VR1, VR0 || VMOV32 mem32, VRa *Complex Multiply with Parallel Store*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Both inputs are complex numbers with a 16-bit real and 16-bit imaginary part.

Input Register	Value
VR0H	16-bit integer representing the real part of the first input: Re(X)
VR0L	16-bit integer representing the imaginary part of the first input: Im(X)
VR1H	16-bit integer representing the real part of the 2nd input: Re(Y)
VR1L	16-bit integer representing the imaginary part of the 2nd input: Im(Y)
VRa	Value to be stored.

The result is a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR2 and VR3 as shown below:

Output Register	Value
VR3	16-bit integer representing the real part of the result: Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
VR2	16-bit integer representing the imaginary part of the result: Im(Z) = Re(X)*Im(Y) + Im(X)*Re(Y)
[mem32]	Contents of VRa. VRa can be VR0-VR7. VRa can not be VR8.

Opcode

```
LSW: 1110 0010 1100 1010
MSW: 0000 0000  mem16
```

Description

Complex 16 x 16 = 32-bit multiply operation with parallel register load.

If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of a 32-bit overflow or underflow.

```
// VR0 = X + jX:   VR0[31:16] = Re(X),   VR0[15:0] = Im(X)
// VR1 = Y + jY:   VR1[31:16] = Re(Y),   VR1[15:0] = Im(Y)
//
// Calculate: Z = (X + jX) * (Y + jY)
//
   VR3 = VR0H * VR1H - VR0L * VR1L;     // Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
   VR2 = VR0H * VR1L + VR0L * VR1H;     // Im(Z) = Re(X)*Im(Y) + Im(X)*Re(Y)
   if(SAT == 1)
   {
      sat32(VR3);
      sat32(VR2);
   }
   VRa = [mem32];
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR3 computation (real part) overflows or underflows.
- OVFI is set if the VR2 computation (imaginary part) overflows or underflows.

Pipeline

This is a 2p/1-cycle instruction. The multply operation takes 2p cycles and the VMOV operation completes in a single cycle. The instruction following this one must not use VR2 or VR3.

Example

```
; Example 1
; X =  4 + 6j
; Y = 12 + 9j
;
; Z = X * Y
; Re(Z) = 4*12 - 6*9  = -6
; Im(Z) = 4*9  + 6*12 = 108
;
    VSATOFF                        ; VSTATUS[SAT] = 0
    VCLEARALL                      ; VR0, VR1...VR8 == 0
    VMOVXI    VR0, #6
    VMOVIX    VR0, #4               ; VR0 = X = 0x00040006 =  4 +  6j
    VMOVXI    VR1, #9
    VMOVIX    VR1, #12              ; VR1 = Y = 0x000C0009 = 12 +  9j
                                    ; VR3 = Re(Z) = 0xFFFFFFFA = -6
    VCMPY     VR3, VR2, VR1, VR0    ; VR2 = Im(Z) = 0x0000006C = 108
||  VMOV32    *XAR7, VR3            ; Location XAR7 points to = VR3 (before
multiply)
    <instruction 1>                 ; <- Must not use VR2, VR3
                                    ; <- VCMPY completes, VR2, VR3 valid
    <instruciton 2>                 ; Can use VR2, VR3
```

See also

VCLROVFI
VCLROVFR
VCMAC VR5, VR4, VR3, VR2, VR1, VR0
VCMAC VR5, VR4, VR3, VR2, VR1, VR0 || VMOV32 VRa, mem32
VSATON
VSATOFF

VCMPY VR3, VR2, VR1, VR0 || VMOV32 VRa, mem32 *Complex Multiply with Parallel Load*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Both inputs are complex numbers with a 16-bit real and 16-bit imaginary part.

Input Register	Value
VR0H	16-bit integer representing the real part of the first input: Re(X)
VR0L	16-bit integer representing the imaginary part of the first input: Im(X)
VR1H	16-bit integer representing the real part of the 2nd input: Re(Y)
VR1L	16-bit integer representing the imaginary part of the 2nd input: Im(Y)
mem32	pointer to 32-bit memory location

The result is a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR2 and VR3 as shown below:

Output Register	Value
VR3	16-bit integer representing the real part of the result: Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
VR2	16-bit integer representing the imaginary part of the result: Im(Z) = Re(X)*Im(Y) + Im(X)*Re(Y)
VRa	32-bit value pointed to by [mem32]. VRa can not be VR2, VR3 or VR8.

Opcode

```
LSW: 1110 0011 1111 0110
MSW: 0000 aaaa  mem32
```

Description

Complex 16 x 16 = 32-bit multiply operation with parallel register load.

If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of a 32-bit overflow or underflow.

```
// VR0 = X + jX:   VR0[31:16] = Re(X),   VR0[15:0] = Im(X)
// VR1 = Y + jY:   VR1[31:16] = Re(Y),   VR1[15:0] = Im(Y)
//
// Calculate: Z = (X + jX) * (Y + jY)
//
   VR3 = VR0H * VR1H - VR0L * VR1L;     // Re(Z) = Re(X)*Re(Y) - Im(X)*Im(Y)
   VR2 = VR0H * VR1L + VR0L * VR1H;     // Im(Z) = Re(X)*Im(Y) + Im(X)*Re(Y)
   if(SAT == 1)
   {
      sat32(VR3);
      sat32(VR2);
   }
   VRa = [mem32];
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR3 computation (real part) overflows or underflows.
- OVFI is set if the VR2 computation (imaginary part) overflows or underflows.

Pipeline

This is a 2p/1-cycle instruction. The multply operation takes 2p cycles and the VMOV operation completes in a single cycle. The instruction following this one must not use VR2 or VR3.

Example

```
; Example 1
; X =  4 + 6j
; Y = 12 + 9j
;
; Z = X * Y
; Re(Z) = 4*12 - 6*9  = -6
; Im(Z) = 4*9  + 6*12 = 108
;
    VSATOFF                        ; VSTATUS[SAT] = 0
    VCLEARALL                      ; VR0, VR1...VR8 == 0
    VMOVXI    VR0, #6
    VMOVIX    VR0, #4             ; VR0 = X = 0x00040006 =  4 +  6j
    VMOVXI    VR1, #9
    VMOVIX    VR1, #12            ; VR1 = Y = 0x000C0009 = 12 +  9j
                                  ; VR3 = Re(Z) = 0xFFFFFFFA = -6
    VCMPY     VR3, VR2, VR1, VR0  ; VR2 = Im(Z) = 0x0000006C = 108
||  VMOV32    VR0, *XAR7          ; VR0 = contents of location XAR7 points to
    <instruction 1>               ; <- Must not use VR2, VR3
                                  ; <- VCMPY completes, VR2, VR3 valid
    <instruciton 2>               ; Can use VR2, VR3
```

See also

VCLROVFI
VCLROVFR
VCMAC VR5, VR4, VR3, VR2, VR1, VR0
VCMAC VR5, VR4, VR3, VR2, VR1, VR0 || VMOV32 VRa, mem32
VSATON
VSATOFF

VNEG VRa *Two's Complement Negate*

Operands

VRa	VRa can be VR0 - VR7. VRa can not be VR8.

Opcode

```
LSW: 1110 0101 0001 aaaa
```

Description

Complex add operation.

```
// SAT    is VSTATUS[SAT]
//
   if (VRa == 0x800000000)
   {
      if(SAT == 1)
      {
         VRa = 0x7FFFFFFF;
      }
      else
      {
         VRa = 0x80000000;
      }
   }
   else
   {
      VRa = - VRa
   }
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the input to the operation is 0x80000000.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLROVFR
VSATON
VSATOFF

VCSUB VR5, VR4, VR3, VR2 *Complex 32 - 32 = 32 Subtraction*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Each complex number includes a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR5	32-bit integer representing the real part of the first input: Re(X)
VR4	32-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)

The result is also a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR5 and VR4 as shown below:

Output Register	Value
VR5	32-bit integer representing the real part of the result: Re(Z) = Re(X) - (Re(Y) >> SHIFTR)
VR4	32-bit integer representing the imaginary part of the result: Im(Z) = Im(X) - (Im(Y) >> SHIFTR)

Opcode

```
LSW: 1110 0101 0000 0011
```

Description

Complex 32 - 32 = 32-bit subtraction operation.

The second input operand (stored in VR3 and VR2) is shifted right by VSTATUS[SHIFR] bits before the subtraction. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of an overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
//
   if (RND == 1)
   {
      VR5 = VR5 - round(VR3 >> SHIFTR);
      VR4 = VR4 - round(VR2 >> SHIFTR);
   }
   else
   {
      VR5 = VR5 - (VR3 >> SHIFTR);
      VR4 = VR4 - (VR2 >> SHIFTR);
   }
   if (SAT == 1)
   {
      sat32(VR5);
      sat32(VR4);
   }
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR5 computation (real part) overflows or underflows.
- OVFI is set if the VR6 computation (imaginary part) overflows or underflows.

Pipeline

This is a single-cycle instruction.

Example

See also

VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
VCADD VR7, VR6, VR5, VR4
VCSUB VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
VCLROVFI

VCLROVFR
VRNDOFF
VRNDON
VSATON
VSATOFF
VSETSHR #5-bit

VCSUB VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32 *Complex Subtraction*

Operands

Before the operation, the inputs should be loaded into registers as shown below. Each complex number includes a 32-bit real and a 32-bit imaginary part.

Input Register	Value
VR5	32-bit integer representing the real part of the first input: Re(X)
VR4	32-bit integer representing the imaginary part of the first input: Im(X)
VR3	32-bit integer representing the real part of the 2nd input: Re(Y)
VR2	32-bit integer representing the imaginary part of the 2nd input: Im(Y)
mem32	pointer to a 32-bit memory location

The result is also a complex number with a 32-bit real and a 32-bit imaginary part. The result is stored in VR5 and VR4 as shown below:

Output Register	Value
VR5	32-bit integer representing the real part of the result: Re(Z) = Re(X) - (Re(Y) >> SHIFTR)
VR4	32-bit integer representing the imaginary part of the result: Im(Z) = Im(X) - (Im(Y) >> SHIFTR)
VRa	contents of the memory pointed to by [mem32]. VRa can not be VR5, VR4 or VR8.

Opcode

```
LSW: 1110 0010 1100 1010
MSW: 0000 0000  mem16
```

Description

Complex 32 - 32 = 32-bit subtraction operation with parallel load.

The second input operand (stored in VR3 and VR2) is shifted right by VSTATUS[SHIFR] bits before the subtraction. If VSTATUS[RND] is set, then bits shifted out to the right are rounded, otherwise these bits are truncated. The rounding operation is described in . If the VSTATUS[SAT] bit is set, then the result will be saturated in the event of an overflow or underflow.

```
// RND    is VSTATUS[RND]
// SAT    is VSTATUS[SAT]
// SHIFTR is VSTATUS[SHIFTR]
//
   if (RND == 1)
   {
      VR5 = VR5 - round(VR3 >> SHIFTR);
      VR4 = VR4 - round(VR2 >> SHIFTR);
   }
   else
   {
      VR5 = VR5 - (VR3 >> SHIFTR);
      VR4 = VR4 - (VR2 >> SHIFTR);
   }
   if (SAT == 1)
   {
      sat32(VR5);
      sat32(VR4);
   }
   VRa = [mem32];
```

Flags

This instruction modifies the following bits in the VSTATUS register:

- OVFR is set if the VR5 computation (real part) overflows or underflows.
- OVFI is set if the VR6 computation (imaginary part) overflows or underflows.

Pipeline

This is a single-cycle instruction.

Example

See also

VCADD VR5, VR4, VR3, VR2 || VMOV32 VRa, mem32
VCADD VR7, VR6, VR5, VR4
VCSUB VR5, VR4, VR3, VR2
VCLROVFI
VCLROVFR
VRNDOFF
VRNDON
VSATON
VSATOFF
VSETSHR #5-bit

10.5.4 Cyclic Redundancy Check (CRC) Instructions

The instructions are listed alphabetically, preceded by a summary.

Table 10-12. CRC Instructions

Title	Page
VCRC8H_1 mem16 —CRC8, High Byte	771
VCRC8L_1 mem16 —CRC8 , Low Byte	772
VCRC16P1H_1 mem16 —CRC16, Polynomial 1, High Byte	773
VCRC16P1L_1 mem16 —CRC16, Polynomial 1, Low Byte	774
VCRC16P2H_1 mem16 —CRC16, Polynomial 2, High Byte	775
VCRC16P2L_1 mem16 —CRC16, Polynomial 2, Low Byte	776
VCRC32H_1 mem16 —CRC32, High Byte	777
VCRC32L_1 mem16 —CRC32, Low Byte	778
VCRCCLR —Clear CRC Result Register	779
VMOV32 mem32, VCRC —Store the CRC Result Register	780
VMOV32 VCRC, mem32 —Load the CRC Result Register	781

VCRC8H_1 mem16 *CRC8, High Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 1100
MSW: 0000 0000   mem16
```

Description

This instruction uses CRC8 polynomial == 0x07.

Calculate the CRC8 of the most significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC8 (VCRC, mem16[15:8])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VCRC8L_1 mem16

See also

VCRC8L_1 mem16

VCRC8L_1 mem16 *CRC8 , Low Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 1011
MSW: 0000 0000  mem16
```

Description

This instruction uses CRC8 polynomial == 0x07.

Calculate the CRC8 of the least significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC8 (VCRC, mem16[7:0])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
   typedef struct {
      uint32_t *CRCResult;      //  Address where result should be stored
      uint16_t *CRCData;        //  Start of data
      uint16_t CRCLen;          //  Length of data in bytes
   }CRC_CALC;

   CRC_CALC mycrc;
   ...
   CRC8(&mycrc);
   ...

; -------------------
; Calculate the CRC of a block of data
; This function assumes the block is a multiple of 2 16-bit words
;
    .global _CRC8
_CRC8
    VCRCCLR                     ; Clear the result register
    MOV    AL,     *+XAR4[4]    ; AL = CRCLen
    ASR    AL,     2            ; AL = CRCLen/4
    SUBB   AL,     #1           ; AL = CRCLen/4 - 1
    MOVL   XAR7,   *+XAR4[2]    ; XAR7 = &CRCData
    .align 2
    NOP                         ; Align RPTB to an odd address
    RPTB _CRC8_done, AL         ; Execute block of code AL + 1 times
    VCRC8L_1 *XAR7              ; Calculate CRC for 4 bytes
    VCRC8H_1 *XAR7++            ; ...
    VCRC8L_1 *XAR7              ; ...
    VCRC8H_1 *XAR7++            ; ...
_CRC8_done
    MOVL   XAR7, *_+XAR4[0]     ; XAR7 = &CRCResult
    MOV32  *+XAR7[0], VCRC      ; Store the     result
    LRETR                       ; return to caller
```

See also

VCRC8H_1 mem16

VCRC16P1H_1 mem16 *CRC16, Polynomial 1, High Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 1111
MSW: 0000 0000   mem16
```

Description

This instruction uses CRC16 polynomial 1 == 0x8005.

Calculate the CRC16 of the most significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC16 (VCRC, mem16[15:8])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VCRC16P1L_1 mem16.

See also

VCRC16P1L_1 mem16
VCRC16P2H_1 mem16
VCRC16P2L_1 mem16

VCRC16P1L_1 mem16 *CRC16, Polynomial 1, Low Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 1110
MSW: 0000 0000   mem16
```

Description

This instruction uses CRC16 polynomial 1 == 0x8005.

Calculate the CRC16 of the least significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC16 (VCRC, mem16[7:0])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
   typedef struct {
      uint32_t *CRCResult;      //  Address where result should be stored
      uint16_t *CRCData;        //  Start of data
      uint16_t CRCLen;          //  Length of data in bytes
   }CRC_CALC;

   CRC_CALC mycrc;
   ...
   CRC16P1(&mycrc);
   ...

; -------------------
; Calculate the CRC of a block of data
; This function assumes the block is a multiple of 2 16-bit words
;
    .global _CRC16P1
_CRC16P1
    VCRCCLR                     ; Clear the result register
    MOV    AL,      *+XAR4[4]   ; AL = CRCLen
    ASR    AL,      2           ; AL = CRCLen/4
    SUBB   AL,      #1          ; AL = CRCLen/4 - 1
    MOVL   XAR7,    *+XAR4[2]   ; XAR7 = &CRCData
    .align 2
    NOP                         ; Align RPTB to an odd address
    RPTB _CRC16P1_done, AL      ; Execute block of code AL + 1 times
    VCRC16P1L_1 *XAR7           ; Calculate CRC for 4 bytes
    VCRC16P1H_1 *XAR7++         ; ...
    VCRC16P1L_1 *XAR7           ; ...
    VCRC16P1H_1 *XAR7++         ; ...
_CRC16P1_done
    MOVL   XAR7, *_+XAR4[0]     ; XAR7 = &CRCResult
    MOV32  *+XAR7[0], VCRC      ; Store the     result
    LRETR                       ; return to caller
```

See also

VCRC16P1H_1 mem16
VCRC16P2H_1 mem16
VCRC16P2L_1 mem16

VCRC16P2H_1 mem16 *CRC16, Polynomial 2, High Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 1111
MSW: 0001 0000  mem16
```

Description

This instruction uses CRC16 polynomial 2== 0x1021.

Calculate the CRC16 of the most significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC16 (VCRC, mem16[15:8])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VCRC16P2L_1 mem16.

See also

VCRC16P2L_1 mem16
VCRC16P1H_1 mem16
VCRC16P1L_1 mem16

VCRC16P2L_1 mem16 *CRC16, Polynomial 2, Low Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 1110
MSW: 0001 0000   mem16
```

Description

This instruction uses CRC16 polynomial 2== 0x1021.

Calculate the CRC16 of the least significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC16 (VCRC, mem16[7:0])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
   typedef struct {
      uint32_t *CRCResult;      //  Address where result should be stored
      uint16_t *CRCData;        //  Start of data
      uint16_t CRCLen;          //  Length of data in bytes
   }CRC_CALC;

   CRC_CALC mycrc;
   ...
   CRC16P2(&mycrc);
   ...

; -------------------
; Calculate the CRC of a block of data
; This function assumes the block is a multiple of 2 16-bit words
;
    .global _CRC16P2
_CRC16P2
    VCRCCLR                      ; Clear the result register
    MOV    AL,      *+XAR4[4]    ; AL = CRCLen
    ASR    AL,      2            ; AL = CRCLen/4
    SUBB   AL,      #1           ; AL = CRCLen/4 - 1
    MOVL   XAR7,    *+XAR4[2]    ; XAR7 = &CRCData
    .align 2
    NOP                          ; Align RPTB to an odd address
    RPTB _CRC16P2_done, AL       ; Execute block of code AL + 1 times
    VCRC16P2L_1 *XAR7            ; Calculate CRC for 4 bytes
    VCRC16P2H_1 *XAR7++          ; ...
    VCRC16P2L_1 *XAR7            ; ...
    VCRC16P2H_1 *XAR7++          ; ...
_CRC16P2_done
    MOVL   XAR7, *_+XAR4[0]      ; XAR7 = &CRCResult
    MOV32  *+XAR7[0], VCRC       ; Store the     result
    LRETR                        ; return to caller
```

See also

VCRC16P2H_1 mem16
VCRC16P1H_1 mem16
VCRC16P1L_1 mem16

VCRC32H_1 mem16 *CRC32, High Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 0010
MSW: 0000 0000   mem16
```

Description

This instruction uses CRC32 polynomial 1 == 0x04C11DB7

Calculate the CRC16 of the most significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC16 (VCRC, mem16[15:8])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VCRC32L_1 mem16.

See also

VCRC32L_1 mem16

VCRC32L_1 mem16 *CRC32, Low Byte*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0010 1100 0001
MSW: 0000 0000   mem16
```

Description

This instruction uses CRC32 polynomial 1 == 0x04C11DB7

Calculate the CRC32 of the least significant byte pointed to by mem16 and accumulate it with the value in the VCRC register. Store the result in VCRC.

```
VCRC = CRC32 (VCRC, mem16[7:0])
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
   typedef struct {
      uint32_t *CRCResult;      //  Address where result should be stored
      uint16_t *CRCData;        //  Start of data
      uint16_t CRCLen;          //  Length of data in bytes
   }CRC_CALC;

   CRC_CALC mycrc;
   ...
   CRC32(&mycrc);
   ...

; -------------------
; Calculate the CRC of a block of data
; This function assumes the block is a multiple of 2 16-bit words
;
    .global _CRC32
_CRC32
    VCRCCLR                    ; Clear the result register
    MOV    AL,     *+XAR4[4]   ; AL = CRCLen
    ASR    AL,     2           ; AL = CRCLen/4
    SUBB   AL,     #1          ; AL = CRCLen/4 - 1
    MOVL   XAR7,   *+XAR4[2]   ; XAR7 = &CRCData
    .align 2
    NOP                        ; Align RPTB to an odd address
    RPTB _CRC16P2_done, AL     ; Execute block of code AL + 1 times
    VCRC32_1 *XAR7             ; Calculate CRC for 4 bytes
    VCRC32_1 *XAR7++           ; ...
    VCRC32_1 *XAR7             ; ...
    VCRC32_1 *XAR7++           ; ...
_CRC32_done
    MOVL   XAR7, *_+XAR4[0]    ; XAR7 = &CRCResult
    MOV32  *+XAR7[0], VCRC     ; Store the     result
    LRETR                      ; return to caller
```

See also

VCRC32H_1 mem16

VCRCCLR *Clear CRC Result Register*

Operands

mem16	16-bit memory location

Opcode

```
LSW: 1110 0101 0010 0100
```

Description

Clear the VCRC register.

```
VCRC = 0x0000
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VCRC32L_1 mem16.

See also

VMOV32 mem32, VCRC
VMOV32 VCRC, mem32

VMOV32 mem32, VCRC *Store the CRC Result Register*

Operands

mem32	32-bit memory destination
VCRC	CRC result register

Opcode

```
LSW: 1110 0010 0000 0110
MSW: 0000 0000   mem32
```

Description

Store the VCRC register.

```
[mem32] = VCRC
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
x
```

See also

VCRCCLR
VMOV32 VCRC, mem32

VMOV32 VCRC, mem32 *Load the CRC Result Register*

Operands

mem32	32-bit memory destination
VCRC	CRC result register

Opcode

```
LSW: 1110 0011 1111 0110
MSW: 0000 0000   mem32
```

Description

Load the VCRC register.

```
VCRC = [mem32]
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VCRCCLR
VMOV32 mem32, VCRC

10.5.5 Viterbi Instructions

The instructions are listed alphabetically, preceded by a summary.

Table 10-13. Viterbi Instructions

Title	Page
VITBM2 VR0 —Code Rate 1:2 Branch Metric Calculation	783
VITBM2 VR0 || VMOV32 VR2, mem32 — Code Rate 1:2 Branch Metric Calculation with Parallel Load	784
VITBM3 VR0, VR1, VR2 —Code Rate 1:3 Branch Metric Calculation	785
VITBM3 VR0, VR1, VR2 || VMOV32 VR2, mem32 —Code Rate 1:3 Branch Metric Calculation with Parallel Load	786
VITDHADDSUB VR4, VR3, VR2, VRa —Viterbi Double Add and Subtract, High	787
VITDHADDSUB VR4, VR3, VR2, VRa || mem32 VRb —Viterbi Add and Subtract High with Parallel Store	789
VITDHSUBADD VR4, VR3, VR2, VRa —Viterbi Add and Subtract Low	790
VITDHSUBADD VR4, VR3, VR2, VRa || mem32 VRb —Viterbi Subtract and Add, High with Parallel Store	791
VITDLADDSUB VR4, VR3, VR2, VRa —Viterbi Add and Subtract Low	792
VITDLADDSUB VR4, VR3, VR2, VRa || mem32 VRb —Viterbi Add and Subtract Low with Parallel Load	793
VITDLSUBADD VR4, VR3, VR2, VRa —Viterbi Subtract and Add Low	794
VITDLSUBADD VR4, VR3, VR2, VRa || mem32 VRb —Viterbi Subtract and Add, Low with Parallel Store	795
VITHSEL VRa, VRb, VR4, VR3 —Viterbi Select High	796
VITHSEL VRa, VRb, VR4, VR3 || VMOV32 VR2, mem32 —Viterbi Select High with Parallel Load	797
VITLSEL VRa, VRb, VR4, VR3 —Viterbi Select, Low Word	798
VITLSEL VRa, VRb, VR4, VR3 || VMOV32 VR2, mem32 —Viterbi Select Low with Parallel Load	799
VTCLEAR —Clear Transition Bit Registers	800
VTRACE mem32, VR0, VT0, VT1 —Viterbi Traceback, Store to Memory	801
VTRACE VR1, VR0, VT0, VT1 —Viterbi Traceback, Store to Register	803

VITBM2 VR0 *Code Rate 1:2 Branch Metric Calculation*

Operands

Before the operation, the inputs are loaded into the registers as shown below. Each operand for the branch metric calculation is 16-bits.

Input Register	Value
VR0L	16-bit decoder input 0
VR0H	16-bit decoder input 1

The result of the operation is also stored in VR0 as shown below:

Output Register	Value
VR0L	16-bit branch metric 0 = VR0L + VR0H
VR0H	16-bit branch metric 1 = VR0L - VR0L

Opcode

```
LSW: 1110 0101 0000 1100
```

Description

Branch metric calculation for code rate = 1/2.

```
// SAT  is VSTATUS[SAT]
// VR0L is decoder input 0
// VR0H is decoder input 1
//
// Calculate the branch metrics by performing 16-bit signed
// addition and subtraction
//
   VR0L = VR0L + VR0H;     // VR0L = branch metric 0
   VR0H = VR0L - VR0L;     // VR0H = branch metric 1
   if (SAT == 1)
   {
      sat16(VR0L);
      sat16(VR0H);
   }
```

Flags

This instruction sets the real overflow flag, VSTATUS[OVFR] in the event of an overflow or underflow.

Pipeline

This is a single-cycle instruction.

Example

See also

VITBM2 VR0 || VMOV32 VR2, mem32
VITBM3 VR0, VR1, VR2

VITBM2 VR0 || VMOV32 VR2, mem32 *Code Rate 1:2 Branch Metric Calculation with Parallel Load*

Operands

Before the operation, the inputs are loaded into the registers as shown below. Each operand for the branch metric calculation is 16-bits.

Input Register	Value
VR0L	16-bit decoder input 0
VR0H	16-bit decoder input 1
[mem32]	pointer to 32-bit memory location.

The result of the operation is stored in VR0 as shown below:

Output Register	Value
VR0L	16-bit branch metric 0 = VR0L + VR0H
VR0H	16-bit branch metric 1 = VR0L - VR0L
VR2	contents of memory pointed to by [mem32]

Opcode

```
LSW: 1110 0011 1111 1100
MSW: 0000 aaaa   mem32
```

Description

Branch metric calculation for a code rate of 1/2 with parallel register load.

```
// SAT  is VSTATUS[SAT]
// VR0L is decoder input 0
// VR0H is decoder input 1
//
// Calculate the branch metrics by performing 16-bit signed
// addition and subtraction
//
   VR0L = VR0L + VR0H;     // VR0L = branch metric 0
   VR0H = VR0L - VR0L;     // VR0H = branch metric 1
   if (SAT == 1)
   {
      sat16(VR0L);
      sat16(VR0H);
   }
   VR2 = [mem32]           // Load VR2L and VR2H with the next state metrics
```

Flags

This instruction sets the real overflow flag, VSTATUS[OVFR] in the event of an overflow or underflow.

Pipeline

Both operations complete in a single cycle.

Example

See also

VITBM2 VR0
VITBM3 VR0, VR1, VR2
VITBM3 VR0, VR1, VR2 || VMOV32 VR2, mem32

VITBM3 VR0, VR1, VR2 *Code Rate 1:3 Branch Metric Calculation*

Operands

Before the operation, the inputs are loaded into the registers as shown below. Each operand for the branch metric calculation is 16-bits.

Input Register	Value
VR0L	16-bit decoder input 0
VR1L	16-bit decoder input 1
VR2L	16-bit decoder input 2

The result of the operation is stored in VR0 and VR1 as shown below:

Output Register	Value
VR0L	16-bit branch metric 0 = VR0L + VR1L + VR2L
VR0H	16-bit branch metric 1 = VR0L + VR1L - VR2L
VR1L	16-bit branch metric 2 = VR0L - VR1L + VR2L
VR1H	16-bit branch metric 3 = VR0L - VR1L - VR2L

Opcode

```
LSW: 1110 0101 0000    1101
```

Description

Calculate the four branch metrics for a code rate of 1/3.

```
// SAT  is VSTATUS[SAT]
// VR0L is decoder input 0
// VR1L is decoder input 1
// VR2L is decoder input 2
//
// Calculate the branch metrics by performing 16-bit signed
// addition and subtraction
//
   VR0L = VR0L + VR1L + VR2L;   // VR0L = branch Metric 0
   VR0H = VR0L + VR1L - VR2L;   // VR0H = branch Metric 1
   VR1L = VR0L - VR1L + VR2L;   // VR1L = branch Metric 2
   VR1H = VR0L - VR1L - VR2L;   // VR1H = branch Metric 3
   if(SAT == 1)
   {
      sat16(VR0L);
      sat16(VR0H);
      sat16(VR1L);
      sat16(VR1H);
   }
```

Flags

This instruction sets the real overflow flag, VSTATUS[OVFR] in the event of an overflow or underflow.

Pipeline

This is a 2p-cycle instruction. The instruction following VITBM3 must not use VR0 or VR1.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITBM2 VR0
VITBM2 VR0 || VMOV32 VR2, mem32

VITBM3 VR0, VR1, VR2 || VMOV32 VR2, mem32 *Code Rate 1:3 Branch Metric Calculation with Parallel Load*

Operands

Before the operation, the inputs are loaded into the registers as shown below. Each operand for the branch metric calculation is 16-bits.

Input Register	Value
VR0L	16-bit decoder input 0
VR1L	16-bit decoder input 1
[mem32]	pointer to a 32-bit memory location

The result of the operation is stored in VR0 and VR1 and VR2 as shown below:

Output Register	Value
VR0L	16-bit branch metric 0 = VR0L + VR1L + VR2L
VR0H	16-bit branch metric 1 = VR0L + VR1L - VR2L
VR1L	16-bit branch metric 2 = VR0L - VR1L + VR2
VR1H	16-bit branch metric 3 = VR0L - VR1L - VR2L
VR2	Contents of the memory pointed to by [mem32]

Opcode

```
LSW: 1110 0011 1111    1101
MSW: 0000 aaaa   mem32
```

Description

Calculate the four branch metrics for a code rate of 1/3 with parallel register load.

```
// SAT  is VSTATUS[SAT]
// VR0L is decoder input 0
// VR1L is decoder input 1
// VR2L is decoder input 2
//
// Calculate the branch metrics by performing 16-bit signed
// addition and subtraction
//
   VR0L = VR0L + VR1L + VR2L;   // VR0L = branch Metric 0
   VR0H = VR0L + VR1L - VR2L;   // VR0H = branch Metric 1
   VR1L = VR0L - VR1L + VR2L;   // VR1L = branch Metric 2
   VR1H = VR0L - VR1L - VR2L;   // VR1H = branch Metric 3
   if(SAT == 1)
   {
      sat16(VR0L);
      sat16(VR0H);
      sat16(VR1L);
      sat16(VR1H);
   }
   VR2 = [mem32];
```

Flags

This instruction sets the real overflow flag, VSTATUS[OVFR] in the event of an overflow or underflow.

Pipeline

This is a 2p/1-cycle instruction. The VBITM3 operation takes 2p cycles and the VMOV32 completes in a single cycle. The next instruction must not use VR0 or VR1.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITBM2 VR0
VITBM2 VR0 || VMOV32 VR2, mem32

VITDHADDSUB VR4, VR3, VR2, VRa *Viterbi Double Add and Subtract, High*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaH.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaH	Branch metric 1. VRa must be VR0 or VR1.

The result of the operation is stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0 = VR2L + VRaH
VR3H	16-bit path metric 1 = VR2H - VRaH
VR4L	16-bit path metric 2 = VR2L - VRaH
VR4H	16-bit path metric 3 = VR2H +VRaH

Opcode

```
LSW: 1110 0101 0111    aaaa
```

Description

Viterbi high add and subtract. This instruction is used to calculate four path metrics.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaH with the branch metric.
//
            VR3L = VR2L + VRaH        // Path metric 0
            VR3H = VR2H - VRaH     // Path metric 1
            VR4L = VR2L - VRaH     // Path metric 2
            VR4H = VR2H + VRaH     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
; Example Viterbi decoder code fragment
; Viterbi butterfly calculations
; Loop once for each decoder input pair
;
; Branch metrics = BM0 and BM1
; XAR5 points to the input stream to the decoder
 ...
 ...
_loop:
   VMOV32 VR0, *XAR5++            ; Load two inputs into VR0L, VR0H
   VITBM2 VR0                     ; VR0L = BM0    VR0H = BM1
|| VMOV32 VR2, *XAR1++            ; Load previous state metrics

;
; 2 cycle Viterbi butterfly
;
   VITDLADDSUB VR4,VR3,VR2,VR0 ; Perform add/sub
   VITLSEL VR6,VR5,VR4,VR3     ; Perform compare/select
|| VMOV32  VR2, *XAR1++        ; Load previous state metrics

;
; 2 cycle Viterbi butterfly, next stage
;
   VITDHADDSUB VR4,VR3,VR2,VR0
   VITHSEL VR6,VR5,VR4,VR3
|| VMOV32 VR2, *XAR1++

;
; 2 cycle Viterbi butterfly, next stage
;
   VITDLADDSUB VR4,VR3,VR2,VR0
|| VMOV32 *XAR2++, VR5
   ...
   ...
```

See also

VITDHSUBADD VR4, VR3, VR2, VRa
VITDLADDSUB VR4, VR3, VR2, VRa
VITDLSUBADD VR4, VR3, VR2, VRa

VITDHADDSUB VR4, VR3, VR2, VRa || mem32 VRb *Viterbi Add and Subtract High with Parallel Store*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaH.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaH	Branch metric 1. VRa must be VR0 or VR1.
VRb	Value to be stored. VRb can be VR5, VR6, VR7 or VR8.

The result of the operation is stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0 = VR2L + VRaH
VR3H	16-bit path metric 1 = VR2H - VRaH
VR4L	16-bit path metric 2 = VR2L - VRaH
VR4H	16-bit path metric 3 = VR2H +VRaH
[mem32]	Contents of VRb. VRb can be VR5, VR6, VR7 or VR8.

Opcode

```
LSW: 1110 0101 0000    1001
MSW: bbbb aaaa    mem32
```

Description

Viterbi high add and subtract. This instruction is used to calculate four path metrics.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaH with the branch metric.
//
            VR3L = VR2L + VRaH        // Path metric 0
            VR3H = VR2H - VRaH     // Path metric 1
            VR4L = VR2L - VRaH     // Path metric 2
            VR4H = VR2H + VRaH     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VITDHSUBADD VR4, VR3, VR2, VRa
VITDLADDSUB VR4, VR3, VR2, VRa
VITDLSUBADD VR4, VR3, VR2, VRa

VITDHSUBADD VR4, VR3, VR2, VRa *Viterbi Add and Subtract Low*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaL.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaL	Branch metric 0. VRa must be VR0 or VR1.

The result of the operation is 4 path metrics stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0 = VR2L - VRaH
VR3H	16-bit path metric 1 = VR2H + VRaH
VR4L	16-bit path metric 2 = VR2L + VRaH
VR4H	16-bit path metric 3 = VR2H - VRaL

Opcode

```
LSW: 1110 0101 1111    aaaa
```

Description

This instruction is used to calculate four path metrics in the Viterbi butterfly. This operation uses the branch metric stored in VRaL.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaL with the branch metric.
//
            VR3L = VR2L - VRaL        // Path metric 0
            VR3H = VR2H + VRaL     // Path metric 1
            VR4L = VR2L + VRaL     // Path metric 2
            VR4H = VR2H - VRaL     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITDHADDSUB VR4, VR3, VR2, VRa
VITDHSUBADD VR4, VR3, VR2, VRa
VITDLSUBADD VR4, VR3, VR2, VRa

VITDHSUBADD VR4, VR3, VR2, VRa || mem32 VRb *Viterbi Subtract and Add, High with Parallel Store*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaH.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaH	Branch metric 1. VRa must be VR0 or VR1.
VRb	Contents to be stored. VRb can be VR5, VR6, VR7 or VR8.

The result of the operation is stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0 = VR2L -VRaH
VR3H	16-bit path metric 1 = VR2H + VRaH
VR4L	16-bit path metric 2 = VR2L + VRaH
VR4H	16-bit path metric 3 = VR2H - VRaH
[mem32]	Contents of VRb. VRb can be VR5, VR6, VR7 or VR8.

Opcode

```
LSW: 1110 0010 0000    0101
MSW: bbbb aaaa    mem32
```

Description

Viterbi high subtract and add. This instruction is used to calculate four path metrics.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaH with the branch metric.
//
            [mem32] = VRb          // Store VRb to memory
   VR3L = VR2L - VRaH        // Path metric 0
            VR3H = VR2H + VRaH     // Path metric 1
            VR4L = VR2L + VRaH     // Path metric 2
            VR4H = VR2H - VRaH     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VITDHADDSUB VR4, VR3, VR2, VRa
VITDLADDSUB VR4, VR3, VR2, VRa
VITDLSUBADD VR4, VR3, VR2, VRa

VITDLADDSUB VR4, VR3, VR2, VRa *Viterbi Add and Subtract Low*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaL.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaL	Branch metric 0. VRa must be VR0 or VR1.

The result of the operation is 4 path metrics stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0 = VR2L + VRaH
VR3H	16-bit path metric 1 = VR2H - VRaH
VR4L	16-bit path metric 2 = VR2L - VRaH
VR4H	16-bit path metric 3 = VR2H + VRaL

Opcode

```
LSW: 1110 0101 0011    aaaa
```

Description

This instruction is used to calculate four path metrics in the Viterbi butterfly. This operation uses the branch metric stored in VRaL.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaL with the branch metric.
//
            VR3L = VR2L + VRaL        // Path metric 0
            VR3H = VR2H - VRaL     // Path metric 1
            VR4L = VR2L - VRaL     // Path metric 2
            VR4H = VR2H + VRaL     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITDHADDSUB VR4, VR3, VR2, VRa
VITDHSUBADD VR4, VR3, VR2, VRa
VITDLSUBADD VR4, VR3, VR2, VRa

VITDLADDSUB VR4, VR3, VR2, VRa || mem32 VRb *Viterbi Add and Subtract Low with Parallel Load*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaL.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaL	Branch metric 0. VRa can be VR0 or VR1.
VRb	Contents to be stored to memory

The result of the operation is 4 path metrics stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0 = VR2L + VRaH
VR3H	16-bit path metric 1 = VR2H - VRaH
VR4L	16-bit path metric 2 = VR2L - VRaH
VR4H	16-bit path metric 3 = VR2H + VRaL
[mem32]	Contents of VRb. VRb can be VR5, VR6, VR7 or VR8.

Opcode

```
LSW: 1110 0010 0000    1000
MSW: bbbb aaaa    mem32
```

Description

This instruction is used to calculate four path metrics in the Viterbi butterfly. This operation uses the branch metric stored in VRaL.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaL with the branch metric.
//
   [mem32] = VRb           // Store VRb
   VR3L = VR2L + VRaL         // Path metric 0
            VR3H = VR2H - VRaL     // Path metric 1
            VR4L = VR2L - VRaL     // Path metric 2
            VR4H = VR2H + VRaL     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITDHADDSUB VR4, VR3, VR2, VRa
VITDHSUBADD VR4, VR3, VR2, VRa
VITDLSUBADD VR4, VR3, VR2, VRa

VITDLSUBADD VR4, VR3, VR2, VRa *Viterbi Subtract and Add Low*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaL.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaL	Branch metric 0. VRa must be VR0 or VR1.

The result of the operation is 4 path metrics stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0= VR2L - VRaH
VR3H	16-bit path metric 1 = VR2H + VRaH
VR4L	16-bit path metric 2 = VR2L + VRaH
VR4H	16-bit path metric 3 = VR2H - VRaL

Opcode

```
LSW: 1110 0101 1110    aaaa
```

Description

This instruction is used to calculate four path metrics in the Viterbi butterfly. This operation uses the branch metric stored in VRaL.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaH with the branch metric.
//
            VR3L = VR2L - VRaL        // Path metric 0
            VR3H = VR2H + VRaL     // Path metric 1
            VR4L = VR2L + VRaL     // Path metric 2
            VR4H = VR2H - VRaL     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITDHADDSUB VR4, VR3, VR2, VRa
VITDHSUBADD VR4, VR3, VR2, VRa
VITDLADDSUB VR4, VR3, VR2, VRa

VITDLSUBADD VR4, VR3, VR2, VRa || mem32 VRb *Viterbi Subtract and Add, Low with Parallel Store*

Operands

Before the operation, the inputs are loaded into the registers as shown below. This operation uses the branch metric stored in VRaL.

Input Register	Value
VR2L	16-bit state metric 0
VR2H	16-bit state metric 1
VRaL	Branch metric 0. VRa must be VR0 or VR1.
VRb	Value to be stored. VRb can be VR5, VR6, VR7 or VR8.

The result of the operation is 4 path metrics stored in VR3 and VR4 as shown below:

Output Register	Value
VR3L	16-bit path metric 0= VR2L - VRaH
VR3H	16-bit path metric 1 = VR2H + VRaH
VR4L	16-bit path metric 2 = VR2L + VRaH
VR4H	16-bit path metric 3 = VR2H - VRaL
[mem32]	Contents of VRb. VRb can be VR5, VR6, VR7 or VR8.

Opcode

```
LSW: 1110 0010 0000    1010
MSW: bbbb aaaa   mem32
```

Description

This instruction is used to calculate four path metrics in the Viterbi butterfly. This operation uses the branch metric stored in VRaL.

```
//
// Calculate the four path metrics by performing 16-bit signed
// addition and subtraction
//
// Before this operation VR2L and VR2H are loaded with the state
// metrics and VRaH with the branch metric.
//
            [mem32] = VRb          // Store VRb into mem32
   VR3L = VR2L - VRaL        // Path metric 0
            VR3H = VR2H + VRaL     // Path metric 1
            VR4L = VR2L + VRaL     // Path metric 2
            VR4H = VR2H - VRaL     // Path metric 3
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITDHADDSUB VR4, VR3, VR2, VRa
VITDHSUBADD VR4, VR3, VR2, VRa
VITDLADDSUB VR4, VR3, VR2, VRa

VITHSEL VRa, VRb, VR4, VR3 *Viterbi Select High*

Operands

Before the operation, the path metrics are loaded into the registers as shown below. Typically this will have been done using a Viterbi AddSub or SubAdd instruction.

Input Register	Value
VR3L	16-bit path metric 0
VR3H	16-bit path metric 1
VR4L	16-bit path metric 2
VR4H	16-bit path metric 3

The result of the operation is the new state metrics stored in VRa and VRb as shown below:

Output Register	Value
VRaH	16-bit state metric 0. VRa can be VR6 or VR8.
VRbH	16-bit state metric 1. VRb can be VR5 or VR7.
VT0	The transition bit is appended to the end of the register.
VT1	The transition bit is appended to the end of the register.

Opcode

```
LSW: 1110 0110 1111    0111
MSW: 0000 0000 bbbb aaaa
```

Description

This instruction computes the new state metrics of a Viterbi butterfly operation and stores them in the higher 16-bits of the VRa and VRb registers. To instead load the state metrics into the low 16-bits use the VITLSEL instruction.

```
T0 = T0 << 1     // Shift previous transition bits left
if (VR3L > VR3H)
{
   VRbH = VR3L;  // New state metric 0
   T0[0:0] = 0;  // Store the transition bit
}
else
{
   VRbH = VR3H;  // New state metric 0
   T0[0:0] = 1;  // Store the transition bit
}

T1 = T1 << 1     // Shift previous transition bits left
if (VR4L > VR4H)
{
   VRaH = VR4L;  // New state metric 1
   T1[0:0] = 0;  // Store the transition bit
}
else
{
   VRaH = VR4H;  // New state metric 1
   T1[0:0] = 1;  // Store the transition bit
}
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITLSEL VRa, VRb, VR4, VR3

VITHSEL VRa, VRb, VR4, VR3 || VMOV32 VR2, mem32 *Viterbi Select High with Parallel Load*

Operands

Before the operation, the path metrics are loaded into the registers as shown below. Typically this will have been done using a Viterbi AddSub or SubAdd instruction.

Input Register	Value
VR3L	16-bit path metric 0
VR3H	16-bit path metric 1
VR4L	16-bit path metric 2
VR4H	16-bit path metric 3
[mem32]	pointer to 32-bit memory location.

The result of the operation is the new state metrics stored in VRa and VRb as shown below:

Output Register	Value
VRaH	16-bit state metric 0. VRa can be VR6 or VR8.
VRbH	16-bit state metric 1. VRb can be VR5 or VR7.
VT0	The transition bit is appended to the end of the register.
VT1	The transition bit is appended to the end of the register.
VR2	Contents of the memory pointed to by [mem32].

Opcode

```
LSW: 1110 0011 1111    1111
MSW: bbbb aaaa   mem32
```

Description

This instruction computes the new state metrics of a Viterbi butterfly operation and stores them in the higher 16-bits of the VRa and VRb registers. To instead load the state metrics into the low 16-bits use the VITLSEL instruction.

```
T0 = T0 << 1     // Shift previous transition bits left
if (VR3L > VR3H)
{
   VRbH = VR3L;  // New state metric 0
   T0[0:0] = 0;  // Store the transition bit
}
else
{
   VRbH = VR3H;  // New state metric 0
   T0[0:0] = 1;  // Store the transition bit
}

T1 = T1 << 1     // Shift previous transition bits left
if (VR4L > VR4H)
{
   VRaH = VR4L;  // New state metric 1
   T1[0:0] = 0;  // Store the transition bit
}
else
{
   VRaH = VR4H;  // New state metric 1
   T1[0:0] = 1;  // Store the transition bit
}
VR2 = [mem32];   // Load VR2
```

Flags This instruction does not modify any flags in the VSTATUS register.

Pipeline This is a single-cycle instruction.

Example Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also VITLSEL VRa, VRb, VR4, VR3

VITLSEL VRa, VRb, VR4, VR3 *Viterbi Select, Low Word*

Operands

Before the operation, the path metrics are loaded into the registers as shown below. Typically this will have been done using a Viterbi AddSub or SubAdd instruction.

Input Register	Value
VR3L	16-bit path metric 0
VR3H	16-bit path metric 1
VR4L	16-bit path metric 2
VR4H	16-bit path metric 3

The result of the operation is the new state metrics stored in VRa and VRb as shown below:

Output Register	Value
VRaL	16-bit state metric 0. VRa can be VR6 or VR8.
VRbL	16-bit state metric 1. VRb can be VR5 or VR7.
VT0	The transition bit is appended to the end of the register.
VT1	The transition bit is appended to the end of the register.

Opcode

```
LSW: 1110 0110 1111    0110
MSW: 0000 0000 bbbb aaaa
```

Description

This instruction computes the new state metrics of a Viterbi butterfly operation and stores them in the higher 16-bits of the VRa and VRb registers. To instead load the state metrics into the low 16-bits use the VITHSEL instruction.

```
T0 = T0 << 1     // Shift previous transition bits left
if (VR3L > VR3H)
{
   VRbL = VR3L;  // New state metric 0
   T0[0:0] = 0;  // Store the transition bit
}
else
{
   VRbL = VR3H;  // New state metric 0
   T0[0:0] = 1;  // Store the transition bit
}

T1 = T1 << 1     // Shift previous transition bits left
if (VR4L > VR4H)
{
   VRaL = VR4L;  // New state metric 1
   T1[0:0] = 0;  // Store the transition bit
}
else
{
   VRaL = VR4H;  // New state metric 1
   T1[0:0] = 1;  // Store the transition bit
}
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also

VITHSEL VRa, VRb, VR4, VR3

VITLSEL VRa, VRb, VR4, VR3 || VMOV32 VR2, mem32 *Viterbi Select Low with Parallel Load*

Operands

Before the operation, the path metrics are loaded into the registers as shown below. Typically this will have been done using a Viterbi AddSub or SubAdd instruction.

Input Register	Value
VR3L	16-bit path metric 0
VR3H	16-bit path metric 1
VR4L	16-bit path metric 2
VR4H	16-bit path metric 3
mem32	Pointer to 32-bit memory location.

The result of the operation is the new state metrics stored in VRa and VRb as shown below:

Output Register	Value
VRaL	16-bit state metric 0. VRa can be VR6 or VR8.
VRbL	16-bit state metric 1. VRb can be VR5 or VR7.
VT0	The transition bit is appended to the end of the register.
VT1	The transition bit is appended to the end of the register.
VR2	Contents of 32-bit memory pointed to by mem32.

Opcode

```
LSW: 1110 0011 1111    1110
MSW: bbbb aaaa    mem32
```

Description

This instruction computes the new state metrics of a Viterbi butterfly operation and stores them in the higher 16-bits of the VRa and VRb registers. To instead load the state metrics into the low 16-bits use the VITHSEL instruction. In parallel the VR2 register is loaded with the contents of memory pointed to by [mem32].

```
T0 = T0 << 1     // Shift previous transition bits left
if (VR3L > VR3H)
{
   VRbL = VR3L;  // New state metric 0
   T0[0:0] = 0;  // Store the transition bit
}
else
{
   VRbL = VR3H;  // New state metric 0
   T0[0:0] = 1;  // Store the transition bit
}

T1 = T1 << 1     // Shift previous transition bits left
if (VR4L > VR4H)
{
   VRaL = VR4L;  // New state metric 1
   T1[0:0] = 0;  // Store the transition bit
}
else
{
   VRaL = VR4H;  // New state metric 1
   T1[0:0] = 1;  // Store the transition bit
}
VR2 = [mem32]
```

Flags This instruction does not modify any flags in the VSTATUS register.

Pipeline This is a single-cycle instruction.

Example Refer to the example for VITDHADDSUB VR4, VR3, VR2, VRa.

See also VITHSEL VRa, VRb, VR4, VR3

VTCLEAR *Clear Transition Bit Registers*

Operands

none

Opcode

```
LSW: 1110 0101 0010    1001
```

Description

Clear the VT0 and VT1 registers.

```
VT0 = 0;
VT1 = 0;
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VCLEARALL
VCLEAR VRa

VTRACE mem32, VR0, VT0, VT1 *Viterbi Traceback, Store to Memory*

Operands

Before the operation, the path metrics are loaded into the registers as shown below using a Viterbi AddSub or SubAdd instruction.

Input Register	Value
VT0	transition bit register 0
VT1	transiton bit register 1
VR0	Initial value is zero. After the first VTRACE, this contains infromation from the previous trace-back.

The result of the operation is the new state metrics stored in VRa and VRb as shown below:

Output Register	Value
[mem32]	Traceback result from the transiton bits.

Opcode

```
LSW: 1110 0010 0000    1100
MSW: 0000 0000   mem32
```

Description

Trace-back from the transition bits stored in VT0 and VT1 registers. Write the result to memory. The transition bits in the VT0 and VT1 registers are stored in the following format by the VITLSEL and VITHSEL instructions:

VT0[31]	Transition bit [State 0]
VT0[30]	Transition bit [State 1]
VT0[29]	Transition bit [State 2]
...	...
VT0[0]	Transition bit [State 31]
VT1[31]	Transition bit [State 32]
VT1[30]	Transition bit [State 33]
VT1[29]	Transition bit [State 34]
...	...
VT1[0]	Transition bit [State 63]

```
//
// Calculate the decoder output bit by performing a
// traceback from the transition bits stored in the VT0 and VT1 registers
//
   S = VR0[5:0];
   VR0[31:6] = 0;
   if (S < 32)
   {
      temp[0] = VT0[31-S];
   }
   else
   {
      temp[0] = VT1[63-S];
   }
   *[mem32][0] = temp;
   *[mem32][31:1] = 0;
   VR0[5:0] = 2*VR0[5:0] + temp[0];
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

```
//
// Example traceback code fragment
```

```
//
// XAR5 points to the beginning of Decoder Output array
//
   VCLEAR VR0
   MOVL XAR5,*+XAR4[0]

//
// To retrieve each original message:
// Load VT0/VT1 with the stored transition values
// and use VTRACE instruction
//

   VMOV32 VT0, *--XAR3
   VMOV32 VT1, *--XAR3
   VTRACE *XAR5++, VR0, VT0, VT1

   VMOV32 VT0, *--XAR3
   VMOV32 VT1, *--XAR3
   VTRACE *XAR5++, VR0, VT0, VT1
   ...
   ...etc for each VT0/VT1 pair
```

See also VTRACE VR1, VR0, VT0, VT1

VTRACE VR1, VR0, VT0, VT1 *Viterbi Traceback, Store to Register*

Operands

Before the operation, the path metrics are loaded into the registers as shown below using a Viterbi AddSub or SubAdd instruction.

Input Register	Value
VT0	transition bit register 0
VT1	transiton bit register 1
VR0	Initial value is zero. After the first VTRACE, this contains infromation from the previous trace-back.

The result of the operation is the output of the decoder stored in VR1:

Output Register	Value
VR1	Traceback result from the transiton bits.

Opcode

```
LSW: 1110 0101 0010    1000
```

Description

Trace-back from the transition bits stored in VT0 and VT1 registers. Write the result to VR1. The transition bits in the VT0 and VT1 registers are stored in the following format by the VITLSEL and VITHSEL instructions:

VT0[31]	Transition bit [State 0]
VT0[30]	Transition bit [State 1]
VT0[29]	Transition bit [State 2]
...	...
VT0[0]	Transition bit [State 31]
VT1[31]	Transition bit [State 32]
VT1[30]	Transition bit [State 33]
VT1[29]	Transition bit [State 34]
...	...
VT1[0]	Transition bit [State 63]

```
//
// Calculate the decoder output bit by performing a
// traceback from the transition bits stored in the VT0 and VT1 registers
//
   S = VR0[5:0];
   VR0[31:6] = 0;
   if (S < 32)
   {
      temp[0] = VT0[31-S];
   }
   else
   {
      temp[0] = VT1[63-S];
   }
   VR1[0] = temp;
   VR1[31:1] = 0;
   VR0[5:0] = 2*VR0[5:0] + temp[0];
```

Flags

This instruction does not modify any flags in the VSTATUS register.

Pipeline

This is a single-cycle instruction.

Example

See also

VTRACE mem32, VR0, VT0, VT1

10.6 Rounding Mode

This section details the rounding operation as applied to a right shift. When the rounding mode is enabled in the VSTATUS register, .5 will be added to the right shifted intermediate value before truncation. If rounding is disabled the right shifted value is only truncated. Table 10-14shows the bit representation of two values, 11.0 and 13.0. The columns marked Bit-1, Bit-2 and Bit-3 hold temporary bits resulting from the right shift operation.

Table 10-14. Example: Values Before Shift Right

	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit-1	Bit-2	Bit -3	Value
Val A	0	0	1	0	1	1	0	0	0	11.000
Val B	0	0	0	0	0	1	0	0	0	13.000

Table 10-14Shows the intermediate values after the right shift has been applied to Val B. The columns marked Bit-1, Bit-2 and Bit-3 hold temporary bits resulting from the right shift operation.

Table 10-15. Example: Values after Shift Right

	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit-1	Bit-2	Bit -3	Value
Val A	0	0	1	0	1	1	0	0	0	11.000
Val B >> 3	0	0	0	0	0	1	1	0	1	1.625

When the rounding mode is enabled, .5 will be added to the intermediate result before truncation. Table 10-16 shows the bit representation of Val A + Val (B >> 3) operation with rounding. Notice .5 is added to the intermediate shifted right value. After the addition, the bits in Bit-1, Bit-2 and Bit-3 are removed. In this case the result of the operation will be 13 which is the truncated value after rounding.

Table 10-16. Example: Addition with Right Shift and Rounding

	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit-1	Bit-2	Bit -3	Value
Val A	0	0	1	0	1	1	0	0	0	11.000
Val B >> 3	0	0	0	0	0	1	1	0	1	1.625
.5	0	0	0	0	0	0	1	0	0	0 .500
Val A + Val B >> 3 + .5	0	0	1	1	0	1	0	0	1	13.125

When the rounding mode is disabled, the value is simply truncated. Table 10-17 shows the bit representation of the operation Val A + (Val B >> 3) without rounding. After the addition, the bits in Bit-1, Bit-2 and Bit-3 are removed. In this case the result of the operation will be 12 which is the truncated value without rounding.

Table 10-17. Example: Addition with Rounding After Shift Right

	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit-1	Bit-2	Bit -3	Value
Val A	0	0	1	0	1	1	0	0	0	11.000
Val B >> 3	0	0	0	0	0	1	1	0	1	1.625
Val A + Val B >> 3	0	0	1	1	0	0	1	0	1	12.625

Table 10-18 shows more examples of the intermediate shifted value along with the result if rounding is enabled or disabled. In each case, the truncated value is without .5 added and the rounded value is with .5 added.

Table 10-18. Shift Right Operation With and Without Rounding

Bit2	Bit1	Bit0	Bit -1	Bit -2	Value	Result with RND = 0	Result with RND = 1
0	1	0	0	0	2.00	2	2
0	0	1	1	1	1.75	1	2
0	0	1	1	0	1.50	1	2

Table 10-18. Shift Right Operation With and Without Rounding (continued)

Bit2	Bit1	Bit0	Bit -1	Bit -2	Value	Result with RND = 0	Result with RND = 1
0	0	1	0	1	1.25	1	1
0	0	0	1	1	0.75	0	1
0	0	0	1	0	0.50	0	1
0	0	0	0	1	0.25	0	0
0	0	0	0	0	0.00	0	0
1	1	1	1	1	-0.25	0	0
1	1	1	1	0	-0.50	0	0
1	1	1	0	1	-0.75	0	-1
1	1	1	0	0	-1.00	-1	-1
1	1	0	1	1	-1.25	-1	-1
1	1	0	1	0	-1.50	-1	-1
1	1	0	0	1	-1.75	-1	-2
1	1	0	0	0	-2.00	-2	-2

Direct Memory Access (DMA) Module

The direct memory access (DMA) module provides a hardware method of transferring data between peripherals and/or memory without intervention from the CPU, thereby freeing up bandwidth for other system functions. Additionally, the DMA has the capability to orthogonally rearrange the data as it is transferred as well as "ping-pong" data between buffers. These features are useful for structuring data into blocks for optimal CPU processing.

Topic **Page**

11.1 Introduction

The strength of a controller is not measured purely in processor speed, but in total system capabilities. As a part of the equation, any time the CPU bandwidth for a given function can be reduced, the greater the system capabilities. Many times applications spend a significant amount of their bandwidth moving data, whether it is from off-chip memory to on-chip memory, or from a peripheral such as an analog-to-digital converter (ADC) to RAM, or even from one peripheral to another. Furthermore, many times this data comes in a format that is not conducive to the optimum processing powers of the CPU. The DMA module described in this reference guide has the ability to free up CPU bandwidth and rearrange the data into a pattern for more streamlined processing.

The DMA module is an event-based machine, meaning it requires a peripheral interrupt trigger to start a DMA transfer. Although it can be made into a periodic time-driven machine by configuring a timer as the interrupt trigger source, there is no mechanism within the module itself to start memory transfers periodically. The interrupt trigger source for each of the six DMA channels can be configured separately and each channel contains its own independent PIE interrupt to let the CPU know when a DMA transfers has either started or completed. Five of the six channels are exactly the same, while Channel 1 has one additional feature: the ability to be configured at a higher priority than the others. At the heart of the DMA is a state machine and tightly coupled address control logic. It is this address control logic that allows for rearrangement of the block of data during the transfer as well as the process of *ping-ponging* data between buffers. Each of these features, along with others will be discussed in detail in this document.

DMA Overview:

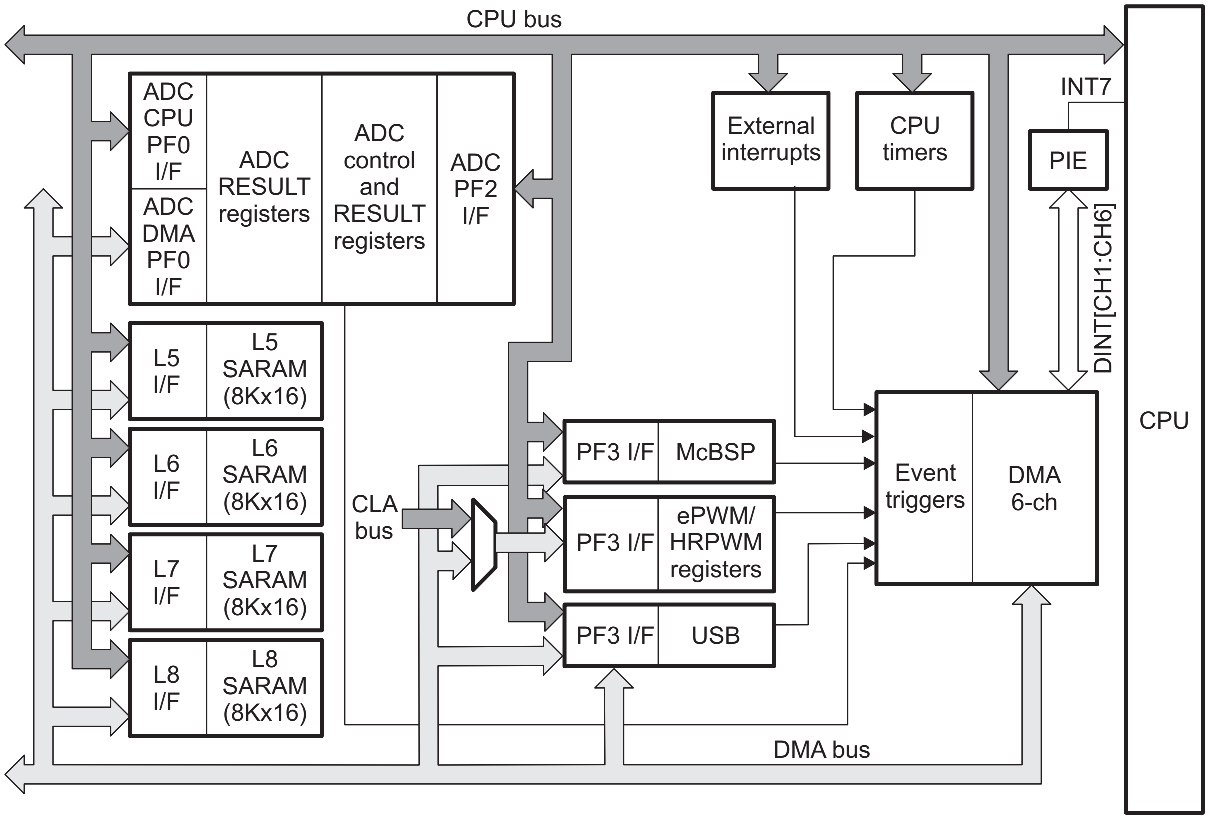
- 6 channels with independent PIE interrupts
- Peripheral interrupt trigger sources
  - ADC interrupts 1 and 2
  - Multichannel Buffered Serial Port transmit and receive
  - XINT1-3
  - CPU Timers
  - ePWM1-6 ADCSOCA and ADSOCB signals
  - USB endpoints 1-3 transmit and receive
  - Software
- Data sources/destinations:
  - L5-L8 32K x 16 SARAM
  - ADC memory bus mapped result registers
  - McBSP transmit and receive buffers
  - ePWM1-8 / HRPWM1-8
- Word Size: 16-bit or 32-bit (McBSP limited to 16-bit)
- Throughput: 4 cycles/word (5 cycles/word for McBSP reads)

11.2 Architecture

11.2.1 Block Diagram

Figure 11-1 shows a device level block diagram of the DMA.

Figure 11-1. DMA Block Diagram



11.2.2 Peripheral Interrupt Event Trigger Sources

The peripheral interrupt event trigger can be independently configured as one twenty-nine different sources for each of the six DMA channels. Included in these sources are three external interrupt signals which can be connected to most of the general-purpose input/output (GPIO) pins on the device. This adds significant flexibility to the event trigger capabilities. A bit field called PERINTSEL in the MODE register of each channel is used to select that channels interrupt trigger source. An active peripheral interrupt trigger will be latched into the PERINTFLG bit of the CONTROL register, and if the respective interrupt and DMA channel is enabled (see the MODE.CHx[PERINTE] and CONTROL.CHx[RUNSTS] bits), it will be serviced by the DMA channel. Upon receipt of a peripheral interrupt event signal, the DMA will automatically send a clear signal to the interrupt source so that subsequent interrupt events will occur.

Regardless of the value of the MODE.CHx[PERINTSEL] bit field, software can always force a trigger by using the CONTROL.CHx[PERINTFRC] bit. Likewise, software can always clear a pending DMA trigger using the CONTROL.CHx[PERINTCLR] bit.

Once a particular interrupt trigger sets a channel's PERINTFLG bit, the bit stays pending until the priority logic of the state machine starts the burst transfer for that channel. Once the burst transfer starts, the flag is cleared. If a new interrupt trigger is generated while a burst is in progress, the burst will complete before responding to the new interrupt trigger (after proper prioritization). If a third interrupt trigger occurs before the pending interrupt is serviced, an error flag is set in the CONTROL.CHx[OVRFLG] bit. If a peripheral interrupt trigger occurs at the same time as the latched flag is being cleared, the peripheral interrupt trigger has priority and the PERINTFLG will remain set.

Figure 11-2 shows a diagram of the trigger select circuit. See the MODE.CHx[PERINTSEL] bit field description for the complete list of peripheral interrupt trigger sources.

Figure 11-2. Peripheral Interrupt Trigger Input Diagram

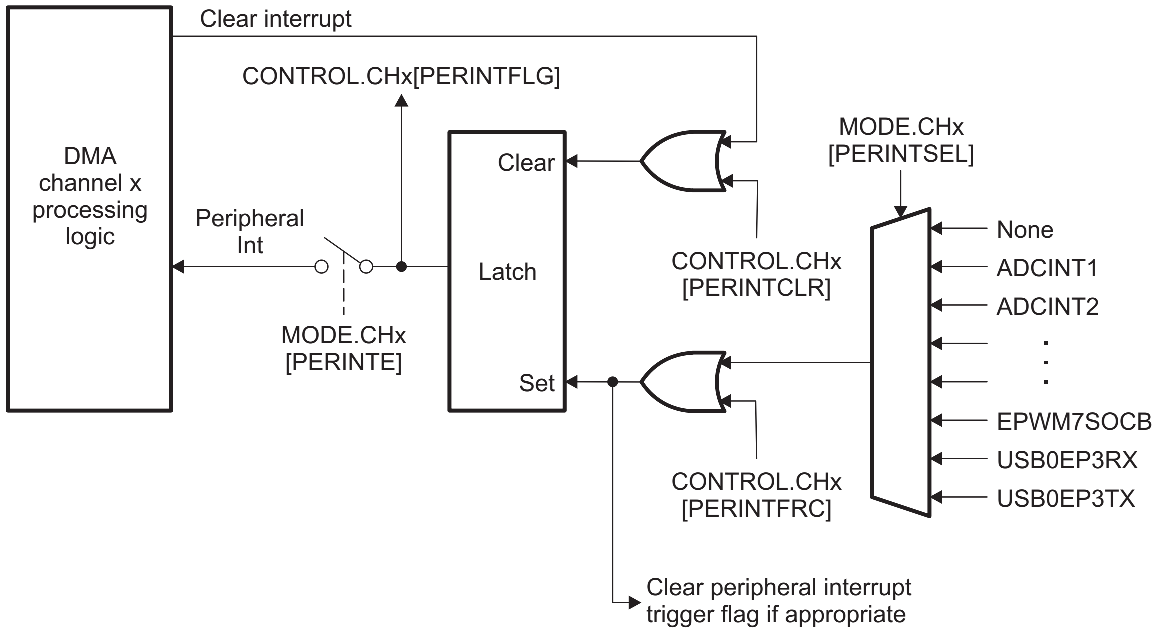


Table 11-1 shows the interrupt trigger source options that are available for each channel.

Table 11-1. Peripheral Interrupt Trigger Source Options

Peripheral	Interrupt Trigger Source
CPU	DMA Software bit (CHx.CONTROL.PERINTFRC) only
ADC	ADCINT 1
	ADCINT 2
External Interrupts	External Interrupt 1
	External Interrupt 2
	External Interrupt 3
CPU Timers	Timer 0 Overflow
	Timer 1 Overflow
	Timer 2 Overflow
McBSP	McBSP Transmit Buffer Empty
	McBSP Receive Buffer Full
ePWM2	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM3	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM4	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM5	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM6	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM7	ADC Start of Conversion A
	ADC Start of Conversion B
USB	USB Endpoint 1 Receive Full
USB	USB Endpoint 1 Transmit Empty
USB	USB Endpoint 2 Receive Full
USB	USB Endpoint 2 Transmit Empty
USB	USB Endpoint 3 Receive Full
USB	USB Endpoint 3 Transmit Empty

11.2.3 DMA Bus

The DMA bus architecture consists of a 22-bit address bus, a 32-bit data read bus, and a 32-bit data write bus. Memories and register locations connected to the DMA bus are via interfaces that sometimes share resources with the CPU memory or peripheral bus. Arbitration rules are defined in Section 11.4. The following resources are connected to the DMA bus:

- L5 SARAM
- L6 SARAM
- L7 SARAM
- L8 SARAM
- ADC Memory Mapped Result Registers
- McBSP Data Receive Registers (DRR2/DRR1) and Data Transmit Registers (DXR2/DXR1)
- ePWM1-8/HRPWM1-8 Registers
- USB Transmit and Receive Endpoints 1-3

11.3 Pipeline Timing and Throughput

The DMA consists of a 4-stage pipeline as shown in Figure 11-3. The one exception to this is when a DMA channel is configured to have the McBSP as its data source. A read of a McBSP DRR register stalls the DMA bus for one cycle during the read portion of the transfer, as shown in Figure 11-4.

Figure 11-3. 4-Stage Pipeline DMA Transfer

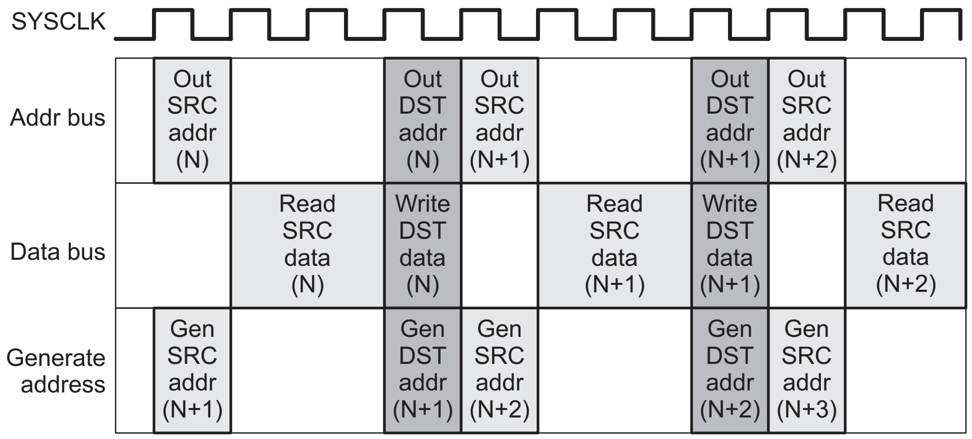
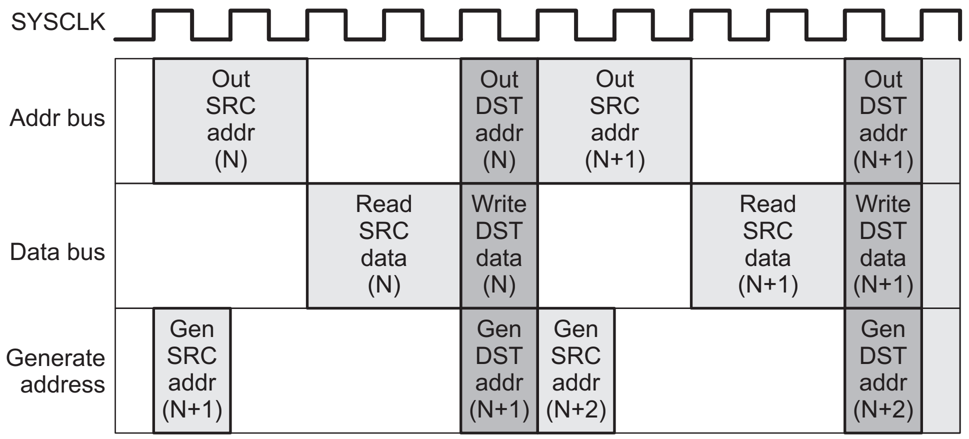


Figure 11-4. 4-Stage Pipeline With One Read Stall (McBSP as source)



In addition to the pipeline there are a few other behaviors of the DMA that affect it's total throughput

- A 1-cycle delay is added at the beginning of each burst
- A 1-cycle delay is added when returning from a CH1 high priority interrupt
- 32-bit transfers run at double the speed of a 16-bit transfer (i.e., it takes the same amount of time to transfer a 32-bit word as it does a 16-bit word)
- Collisions with the CPU may add delay slots (see Section 11.4)

For example, to transfer 128 16-bit words from ADC to RAM a channel can be configured to transfer 8 bursts of 16 words/burst. This will give:

8 bursts * [(4 cycles/word * 16 words/burst) + 1] = 520 cycles

If instead the channel were configured to transfer the same amount of data 32 bits at a time (the word size is configured to 32 bits) the transfer would take:

8 bursts * [(4 cycles/word * 8 words/burst) + 1] = 264 cycles

11.4 CPU Arbitration

Typically, DMA activity is independent of the CPU activity. Under the circumstance where both the DMA and the CPU are attempting to access memory or a peripheral register within the same interface concurrently, an arbitration procedure will occur. The one exception is with the memory mapped (PF0) ADC registers, which do not create a conflict when read by both the CPU and the DMA simultaneously, even if different addresses are accessed. Any combined accesses between the different interfaces, or where the CPU access is outside of the interface that the DMA is accessing do not create a conflict.

The interfaces which internally contain conflicts are:

- L5 RAM
- L6 RAM
- L7 RAM
- L8 RAM
- McBSP Peripheral Frame 3
- ePWM/HRPWM Peripheral Frame 3
- USB Peripheral Frame 3

If the CPU and the DMA make an access to the same interface in the same cycle, the DMA has priority and the CPU is stalled.

If a CPU access to an interface is in progress and another CPU access to the same interface is pending, for example, the CPU is performing a write operation and a read operation from the CPU is pending, then a DMA access to that same interface has priority over the pending CPU access when the current CPU access completes.

NOTE: If the CPU is performing a read-modify-write operation and the DMA performs a write to the same location, the DMA write may be lost if the operation occurs in between the CPU read and the CPU write. For this reason, it is advised not to mix such CPU accesses with DMA accesses to the same locations.

In the case of RAM, a ping-pong scheme can be implemented to avoid the CPU and the DMA accessing the same RAM block concurrently, thus avoiding any stalls or corruption issues.

11.5 Channel Priority

Two priority schemes exist when determining channel priority: Round-robin mode and Channel 1 high-priority mode.

11.5.1 Round-Robin Mode

In this mode, all channels have *equal* priority and each enabled channel is serviced in round-robin fashion as follows:

CH1 → CH2 → CH3 → CH4 → CH5 → CH6 → CH1 → CH2 → …

In the case above, after each channel has transferred a burst of words, the next channel is serviced. You can specify the size of the burst for each channel. Once CH6 (or the last enabled channel) has been serviced, and no other channels are pending, the round-robin state machine enters an idle state.

From the idle state, channel 1 (if enabled) is always serviced first. However, if the DMA is currently processing another channel x, all other pending channels between x and the end of the round are serviced before CH1. It is in this sense that all the channels are of *equal* priority. For instance, take an example where CH1, CH4, and CH5 are enabled in round-robin mode and CH4 is currently being processed. Then CH1 and CH5 both receive an interrupt trigger from their respective peripherals before CH4 completes. CH1 and CH5 are now both pending. When CH4 completes its burst, CH5 will be serviced next. Only after CH5 completes will CH1 be serviced. Upon completion of CH1, if there are no more channels pending, the round-robin state machine will enter an idle state.

A more complicated example is shown below:

- Assume all channels are enabled, and the DMA is in an idle state,
- Initially a trigger occurs on CH1, CH3, and CH5 on the same cycle,
- When the CH1 burst transfer starts, requests from CH3 and CH5 are pending,
- Before completion of the CH1 burst, the DMA receives a request from CH2. Now the pending requests are from CH2, CH3, and CH5,
- After completing the CH1 burst, CH2 will be serviced since it is next in the round-robin scheme after CH1.
- After the burst from CH2 is finished, the CH3 burst will be serviced, followed by CH5 burst.
- Now while the CH5 burst is being serviced, the DMA receives a request from CH1, CH3, and CH6.
- The burst from CH6 will start after the completion of the CH5 burst since it is the next channel after CH5 in the round-robin scheme.
- This will be followed by the CH1 burst and then the CH3 burst
- After the CH3 burst finishes, assuming no more triggers have occurred, the round-robin state machine will enter an idle state.

The round-robin state machine may be reset to the idle state via the DMACTRL[PRIORITYRESET] bit.

11.5.2 Channel 1 High Priority Mode

In this mode, if a CH1 trigger occurs, the current word transfer on any other channel is completed (not the complete burst), execution is halted, and CH1 is serviced for the complete burst count. When the CH1 burst is complete, execution returns to the channel that was active when the CH1 trigger occurred. All other channels have equal priority and each enabled channel is serviced in round-robin fashion as follows:

Higher Priority:	CH1
Lower priority:	CH2 → CH3 → CH4 → CH5 → CH6 → CH2 → …

Given an example where CH1, CH4 and CH5 are enabled in Channel 1 High Priority Mode and CH4 is currently being processed. Then CH1 and CH5 both receive an interrupt trigger from their respective peripherals before CH4 completes. CH1 and CH5 are now both pending. When the current CH4 word transfer is completed, regardless of whether the DMA has completed the entire CH4 burst, CH4 execution will be suspended and CH1 will be serviced. After the CH1 burst completes, CH4 will resume execution.

Upon completion of CH4, CH5 will be serviced. After CH5 completes, if there are no more channels pending, the round-robin state machine will enter an idle state.

Typically Channel 1 would be used in this mode for the ADC, since its data rate is so high. However, Channel 1 High Priority Mode may be used in conjunction with any peripheral.

11.6 Address Pointer and Transfer Control

The DMA state machine is, at its most basic level, two nested loops. The inner loop transfers a burst of data when a peripheral interrupt trigger is received. A burst is the smallest amount of data that can be transferred at one time and its size is defined by the BURST_SIZE register for each channel. The BURST_SIZE register allows a maximum of 32 sixteen-bit words to be transferred in one burst. The outer loop, whose size is set by the TRANSFER_SIZE register for each channel, defines how many bursts are performed in the entire transfer. Since TRANSFER_SIZE is a 16-bit register, the total size of a transfer allowed is well beyond any practical requirement. One CPU interrupt is generated, if enabled, for each transfer. This interrupt can be configured to occur at the beginning or the end of the transfer via the MODE.CHx[CHINTMODE] bit.

In the default setting of the MODE.CHx[ONESHOT] bit, the DMA transfers one burst of data each time a peripheral interrupt trigger is received. After the burst is completed, the state machine moves on to the next pending channel in the priority scheme, even if another trigger for the channel just completed is pending. This feature keeps any single channel from monopolizing the DMA bus. If a transfer of more than the maximum number of words per burst is desired for a single trigger, the MODE.CHx[ONESHOT] bit can be set to complete the entire transfer when triggered. Care is advised when using this mode, since this can create a condition where one trigger uses up the majority of the DMA bandwidth.

Each DMA channel contains a shadowed address pointer for the source and the destination address. These pointers, SRC_ADDR and DST_ADDR, can be independently controlled during the state machine operation. At the beginning of each transfer, the shadowed version of each pointer is copied into its respective active register. During the burst loop, after each word is transferred, the signed value contained in the appropriate source or destination BURST_STEP register is added to the active SRC/DST_ADDR register. During the transfer loop, after each burst is complete, there are two methods that can be used to modify the active address pointer. The first, and default, method is by adding the signed value contained in the SRC/DST_TRANSFER_STEP register to the appropriate pointer. The second is via a process called wrapping, where a wrap address is loaded into the active address pointer. When a wrap procedure occurs, the associated SRC/DST_TRANSFER_STEP register has no effect.

Address wrapping occurs when a number of bursts specified by the appropriate SRC/DST_WRAP_SIZE register completes. Each DMA channel contains two shadowed wrap address pointers, SRC_BEG_ADDR and DST_BEG_ADDR, allowing the source and destination wrapping to be independent of each other. Like the SRC_ADDR and DST_ADDR registers, the active SRC/DST_BEG_ADDR registers are loaded from their shadow counterpart at the beginning of a transfer. When the specified number of bursts has occurred, a two part wrap procedure takes place:

- The appropriate active SRC/DST_BEG_ADDR register is incremented by the signed value contained in the SRC/DST_WRAP_STEP register, then
- The new active SRC/DST_BEG_ADDR register is loaded into the active SRC/DST_ADDR register.

Additionally the wrap counter (SRC/DST_WRAP_COUNT) register is reloaded with the SRC/DST_WRAP_SIZE value to setup the next wrap period. This allows the channel to wrap multiple times within a single transfer. Combined with the first bullet above, this allows the channel to address multiple buffers within a single transfer.

The DMA contains both an active and shadow set of the following address pointers. When a DMA transfer begins, the shadow register set is copied to the active working set of registers. This allows you to program the values of the shadow registers for the next transfer while the DMA works with the active set. It also allows you to implement Ping-Pong buffer schemes without disrupting the DMA channel execution.

Source/Destination Address Pointers (SRC/DST_ADDR)— The value written into the shadow register is the start address of the first location where data is read or written to.

At the beginning of a transfer the shadow register is copied into the active register. The active register performs as the current address pointer.

Source/Destination Begin Address Pointers (SRC/DST_BEG_ADDR)— This is the wrap pointer.

The value written into the shadow register will be loaded into the active register at the start of a transfer. On a wrap condition, the active register will be incremented by the signed value in the appropriate SRC/DST_WRAP_STEP register prior to being loaded into the active SRC/DST_ADDR register.

For each channel, the transfer process can be controlled with the following size values:

Source and Destination Burst Size (BURST_SIZE): — This specifies the number of words to be transferred in a burst.

This value is loaded into the BURST_COUNT register at the beginning of each burst. The BURST_COUNT decrements each word that is transferred and when it reaches a zero value, the burst is complete, indicating that the next channel can be serviced. The behavior of the current channel is defined by the ONE_SHOT bit in the MODE register. The maximum size of the burst is dictated by the type of peripheral. For the ADC, the burst size could be all 16 registers (if all 16 registers are used). For a McBSP peripheral, the burst size is limited to 1 since there is no FIFO and the receive or transmit data register must be loaded or copied every word transferred. For RAM the burst size can be up to the maximum allowed by the BURST_SIZE register, which is 32.

Source and Destination Transfer Size (TRANSFER_SIZE): — This specifies the number of bursts to be transferred before per CPU interrupt (if enabled).

Whether this interrupt is generated at the beginning or the end of the transfer is defined in the CHINTMODE bit in the MODE register. Whether the channel remains enabled or not after the transfer is completed is defined by the CONTINUOUS bit in the MODE register. The TRANSFER_SIZE register is loaded into the TRANSFER_COUNT register at the beginning of each transfer. The TRANSFER_COUNT register keeps track of how many bursts of data the channel has transferred and when it reaches zero, the DMA transfer is complete.

Source/Destination Wrap Size (SRC/DST_WRAP_SIZE)— This specifies the number of bursts to be transferred before the current address pointer wraps around to the beginning.

This feature is used to implement a circular addressing type function. This value is loaded into the appropriate SRC/DST_WRAP_COUNT register at the beginning of each transfer. The SRC/DST_WRAP_COUNT registers keep track of how many bursts of data the channel has transferred and when they reaches zero, the wrap procedure is performed on the appropriate source or destination address pointer. A separate size and count register is allocated for source and destination pointers. To *disable* the wrap function, assign the value of these registers to be larger than the TRANSFER_SIZE.

NOTE: The value written to the SIZE registers is one less than the intended size. So, to transfer three 16-bit words, the value 2 should be placed in the SIZE register.

Regardless of the state of the DATASIZE bit, the value specified in the SIZE registers are for 16-bit addresses. So, to transfer three 32-bit words, the value 5 should be placed in the SIZE register.

For each source/destination pointer, the address changes can be controlled with the following step values:

Source/Destination Burst Step (SRC/DST_BURST_STEP)— Within each burst transfer, the address source and destination step sizes are specified by these registers.

This value is a signed 2's compliment number so that the address pointer can be incremented or decremented as required. If no increment is desired, such as when accessing the McBSP data receive or transmit registers, the value of these registers should be set to zero.

Source/Destination Transfer Step (SRC/DST_TRANSFER_STEP)— This specifies the address offset to start the next burst transfer after completing the current burst transfer.

This is used in cases where registers or data memory locations are spaced at constant intervals. This value is a signed 2's compliment number so that the address pointer can be incremented or decremented as required.

Source/Destination Wrap Step (SRC/DST_WRAP_STEP): — When the wrap counter reaches zero, this value specifies the number of words to add/subtract from the BEG_ADDR pointer and hence sets the new start address.

This implements a circular type of addressing mode, useful in many applications. This value is a signed 2's compliment number so that the address pointer can be incremented or decremented as required.

NOTE: Regardless of the state of the DATASIZE bit, the value specified in the STEP registers are for 16-bit addresses. So, to increment one 32-bit address, a value of 2 should be placed in these registers.

Three modes are provided to control the way the state machine behaves during the burst loop and the transfer loop:

One Shot Mode (ONESHOT)— If one shot mode is enabled when an interrupt event trigger occurs, the DMA will continue transferring data in bursts until TRANSFER_COUNT is zero. If one shot mode is disabled, then an interrupt event trigger is required for each burst transfer and this will continue until TRANSFER_COUNT is zero.

NOTE: When ONESHOT mode is enabled, the DMA will continuously transfer bursts of data on the given channel until the TRANSFER_COUNT value is zero. This could potentially hog the bandwidth of a peripheral and cause long CPU stalls to occur. To avoid this, you could configure a CPU timer (or similar) and disable ONESHOT so as to avoid this situation.

Continuous Mode (CONTINUOUS)— If continuous mode is disabled the RUNSTS bit in the CONTROL register is cleared at the end of the transfer, disabling the DMA channel.

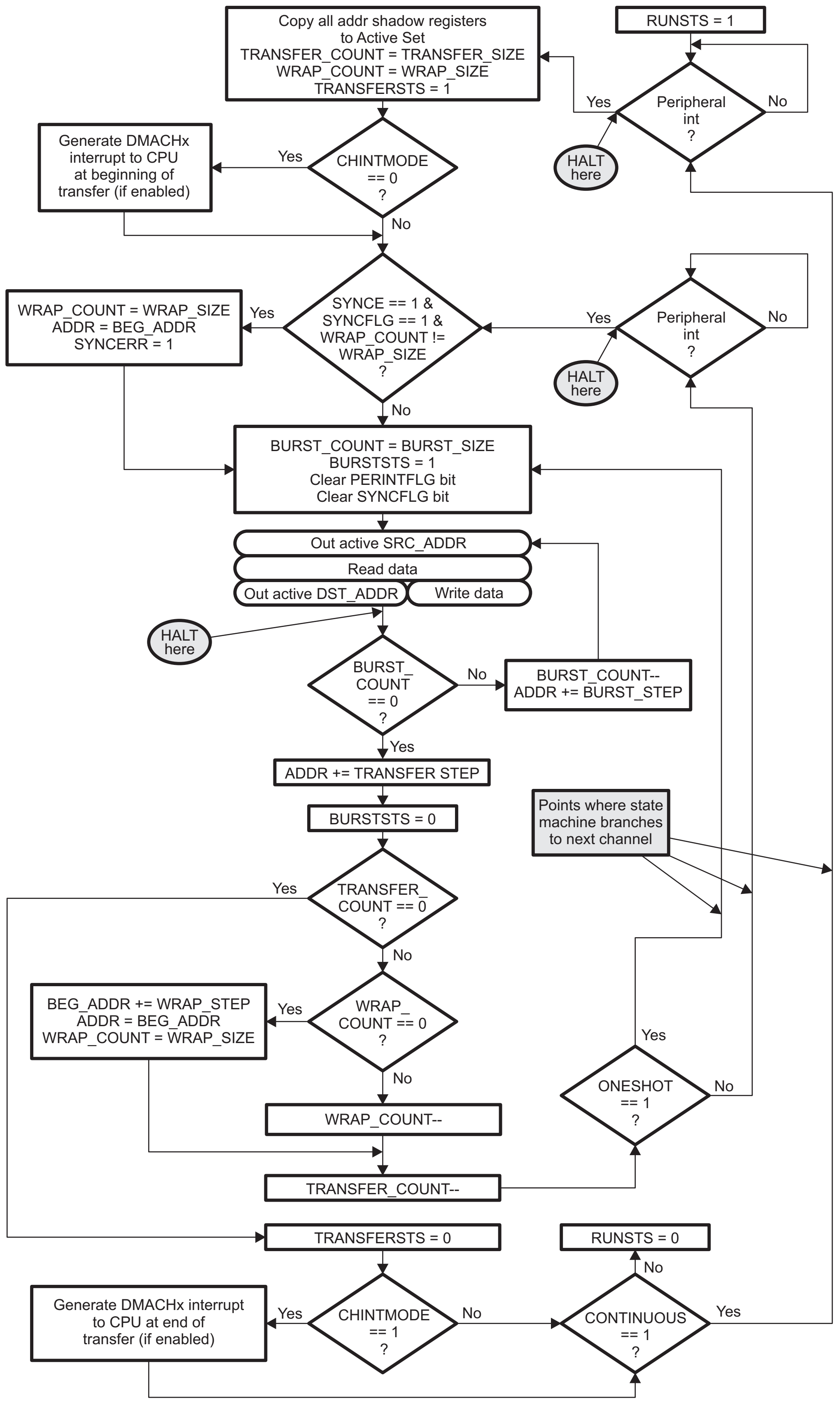
The channel must be re-enabled by setting the RUN bit in the CONTROL register before another transfer can be started on that channel. If the continuous mode is enabled the RUNSTS bit is not cleared at the end of the transfer.

Channel Interrupt Mode (CHINTMODE)— This mode bit selects whether the DMA interrupt from the respective channel is generated at the beginning of a new transfer or at the end of the transfer.

If implementing a ping-pong buffer scheme with continuous mode of operation, then the interrupt would be generated at the beginning, just after the working registers are copied to the shadow set. If the DMA does not operate in continuous mode, then the interrupt is typically generated at the end when the transfer is complete.

All of the above features and modes are shown in Figure 11-5.

Figure 11-5. DMA State Diagram



The following items are in reference to Figure 11-5.

- The *HALT* points represent where the channel halts operation when interrupted by a high priority channel 1 trigger, or when the HALT command is set, or when an emulation halt is issued and the FREE bit is cleared to 0.
- The ADDR registers are not affected by BEG_ADDR at the start of a transfer. BEG_ADDR only affects the ADDR registers on a wrap or sync error. Following is what happens to each of the ADDR registers when a transfer first starts:
  - BEG_ADDR_SHADOW remains unchanged.
  - ADDR_SHADOW remains unchanged.
  - BEG_ADDR = BEG_ADDR_SHADOW
  - ADDR = ADDR_SHADOW
- The active registers get updated when a wrap occurs. The shadow registers remain unchanged. Specifically:
  - BEG_ADDR_SHADOW remains unchanged.
  - ADDR_SHADOW remains unchanged.
  - BEG_ADDR += WRAP_STEP
  - ADDR = BEG_ADDR
- The active registers get updated when a sync error occurs. The shadow registers remain unchanged. Specifically:
  - BEG_ADDR_SHADOW remains unchanged.
  - ADDR_SHADOW remains unchanged.
  - BEG_ADDR remains unchanged.
  - ADDR = BEG_ADDR

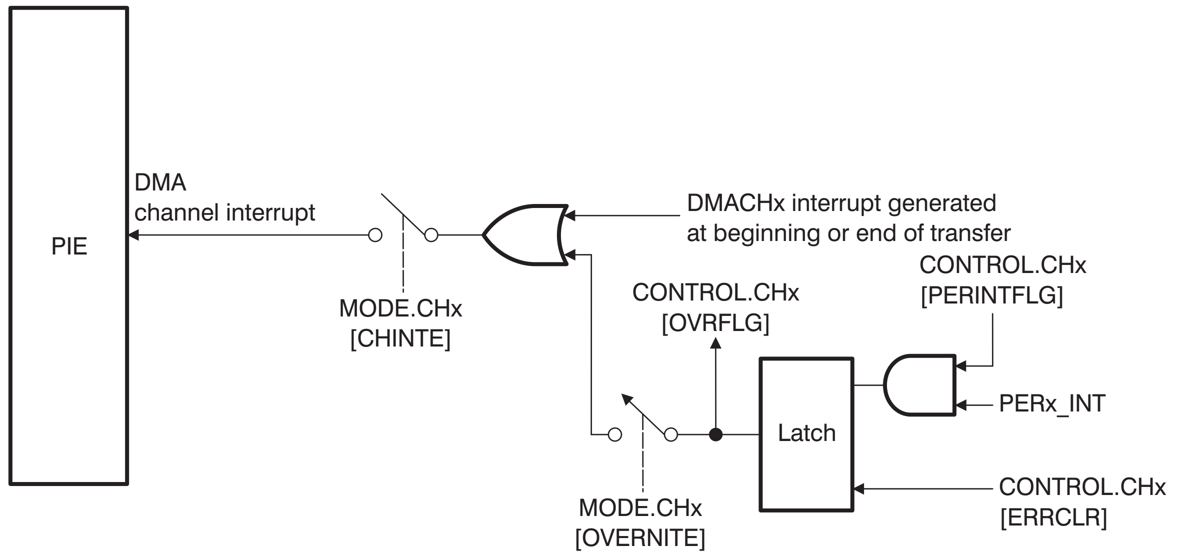
Probably the easiest way to remember all this is that:

- The shadow registers never change except by software.
- The active registers never change except by hardware, and a shadow register is only copied into its own active register, never an active register by another name.

11.7 Overrun Detection Feature

The DMA contains overrun detection logic. When a peripheral event trigger is received by the DMA, the PERINTFLG bit in the CONTROL register is set, pending the channel to the DMA state machine. When the burst for that channel is started, the PERINTFLG is cleared. If however, between the time that the PERINTFLG bit is set by an event trigger and cleared by the start of the burst, an additional event trigger arrives, the second trigger will be lost. This condition will set the OVRFLG bit in the CONTROL register as in Figure 11-6. If the overrun interrupt is enabled then the channel interrupt will be generated to the PIE module.

Figure 11-6. Overrun Detection Logic



11.8 Register Descriptions

The complete DMA register set is shown in Table 11-2.

Table 11-2. DMA Register Summary[1]

Address	Acronym	Description	Section
DMA Control, Mode and Status Registers			
0x1000	DMACTRL	DMA Control Register	Section 11.8.1
0x1001	DEBUGCTRL	Debug Control Register	Section 11.8.2
0x1002	REVISION	Peripheral Revision Register	Section 11.8.3
0x1003	Reserved	Reserved	
0x1004	PRIORITYCTRL1	Priority Control Register 1	Section 11.8.4
0x1005	Reserved	Reserved	
0x1006	PRIORITYSTAT	Priority Status Register	Table 11-7
0x1007 0x101F	Reserved	Reserved	
DMA Channel 1 Registers			
0x1020	MODE	Mode Register	Section 11.8.6
0x1021	CONTROL	Control Register	Section 11.8.7
0x1022	BURST_SIZE	Burst Size Register	Section 11.8.8
0x1023	BURST_COUNT	Burst Count Register	Section 11.8.9
0x1024	SRC_BURST_STEP	Source Burst Step Size Register	Section 11.8.10
0x1025	DST_BURST_STEP	Destination Burst Step Size Register	Section 11.8.11
0x1026	TRANSFER_SIZE	Transfer Size Register	Table 11-13
0x1027	TRANSFER_COUNT	Transfer Count Register	Section 11.8.13
0x1028	SRC_TRANSFER_STEP	Source Transfer Step Size Register	Section 11.8.14
0x1029	DST_TRANSFER_STEP	Destination Transfer Step Size Register	Section 11.8.15
0x102A	SRC_WRAP_SIZE	Source Wrap Size Register	Section 11.8.16
0x102B	SRC_WRAP_COUNT	Source Wrap Count Register	Section 11.8.17
0x102C	SRC_WRAP_STEP	Source Wrap Step Size Register	Section 11.8.18
0x102D	DST_WRAP_SIZE	Destination Wrap Size Register	Section 11.8.16
0x102E	DST_WRAP_COUNT	Destination Wrap Count Register	Section 11.8.17
0x102F	DST_WRAP_STEP	Destination Wrap Step Size Register	Section 11.8.18
0x1030	SRC_BEG_ADDR_SHADOW	Shadow Source Begin and Current Address Pointer Registers	Section 11.8.19
0x1032	SRC_ADDR_SHADOW		Section 11.8.19
0x1034	SRC_BEG_ADDR	Active Source Begin and Current Address Pointer Registers	Section 11.8.20
0x1036	SRC_ADDR		Section 11.8.20
0x1038	DST_BEG_ADDR_SHADOW	Shadow Destination Begin and Current Address Pointer Registers	Section 11.8.21

(1) All DMA register writes are EALLOW protected.

Table 11-2. DMA Register Summary[1] (continued)

Address	Acronym	Description	Section
0x103A	DST_ADDR_SHADOW		Section 11.8.21
0x103C	DST_BEG_ADDR	Active Destination Begin and Current Address Pointer Registers	Section 11.8.22
0x103E	DST_ADDR		Section 11.8.22
0x103F	Reserved	Reserved	
DMA Channel 2 Registers			
0x1040 0x105F	Same as above		
DMA Channel 3 Registers			
0x1060 0x107F	Same as above		
DMA Channel 4 Registers			
0x1080 0x109F	Same as above		
DMA Channel 5 Registers			
0x10A0 0x10BF	Same as above		
DMA Channel 6 Registers			
0x10C0 0x10DF	Same as above		

11.8.1 *DMA Control Register (DMACTRL) — EALLOW Protected*

The DMA control register (DMACTRL) is shown in Figure 11-7 and described in Table 11-3.

Figure 11-7. DMA Control Register (DMACTRL)

15-8
Reserved
R-0

7-2	1	0
Reserved	PRIORITY RESET	HARD RESET
R-0	R0/S-0	R0/S-0

LEGEND: R0/S = Read 0/Set; R = Read only; -*n* = value after reset

Table 11-3. DMA Control Register (DMACTRL) Field Descriptions

Bit	Field	Value	Description
15-2	Reserved		Reserved
1	PRIORITYRESET	0	The priority reset bit resets the round-robin state machine when a 1 is written. Service starts from the first enabled channel. Writes of 0 are ignored and this bit always reads back a 0. When a 1 is written to this bit, any pending burst transfer completes before resetting the channel priority machine. If CH1 is configured as a high priority channel, and this bit is written to while CH1 is servicing a burst, the CH1 burst is completed and then any lower priority channel burst is also completed (if CH1 interrupted in the middle of a burst), before the state machine is reset. In case CH1 is high priority, the *state machine* restarts from CH2 (or the next highest enabled channel).

Table 11-3. DMA Control Register (DMACTRL) Field Descriptions (continued)

Bit	Field	Value	Description
0	HARDRESET	0	Writing a 1 to the hard reset bit resets the whole DMA and aborts any current access (similar to applying a device reset). Writes of 0 are ignored and this bit always reads back a 0. For a *soft* reset, a bit is provided for each channel to perform a gentler reset. Refer to the channel control registers. If the DMA was performing an access to the XINTF and the DMA access was stalled (XREADY not responding), then a HARDRESET would abort the access. The XINTF access would only complete if XREADY is released. When writing to this bit, there is a one cycle delay before it takes effect. Hence at least a one cycle delay (i.e., a NOP instruction) after writing to this bit should be introduced before attempting an access to any other DMA register.

11.8.2 Debug Control Register (DEBUGCTRL) — EALLOW Protected

The debug control register (DEBUGCTRL) is shown in Figure 11-8 and described in Table 11-4.

Figure 11-8. Debug Control Register (DEBUGCTRL)

15	14 — 0
FREE	Reserved
R/W-0	R-0

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

Table 11-4. Debug Control Register (DEBUGCTRL) Field Descriptions

Bit	Field	Value	Description
15	FREE		Emulation Control Bit: This bit specifies the action when an emulation halt event occurs.
		0	DMA runs until the current DMA read-write access is completed and the current status of a DMA is frozen. See the *HALT* points in Figure 11-5 for possible halt states.
		1	DMA is unaffected by emulation suspend (run free)
14-0	Reserved		Reserved

11.8.3 Revision Register (REVISION)

The revision register (REVISION) is shown in Figure 11-9 and described in Table 11-5.

Figure 11-9. Revision Register (REVISION)

15 — 8	7 — 0
TYPE	REV
R	R

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

Table 11-5. Revision Register (REVISION) Field Descriptions

Bit	Field	Value	Description
15-8	TYPE		DMA Type Bits. A type change represents a major functional feature difference in a peripheral module. Within a peripheral type, there may be minor differences between devices which do not affect the basic functionality of the module. These device-specific differences are listed in the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* (SPRU566).
		0x0000	This document describes a Type0 DMA.
7-0	REV		DMA Silicon Revision Bits: These bits specify the DMA revision and are changed if any bug fixes are performed.
		0x0000	First release

11.8.4 Priority Control Register 1 (PRIORITYCTRL1) — EALLOW Protected

The priority control register 1 (PRIORITYCTRL1) is shown in Figure 11-10 and described in Table 11-6.

Figure 11-10. Priority Control Register 1 (PRIORITYCTRL1)

15 ... 1	0
Reserved	CH1 PRIORITY
R-0	R/W-0

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

Table 11-6. Priority Control Register 1 (PRIORITYCTRL1) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved		Reserved
0	CH1PRIORITY		DMA Ch1 Priority: This bit selects whether channel 1 has higher priority or not:
		0	Same priority as all other channels
		1	Highest priority channel
			Channel priority can only be changed when all channels are disabled. A priority reset should be performed before restarting channels after changing priority.

11.8.5 Priority Status Register (PRIORITYSTAT)

The priority status register (PRIORITYSTAT) is shown in Figure 11-11 and described in Table 11-7.

Figure 11-11. Priority Status Register (PRIORITYSTAT)

15–8
Reserved
R-0

7	6–4	3	2–0
Reserved	ACTIVESTS_SHADOW	Reserved	ACTIVESTS
R-0	R-0	R-0	R-0

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

Table 11-7. Priority Status Register (PRIORITYSTAT) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved		Reserved
6-4	ACTIVESTS_SHADOW		Active Channel Status Shadow Bits: These bits are only useful when CH1 is enabled as a higher priority channel. When CH1 is serviced, the ACTIVESTS bits are copied to the shadow bits and indicate which channel was interrupted by CH1. When CH1 service is completed, the shadow bits are copied back to the ACTIVESTS bits. If this bit field is zero or the same as the ACTIVESTS bit field, then no channel is pending due to a CH1 interrupt. When CH1 is not a higher priority channel, these bits should be ignored:
		0,0,0	No channel pending
		0,0,1	CH 1
		0,1,0	CH 2
		0,1,1	CH 3
		1,0,0	CH 4
		1,0,1	CH 5
		1,1,0	CH 6
3	Reserved		Reserved
2-0	ACTIVESTS		Active Channel Status Bits: These bits indicate which channel is currently active or performing a transfer:
		0,0,0	no channel active
		0,0,1	CH 1
		0,1,0	CH 2
		0,1,1	CH 3
		1,0,0	CH 4
		1,0,1	CH 5
		1,1,0	CH 6

11.8.6 Mode Register (MODE) — EALLOW Protected

The mode register (MODE) is shown in Figure 11-12 and described in Table 11-8.

Figure 11-12. Mode Register (MODE)

15	14	13	12	11	10	9	8
CHINTE	DATASIZE	Reserved		CONTINUOUS	ONESHOT	CHINTMODE	PERINTE
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				0
OVRINTE	Reserved		PERINTSEL				
R/W-0	R-0		R/W-0				

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

Table 11-8. Mode Register (MODE) Field Descriptions

Bit	Field	Value	Description
15	CHINTE		Channel Interrupt Enable Bit: This bit enables/disables the respective DMA channel interrupt to the CPU (via the PIE).
		0	Interrupt disabled
		1	Interrupt enabled
14	DATASIZE		Data Size Mode Bit: This bit selects if the DMA channel transfers 16-bits or 32-bits of data at a time.
		0	16-bit data transfer size
		1	32-bit data transfer size
			NOTE: Regardless of the value of this bit all of the registers in the DMA refer to 16-bit words. The only effect this bit causes is whether the databus width is 16 or 32 bits. It is up to you to configure the pointer step increment and size to accommodate 32-bit data transfers. See section Section 11.6 for details.
13-12	Reserved		Reserved
11	CONTINUOUS		Continuous Mode Bit: If this bit is set to 1, then DMA re-initializes when TRANSFER_COUNT is zero and waits for the next interrupt event trigger. If this bit is 0, then the DMA stops and clears the RUNSTS bit to 0.
10	ONESHOT		One Shot Mode Bit: If this bit is set to 1, then subsequent burst transfers occur without additional event triggers after the first event trigger. If this bit is 0 then only one burst transfer is performed per event trigger.
9	CHINTMODE		Channel Interrupt Generation Mode Bit: This bit specifies when the respective DMA channel interrupt should be generated to the CPU (via the PIE).
		0	Generate interrupt at beginning of new transfer
		1	Generate interrupt at end of transfer.
8	PERINTE		Peripheral Interrupt Trigger Enable Bit: This bit enables/disables the selected peripheral interrupt trigger to the DMA.
		0	Interrupt trigger disabled. Neither the selected peripheral nor software can start a DMA burst.
		1	Interrupt trigger enabled.
7	OVRINTE		Overflow Interrupt Enable: This bit when set to 1 enables the DMA to generate an interrupt when an overflow event is detected.
		0	Overflow interrupt disabled
		1	Overflow interrupt enabled
			An overflow interrupt is generated when the PERINTFLG is set and another interrupt event occurs. The PERINTFLG being set indicates a previous peripheral event is latched and has not been serviced by the DMA.
6-5	Reserved		Reserved

Table 11-8. Mode Register (MODE) Field Descriptions (continued)

Bit	Field	Value	Description		
4-0	PERINTSEL		Peripheral Interrupt Source Select Bits: These bits select which interrupt triggers a DMA burst for the given channel. Only one interrupt source can be selected. A DMA burst can also be forced via the PERINTFRC bit.		
		Value	**Interrupt**	**Sync**	**Peripheral**
		0	None	None	No peripheral connection
		1	ADCINT1	None	ADC
		2	ADCINT2	None	
		3	XINT1	None	External Interrupts
		4	XINT2	None	
		5	XINT3	None	
		6	Reserved	None	No peripheral connection
		7	USB0EP1RX	None	USB-0
		8	USB0EP1TX	None	
		9	USB0EP2RX	None	
		10	USB0EP2TX	None	
		11	TINT0	None	CPU Timers
		12	TINT1	None	
		13	TINT2	None	
		14	MXEVTA	None	McBSP-A
		15	MREVTA	None	
		16	Reserved	None	No peripheral conneciton
		17	Reserved	None	
		18	ePWM2SOCA	None	ePWM2
		19	ePWM2SOCB	None	
		20	ePWM3SOCA	None	ePWM3
		21	ePWM3SOCB	None	
		22	ePWM4SOCA	None	ePWM4
		23	ePWM4SOCB	None	
		24	ePWM5SOCA	None	ePWM5
		25	ePWM5SOCB	None	
		26	ePWM6SOCA	None	ePWM6
		27	ePWM6SOCB	None	
		28	ePWM7SOCA	None	ePWM7
		29	ePWM7 SOCB	None	
		30	USB0EP3RX	None	USB-0
		31	USB0EP3TX	None	

11.8.7 Control Register (CONTROL) — EALLOW Protected

The control register (CONTROL) is shown in Figure 11-13 and described in Table 11-9.

Figure 11-13. Control Register (CONTROL)

15	14	13	12	11	10	9	8
Reserved	OVRFLG	RUNSTS	BURSTSTS	TRANSFERSTS	Reserved		PERINTFLG
R-0	R-0	R-0	R-0	R-0	R-0		R-0

7	6	5	4	3	2	1	0
ERRCLR	Reserved		PERINTCLR	PERINTFRC	SOFTRESET	HALT	RUN
R0/S-0	R-0		R0/S-0	R0/S-0	R0/S-0	R0/S-0	R0/S-0

LEGEND: R0/S = Read 0/Set; R = Read only; -*n* = value after reset

Table 11-9. Control Register (CONTROL) Field Descriptions

Bit	Field	Value	Description
15	Reserved		Reserved
14	OVRFLG		Overflow Flag Bit: This bit indicates if a peripheral interrupt event trigger is received from the selected peripheral and the PERINTFLG is already set.
		0	No overflow event
		1	Overflow event
			The ERRCLR bit can be used to clear the state of this bit to 0. The OVRFLG bit is not affected by the PERINTFRC event.
13	RUNSTS		Run Status Bit: This bit is set to 1 when the RUN bit is written to with a 1. This indicates the DMA channel is now ready to process peripheral interrupt event triggers. This bit is cleared to 0 when TRANSFER_COUNT reaches zero and CONTINUOUS mode bit is set to 0. This bit is also cleared to 0 when either the HARDRESET bit, the SOFTRESET bit, or the HALT bit is activated.
		0	Chanel is disabled.
		1	Channel is enabled.
12	BURSTSTS		Burst Status Bit: This bit is set to 1 when a DMA burst transfer begins and the BURST_COUNT is initialized with the BURST_SIZE. This bit is cleared to zero when BURST_COUNT reaches zero. This bit is also cleared to 0 when either the HARDRESET or the SOFTRESET bit is activated.
		0	No burst activity
		1	The DMA is currently servicing or suspending a burst transfer from this channel.
11	TRANSFERSTS		Transfer Status Bit: This bit is set to 1 when a DMA transfer begins and the address registers are copied to the shadow set and the TRANSFER_COUNT is initialized with the TRANSFER_SIZE. This bit is cleared to zero when TRANSFER_COUNT reaches zero. This bit is also cleared to 0 when either the HARDRESET or the SOFTRESET bit is activated.
		0	No transfer activity
		1	The channel is currently in the middle of a transfer regardless of whether a burst of data is actively being transferred or not.
10-9	Reserved		Reserved
8	PERINTFLG		Peripheral Interrupt Trigger Flag Bit: This bit indicates if a peripheral interrupt event trigger has occurred. This flag is automatically cleared when the first burst transfer begins.
		0	No interrupt event trigger
		1	Interrupt event trigger
			The PERINTFRC bit can be used to set the state of this bit to 1 and force a software DMA event. The PERINTCLR bit can be used to clear the state of this bit to 0.
7	ERRCLR	0	Error Clear Bit: Writing a 1 to this bit will clear any latched sync error event and clear the SYNCERR bit. This bit will also clear the OVRFLG bit. This bit would normally be used when initializing the DMA for the first time or if an overflow condition is detected. If an ADCSYNC error event or overflow event occurs at the same time as writing to this bit, the ADC or overrun has priority and the SYNCERR or OVRFLG bit is set.
6-5	Reserved		Reserved

Table 11-9. Control Register (CONTROL) Field Descriptions (continued)

Bit	Field	Value	Description
4	PERINTCLR	0	Peripheral Interrupt Clear Bit: Writing a 1 to this bit clears any latched peripheral interrupt event and clears the PERINTFLG bit. This bit would normally be used when initializing the DMA for the first time. If a peripheral event occurs at the same time as writing to this bit, the peripheral has priority and the PERINTFLG bit is set.
3	PERINTFRC	0	Peripheral Interrupt Force Bit: Writing a 1 to this bit latches a peripheral interrupt event trigger and sets the PERINTFLG bit. If the PERINTE bit is set, this bit can be used like a software force for a DMA burst transfer.

2	SOFTRESET	0	Channel Soft Reset Bit: Writing a 1 to this bit completes current read-write access and places the channel into a default state as follows:
			RUNSTS = 0
			TRANSFERSTS = 0
			BURSTSTS = 0
			BURST_COUNT = 0
			TRANSFER_COUNT = 0
			SRC_WRAP_COUNT = 0
			DST_WRAP_COUNT = 0
			This is a *soft* reset that basically allows the DMA to complete the current read-write access and then places the DMA channel into the default reset state.

1	HALT	0	Channel Halt Bit: Writing a 1 to this bit halts the DMA at the current state and any current read-write access is completed. See Figure 11-5 for the various positions the state machine can be at when HALTED. The RUNSTS bit is set to 0. To take the device out of HALT, the RUN bit needs to be activated.
0	RUN	0	Channel Run Bit: Writing a 1 to this bit starts the DMA channel. The RUNSTS bit is set to 1. This bit is also used to take the device out of HALT. The RUN bit is typically used to start the DMA running after you have configured the DMA. It will then wait for the first interrupt event (PERINTFLG == 1) to start operation. The RUN bit can also be used to take the DMA channel out of a HALT condition See Figure 11-5 for the various positions the state machine can be at when HALTED.

11.8.8 Burst Size Register (BURST_SIZE) — EALLOW Protected

The burst size register (BURST_SIZE) is shown in Figure 11-14 and described in Table 11-10.

Figure 11-14. Burst Size Register (BURST_SIZE)

15 5	4 0
Reserved	BURSTSIZE
R-0	R/W-0

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

Table 11-10. Burst Size Register (BURST_SIZE) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved		Reserved
4-0	BURSTSIZE		These bits specify the burst transfer size:
		0	Transfer 1 word in a burst
		1	Transfer 2 words in a burst
		...	...
		31	Transfer 32 words in a burst

11.8.9 BURST_COUNT Register

The burst count register (BURST_COUNT) is shown in Figure 11-15 and described in Table 11-11.

Figure 11-15. Burst Count Register (BURST_COUNT)

15 5	4 0
Reserved	BURSTCOUNT
R-0	R/W-0

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

Table 11-11. Burst Count Register (BURST_COUNT) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved		Reserved
4-0	BURSTCOUNT		These bits indicate the current burst counter value:
		0	0 word left in a burst
		1	1 word left in a burst
		2	2 words left in a burst
		...	...
		31	31 words left in a burst
			The above values represent the state of the counter at the HALT conditions.

11.8.10 Source Burst Step Register Size (SRC_BURST_STEP) — EALLOW Protected

The source burst step size register (SRC_BURST_STEP) is shown in Figure 11-16 and described in Table 11-12.

Figure 11-16. Source Burst Step Size Register (SRC_BURST_STEP)

15 — 0
SRCBURSTSTEP
R/W-0

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

Table 11-12. Source Burst Step Size Register (SRC_BURST_STEP) Field Descriptions

Bit	Field	Value	Description
15-0	SRCBURSTSTEP		These bits specify the source address post-increment/decrement step size while processing a burst of data:
		0x0FFF	Add 4095 to address
		...	...
		0x0002	Add 2 to address
		0x0001	Add 1 to address
		0x0000	No address change
		0xFFFF	Sub 1 from address
		0xFFFE	Sub 2 from address
		...	...
		0xF000	Sub 4096 from address
			Only values from -4096 to 4095 are valid.

11.8.11 Destination Burst Step Register Size (DST_BURST_STEP) — EALLOW Protected

The destination burst step register size (DST_BURST_STEP) is shown in Figure 11-17 and described in Table 11-13.

Figure 11-17. Destination Burst Step Register Size (DST_BURST_STEP)

15 ... 0
DSTBURSTSTEP
R/W-0

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

Table 11-13. Destination Burst Step Register Size (DST_BURST_STEP) Field Descriptions

Bit	Field	Value	Description
15-0	DSTBURSTSTEP		These bits specify the destination address post-increment/decrement step size while processing a burst of data:
		0x0FFF	Add 4095 to address
		...	...
		0x0002	Add 2 to address
		0x0001	Add 1 to address
		0x0000	No address change
		0xFFFF	Sub 1 from address
		0xFFFE	Sub 2 from address
		...	...
		0xF000	Sub 4096 from address
			Only values from -4096 to 4095 are valid.

11.8.12 Transfer Size Register (TRANSFER_SIZE) — EALLOW Protected

The transfer size register (TRANSFER_SIZE) is shown in Figure 11-18 and described in Table 11-14.

Figure 11-18. Transfer Size Register (TRANSFER_SIZE)

15 ... 0
TRANSFERSIZE
R/W-0

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

Table 11-14. Transfer Size Register (TRANSFER_SIZE) Field Descriptions

Bit	Field	Value	Description
15-0	TRANSFERSIZE		These bits specify the number of bursts to transfer:
		0x0000	Transfer 1 burst
		0x0001	Transfer 2 bursts
		0x0002	Transfer 3 bursts
		...	...
		0xFFFF	Transfer 65536 bursts

11.8.13 Transfer Count Register (TRANSFER_COUNT)

The transfer count register (TRANSFER_COUNT) is shown in Figure 11-19 and described in Table 11-15.

Figure 11-19. Transfer Count Register (TRANSFER_COUNT)

15 ... 0
TRANSFERCOUNT
R/W-0

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

Table 11-15. Transfer Count Register (TRANSFER_COUNT) Field Descriptions

Bit	Field	Value	Description
15-0	TRANSFERCOUNT		These bits specify the current transfer counter value:
		0x0000	0 bursts left to transfer
		0x0001	1 burst left to transfer
		0x0002	2 bursts left to transfer
		...	...
		0xFFFF	65535 bursts left to transfer
			The above values represent the state of the counter at the HALT conditions.

11.8.14 Source Transfer Step Size Register (SRC_TRANSFER_STEP) — EALLOW Protected

The source transfer step size register (SRC_TRANSFER_STEP) is shown in Figure 11-20 and described in Table 11-16.

Figure 11-20. Source Transfer Step Size Register (SRC_TRANSFER_STEP)

15 ... 0
SRCTRANSFERSTEP
R/W-0

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

Table 11-16. Source Transfer Step Size Register (SRC_TRANSFER_STEP) Field Descriptions

Bit	Field	Value	Description
15-0	SRCTRANSFERSTEP		These bits specify the source address pointer post-increment/decrement step size after processing a burst of data:
		0x0FFF	Add 4095 to address
		...	...
		0x0002	Add 2 to address
		0x0001	Add 1 to address
		0x0000	No address change
		0xFFFF	Sub 1 from address
		0xFFFE	Sub 2 from address
		...	...
		0xF000	Sub 4096 from address
			Only values from -4096 to 4095 are valid.

11.8.15 Destination Transfer Step Size Register (DST_TRANSFER_STEP) — EALLOW Protected

The destination transfer step size register (DST_TRANSFER_STEP) is shown in Figure 11-21 and described in Table 11-17.

Figure 11-21. Destination Transfer Step Size Register (DST_TRANSFER_STEP)

15 ... 0
DSTTRANSFERSTEP
R/W-0

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

Table 11-17. Destination Transfer Step Size Register (DST_TRANSFER_STEP) Field Descriptions

Bit	Field	Value	Description
15-0	DSTTRANSFERSTEP		These bits specify the destination address pointer post-increment/decrement step size after processing a burst of data:
		0x0FFF	Add 4095 to address
		...	...
		0x0002	Add 2 to address
		0x0001	Add 1 to address
		0x0000	No address change
		0xFFFF	Sub 1 from address
		0xFFFE	Sub 2 from address
		...	...
		0xF000	Sub 4096 from address
			Only values from -4096 to 4095 are valid.

11.8.16 Source/Destination Wrap Size Register (SRC/DST_WRAP_SIZE) — EALLOW protected)

The source/destination wrap size register is shown in Figure 11-22 and described in Table 11-18.

Figure 11-22. Source/Destination Wrap Size Register (SRC/DST_WRAP_SIZE)

15 ... 0
WRAPSIZE
R/W-0

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

Table 11-18. Source/Destination Wrap Size Register (SRC/DST_WRAP_SIZE) Field Descriptions

Bit	Field	Value	Description
15-0	WRAPSIZE		These bits specify the number of bursts to transfer before wrapping back to begin address pointer:
		0x0000	Wrap after 1 burst
		0x0001	Wrap after 2 bursts
		0x0002	Wrap after 3 bursts
		...	...
		0xFFFF	Wrap after 65536 bursts
			To *disable* the wrap function, set the WRAPSIZE bit field to a number larger than the TRANSFERSIZE bit field.

11.8.17 Source/Destination Wrap Count Register (SCR/DST_WRAP_COUNT)

The source/destination wrap count register (SCR/DST_WRAP_COUNT) is shown in Figure 11-23 and described in Table 11-19.

Figure 11-23. Source/Destination Wrap Count Register (SCR/DST_WRAP_COUNT)

15 ... 0
WRAPCOUNT
R/W-0

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

Table 11-19. Source/Destination Wrap Count Register (SCR/DST_WRAP_COUNT) Field Descriptions

Bit	Field	Value	Description
15-0	WRAPCOUNT		These bits indicate the current wrap counter value:
		0x0000	Wrap complete
		0x0001	1 burst left
		0x0002	2 burst left
		...	...
		0xFFFF	65535 burst left
			The above values represent the state of the counter at the HALT conditions.

11.8.18 Source/Destination Wrap Step Size Registers (SRC/DST_WRAP_STEP) — EALLOW Protected

The source/destination wrap step size register (SRC/DST_WRAP_STEP) are shown in Figure 11-24 and described in Table 11-20.

Figure 11-24. Source/Destination Wrap Step Size Registers (SRC/DST_WRAP_STEP)

15 ... 0
WRAPSTEP
R/W-0

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

Table 11-20. Source/Destination Wrap Step Size Registers (SRC/DST_WRAP_STEP) Field Descriptions

Bit	Field	Value	Description
15-0	WRAPSTEP		These bits specify the source begin address pointer post-increment/decrement step size after wrap counter expires:
		0x0FFF	Add 4095 to address
		...	...
		0x0002	Add 2 to address
		0x0001	Add 1 to address
		0x0000	No address change
		0xFFFF	Sub 1 from address
		0xFFFE	Sub 2 from address
		...	...
		0xF000	Sub 4096 from address
			Only values from -4096 to 4095 are valid.

11.8.19 Shadow Source Begin and Current Address Pointer Registers (SRC_BEG_ADDR_SHADOW/DST_BEG_ADDR_SHADOW) — All EALLOW Protected

The shadow source begin and current address pointer registers (SRC_BEG_ADDR_SHADOW/DST_BEG_ADDR_SHADOW) are shown in Figure 11-25 and described in Table 11-21.

Figure 11-25. Shadow Source Begin and Current Address Pointer Registers (SRC_BEG_ADDR_SHADOW/DST_BEG_ADDR_SHADOW)

31 22	21 0
Reserved	BEGADDR
R-0	R/W-0

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

Table 11-21. Shadow Source Begin and Current Address Pointer Registers (SRC_BEG_ADDR_SHADOW/DST_BEG_ADDR_SHADOW) Field Descriptions

Bit	Field	Value	Description
31-22	Reserved		Reserved
21-0	BEGADDR		22-bit address value

11.8.20 Active Source Begin and Current Address Pointer Registers (SRC_BEG_ADDR/DST_BEG_ADDR)

The active source begin and current address pointer registers (SRC_BEG_ADDR/DST_BEG_ADDR) are shown in Table 11-22 and described in Table 11-22.

Figure 11-26. Active Source Begin and Current Address Pointer Registers (SRC_BEG_ADDR/DST_BEG_ADDR)

31 22	21 0
Reserved	BEGADDR
R-0	R-0

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

Table 11-22. Active Source Begin and Current Address Pointer Registers (SRC_BEG_ADDR/DST_BEG_ADDR) Field Descriptions

Bit	Field	Value	Description
31-22	Reserved		Reserved
21-0	BEGADDR		22-bit address value

11.8.21 Shadow Destination Begin and Current Address Pointer Registers (SRC_ADDR_SHADOW/DST_ADDR_SHADOW) — All EALLOW Protected

The shadow destination begin and current address pointer registers (SRC_ADDR_SHADOW/DST_ADDR_SHADOW) are shown in Figure 11-27 and described in Table 11-23.

Figure 11-27. Shadow Destination Begin and Current Address Pointer Registers (SRC_ADDR_SHADOW/DST_ADDR_SHADOW)

31 22	21 0
Reserved	ADDR
R-0	R/W-0

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

Table 11-23. Shadow Destination Begin and Current Address Pointer Registers (SRC_ADDR_SHADOW/DST_ADDR_SHADOW) Field Descriptions

Bit	Field	Value	Description
31-22	Reserved		Reserved
21-0	ADDR		22-bit address value

11.8.22 Active Destination Begin and Current Address Pointer Registers (SRC_ADDR/DST_ADDR)

The active destination begin and current address pointer registers (SRC_ADDR/DST_ADDR) are shown in Figure 11-28 and described in Table 11-24.

Figure 11-28. Active Destination Begin and Current Address Pointer Registers (SRC_ADDR/DST_ADDR)

31 22	21 0
Reserved	ADDR
R-0	R-0

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

Table 11-24. Active Destination Begin and Current Address Pointer Registers (SRC_ADDR/DST_ADDR) Field Descriptions

Bit	Field	Value	Description
31-22	Reserved		Reserved
21-0	ADDR		22-bit address value

Serial Peripheral Interface (SPI)

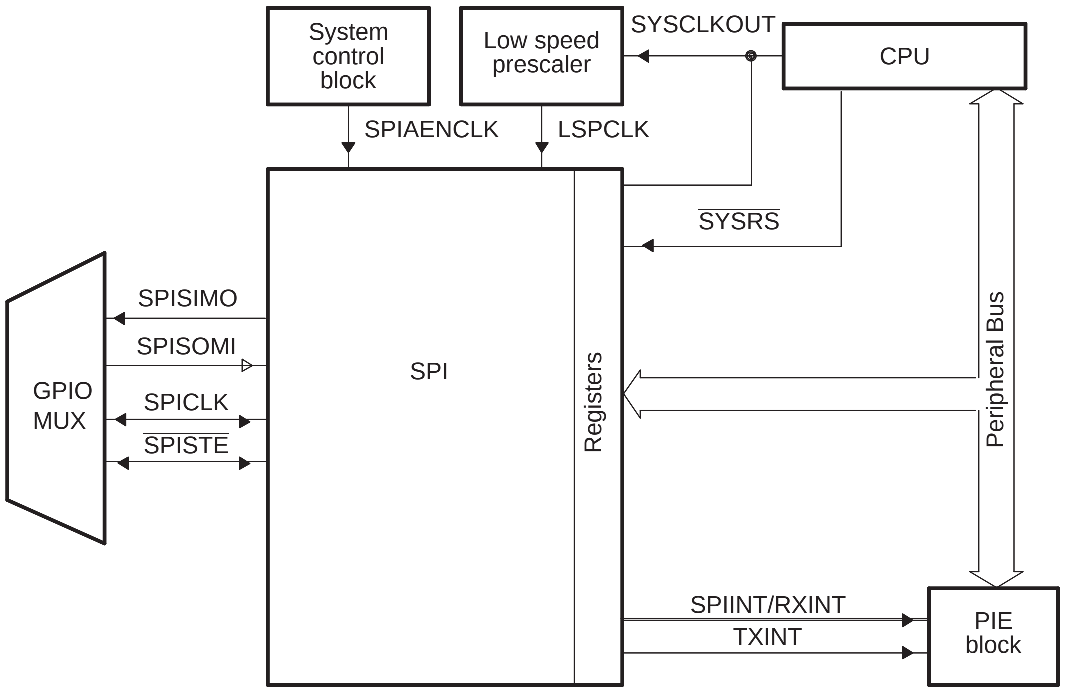
The serial peripheral interface (SPI) is a high-speed synchronous serial input/ output (I/O) port that allows a serial bit stream of programmed length (one to 16 bits) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI is normally used for communications between the DSP controller and external peripherals or another controller. Typical applications include external I/O or peripheral expansion via devices such as shift registers, display drivers, and analog-to-digital converters (ADCs). Multi-device communications are supported by the master/slave operation of the SPI. On the C28x, the port supports a 4-level, receive and transmit FIFO for reducing CPU servicing overhead.

Topic **Page**

12.1 Enhanced SPI Module Overview

Figure 12-1 shows the SPI CPU interfaces.

Figure 12-1. SPI CPU Interface



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: one to sixteen 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 falling 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 algorithms.
- 12 SPI module control registers: Located in control register frame beginning at address 7040h.

NOTE: All registers in this module are 16-bit registers that are connected to Peripheral Frame 2. When a register is accessed, the register data is in the lower byte (7−0), and the upper byte (15−8) is read as zeros. Writing to the upper byte has no effect.

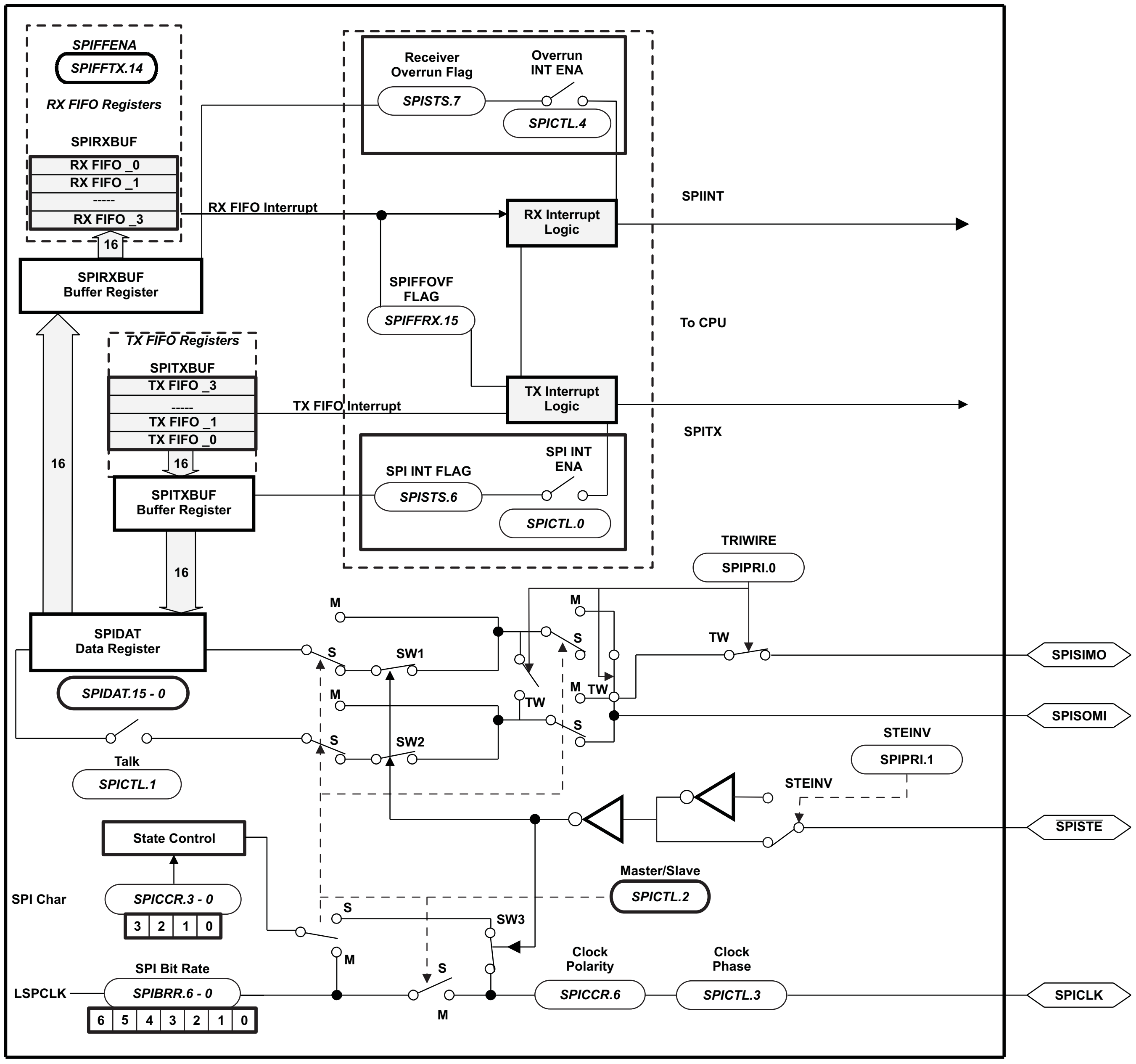
Enhanced Features:

- 4-level transmit/receive FIFO
- Delayed transmit control
- 3-wire SPI mode
- $\overline{\text{SPISTE}}$ inversion for digital audio interface receive mode on devices with two SPI modules.

12.1.1 SPI Block Diagram

Figure 12-2 is a block diagram of the SPI in slave mode, showing the basic control blocks available on the SPI module.

Figure 12-2. Serial Peripheral Interface Module Block Diagram



A SPISTE of a slave device is driven low by the master.

12.1.2 SPI Module Signal Summary

Table 12-1. SPI Module Signal Summary

Signal Name	Description
External Signals	
SPICLK	SPI clock
SPISIMO/SPIMOMI[(1)]	SPI slave in, master out/ SPI master out, master in
SPISOMI/SPISISO[(1)]	SPI slave out, master in/ SPI slave in, slave out
$\overline{\text{SPISTE}}$	SPI slave transmit enable
Control	
SPI Clock Rate	LSPCLK
Interrupt signals	
SPIRXINT	Transmit interrupt/ Receive Interrupt in non FIFO mode (referred to as SPI INT)
	Receive in interrupt in FIFO mode
SPITXINT	Transmit interrupt – FIFO

(1) In 3-wire master mode, the SPISIMO pin becomes the SPIMOMI pin and the SPISOMI pin becomes a general purpose input/output (GPIO) pin. In 3-wire slave mode, the SPISOMI pin becomes the SPISISO pin and the SPISIMO pin becomes a GPIO pin.

12.1.3 Overview of SPI Module Registers

The SPI port operation is configured and controlled by the registers listed in Table 12-2.

Table 12-2. SPI Registers

Name	Address Range	Size (x16)	Description
SPICCR	0x0000-7040	1	SPI Configuration Control Register
SPICTL	0x0000-7041	1	SPI Operation Control Register
SPIST	0x0000-7042	1	SPI Status Register
SPIBRR	0x0000-7044	1	SPI Baud Rate Register
SPIEMU	0x0000-7046	1	SPI Emulation Buffer Register
SPIRXBUF	0x0000-7047	1	SPI Serial Input Buffer Register
SPITXBUF	0x0000-7048	1	SPI Serial Output Buffer Register
SPIDAT	0x0000-7049	1	SPI Serial Data Register
SPIFFTX	0x0000-704A	1	SPI FIFO Transmit Register
SPIFFRX	0x0000-704B	1	SPI FIFO Receive Register
SPIFFCT	0x0000-704C	1	SPI FIFO Control Register
SPIPRI	0x0000-704F	1	SPI Priority Control Register

This SPI has 16-bit transmit and receive capability, with double-buffered transmit and double-buffered receive. All data registers are 16-bits wide.

The SPI is no longer limited to a maximum transmission rate of LSPCLK/8 in slave mode. The maximum transmission rate in both slave mode and master mode is now LSPCLK/4.

Writes of transmit data to the serial data register, SPIDAT (and the new transmit buffer, SPITXBUF), must be left-justified within a 16-bit register.

The control and data bits for general-purpose bit I/O multiplexing have been removed from this peripheral, along with the associated registers, SPIPC1 (704Dh) and SPIPC2 (704Eh). These bits are now in the General-Purpose I/O registers.

Twelve registers inside the SPI module control the SPI operations:

- SPICCR (SPI configuration control register). Contains control bits used for SPI configuration
  - SPI module software reset

  - SPICLK polarity selection
  - Four SPI character-length control bits
- SPICTL (SPI operation control register). Contains control bits for data transmission
  - Two SPI interrupt enable bits
  - SPICLK phase selection
  - Operational mode (master/slave)
  - Data transmission enable
- SPISTS (SPI status register). Contains two receive buffer status bits and one transmit buffer status bit
  - RECEIVER OVERRUN
  - SPI INT FLAG
  - TX BUF FULL FLAG
- SPIBRR (SPI baud rate register). Contains seven bits that determine the bit transfer rate
- SPIRXEMU (SPI receive emulation buffer register). Contains the received data. This register is used for emulation purposes only. The SPIRXBUF should be used for normal operation
- SPIRXBUF (SPI receive buffer — the serial receive buffer register). Contains the received data
- SPITXBUF (SPI transmit buffer — the serial transmit buffer register). Contains the next character to be transmitted
- SPIDAT (SPI data register). Contains data to be transmitted by the SPI, acting as the transmit/receive shift register. Data written to SPIDAT is shifted out on subsequent SPICLK cycles. For every bit shifted out of the SPI, a bit from the receive bit stream is shifted into the other end of the shift register
- SPIPRI (SPI priority register). Contains bits that specify interrupt priority and determine SPI operation on the XDS emulator during program suspensions. This register also contains bit to enable 3-wire mode and the $\overline{\text{SPISTE}}$ inversion bit.

12.1.4 SPI Operation

This section describes the operation of the SPI. Included are explanations of the operation modes, interrupts, data format, clock sources, and initialization. Typical timing diagrams for data transfers are given.

12.1.4.1 Introduction to Operation

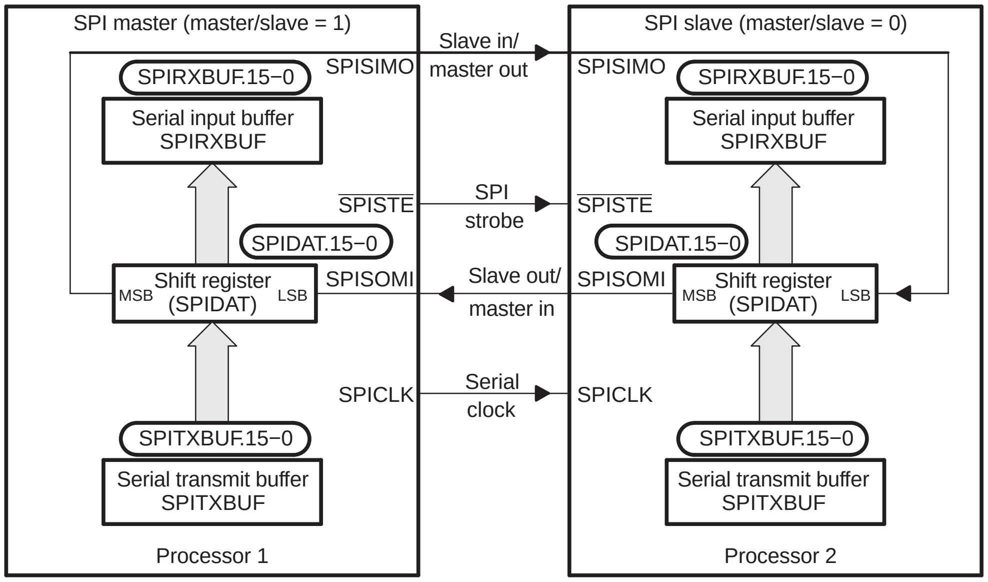
Figure 12-3 shows typical connections of the SPI for communications between two controllers: a master and a slave.

The master initiates data transfer 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 CLOCK PHASE bit (SPICTL.3) is high, data is transmitted and received a half-cycle before the SPICLK transition (see Section 12.1.4.2). 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.

Figure 12-3. SPI Master/Slave Connection


SPI master (master/slave = 1)
SPIRXBUF.15−0
Serial input buffer SPIRXBUF
SPISIMO
SPISTE
SPIDAT.15−0
MSB Shift register (SPIDAT) LSB
SPISOMI
SPICLK
SPITXBUF.15−0
Serial transmit buffer SPITXBUF
Processor 1
Slave in/ master out
SPI strobe
Slave out/ master in
Serial clock
SPI slave (master/slave = 0)
SPISIMO
SPIRXBUF.15−0
Serial input buffer SPIRXBUF
SPISTE
SPIDAT.15−0
SPISOMI
MSB Shift register (SPIDAT) LSB
SPICLK
SPITXBUF.15−0
Serial transmit buffer SPITXBUF
Processor 2

12.1.4.2 SPI Module Slave and Master Operation Modes

The SPI can operate in master or slave mode. The MASTER/SLAVE bit (SPICTL.2) selects the operating mode and the source of the SPICLK signal.

12.1.4.2.1 Master Mode

In the 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 126 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.
- SPI INT FLAG bit (SPISTS.6) is set to 1.
- If there is valid data in the transmit buffer SPITXBUF, as indicated by the TXBUF FULL bit in SPISTS, this data is transferred to SPIDAT and is transmitted; otherwise, SPICLK stops after all bits have been shifted out of SPIDAT.
- If the SPI INT ENA bit (SPICTL.0) 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 slave SPI device. This pin is driven low by the master before transmitting data to the slave and is taken high after the transmission is complete.

12.1.4.2.2 Slave Mode

In the 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. Data written to the SPITXBUF register will be transferred to the SPIDAT register when all bits of the character to be transmitted have been shifted out of SPIDAT. If no character is currently being transmitted when SPITXBUF is written to, the data will be transferred immediately to SPIDAT. 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 SPITXBUF has not been previously loaded, the data must be written to SPITXBUF or SPIDAT before the beginning of the SPICLK signal.

When the TALK bit (SPICTL.1) 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.

12.1.5 SPI Interrupts

This section includes information on the control bits that initialize interrupts, data format, clocking, initialization, and data transfer.

12.1.5.1 SPI Interrupt Control Bits

Five control bits are used to initialize the SPI interrupts:

- SPI INT ENA bit (SPICTL.0)
- SPI INT FLAG bit (SPISTS.6)
- OVERRUN INT ENA bit (SPICTL.4)
- RECEIVER OVERRUN FLAG bit (SPISTS.7)

12.1.5.1.1 SPI INT ENA Bit (SPICTL.0)

When the SPI interrupt-enable bit is set and an interrupt condition occurs, the corresponding interrupt is asserted.

0	Disable SPI interrupts
1	Enable SPI interrupts

12.1.5.1.2 SPI INT FLAG Bit (SPISTS.6)

This status flag indicates that a character has been placed in the SPI receiver buffer and is ready to be read.

When a complete character has been shifted into or out of SPIDAT, the SPI INT FLAG bit (SPISTS.6) is set, and an interrupt is generated if enabled by the SPI INT ENA bit (SPICTL.0). The interrupt flag remains set until it is cleared by one of the following events:

- The interrupt is acknowledged (this is different from the C240).
- The CPU reads the SPIRXBUF (reading the SPIRXEMU does not clear the SPI INT FLAG bit).
- The device enters IDLE2 or HALT mode with an IDLE instruction.
- Software clears the SPI SW RESET bit (SPICCR.7).
- A system reset occurs.

When the SPI INT FLAG bit is set, a character has been placed into the SPIRXBUF and is ready to be read. If the CPU does not read the character by the time the next complete character has been received, the new character is written into SPIRXBUF, and the RECEIVER OVERRUN Flag bit (SPISTS.7) is set.

12.1.5.1.3 OVERRUN INT ENA Bit (SPICTL.4)

Setting the overrun interrupt enable bit allows the assertion of an interrupt whenever the RECEIVER OVERRUN Flag bit (SPISTS.7) is set by hardware. Interrupts generated by SPISTS.7 and by the SPI INT FLAG bit (SPISTS.6) share the same interrupt vector.

0	Disable RECEIVER OVERRUN Flag bit interrupts
1	Enable RECEIVER OVERRUN Flag bit interrupts

12.1.5.1.4 RECEIVER OVERRUN FLAG Bit (SPISTS.7)

The RECEIVER OVERRUN Flag bit is set whenever a new character is received and loaded into the SPIRXBUF before the previously received character has been read from the SPIRXBUF. The RECEIVER OVERRUN Flag bit must be cleared by software.

12.1.5.2 Data Format

Four bits (SPICCR.3–0) specify the number of bits (1 to 16) in the data character. 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 12-1).

Example 12-1. Transmission of Bit From SPIRXBUF

Conditions:

1. Transmission character length = 1 bit (specified in bits SPICCR.3−0)
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[(1)]	← (RXed)
SPIRXBUF (after transmission)																	
	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	x[(1)]	

(1) x = 1 if SPISOMI data is high; x = 0 if SPISOMI data is low; master mode is assumed.

12.1.5.3 Baud Rate and Clocking Schemes

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.

Example 12-2 shows how to determine the SPI baud rates.

Example 12-2. Baud Rate Determination

For SPIBRR = 3 to 127:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{(\text{SPIBRR} + 1)} \qquad (5)$$

For SPIBRR = 0, 1, or 2:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{4} \qquad (6)$$

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 (which is device-specific) and the baud rate at which you will be operating.

Example 1−2 shows how to determine the maximum baud rate at which a 240xA can communicate. Assume that LSPCLK = 40 MHz.

Example 12-3. Maximum Baud-Rate Calculation

$$\begin{aligned}\text{Maximum SPI Baud Rate} &= \frac{\text{LSPCLK}}{4}\\ &= \frac{40 \times 10^6}{4}\\ &= 10 \times 10^6 \text{ bps}\end{aligned} \qquad (7)$$

12.1.5.3.1 SPI Clocking Schemes

The CLOCK POLARITY bit (SPICCR.6) and the CLOCK PHASE bit (SPICTL.3) control four different clocking schemes on the SPICLK pin. The CLOCK POLARITY bit selects the active edge, either rising or falling, of the clock. The CLOCK PHASE bit selects a half-cycle delay of the clock. The four different clocking schemes are as follows:

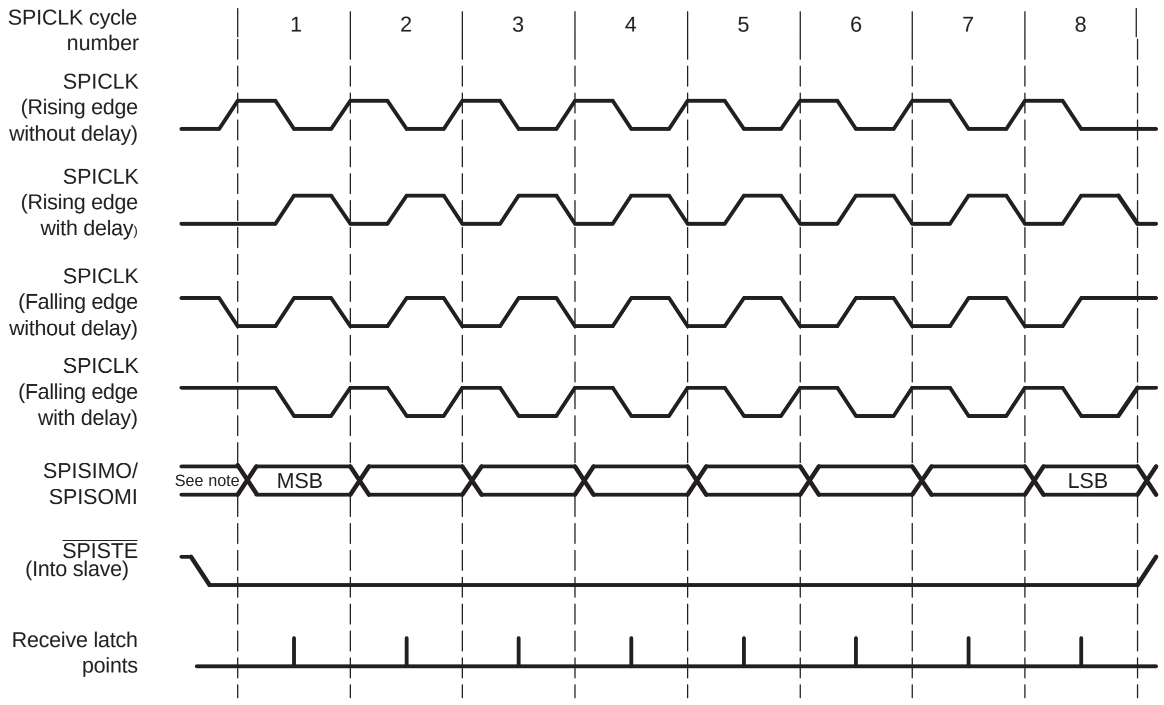
- Falling Edge Without Delay. The SPI transmits data on the falling edge of the SPICLK and receives data on the rising edge of the SPICLK.
- Falling Edge With Delay. The SPI transmits data one half-cycle ahead of the falling edge of the SPICLK signal and receives data on the falling edge of the SPICLK 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 12-3. Examples of these four clocking schemes relative to transmitted and received data are shown in Figure 12-4.

Table 12-3. SPI Clocking Scheme Selection Guide

SPICLK Scheme	CLOCK POLARITY (SPICCR.6)	CLOCK PHASE (SPICTL.3)
Rising edge without delay	0	0
Rising edge with delay	0	1
Falling edge without delay	1	0
Falling edge with delay	1	1

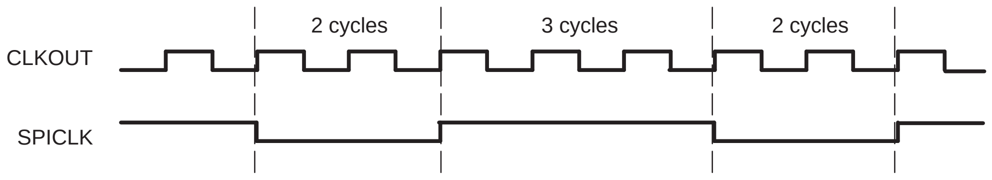
Figure 12-4. SPICLK Signal Options


SPICLK cycle number
1
2
3
4
5
6
7
8
SPICLK (Rising edge without delay)
SPICLK (Rising edge with delay)
SPICLK (Falling edge without delay)
SPICLK (Falling edge with delay)
SPISIMO/ SPISOMI
See note
MSB
LSB
SPISTE (Into slave)
Receive latch points

Note: Previous data bit

For the SPI, 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 CLKOUT longer than the high pulse when the CLOCK POLARITY bit is clear (0). When the CLOCK POLARITY bit is set to 1, the high pulse of the SPICLK is one CLKOUT longer than the low pulse, as shown in Figure 12-5.

Figure 12-5. SPI: SPICLK-CLKOUT Characteristic When (BRR + 1) is Odd, BRR > 3, and CLOCK POLARITY = 1



12.1.5.4 Initialization Upon Reset

A system reset forces the SPI peripheral module 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
- SPI module pin functions are selected as general-purpose inputs (this is done in I/O MUX control register B [MCRB])

To change this SPI configuration:

Step 1. Clear the SPI SW RESET bit (SPICCR.7) to 0 to force the SPI to the reset state.

Step 2. Initialize the SPI configuration, format, baud rate, and pin functions as desired.

Step 3. Set the SPI SW RESET bit to 1 to release the SPI from the reset state.

Step 4. Write to SPIDAT or SPITXBUF (this initiates the communication process in the master).

Step 5. Read SPIRXBUF after the data transmission has completed (SPISTS.6 = 1) to determine what data was received.

To prevent unwanted and unforeseen events from occurring during or as a result of initialization changes, clear the SPI SW RESET bit (SPICCR.7) before making initialization changes, and then set this bit after initialization is complete.

NOTE: Do not change the SPI configuration when communication is in progress.

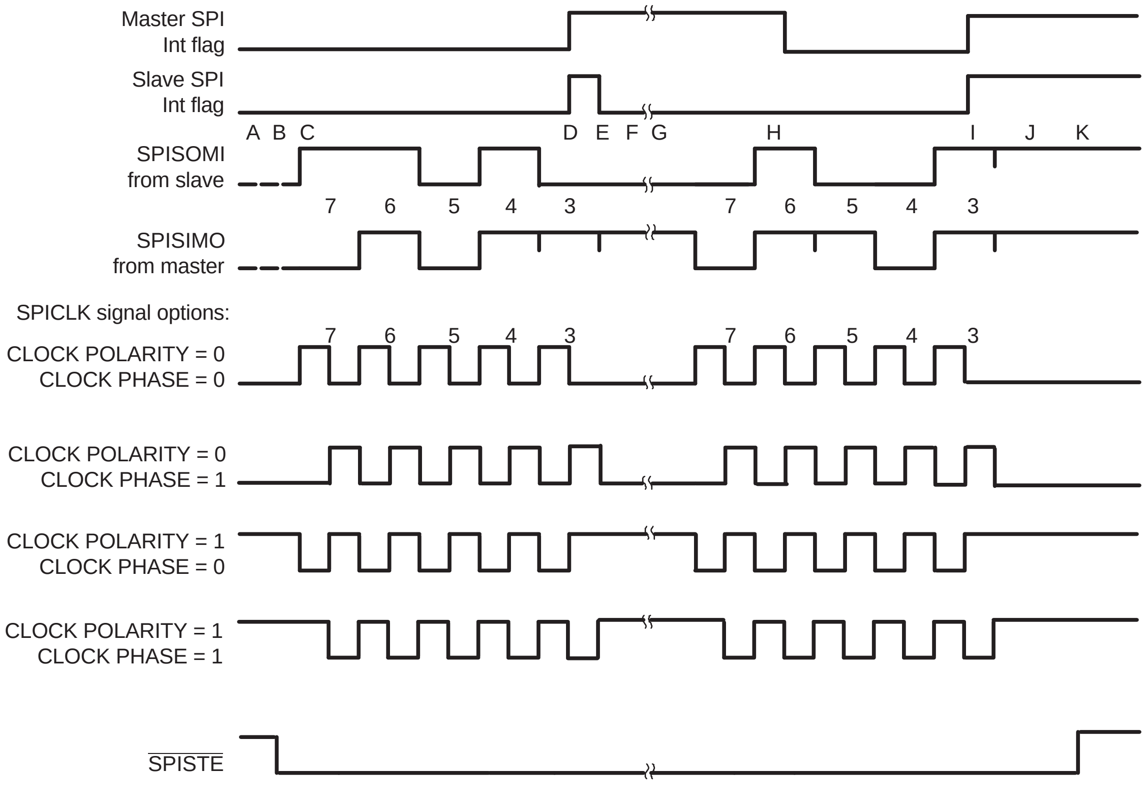
12.1.5.5 Data Transfer Example

The timing diagram shown in Figure 12-6 illustrates an SPI data transfer between two devices using a character length of five bits with the SPICLK being symmetrical.

The timing diagram with SPICLK unsymmetrical (Figure 12-5) shares similar characterizations with Figure 12-6 except that the data transfer is one CLKOUT cycle longer per bit during the low pulse (CLOCK POLARITY = 0) or during the high pulse (CLOCK POLARITY = 1) of the SPICLK.

Figure 12-6 is applicable for 8-bit SPI only and is not for 24x devices that are capable of working with 16-bit data. The figure is shown for illustrative purposes only.

Figure 12-6. Five Bits per Character



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's 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).

12.1.6 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, the FIFO function is disabled. The FIFO registers SPIFFTX, SPIFFRX and SPIFFCT remain inactive.
2. Standard SPI. The standard 240x SPI mode will work with SPIINT/SPIRXINT as the interrupt source.
3. Mode change. FIFO mode is enabled by setting the SPIFFEN 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 transmit FIFO, SPITXINT and one for receive FIFO, SPIINT/SPIRXINT. SPIINT/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.

6. Buffers. Transmit and receive buffers are supplemented with two FIFOs. The one-word transmit buffer (TXBUF) 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) FFTXDLY7−FFTXDLY0 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 serial clock cycles and a maximum of 255 serial clock 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 SPI clocks between each words. The programmable delay facilitates glueless interface to various slow SPI peripherals, such as EEPROMs, ADC, DAC, etc.
8. FIFO status bits. Both transmit and receive FIFOs have status bits TXFFST or RXFFST (bits 12− 0) 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 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 SPI. The default value for these trigger level bits will be 0x11111 for receive FIFO and 0x00000 for transmit FIFO respectively.

12.1.6.1 SPI Interrupts

Figure 12-7. SPI FIFO Interrupt Flags and Enable Logic Generation

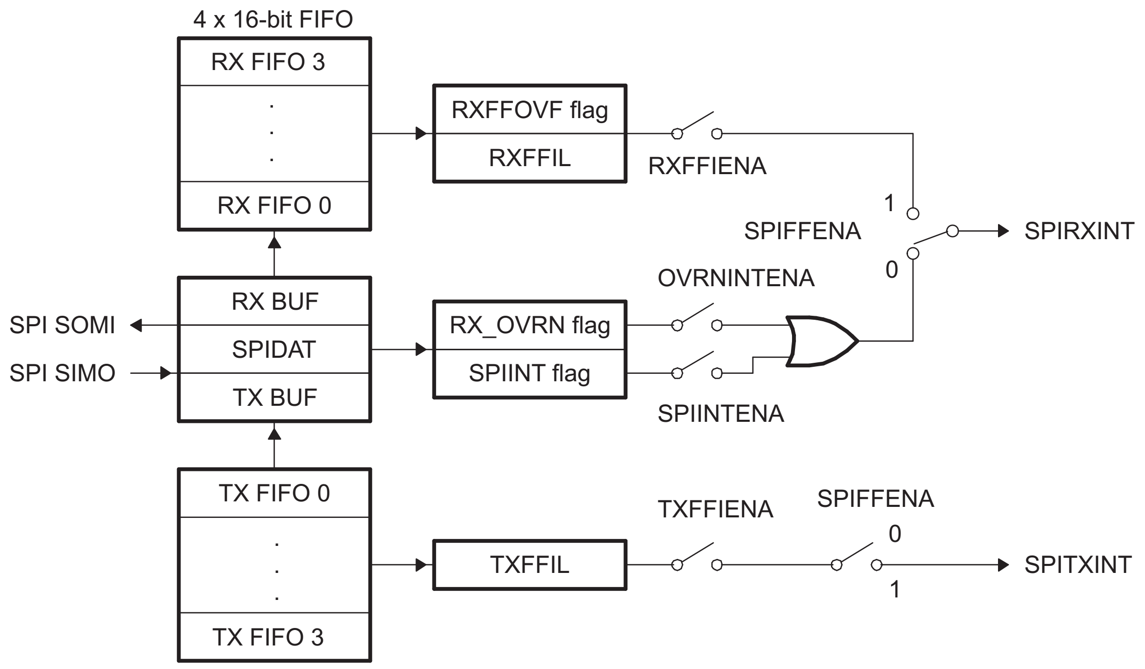


Table 12-4. SPI Interrupt Flag Modes

FIFO Options	SPI Interrupt Source	Interrupt Flags	Interrupt Enables	FIFO Enable SPIFFENA	Interrupt[(1)] line
SPI without FIFO					
	Receive overrun	RXOVRN	OVRNINTENA	0	SPIRXINT
	Data receive	SPIINT	SPIINTENA	0	SPIRXINT
	Transmit empty	SPIINT	SPIINTENA	0	SPIRXINT

(1) In non FIFO mode, SPIRXINT is the same as the SPIINT interrupt in 240x devices.

Table 12-4. SPI Interrupt Flag Modes (continued)

	SPI Interrupt	Interrupt	Interrupt	FIFO Enable	Interrupt[1]
SPI FIFO mode					
	FIFO receive	RXFFIL	RXFFIENA	1	SPIRXINT
	Transmit empty	TXFFIL	TXFFIENA	1	SPITXINT

12.1.7 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 (SPIPRI.0) bit is set, enabling 3-wire SPI mode, SPISIMOx 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 SPISIMOx is no longer used by the SPI.

The table below indicates the pin function differences between 3-wire and 4-wire SPI mode for a master and slave SPI.

Table 12-5. 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
SPISIMOx	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 (SPICTL.1) 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 12-8. SPI 3-wire Master Mode

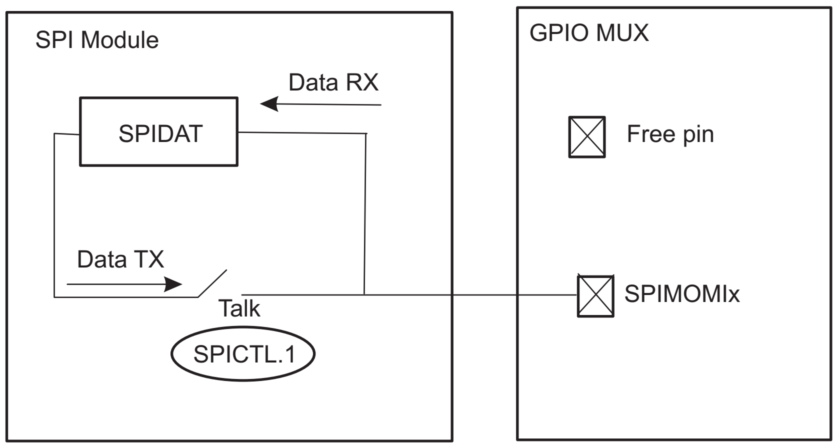


Figure 12-9. SPI 3-wire Slave Mode

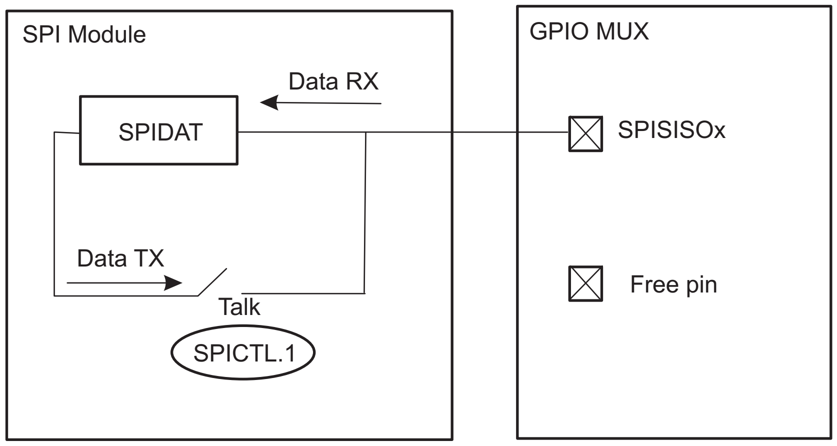


Table 12-6 indicates how data is received or transmitted in the various SPI modes while the TALK bit is set or cleared.

Table 12-6. 3-Wire SPI Pin Configuration

Pin Mode	SPIPRI[TRIWIRE]	SPICTL[TALK]	SPISIMO	SPISOMI
		Master Mode		
4-wire	0	X	TX	RX
3-pin mode	1	0	RX	Disconnect from SPI
		1	TX/RX	
		Slave Mode		
4-wire	0	X	RX	TX
3-pin mode	1	0	Disconnect from SPI	RX
		1		TX/RX

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 12-4. 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 12-5. 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 SPISIMOA 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 12-6. 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.

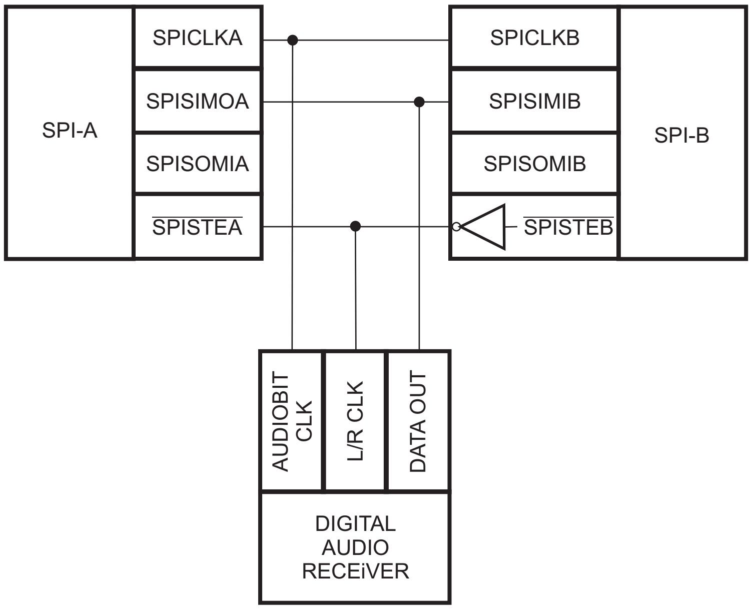
Example 12-7. - 3-Wire Slave Mode Receive

```
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
```

12.1.8 SPI STEINV Bit in Digital Audio Transfers

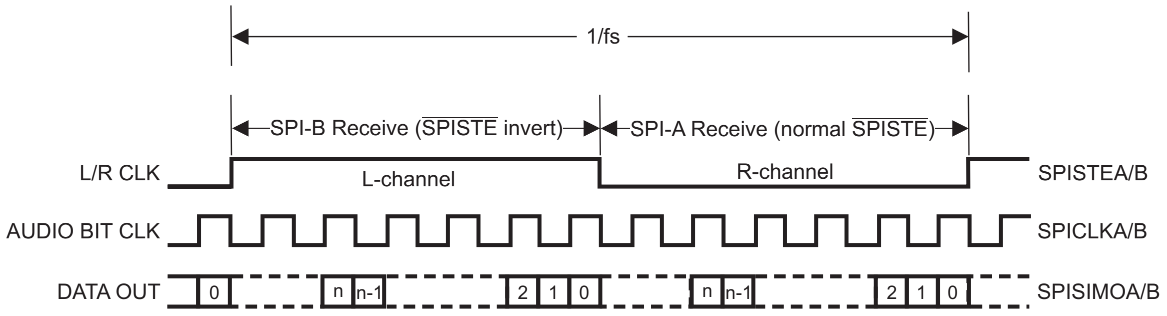
On those devices with two SPI modules, enabling the STEINV bit (SPIPRI.1) on one of the SPI modules allows the pair of SPIs to receive both left and right-channel digital audio data in slave mode. The SPI module that receives a normal active-low $\overline{\text{SPISTE}}$ signal stores right-channel data, and the SPI module that receives an inverted active-high $\overline{\text{SPISTE}}$ signal stores left-channel data from the master. To receive digital audio data from a digital audio interface receiver, the SPI modules can be connected as shown in Figure 12-10.

Figure 12-10. SPI Digital Audio Receiver Configuration Using 2 SPIs



Standard 28x SPI timing requirements limit the number of digital audio interface formats supported using the 2-SPI configuration with the STEINV bit. See your device-specific data sheet electricals for SPI timing requirements. With the SPI clock phase configured such that the CLOCK POLARITY (SPICCR.6) bit is 0 and the CLOCK PHASE (SPICTL.3) bit is 1 (data latched on rising edge of clock), standard right-justified digital audio interface data format is supported as shown in Figure 12-11.

Figure 12-11. Standard Right-Justified Digital Audio Data Format


1/fs
SPI-B Receive ($\overline{\text{SPISTE}}$ invert)
SPI-A Receive (normal $\overline{\text{SPISTE}}$)
L/R CLK
L-channel
R-channel
SPISTEA/B
AUDIO BIT CLK
SPICLKA/B
DATA OUT
0
n
n-1
2
1
0
n
n-1
2
1
0
SPISIMOA/B

12.2 SPI Registers and Waveforms

This section contains the registers, bit descriptions, and waveforms.

12.2.1 SPI Example Waveforms

Figure 12-12. CLOCK POLARITY = 0, CLOCK PHASE = 0 (All data transitions are during the rising edge, non-delayed clock. Inactive level is low.)

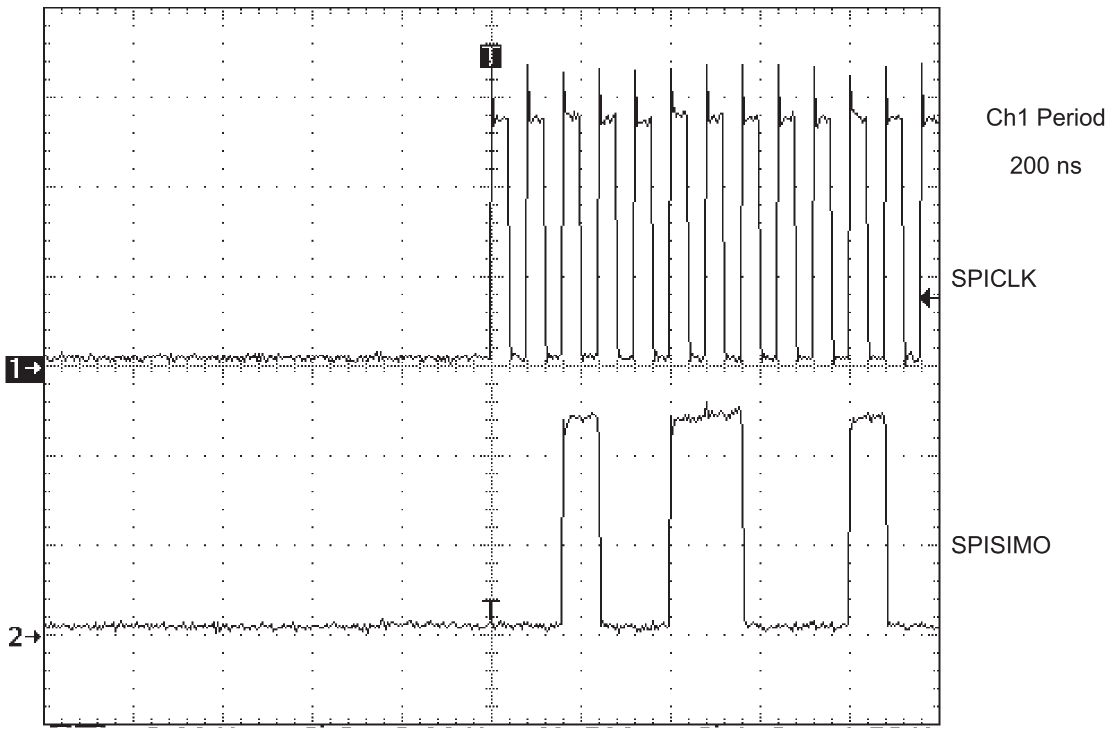


Figure 12-13. CLOCK POLARITY = 0, CLOCK PHASE = 1 (All data transitions are during the rising edge, but delayed by half clock cycle. Inactive level is low.)

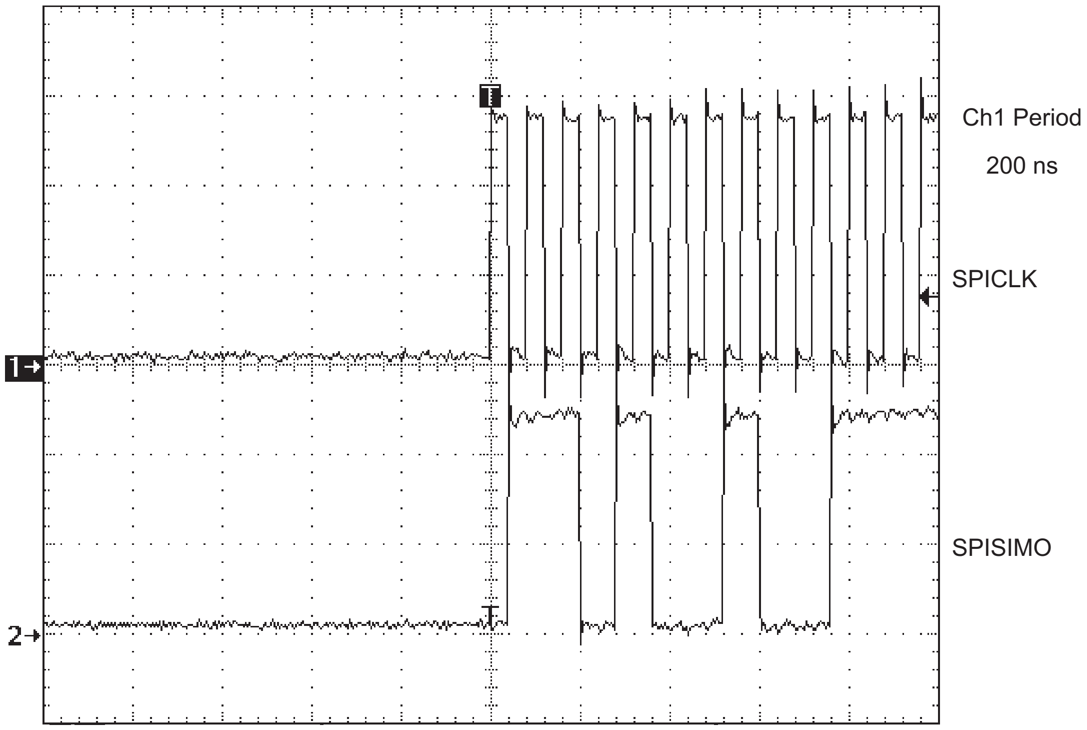


Figure 12-14. CLOCK POLARITY = 1, CLOCK PHASE = 0 (All data transitions are during the falling edge. Inactive level is high.)

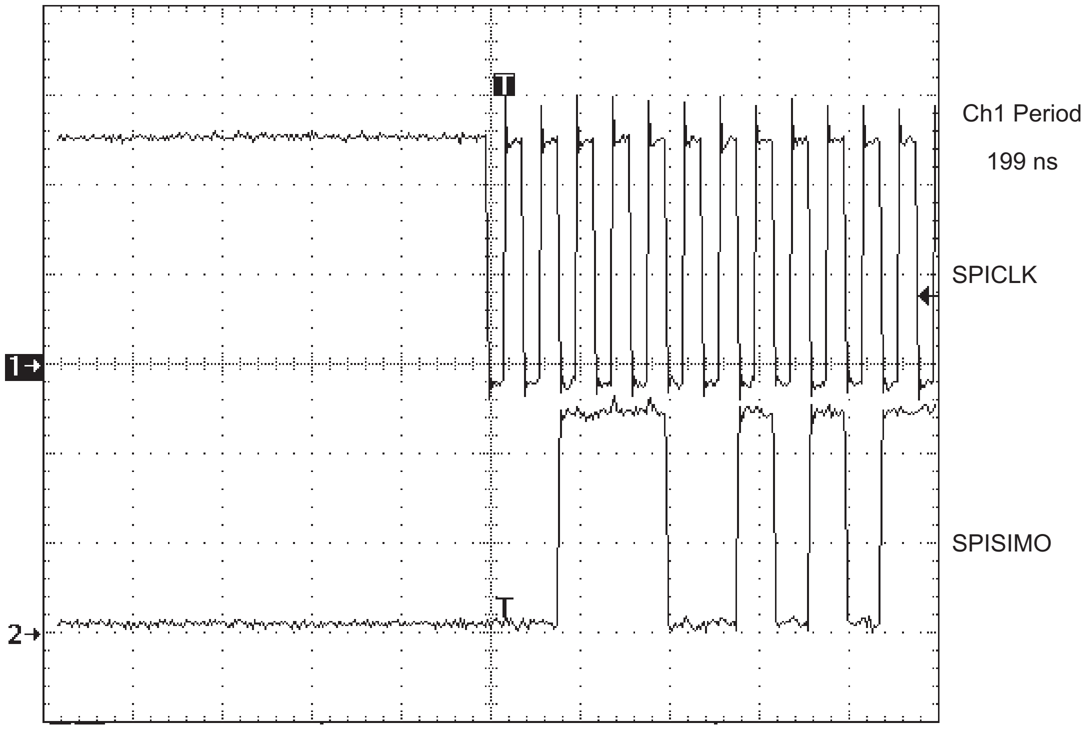


Figure 12-15. CLOCK POLARITY = 1, CLOCK PHASE = 1 (All data transitions are during the falling edge, but delayed by half clock cycle. Inactive level is high.)

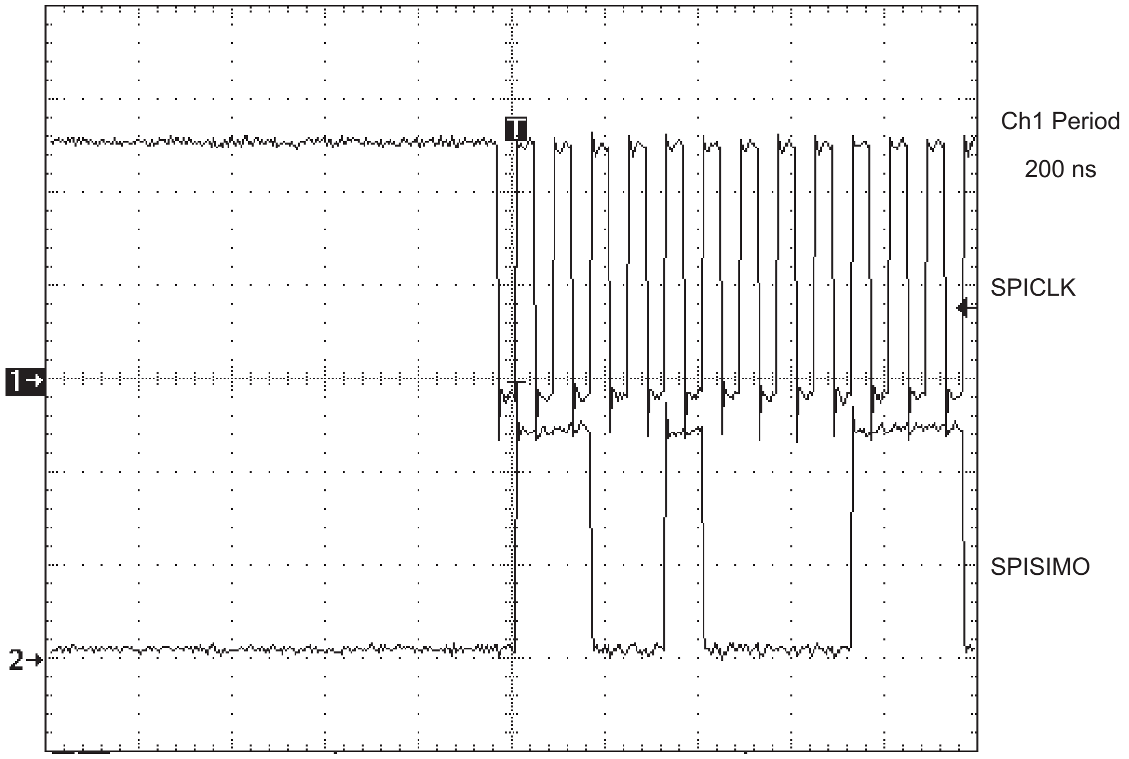


Figure 12-16. $\overline{\text{SPISTE}}$ Behavior in Master Mode (Master lowers $\overline{\text{SPISTE}}$ during the entire 16 bits of transmission.)

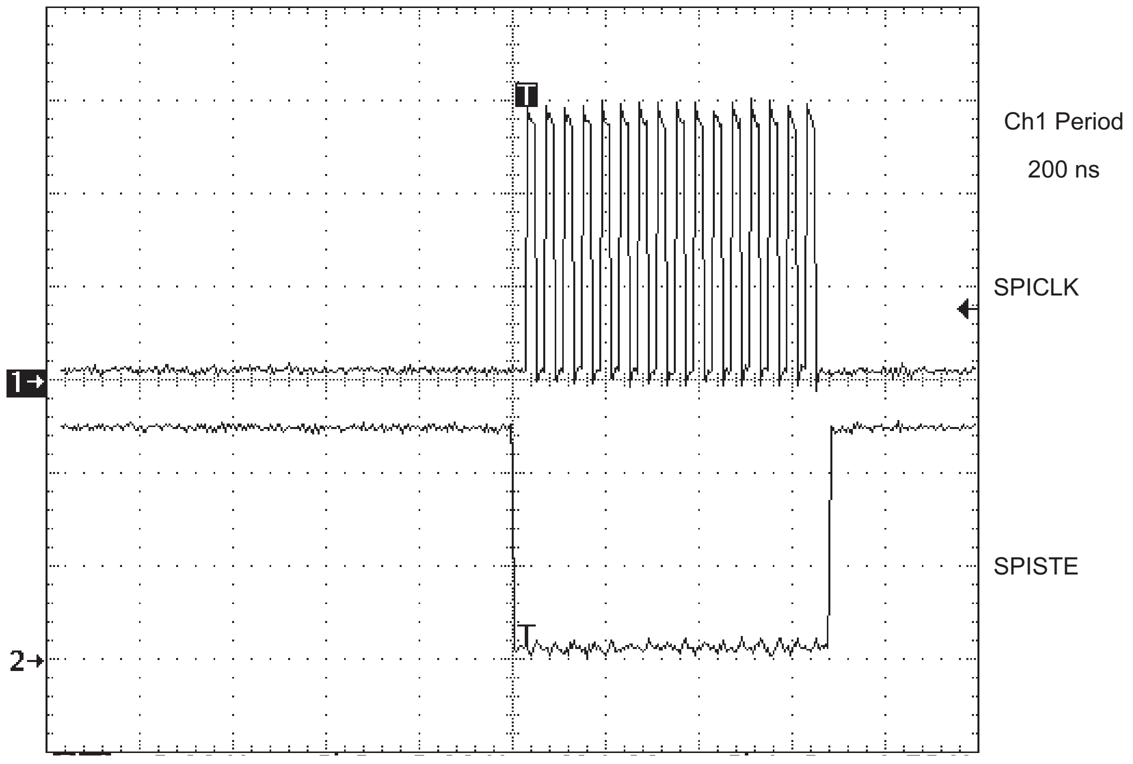
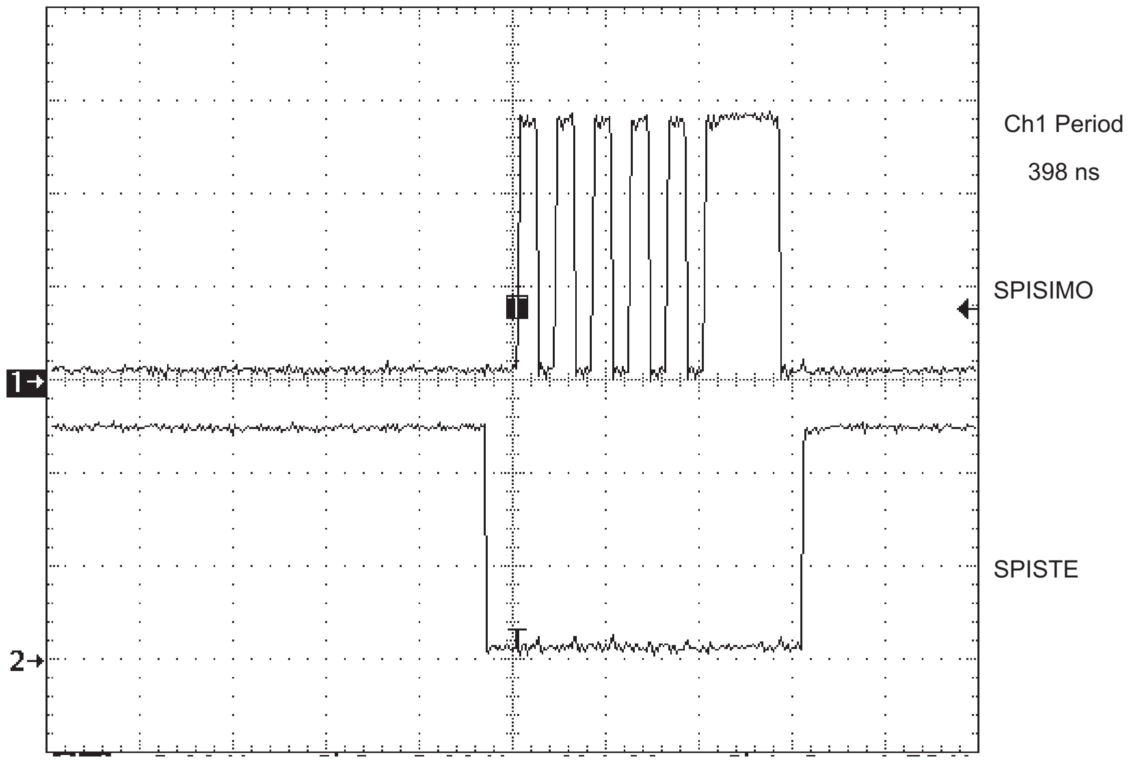


Figure 12-17. $\overline{\text{SPISTE}}$ Behavior in Slave Mode (Slave's $\overline{\text{SPISTE}}$ is lowered during the entire 16 bits of transmission.)



12.2.2 *SPI Control Registers*

The SPI is controlled and accessed through registers in the control register file.

12.2.2.1 SPI Configuration Control Register (SPICCR)

SPICCR controls the setup of the SPI for operation.

Figure 12-18. SPI Configuration Control Register (SPICCR) — Address 7040h

7	6	5	4	3	2	1	0
SPI SW Reset	CLOCK POLARITY	Reserved	SPILBK	SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0
R/W-0	R/W-0	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 12-7. SPI Configuration Control Register (SPICCR) Field Descriptions

Bit	Field	Value	Description
7	SPI SW RESET		SPI software reset. When changing configuration, you should clear this bit before the changes and set this bit before resuming operation.
		0	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. The SPI configuration remains unchanged.
		1	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.
6	CLOCK POLARITY		Shift Clock Polarity. This bit controls the polarity of the SPICLK signal. CLOCK POLARITY and CLOCK PHASE (SPICTL.3) control four clocking schemes on the SPICLK pin. See Section 12.1.5.3.
		0	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: • 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. • 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.
		1	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: • 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. • 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.
5	Reserved		Reads return zero; writes have no effect.
4	SPILBK		SPI loopback. Loop back mode allows module validation during device testing. This mode is valid only in master mode of the SPI.
		0	SPI loop back mode disabled – default value after reset
		1	SPI loop back mode enabled, SIMO/SOMI lines are connected internally. Used for module self tests.
3-0	SPI CHAR3 − SPI CHAR0		Character Length Control Bits 3-0. These four bits determine the number of bits to be shifted in or out as a single character during one shift sequence. Table 12-8 lists the character length selected by the bit values.

Table 12-8. Character Length Control Bit Values

SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0	Character Length
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11
1	0	1	1	12
1	1	0	0	13
1	1	0	1	14
1	1	1	0	15

Table 12-8. Character Length Control Bit Values (continued)

SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0	Character Length
1	1	1	1	16

12.2.2.2 SPI Operation Control Register (SPICTL)

SPICTL controls data transmission, the SPI's ability to generate interrupts, the SPICLK phase, and the operational mode (slave or master).

Figure 12-19. SPI Operation Control Register (SPICTL) — Address 7041h

7	6	5	4	3	2	1	0
Reserved			OVERRUN INT ENA	CLOCK PHASE	MASTER/ SLAVE	TALK	SPI INT ENA
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 12-9. SPI Operation Control Register (SPICTL) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved		Reads return zero; writes have no effect.
4	Overrun INT ENA		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.
		0	Disable RECEIVER OVERRUN Flag bit (SPISTS.7) interrupts
		1	Enable RECEIVER OVERRUN Flag bit (SPISTS.7) interrupts
3	CLOCK PHASE		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 Figure 12-4). 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.
		0	Normal SPI clocking scheme, depending on the CLOCK POLARITY bit (SPICCR.6)
		1	SPICLK signal delayed by one half-cycle; polarity determined by the CLOCK POLARITY bit
2	MASTER / SLAVE		SPI Network Mode Control. This bit determines whether the SPI is a network master or slave. During reset initialization, the SPI is automatically configured as a network slave.
		0	SPI configured as a slave.
		1	SPI configured as a master.
1	TALK		Master/Slave Transmit Enable. 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.
		0	Disables transmission: • 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. • 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.
		1	Enables transmission For the 4-pin option, ensure to enable the receiver's $\overline{\text{SPISTE}}$ input pin.
0	SPI INT ENA		SPI Interrupt Enable. 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.
		0	Disables interrupt
		1	Enables interrupt

12.2.2.3 SPI Status Register (SPIST)

Figure 12-20. SPI Status Register (SPIST) — Address 7042h

7	6	5	4-0
RECEIVER OVERRUN FLAG[1] [2]	SPI INT FLAG[1] [2]	TX BUF FULL FLAG[2]	Reserved
R/C-0	R/C-0	R/C-0	R-0

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

[1] The RECEIVER OVERRUN FLAG bit and the SPI INT FLAG bit share the same interrupt vector.

[2] Writing a 0 to bits 5, 6, and 7 has no effect.

Table 12-10. SPI Status Register (SPIST) Field Descriptions

Bit	Field	Value	Description
7	RECEIVER OVERRUN FLAG		SPI Receiver Overrun Flag. 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 indicates that 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). The SPI requests one interrupt sequence each time this bit is set if the OVERRUN INT ENA bit (SPICTL.4) is set high. The bit is cleared in one of three ways: • Writing a 1 to this bit • Writing a 0 to SPI SW RESET (SPICCR.7) • Resetting the system 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.
		0	Writing a 0 has no effect
		1	Clears 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.
6	SPI INT FLAG		SPI Interrupt Flag. SPI INT FLAG is a read-only flag. The SPI hardware sets this bit to indicate that it has completed sending or receiving the last bit and is ready to be serviced. The received character is placed in the receiver buffer at the same time this bit is set. This flag causes an interrupt to be requested if the SPI INT ENA bit (SPICTL.0) is set.
		0	Writing a 0 has no effect
		1	This bit is cleared in one of three ways: • Reading SPIRXBUF • Writing a 0 to SPI SW RESET (SPICCR.7) • Resetting the system
5	TX BUF FULL FLAG		SPI Transmit Buffer Full Flag. 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.
		0	Writing a 0 has no effect
		1	This bit is cleared at reset.
4-0	Reserved	0	Reads return zero; writes have no effect.

12.2.2.4 SPI Baud Rate Register (SPIBRR)

SPIBRR contains the bits used for baud-rate selection.

Figure 12-21. SPI Baud Rate Register (SPIBRR) — Address 7044h

7	6	5	4	3	2	1	0
Reserved	SPI BIT RATE 6	SPI BIT RATE 5	SPI BIT RATE 4	SPI BIT RATE 3	SPI BIT RATE 2	SPI BIT RATE 1	SPI BIT RATE 0
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

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

Table 12-11. Field Descriptions

Bit	Field	Value	Description
7	Reserved		Reads return zero; writes have no effect.
6-0	SPI BIT RATE 6− SPI BIT RATE 0		SPI Bit Rate (Baud) Control. These bits determine the bit transfer rate if the SPI is the network 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.) 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 SPICLK signal divided by 4. 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: For SPIBRR = 3 to 127: $\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{(\text{SPIBRR} + 1)}$ For SPIBRR = 0, 1, or 2: $\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{4}$ where: LSPCLK = Function of CPU clock frequency X low-speed peripheral clock of the device SPIBRR = Contents of the SPIBRR in the master SPI device

12.2.2.5 SPI Emulation Buffer Register (SPIRXEMU)

SPIRXEMU contains the received data. Reading SPIRXEMU does not clear the SPI INT FLAG bit (SPISTS.6). This is not a real register but a dummy address from which the contents of SPIRXBUF can be read by the emulator without clearing the SPI INT FLAG.

Figure 12-22. SPI Emulation Buffer Register (SPIRXEMU) — Address 7046h

15	14	13	12	11	10	9	8
ERXB15	ERXB14	ERXB13	ERXB12	ERXB11	ERXB10	ERXB9	ERXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
ERXB7	ERXB6	ERXB5	ERXB4	ERXB3	ERXB2	ERXB1	ERXB0
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 12-12. SPI Emulation Buffer Register (SPIRXEMU) Field Descriptions

Bit	Field	Value	Description
15-0	ERXB15− ERXB0		Emulation Buffer Received Data. 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. This mirror register was created to support emulation. Reading SPIRXBUF clears the SPI INT FLAG bit (SPISTS.6). In the normal operation of the emulator, the control registers are read to continually update the contents of these registers on the display screen. SPIRXEMU was created so that the emulator 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 emulator to emulate the true operation of the SPI more accurately. It is recommended that you view SPIRXEMU in the normal emulator run mode.

12.2.2.6 SPI Serial Receive Buffer Register (SPIRXBUF)

SPIRXBUF contains the received data. Reading SPIRXBUF clears the SPI INT FLAG bit (SPISTS.6).

Figure 12-23. SPI Serial Receive Buffer Register (SPIRXBUF) — Address 7047h

15	14	13	12	11	10	9	8
RXB15	RXB14	RXB13	RXB12	RXB11	RXB10	RXB9	RXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
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 12-13. SPI Serial Receive Buffer Register (SPIRXBUF) Field Descriptions

Bit	Field	Value	Description
15-0	RXB15 − RXB0		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.

12.2.2.7 SPI Serial Transmit Buffer Register (SPITXBUF)

SPITXBUF stores the next character to be transmitted. Writing to this register sets the TX BUF FULL Flag bit (SPISTS.5). When 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 12-24. SPI Serial Transmit Buffer Register (SPITXBUF) — Address 7048h

15	14	13	12	11	10	9	8
TXB15	TXB14	TXB13	TXB12	TXB11	TXB10	TXB9	TXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
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 12-14. SPI Serial Transmit Buffer Register (SPITXBUF) Field Descriptions

Bit	Field	Value	Description
15-0	TXB15 − TXB0		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.

12.2.2.8 SPI Serial Data Register (SPIDAT)

SPIDAT is the transmit/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 12-25. SPI Serial Data Register (SPIDAT) — Address 7049h

15	14	13	12	11	10	9	8
SDAT15	SDAT14	SDAT13	SDAT12	SDAT11	SDAT10	SDAT9	SDAT8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
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 12-15. SPI Serial Data Register (SPIDAT) Field Descriptions

Bit	Field	Value	Description
15-0	SDAT15 − SDAT0		Serial data. Writing to the SPIDAT performs two functions: • 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, see the CLOCK POLARITY bit (SPICCR.6) described in Section 12.2.2.1 and the CLOCK PHASE bit (SPICTL.3) described in Section 12.2.2.2, 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.

12.2.2.9 SPI FIFO Transmit, Receive, and Control Registers

Figure 12-26. SPI FIFO Transmit (SPIFFTX) Register − Address 704Ah

15	14	13	12	11	10	9	8
SPIRST	SPIFFENA	TXFIFO	TXFFST4	TXFFST3	TXFFST2	TXFFST1	TXFFST0
R/W-1	R/W−0	R/W-1	R−0	R−0	R−0	R−0	R−0

7	6	5	4	3	2	1	0
TXFFINT Flag	TXFFINT CLR	TXFFIENA	TXFFIL4	TXFFIL3	TXFFIL2	TXFFIL1	TXFFIL0
R/W-0	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 12-16. SPI FIFO Transmit (SPIFFTX) Register Field Descriptions

Bit	Field	Value	Description
15	SPIRST		SPI reset
		0	Write 0 to reset the SPI transmit and receive channels. The SPI FIFO register configuration bits will be left as is.
		1	SPI FIFO can resume transmit or receive. No effect to the SPI registers bits.
14	SPIFFENA		SPI FIFO enhancements enable
		0	SPI FIFO enhancements are disabled
		1	SPI FIFO enhancements are enabled
13	TXFIFO Reset		Transmit FIFO reset
		0	Write 0 to reset the FIFO pointer to zero, and hold in reset.
		1	Re-enable Transmit FIFO operation
12-8	TXFFST4−0		Transmit FIFO status
		00000	Transmit FIFO is empty.
		00001	Transmit FIFO has 1 word.
		00010	Transmit FIFO has 2 words.
		00011	Transmit FIFO has 3 words.
		00100	Transmit FIFO has 4 words, which is the maximum.
7	TXFFINT		TXFIFO interrupt
		0	TXFIFO interrupt has not occurred, This is a read-only bit.
		1	TXFIFO interrupt has occurred, This is a read-only bit.
6	TXFFINT CLR		TXFIFO clear
		0	Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero.
		1	Write 1 to clear TXFFINT flag in bit 7.
5	TXFFIENA		TX FIFO interrupt enable
		0	TX FIFO interrupt based on TXFFIVL match (less than or equal to) will be disabled .
		1	TX FIFO interrupt based on TXFFIVL match (less than or equal to) will be enabled.
4-0	TXFFIL4−0		TXFFIL4−0 transmit FIFO interrupt level bits. Transmit FIFO will generate interrupt when the FIFO status bits (TXFFST4−0) and FIFO level bits (TXFFIL4−0) match (less than or equal to).
		00000	Default value is 0x00000.

Figure 12-27. SPI FIFO Receive (SPIFFRX) Register − Address 704Bh

15	14	13	12	11	10	9	8
RXFFOVF Flag	RXFFOVF CLR	RXFIFO Reset	RXFFST4	RXFFST3	RXFFST2	RXFFST1	RXFFST0
R-0	W−0	R/W−1	R−0	R−0	R−0	R−0	R−0

7	6	5	4	3	2	1	0
RXFFINT Flag	RXFFINT CLR	RXFFIENA	RXFFIL4	RXFFIL3	RXFFIL2	RXFFIL1	RXFFIL0
R-0	W−0	R/W−0	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 12-17. SPI FIFO Receive (SPIFFRX) Register Field Descriptions

Bit	Field	Value	Description
15	RXFFOVF		Receive FIFO overflow flag
		0	Receive FIFO has not overflowed. This is a read-only bit.
		1	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	RXFFOVF CLR		Receive FIFO overflow clear
		0	Write 0 does not affect RXFFOVF flag bit, Bit reads back a zero
		1	Write 1 to clear RXFFOVF flag in bit 15
13	RXFIFO Reset		Receive FIFO reset
		0	Write 0 to reset the FIFO pointer to zero, and hold in reset.
		1	Re-enable receive FIFO operation
12-8	RXFFST4−0		Receive FIFO Status
		00000	Receive FIFO is empty.
		00001	Receive FIFO has 1 word.
		00010	Receive FIFO has 2 words.
		00011	Receive FIFO has 3 words.
		00100	Receive FIFO has 4 words. Receive FIFO has a maximum of 4 words.
7	RXFFINT		Receive FIFO interrupt
		0	RXFIFO interrupt has not occurred. This is a read-only bit.
		1	RXFIFO interrupt has occurred. This is a read-only bit.
6	RXFFINT CLR		Receive FIFO interrupt clear
		0	Write 0 has no effect on RXFIFINT flag bit, Bit reads back a zero.
		1	Write 1 to clear RXFFINT flag in bit 7.
5	RXFFIENA		RX FIFO interrupt enable
		0	RX FIFO interrupt based on RXFFIL match (greater than or equal to) will be disabled.
		1	RX FIFO interrupt based on RXFFIL match (greater than or equal to) will be enabled.
4-0	RXFFIL4−0		Receive FIFO interrupt level bits
		11111	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. This avoids frequent interrupts after reset, as the receive FIFO will be empty most of the time.

Figure 12-28. SPI FIFO Control (SPIFFCT) Register − Address 704Ch

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
FFTXDLY7	FFTXDLY6	FFTXDLY5	FFTXDLY4	FFTXDLY3	FFTXDLY2	FFTXDLY1	FFTXDLY0
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 12-18. SPI FIFO Control (SPIFFCT) Register Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		Reads return zero; writes have no effect.
7-0	FFTXDLY7−0		FIFO transmit delay bits
		0	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.
		1	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.

12.2.2.10 SPI Priority Control Register (SPIPRI)

Figure 12-29. SPI Priority Control Register (SPIPRI) — Address 704Fh

15–8
Reserved
R-0

7–6	5	4	3–2	1	0
Reserved	SPI SUSP SOFT	SPI SUSP FREE	Reserved	STEINV	TRIWIRE
R-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

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

Table 12-19. SPI Priority Control Register (SPIPRI) Field Descriptions

Bit	Field	Value	Description
15-6	Reserved		Reads return zero; writes have no effect.
5-4	SPI SUSP SOFT SPI SUSP FREE		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.
		0 0	Transmission stops after 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. The SCI module operates differently.
		1 0	If the emulation suspend occurs before the start of a transmission, (i.e., 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. *Standard SPI mode:* Stop after transmitting the words in the shift register and buffer. That is, after TXBUF and SPIDAT are empty. *In FIFO mode:* Stop after transmitting the words in the shift register and buffer. That is, after TX FIFO and SPIDAT are empty.
		x 1	Free run, continue SPI operation regardless of suspend or when the suspend occurred.
3-2	Reserved		
1	STEINV		$\overline{\text{SPISTE}}$ inversion bit. On devices with 2 SPI modules, inverting the $\overline{\text{SPISTE}}$ signal on one of the modules allows the device to receive left and right- channel digital audio data.
		0	$\overline{\text{SPISTE}}$ is active low (normal)
		1	$\overline{\text{SPISTE}}$ is active high (inverted)
0	TRIWIRE		SPI 3-wire mode enable
		0	Normal 4-wire SPI mode
		1	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 SIISOMI pin becomes the SPISISO (slave receive and transmit) pin and SPISIMO is free for non-SPI use.

Serial Communications Interface (SCI)

The serial communications interface (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.

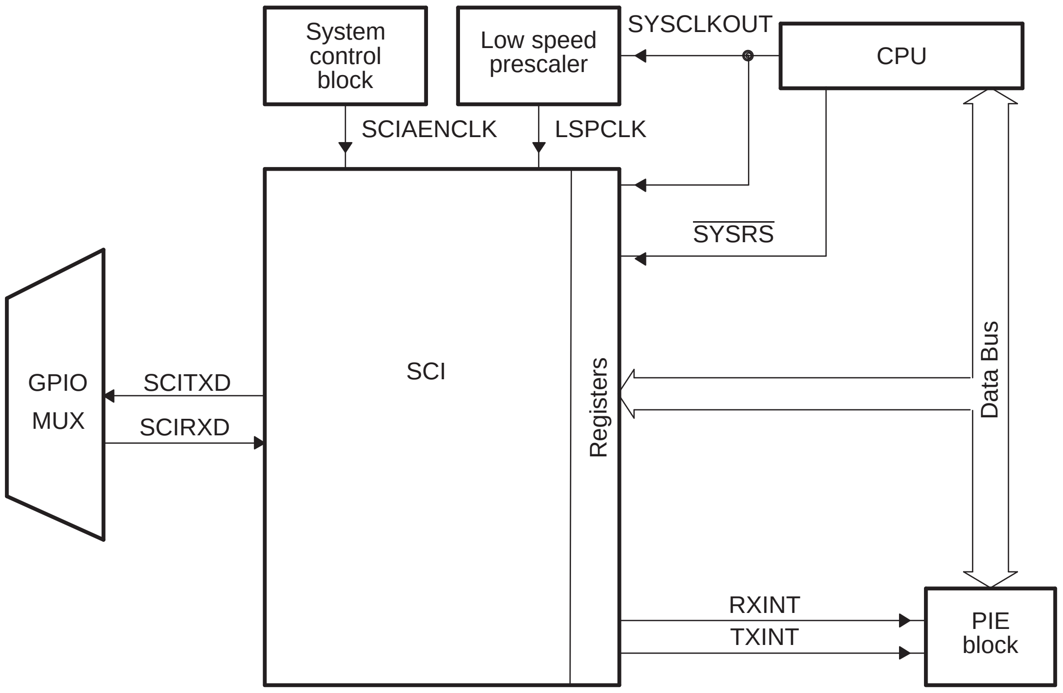
NOTE: The 28x SCI features several enhancements compared to the 240xA SCI. See Section 13.1.1.10 for a description of these features.

Topic **Page**

13.1 Enhanced SCI Module Overview

The SCI interfaces are shown in Figure 13-1.

Figure 13-1. SCI CPU Interface



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
- 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
- 13 SCI module control registers located in the control register frame beginning at address 7050h

  All registers in this module are 8-bit registers that are connected to Peripheral Frame 2. When a register is accessed, the register data is in the lower byte (7−0), and the upper byte (15−8) is read as zeros. Writing to the upper byte has no effect.

Enhanced features:

- Auto-baud-detect hardware logic
- 4-level transmit/receive FIFO

Figure 13-2 shows the SCI module block diagram. The SCI port operation is configured and controlled by the registers listed in Table 13-1 and Table 13-2.

Figure 13-2. Serial Communications Interface (SCI) Module Block Diagram

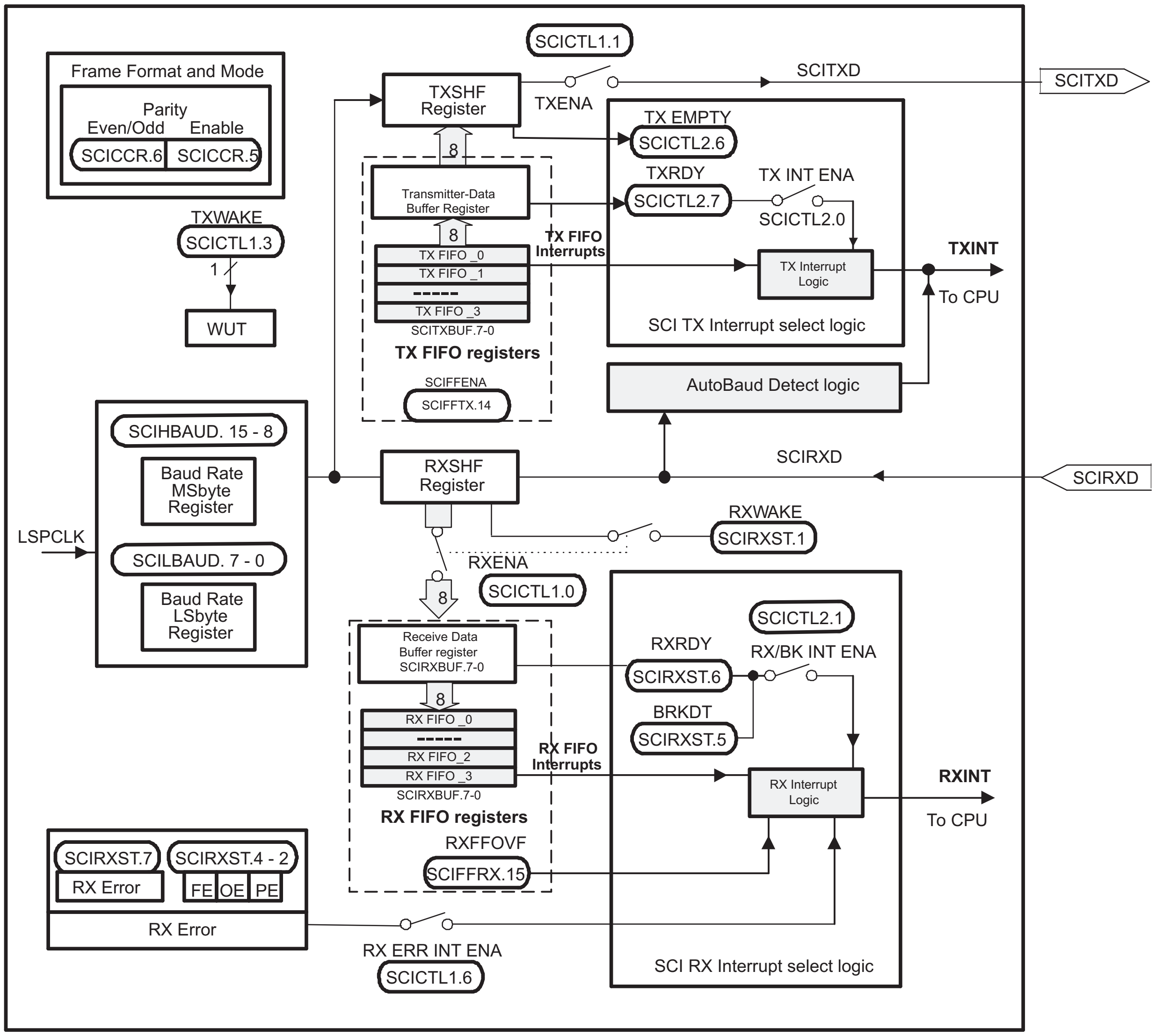


Table 13-1. SCI-A Registers

Name	Address Range	Size (x16)	Description
SCICCR	0x0000-7050	1	SCI-A Communications Control Register
SCICTL1	0x0000-7051	1	SCI-A Control Register 1
SCIHBAUD	0x0000-7052	1	SCI-A Baud Register, High Bits
SCILBAUD	0x0000-7053	1	SCI-A Baud Register, Low Bits
SCICTL2	0x0000-7054	1	SCI-A Control Register 2
SCIRXST	0x0000-7055	1	SCI-A Receive Status Register
SCIRXEMU	0x0000-7056	1	SCI-A Receive Emulation Data Buffer Register
SCIRXBUF	0x0000-7057	1	SCI-A Receive Data Buffer Register
SCITXBUF	0x0000-7059	1	SCI-A Transmit Data Buffer Register
SCIFFTX	0x0000-705A	1	SCI-A FIFO Transmit Register
SCIFFRX	0x0000-705B	1	SCI-A FIFO Receive Register
SCIFFCT	0x0000-705C	1	SCI-A FIFO Control Register
SCIPRI	0x0000-705F	1	SCI-A Priority Control Register

Table 13-2. SCI-B Registers

Name	Address Range	Size (x16)	Description[1] [2]
SCICCR	0x0000-7750	1	SCI-B Communications Control Register
SCICTL1	0x0000-7751	1	SCI-B Control Register 1
SCIHBAUD	0x0000-7752	1	SCI-B Baud Register, High Bits
SCILBAUD	0x0000-7753	1	SCI-B Baud Register, Low Bits
SCICTL2	0x0000-7754	1	SCI-B Control Register 2
SCIRXST	0x0000-7755	1	SCI-B Receive Status Register
SCIRXEMU	0x0000-7756	1	SCI-B Receive Emulation Data Buffer Register
SCIRXBUF	0x0000-7757	1	SCI-B Receive Data Buffer Register
SCITXBUF	0x0000-7759	1	SCI-B Transmit Data Buffer Register
SCIFFTX	0x0000-775A	1	SCI-B FIFO Transmit Register
SCIFFRX	0x0000-775B	1	SCI-B FIFO Receive Register
SCIFFCT	0x0000-775C	1	SCI-B FIFO Control Register
SCIPRI	0x0000-775F	1	SCI-B Priority Control Register

[1] The registers are mapped to peripheral frame 2. This frame allows only 16-bit accesses. Using 32-bit accesses will produce undefined results.

[2] SCIB is an optional peripheral. In some devices this may not be present. See the device-specific data sheet for peripheral availability.

13.1.1 Architecture

The major elements used in full-duplex operation are shown in Figure 13-2 and include:

- A transmitter (TX) and its major registers (upper half of Figure 13-2)
  - 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 13-2)
  - 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
- Data-memory-mapped control and status registers

The SCI receiver and transmitter can operate either independently or simultaneously.

13.1.1.1 SCI Module Signal Summary

Table 13-3. SCI Module Signal Summary

Signal Name	Description
	External signals
SCIRXD	SCI Asynchronous Serial Port receive data
SCITXD	SCI Asynchronous Serial Port transmit data
	Control
Baud clock	LSPCLK Prescaled clock
	Interrupt signals
TXINT	Transmit interrupt
RXINT	Receive Interrupt

13.1.1.2 Multiprocessor and Asynchronous Communication Modes

The SCI has two multiprocessor protocols, the idle-line multiprocessor mode (see Section 13.1.1.5) and the address-bit multiprocessor mode (see Section 13.1.1.6). 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 13.1.1.7) 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

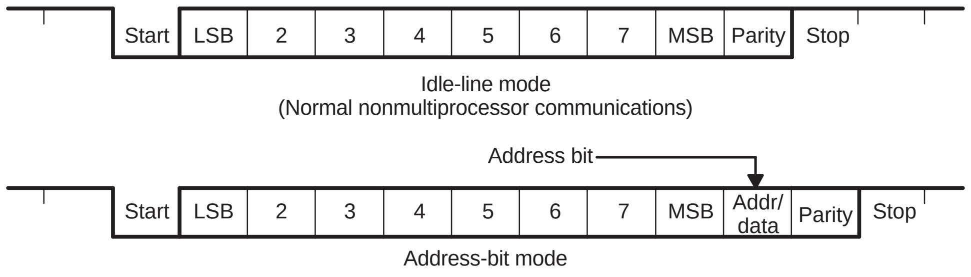
13.1.1.3 SCI Programmable Data Format

SCI data, both receive and transmit, is in NRZ (non-return-to-zero) format. The NRZ data format, shown in Figure 13-3, 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 13-3.

Figure 13-3. 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 13-4.

Table 13-4. Programming the Data Format Using SCICCR

Bit(s)	Bit Name	Designation	Functions
2-0	SCI CHAR2-0	SCICCR.2:0	Select the character (data) length (one to eight bits).
5	PARITY ENABLE	SCICCR.5	Enables the parity function if set to 1, or disables the parity function if cleared to 0.
6	EVEN/ODD PARITY	SCICCR.6	If parity is enabled, selects odd parity if cleared to 0 or even parity if set to 1.
7	STOP BITS	SCICCR.7	Determines the number of stop bits transmitted—one stop bit if cleared to 0 or two stop bits if set to 1.

13.1.1.4 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.

13.1.1.4.1 Recognizing the Address Byte

A processor recognizes an address byte differently, depending on the multiprocessor mode used. For example:

- The idle-line mode (Section 13.1.1.5) 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 13.1.1.6) 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.

13.1.1.4.2 *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.

13.1.1.4.3 *Receipt Sequence*

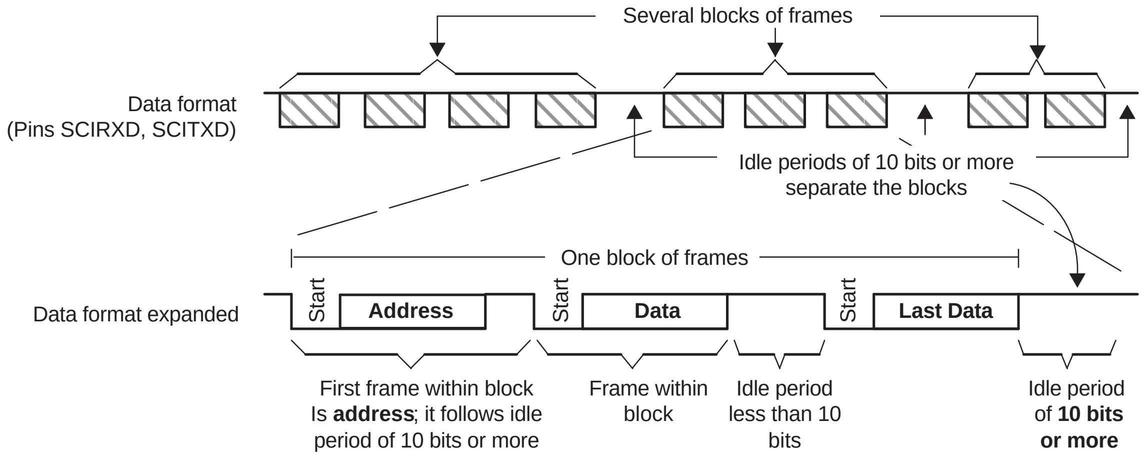
In both multiprocessor modes, the receive sequence is:

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.

13.1.1.5 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 13-4 (ADDR/IDLE MODE bit is bit 3 of SCICCR).

Figure 13-4. Idle-Line Multiprocessor Communication Format



13.1.1.5.1 *Idle-Line Mode Steps*

The steps followed by the idle-line mode:

Step 1. SCI wakes up after receipt of the block-start signal.

Step 2. The processor recognizes the next SCI interrupt.

Step 3. The interrupt service routine compares the received address (sent by a remote transmitter) to its own.

Step 4. If the CPU is being addressed, the service routine clears the SLEEP bit and receives the rest of the data block.

Step 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.

13.1.1.5.2 Block Start Signal

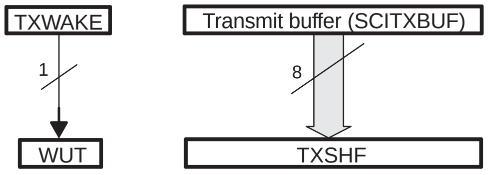
There are two ways to send a block-start signal:

1. 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.
2. 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.)

13.1.1.5.3 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 13-5.

Figure 13-5. Double-Buffered WUT and TXSHF



A WUT = wake-up temporary

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.

13.1.1.5.4 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.

13.1.1.6 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 13-6).

13.1.1.6.1 Sending an Address

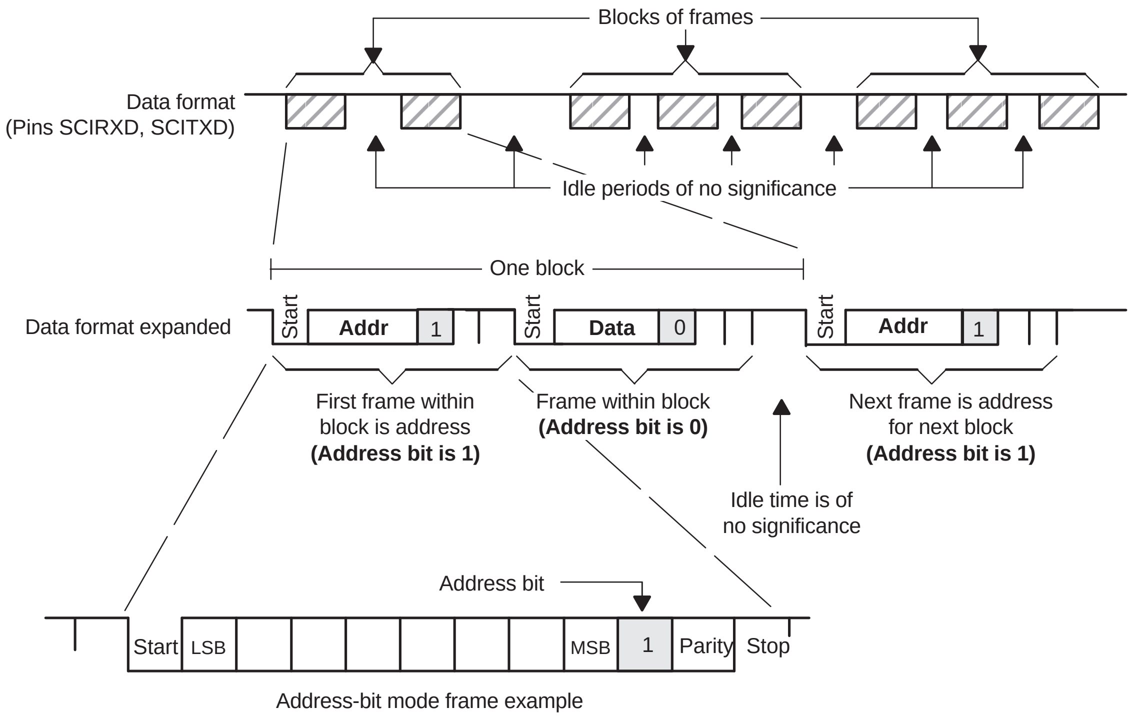
The TXWAKE bit value is placed in the address bit. During transmission, when the SCITXBUF register and TXWAKE are loaded into the TXSHF register and WUT respectively, TXWAKE is reset to 0 and WUT becomes the value of the address bit of the current frame. Thus, to send an address:

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 13-6. Address-Bit Multiprocessor Communication Format



13.1.1.7 SCI Communication Format

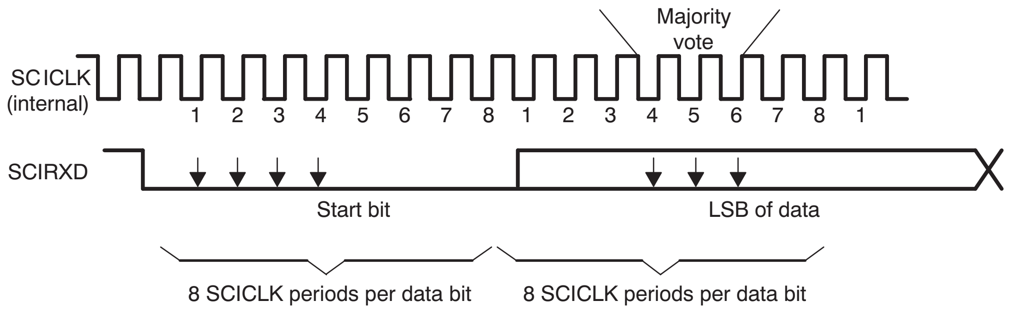
The SCI asynchronous communication format uses either single line (one way) or two line (two way) communications. In this mode, the frame consists of a start bit, one to eight data bits, an optional even/odd parity bit, and one or two stop bits (shown in Figure 13-7). There are eight SCICLK periods per 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 13-7. 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 13-7 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 13-7. SCI Asynchronous Communications Format

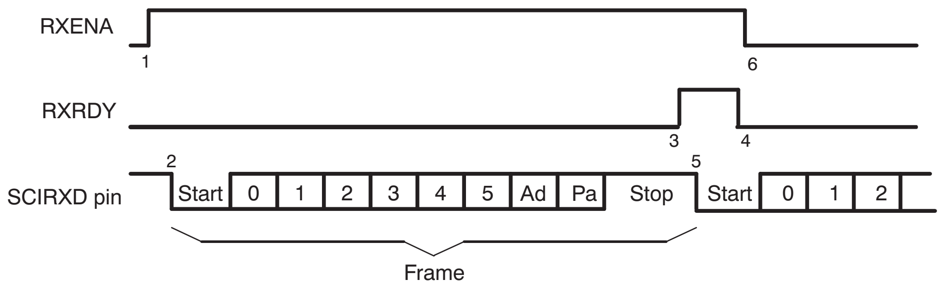


13.1.1.7.1 Receiver Signals in Communication Modes

Figure 13-8 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 13-8. SCI RX Signals in Communication Modes



(1) 2) Data arrives on the SCIRXD pin, start bit detected.

(2) 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.

Notes:

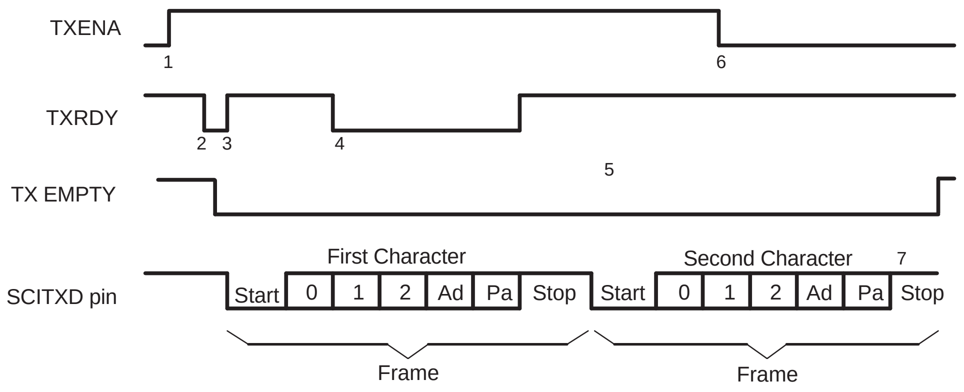
1. Flag bit RXENA (SCICTL1, bit 0) goes high to enable the receiver.
2. Data arrives on the SCIRXD pin, start bit detected.
3. Data is shifted from RXSHF to the receiver buffer register (SCIRXBUF); an interrupt is requested. Flag bit RXRDY (SCIRXST, bit 6) goes high to signal that a new character has 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.

13.1.1.7.2 Transmitter Signals in Communication Modes

Figure 13-9 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 13-9. 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.

13.1.1.8 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 which 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 which 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.

- 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 ten 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).

13.1.1.9 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 Section 13.2.4, for the formula to use when calculating the SCI asynchronous baud. Table 13-5 shows the baud-select values for common SCI bit rates.

Table 13-5. Asynchronous Baud Register Values for Common SCI Bit Rates

	LSPCLK Clock Frequency, 15 MHz		
Ideal Baud	BRR	Actual Baud	% Error
2400	780(30Ch)	2401	0.03
4800	390(186h)	4795	-0.10
9600	194(C2h)	9615	0.16
19200	97(61h)	19133	-0.35
38400	48(30h)	38265	-0.35

13.1.1.10 SCI Enhanced Features

The 28x SCI features autobaud detection and transmit/receive FIFO. The following section explains the FIFO operation.

13.1.1.10.1 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 F24x 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 -level FIFOs. The transmit FIFO registers are 8 bits wide and receive FIFO registers are 10 bits wide. The one word transmit buffer of the standard SCI functions as a transition buffer between the transmit FIFO and shift register. The one word transmit buffer is loaded from transmit FIFO only after the last bit of the shift register is shifted out. With the FIFO enabled, TXSHF is directly loaded after an optional delay value (SCIFFCT), TXBUF is not used.
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 12−0) 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 one.
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 0x11111 for receive FIFO and 0x00000 for transmit FIFO, respectively.

Figure 13-10 and Table 13-6 explain the operation/configuration of SCI interrupts in nonFIFO/FFO mode.

Figure 13-10. SCI FIFO Interrupt Flags and Enable Logic

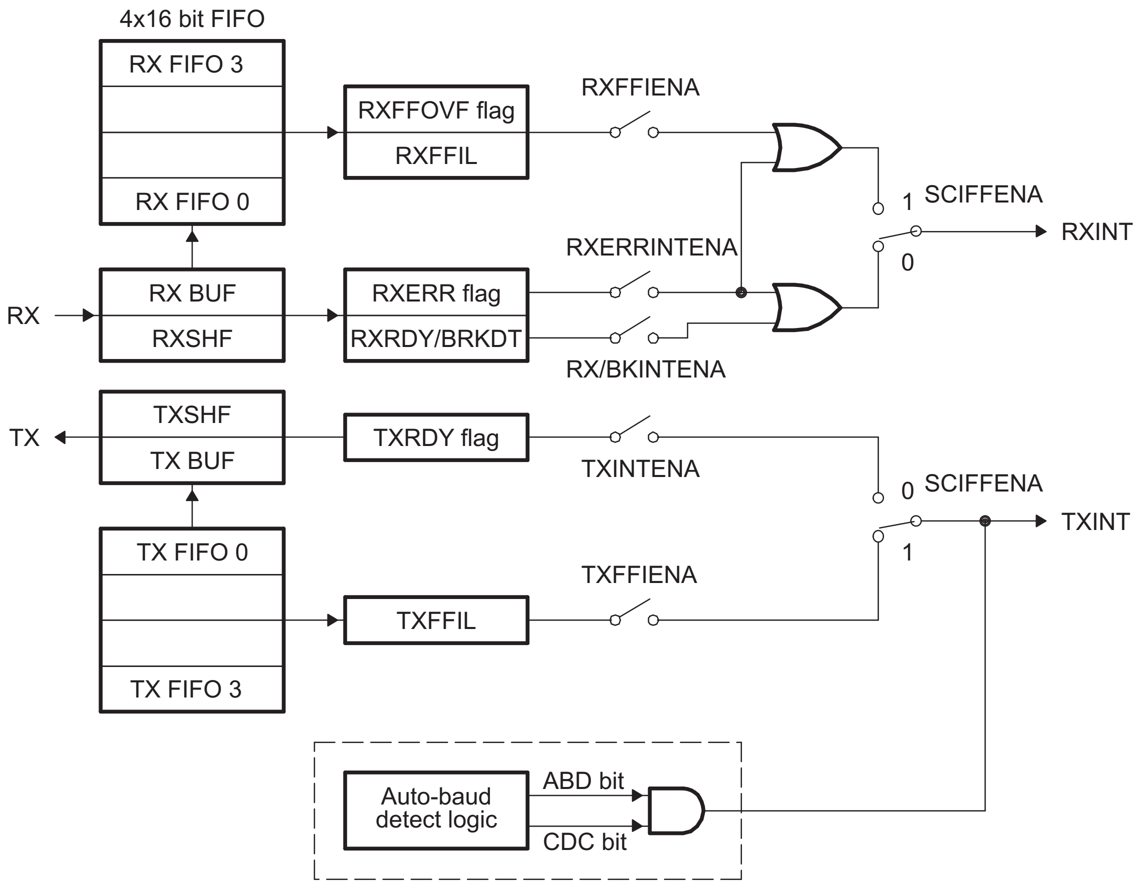


Table 13-6. SCI Interrupt Flags

FIFO Options[(1)]	SCI Interrupt Source	Interrupt Flags	Interrupt Enables	FIFO Enable SCIFFENA	Interrupt Line
SCI without FIFO	Receive error	RXERR [(2)]	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

13.1.1.10.2 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.

13.1.1.10.3 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 followng is recommended:

- Achieve a baud-lock between the host and 28x SCI boot loader using a lower baud rate.
- The host may then handshake with the loaded 28x application to set the SCI baud rate register to the desired higher baud rate.

13.2 SCI Registers

The functions of the SCI are software configurable. Sets of control bits, organized into dedicated bytes, are programmed to initialize the desired SCI communications format. This includes operating mode and protocol, baud value, character length, even/odd parity or no parity, number of stop bits, and interrupt priorities and enables.

13.2.1 SCI Module Register Summary

The SCI is controlled and accessed through registers listed in Table 13-7 and Table 13-8, which are described in the sections that follow.

Table 13-7. SCIA Registers

Register Mnemonic	Address	Number of Bits	Description
SCICCR	0x0000-7050	1	SCI-A Communications Control Register
SCICTL1	0x0000-7051	1	SCI-A Control Register 1
SCIHBAUD	0x0000-7052	1	SCI-A Baud Register, High Bits
SCILBAUD	0x0000-7053	1	SCI-A Baud Register, Low Bits
SCICTL2	0x0000-7054	1	SCI-A Control Register 2
SCIRXST	0x0000-7055	1	SCI-A Receive Status Register
SCIRXEMU	0x0000-7056	1	SCI-A Receive Emulation Data Buffer Register
SCIRXBUF	0x0000-7057	1	SCI-A Receive Data Buffer Register
SCITXBUF	0x0000-7059	1	SCI-A Transmit Data Buffer Register
SCIFFTX [(1)]	0x0000-705A	1	SCI-A FIFO Transmit Register
SCIFFRX[(1)]	0x0000-705B	1	SCI-A FIFO Receive Register
SCIFFCT [(1)]	0x0000-705C	1	SCI-A FIFO Control Register
SCIPRI	0x0000-705F	1	SCI-A Priority Control Register

(1) These registers operate in enhanced mode.

Table 13-8. SCIB Registers

Name	Address Range	Number of Bits	Description
SCICCR	0x0000-7750	1	SCI-B Communications Control Register
SCICTL1	0x0000-7751	1	SCI-B Control Register 1
SCIHBAUD	0x0000-7752	1	SCI-B Baud Register, High Bits
SCILBAUD	0x0000-7753	1	SCI-B Baud Register, Low Bits
SCICTL2	0x0000-7754	1	SCI-B Control Register 2
SCIRXST	0x0000-7755	1	SCI-B Receive Status Register
SCIRXEMU	0x0000-7756	1	SCI-B Receive Emulation Data Buffer Register
SCIRXBUF	0x0000-7757	1	SCI-B Receive Data Buffer Register
SCITXBUF	0x0000-7759	1	SCI-B Transmit Data Buffer Register
SCIFFTX	0x0000-775A	1	SCI-B FIFO Transmit Register
SCIFFRX	0x0000-775B	1	SCI-B FIFO Receive Register
SCIFFCT	0x0000-775C	1	SCI-B FIFO Control Register
SCIPRI	0x0000-775F	1	SCI-B Priority Control Register

13.2.2 SCI Communication Control Register (SCICCR)

SCICCR defines the character format, protocol, and communications mode used by the SCI.

Figure 13-11. SCI Communication Control Register (SCICCR) — Address 7050h

7	6	5	4	3	2	1	0
STOP BITS	EVEN/ODD PARITY	PARITY ENABLE	LOOPBACK ENA	ADDR/IDLE MODE	SCICHAR2	SCICHAR1	SCICHAR0
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 13-9. SCI Communication Control Register (SCICCR) Field Descriptions

Bit	Field	Value	Description
7	STOP BITS		SCI number of stop bits. This bit specifies the number of stop bits transmitted. The receiver checks for only one stop bit.
		0	One stop bit
		1	Two stop bits
6	EVEN/ODD PARITY		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).
		0	Odd parity
		1	Even parity
5	PARITY ENABLE		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.
		0	Parity disabled; no parity bit is generated during transmission or is expected during reception
		1	Parity is enabled
4	LOOP BACK ENA		Loop Back test mode enable. This bit enables the Loop Back test mode where the Tx pin is internally connected to the Rx pin.
		0	Loop Back test mode disabled
		1	Loop Back test mode enabled
3	ADDR/IDLE MODE		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.
		0	Idle-line mode protocol selected
		1	Address-bit mode protocol selected
2-0	SCI CHAR2−0		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. The bit values and character lengths for SCI CHAR2-0 bits are as follows:

SCI CHAR2−0 Bit Values (Binary)			
SCI CHAR2	SCI CHAR1	SCI CHAR0	Character Length (Bits)
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

13.2.3 SCI Control Register 1 (SCICTL1)

SCICTL1 controls the receiver/transmitter enable, TXWAKE and SLEEP functions, and the SCI software reset.

Figure 13-12. SCI Control Register 1 (SCICTL1) — Address 7051h

7	6	5	4	3	2	1	0
Reserved	RX ERR INT ENA	SW RESET	Reserved	TXWAKE	SLEEP	TXENA	RXENA
R-0	R/W-0	R/W-0	R-0	R/S-0	R/W-0	R/W-0	R/W-0

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

Table 13-10. SCI Control Register 1 (SCICTL1) Field Descriptions

Bit	Field	Value	Description
7	Reserved		Reads return zero; writes have no effect.
6	RX ERR INT ENA		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.
		0	Receive error interrupt disabled
		1	Receive error interrupt enabled
5	SW RESET		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 RESET	SCI Flag	Register Bit
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

Bit	Field	Value	Description
		0	Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition.
		1	After a system reset, re-enable the SCI by writing a 1 to this bit.
4	Reserved		Reads return zero; writes have no effect.
3	TXWAKE		SCI transmitter wake-up method select. 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)
		0	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
		1	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); it is cleared by a system reset or the transfer of TXWAKE to the WUT flag.

Table 13-10. SCI Control Register 1 (SCICTL1) Field Descriptions (continued)

Bit	Field	Value	Description
2	SLEEP		SCI sleep. 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.
		0	Sleep mode disabled
		1	Sleep mode enabled
1	TXENA		SCI transmitter enable. 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.
		0	Transmitter disabled0
		1	Transmitter enabled
0	RXENA		SCI receiver enable. 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). 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.
		0	Prevent received characters from transfer into the SCIRXEMU and SCIRXBUF receiver buffers
		1	Send received characters to SCIRXEMU and SCIRXBUF

13.2.4 SCI Baud-Select Registers (SCIHBAUD, SCILBAUD)

The values in SCIHBAUD and SCILBAUD specify the baud rate for the SCI.

Figure 13-13. Baud-Select MSbyte Register (SCIHBAUD) — Address 7052h

15	14	13	12	11	10	9	8
BAUD15 (MSB)	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10	BAUD9	BAUD8
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

Figure 13-14. Baud-Select LSbyte Register (SCILBAUD) — Address 7053h

7	6	5	4	3	2	1	0
BAUD7	BAUD6	BAUD5	BAUD4	BAUD3	BAUD2	BAUD1	BAUD10 (LSB)
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 13-11. Baud-Select Register Field Descriptions

Bit	Field	Value	Description
15-0	BAUD15− BAUD0		SCI 16-bit baud selection Registers SCIHBAUD (MSbyte) and SCILBAUD (LSbyte) are concatenated to form a 16-bit baud value, BRR. 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. The SCI baud rate is calculated using the following equation: $$\text{SCI Asynchronous Baud} = \frac{\text{LSPCLK}}{(\text{BRR} + 1) \times 8} \quad (8)$$ Alternatively, $$\text{BRR} = \frac{\text{LSPCLK}}{\text{SCI Asynchronous Baud} \times 8} - 1 \quad (9)$$ Note that the above formulas are applicable only when 1 ≤ BRR ≤ 65535. If BRR = 0, then $$\text{SCI Asynchronous Baud} = \frac{\text{LSPCLK}}{16} \quad (10)$$ Where: BRR = the 16-bit value (in decimal) in the baud-select registers.

13.2.5 SCI Control Register 2 (SCICTL2)

SCICTL2 enables the receive-ready, break-detect, and transmit-ready interrupts as well as transmitter-ready and -empty flags.

Figure 13-15. SCI Control Register 2 (SCICTL2) — Address 7054h

7	6	5-2	1	0
TXRDY	TX EMPTY	Reserved	RX/BK INT ENA	TX INT ENA
R-1	R-1	R-0	R/W-0	R/W-0

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

Table 13-12. SCI Control Register 2 (SCICTL2) Field Descriptions

Bit	Field	Value	Description
7	TXRDY		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.
		0	SCITXBUF is full
		1	SCITXBUF is ready to receive the next character
6	TX EMPTY		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.
		0	Transmitter buffer or shift register or both are loaded with data
		1	Transmitter buffer and shift registers are both empty
5-2	Reserved		
1	RX/BK INT ENA		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.
		0	Disable RXRDY/BRKDT interrupt
		1	Enable RXRDY/BRKDT interrupt
0	TX INT ENA		SCITXBUF-register interrupt enable.This bit controls the interrupt request caused by setting the TXRDY flag bit (SCICTL2.7). However, it does not prevent the TXRDY flag from being set (being set indicates that register SCITXBUF is ready to receive another character).
		0	Disable TXRDY interrupt
		1	Enable TXRDY interrupt

13.2.6 SCI Receiver Status Register (SCIRXST)

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 13-17 shows the relationships between several of the register's bits.

Figure 13-16. SCI Receiver Status Register (SCIRXST) — Address 7055h

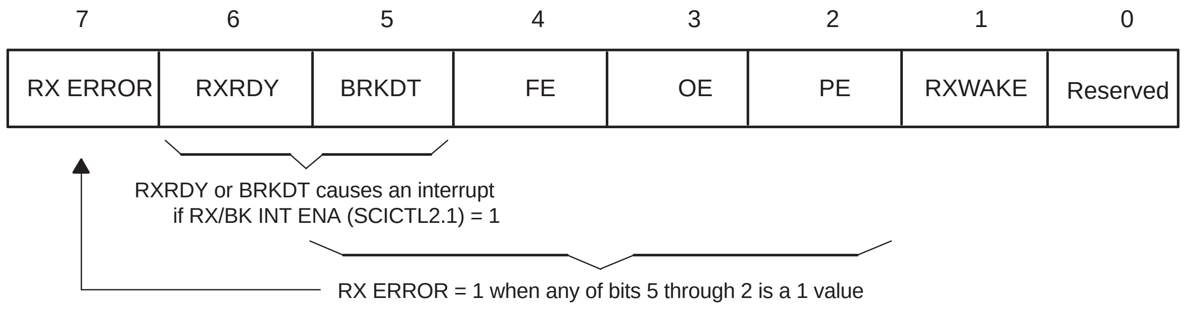
7	6	5	4	3	2	1	0
RX ERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	Reserved
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 13-13. SCI Receiver Status Register (SCIRXST) Field Descriptions

Bit	Field	Value	Description
7	RX ERROR		SCI receiver error flag. 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). 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; it is cleared by an active SW RESET or by a system reset.
		0	No error flags set
		1	Error flag(s) set
6	RXRDY		SCI receiver-ready flag. 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.
		0	No new character in SCIRXBUF
		1	Character ready to be read from SCIRXBUF
5	BRKDT		SCI break-detect flag. 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.
		0	No break condition
		1	Break condition occurred
4	FE		SCI framing-error flag. 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.
		0	No framing error detected
		1	Framing error detected
3	OE		SCI overrun-error flag. 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.
		0	No overrun error detected
		1	Overrun error detected
2	PE		SCI parity-error flag. 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.!
		0	No parity error or parity is disabled
		1	Parity error is detected
1	RXWAKE		Receiver wake-up-detect flag
		0	No detection of a receiver wake-up condition
		1	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. RXWAKE is a read-only flag, cleared by one of the following: • The transfer of the first byte after the address byte to SCIRXBUF (only in non-FIFO mode) • The reading of SCIRXBUF • An active SW RESET • A system reset
0	Reserved		Reads return zero; writes have no effect.

Figure 13-17. Register SCIRXST Bit Associations — Address 7055h


7
6
5
4
3
2
1
0
RX ERROR
RXRDY
BRKDT
FE
OE
PE
RXWAKE
Reserved
RXRDY or BRKDT causes an interrupt
if RX/BK INT ENA (SCICTL2.1) = 1
RX ERROR = 1 when any of bits 5 through 2 is a 1 value

13.2.7 Receiver Data Buffer Registers (SCIRXEMU, SCIRXBUF)

Received data is transferred from RXSHF to SCIRXEMU and SCIRXBUF. When the transfer is complete, the RXRDY flag (bit SCIRXST.6) is set, indicating that the received data is ready to be read. Both registers contain the same data; they have separate addresses but are not physically separate buffers. The only difference is that reading SCIRXEMU does not clear the RXRDY flag; however, reading SCIRXBUF clears the flag.

13.2.7.1 Emulation Data Buffer (SCIRXEMU)

Normal SCI data-receive operations read the data received from the SCIRXBUF register. The SCIRXEMU register is used principally by the emulator (EMU) 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 an emulator 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 13-18. Emulation Data Buffer Register (SCIRXEMU) — Address 7056h

7	6	5	4	3	2	1	0
ERXDT7	ERXDT6	ERXDT5	ERXDT4	ERXDT3	ERXDT2	ERXDT1	ERXDT0
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

13.2.7.2 Receiver Data Buffer (SCIRXBUF)

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 RX/BK INT ENA 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 13-19. SCI Receive Data Buffer Register (SCIRXBUF) — Address 7057h

15	14	13					8
SCIFFFE[(1)]	SCIFFPE[(1)]	Reserved					
R−0	R−0	R−0					
7	6	5	4	3	2	1	0
RXDT7	RXDT6	RXDT5	RXDT4	RXDT3	RXDT2	RXDT1	RXDT0
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

(1) Applicable only if the FIFO is enabled.

Table 13-14. SCI Receive Data Buffer Register (SCIRXBUF) Field Descriptions

Bit	Field	Value	Description
15	SCIFFFE		SCIFFFE. SCI FIFO Framing error flag bit (applicable only if the FIFO is enabled)
		0	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.
		1	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		SCIFFPE. SCI FIFO parity error flag bit (applicable only if the FIFO is enabled)
		0	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.
		1	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		
7-0	RXDT7−0		Receive Character bits

13.2.8 SCI Transmit Data Buffer Register (SCITXBUF)

Data bits to be transmitted are written to SCITXBUF. These bits must be rightjustified 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 TX INT ENA (SCICTL2.0) is set, this data transfer also causes an interrupt.

Figure 13-20. Transmit Data Buffer Register (SCITXBUF) — Address 7059h

7	6	5	4	3	2	1	0
TXDT7	TXDT6	TXDT5	TXDT4	TXDT3	TXDT2	TXDT1	TXDT0
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

13.2.9 SCI FIFO Registers (SCIFFTX, SCIFFRX, SCIFFCT)

Figure 13-21. SCI FIFO Transmit (SCIFFTX) Register — Address 705Ah

15	14	13	12	11	10	9	8
SCIRST	SCIFFENA	TXFIFO Reset	TXFFST4	TXFFST3	TXFFST2	TXFFST1	TXFFST0
R/W-1	R/W-0	R/W-1	R−0	R−0	R−0	R−0	R−0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TXFFINT Flag	TXFFINT CLR	TXFFIENA	TXFFIL4	TXFFIL3	TXFFIL2	TXFFIL1	TXFFIL0
R−0	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 13-15. Field Descriptions

Bit	Field	Value	Description
15	SCIRST		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.
14	SCIFFENA		SCI FIFO enable
		0	SCI FIFO enhancements are disabled
		1	SCI FIFO enhancements are enabled
13	TXFIFO Reset		Transmit FIFO reset
		0	Reset the FIFO pointer to zero and hold in reset
		1	Re-enable transmit FIFO operation

Table 13-15. Field Descriptions (continued)

Bit	Field	Value	Description
12-8	TXFFST4-0	00000	Transmit FIFO is empty.
		00001	Transmit FIFO has 1 words
		00010	Transmit FIFO has 2 words
		00011	Transmit FIFO has 3 words
		00100	Transmit FIFO has 4 words
7	TXFFINT Flag		Transmit FIFO interrupt
		0	TXFIFO interrupt has not occurred, read-only bit
		1	TXFIFO interrupt has occurred, read-only bit
6	TXFFINT CLR		Transmit FIFO clear
		0	Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero
		1	Write 1 to clear TXFFINT flag in bit 7
5	TXFFIENA		Transmit FIFO interrrupt enable
		0	TX FIFO interrupt based on TXFFIVL match (less than or equal to) is disabled
		1	TX FIFO interrupt based on TXFFIVL match (less than or equal to) is enabled.
4-0	TXFFIL4-0		TXFFIL4−0 Transmit FIFO interrupt level bits. Transmit FIFO will generate interrupt when the FIFO status bits (TXFFST4−0) and FIFO level bits (TXFFIL4−0) match (less than or equal to). Because the 2802x SCI has a 4-level transmit FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels.
			Default value should be 0x00000.

Figure 13-22. SCI FIFO Receive (SCIFFRX) Register — Address 705Bh

15	14	13	12	11	10	9	8
RXFFOVF	RXFFOVR CLR	RXFIFO Reset	RXFIFST4	RXFFST3	RXFFST2	RXFFST1	RXFFST0
R-0	W−0	R/W−1	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
RXFFINT Flag	RXFFINT CLR	RXFFIENA	RXFFIL4	RXFFIL3	RXFFIL2	RXFFIL1	RXFFIL0
R-0	W−0	R/W−0	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 13-16. SCI FIFO Receive (SCIFFRX) Register Field Descriptions

Bit	Field	Value	Description
15	RXFFOVF		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.
		0	Receive FIFO has not overflowed, read-only bit
		1	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	RXFFOVF CLR		RXFFOVF clear
		0	Write 0 has no effect on RXFFOVF flag bit, Bit reads back a zero
		1	Write 1 to clear RXFFOVF flag in bit 15
13	RXFIFO Reset		Receive FIFO reset
		0	Write 0 to reset the FIFO pointer to zero, and hold in reset.
		1	Re-enable receive FIFO operation
12-8	RXFFST4−0	00000	Receive FIFO is empty
		00001	Receive FIFO has 1 word
		00010	Receive FIFO has 2 words
		00011	Receive FIFO has 3 words
		00100	Receive FIFO has 4 words, the maximum allowed.

Table 13-16. SCI FIFO Receive (SCIFFRX) Register Field Descriptions (continued)

Bit	Field	Value	Description
7	RXFFINT		Receive FIFO interrupt
		0	RXFIFO interrupt has not occurred, read-only bit
		1	RXFIFO interrupt has occurred, read-only bit
6	RXFFINT CLR		Receive FIFO interrupt clear
		0	Write 0 has no effect on RXFIFINT flag bit. Bit reads back a zero.
		1	Write 1 to clear RXFFINT flag in bit 7
5	RXFFIENA		Receive FIFO interrupt enable
		0	RX FIFO interrupt based on RXFFIVL match (less than or equal to) will be disabled
		1	RX FIFO interrupt based on RXFFIVL match (less than or equal to) will be enabled.
4-0	RXFFIL4−0		Receive FIFO interrupt level bits
		11111	Receive FIFO generates interrupt when the FIFO status bits (RXFFST4−0) and FIFO level bits (RXFFIL4−0) match (i.e., are greater than or equal to). Default value of these bits after reset − 11111. This will avoid frequent interrupts, after reset, as the receive FIFO will be empty most of the time. Because the 2802x SCI has a 4-level receive FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels.

Figure 13-23. SCI FIFO Control (SCIFFCT) Register — Address 705Ch

15	14	13	12–8
ABD	ABD CLR	CDC	Reserved
R-0	W−0	R/W−0	R-0

7	6	5	4	3	2	1	0
FFTXDLY7	FFTXDLY6	FFTXDLY5	FFTXDLY4	FFTXDLY3	FFTXDLY2	FFTXDLY1	FFTXDLY0
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 13-17. SCI FIFO Control (SCIFFCT) Register Field Descriptions

Bit	Field	Value	Description
15	ABD		Auto-baud detect (ABD) bit.
		0	Auto-baud detection is not complete. ”A”,”a” character has not been received successfully.
		1	Auto-baud hardware has detected ”A” or ”a” character on the SCI receive register. Auto-detect is complete.
14	ABD CLR		ABD-clear bit
		0	Write 0 has no effect on ABD flag bit. Bit reads back a zero.
		1	Write 1 to clear ABD flag in bit 15.
13	CDC		CDC calibrate A-detect bit
		0	Disables auto-baud alignment
		1	Enables auto-baud alignment
12-8	Reserved		Reserved

Table 13-17. SCI FIFO Control (SCIFFCT) Register Field Descriptions (continued)

Bit	Field	Value	Description
7−0	FFTXDLY7−0		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.

13.2.10 Priority Control Register (SCIPRI)

Figure 13-24. SCI Priority Control Register (SCIPRI) — Address 705Fh

7 - 5	4	3	2 - 0
Reserved	SCI SOFT	SCI FREE	Reserved
R-0	R/W-0	R/W-0	R-0

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

Table 13-18. Field Descriptions

Bit	Field	Value	Description
7-5	Reserved		Reads return zero; writes have no effect.
4-3	SOFT and FREE		These bits determine what occurs when an emulation suspend event 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.
		00	Immediate stop on suspend
		10	Complete current receive/transmit sequence before stopping
		x1	Free run. Continues SCI operation regardless of suspend
2-0	Reserved		Reads return zero; writes have no effect.

Inter-Integrated Circuit Module (I2C)

This guide describes the features and operation of the inter-integrated circuit (I2C) module. bi-directional data transfer through a two-wire design (a serial data line SDA and a serial clock line SCL), and interfaces to external I2C devices such as serial memory (RAMs and ROMs), networking devices, LCDs, tone generators, and so on. The I2C bus may also be used for system testing and diagnostic purposes in product development and manufacture.

The two I2C modules provide the ability to interact (both transmit and receive) with other I2C devices on the bus. They include the following features:

Devices on the I2C bus can be designated as either a master or a slave – Supports both transmitting and receiving data as either a master or a slave – Supports simultaneous master and slave operation

- Four I2C modes – Master transmit – Master receive – Slave transmit – Slave receive
- Two transmission speeds: Standard (100 Kbps) and Fast (400 Kbps)
- Master and slave interrupt generation – Master generates interrupts when a transmit or receive operation completes (or aborts due to an error) – Slave generates interrupts when data has been transferred or requested by a master or when a START or STOP condition is detected
- Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode

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 via the BC bits of the mode register, I2CMDR.)

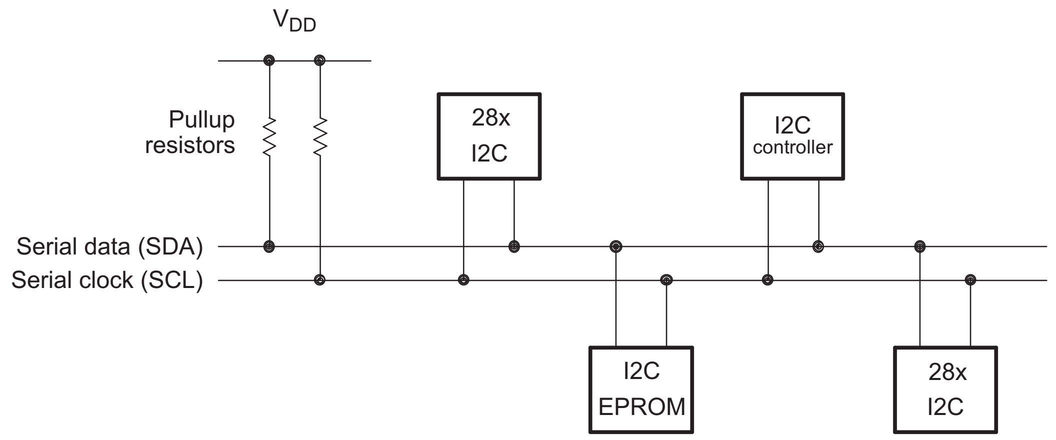
This reference guide is applicable for the I2C found on the TMS320x28x family of processors.

Topic **Page**

14.1 Introduction to the I2C Module

The I2C module supports any slave or master I2C-compatible device. Figure 14-1 shows an example of multiple I2C modules connected for a two-way transfer from one device to other devices.

Figure 14-1. Multiple I2C Modules Connected



14.1.1 Features

The I2C module has the following features:

- Compliance with the Philips 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 of from 10 kbps up to 400 kbps (Philips Fast-mode rate)
- One 4-level receive FIFO and one 4-level transmit FIFO
- One interrupt that can always be used by the CPU. This interrupt can be generated as a result of one of the following conditions: transmit-data ready, receive-data ready, register-access ready, no-acknowledgment received, arbitration lost, stop condition detected, addressed as slave.
- An additional interrupt that can be used by the CPU when in FIFO mode
- Module enable/disable capability
- Free data format mode

14.1.2 Features Not Supported

The I2C module does not support:

- High-speed mode (Hs-mode)
- CBUS-compatibility mode

14.1.3 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 14-2. These two pins carry information between the 28x 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 Table 14-5 for register information.
- Repeat Mode: Keep sending data values until you use software to initiate a STOP condition or a new START condition. See Table 14-5 for RM bit information.

.

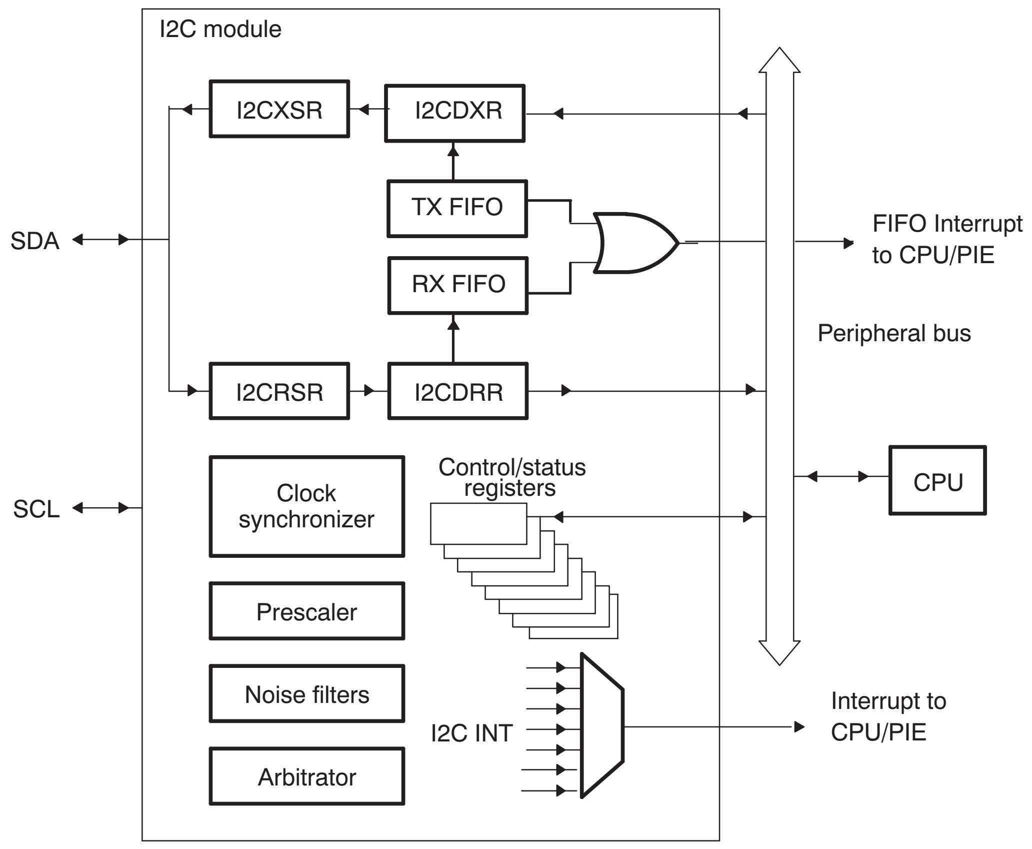
The I2C module consists of the following primary blocks:

- 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 14-2 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 a time. When the I2C module is configured as a receiver, received data is shifted into I2CRSR and then copied to I2CDRR.

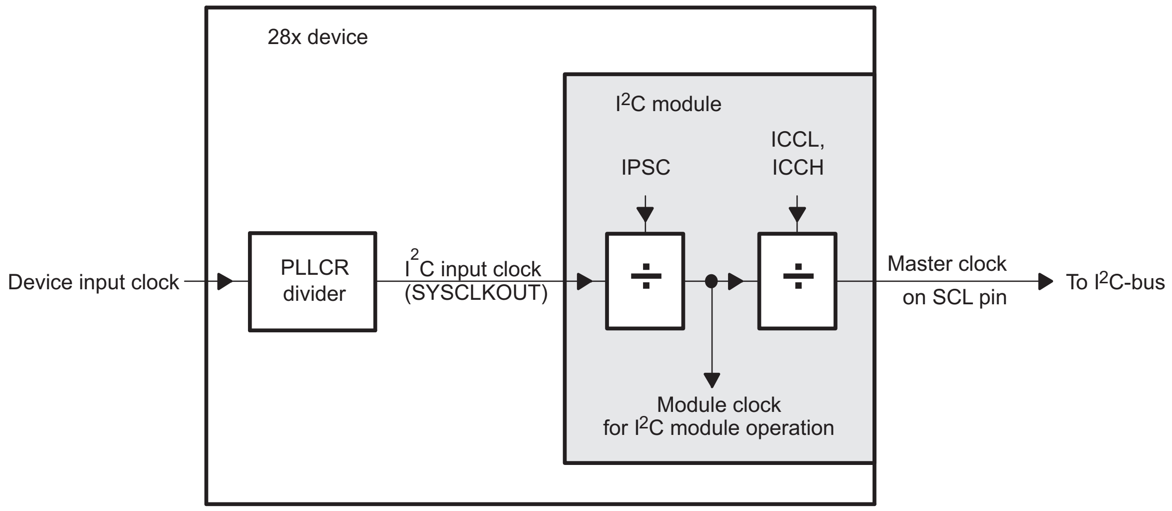
Figure 14-2. I2C Module Conceptual Block Diagram



14.1.4 Clock Generation

As shown in Figure 14-3, the device clock generator receives a signal from an external clock source and produces an I2C input clock with a programmed frequency. The I2C input clock is equivalent to the CPU clock and is then divided twice more inside the I2C module to produce the module clock and the master clock.

Figure 14-3. Clocking Diagram for the I2C Module



The module clock determines the frequency at which the I2C module operates. A programmable prescaler in the I2C module divides down the I2C input clock to produce the module clock. To specify the divide-down value, initialize the IPSC field of the prescaler register, I2CPSC. The resulting frequency is:

$$\text{module clock frequency} = \frac{\text{I2C input clock frequency}}{(\text{IPSC} + 1)}$$

NOTE: To meet all of the I2C protocol timing specifications, the module clock must be configured between 7 - 12 MHz.

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 14-3, 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 Section 14.5.7.1 for the master clock frequency equation.

14.2 I2C Module Operational Details

This section provides an overview of the I2C-bus protocol and how it is implemented.

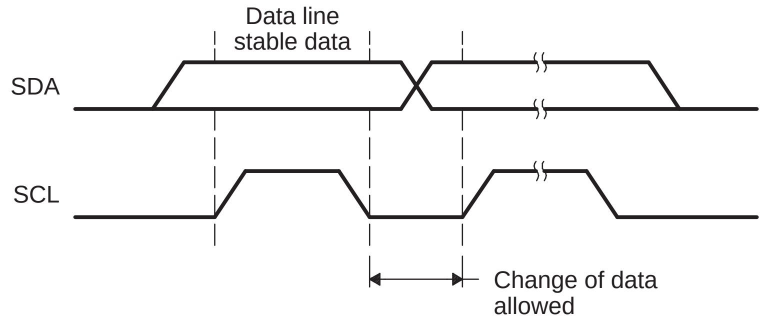
14.2.1 Input and Output Voltage Levels

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 the data manual for your particular device.

14.2.2 Data Validity

The data on SDA must be stable during the high period of the clock (see Figure 14-4). The high or low state of the data line, SDA, should change only when the clock signal on SCL is low.

Figure 14-4. Bit Transfer on the I2C-Bus



14.2.3 Operating Modes

The I2C module has four basic operating modes to support data transfers as a master and as a slave. See Table 14-1 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 14-1. Operating Modes of the I2C Module

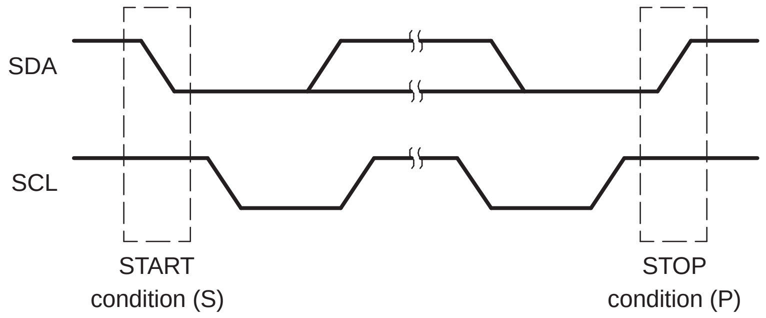
Operating Mode	Description
Slave-receiver modes	The I2C module is a slave and receives data from a master. All slaves begin in this mode. In this mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master. As a slave, the I2C module does not generate 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 section Section 14.2.7 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/$\overline{W}$ = 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 section Section 14.2.7 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/$\overline{W}$ = 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 modes	The IC 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.

14.2.4 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 14-5:

- 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 14-5. 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 14.5.1.

14.2.5 Serial Data Formats

Figure 14-6 shows an example of a data transfer on the I2C-bus. The I2C module supports 1 to 8-bit data values. In Figure 14-6, 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 14-6 is the 7-bit addressing format. The I2C module supports the formats shown in Figure 14-7 through Figure 14-9 and described in the paragraphs that follow the figures.

NOTE: In Figure 14-6 through Figure 14-9, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

Figure 14-6. I2C Module Data Transfer (7-Bit Addressing with 8-bit Data Configuration Shown)

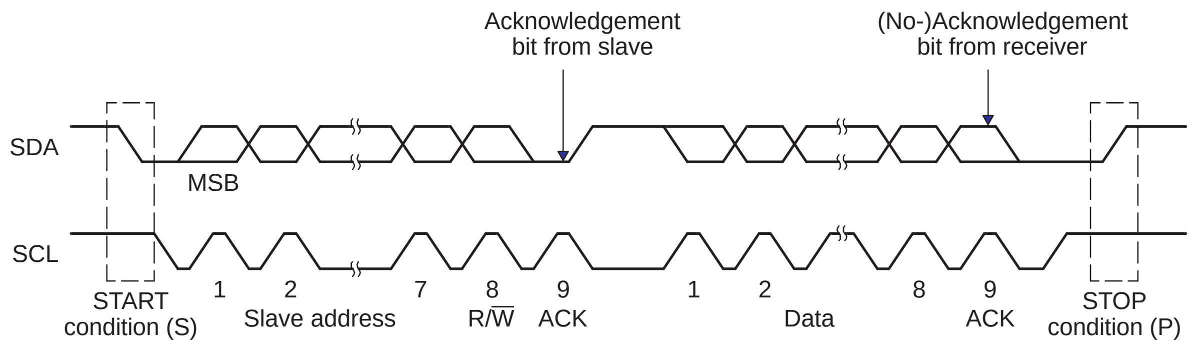


Figure 14-7. I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMDR)

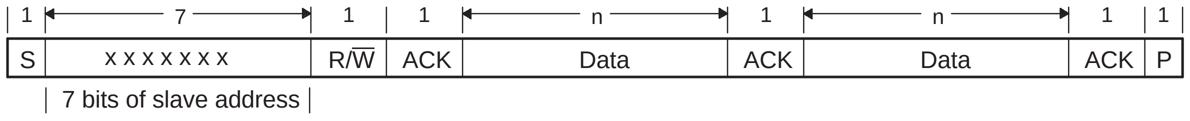


Figure 14-8. I2C Module 10-Bit Addressing Format (FDF = 0, XA = 1 in I2CMDR)

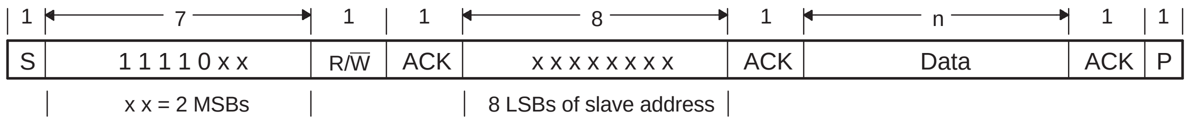
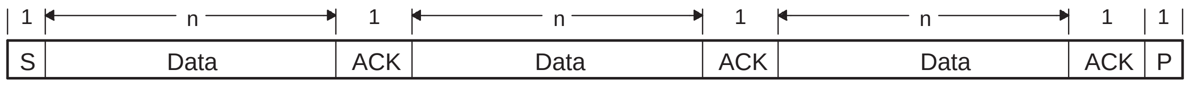


Figure 14-9. I2C Module Free Data Format (FDF = 1 in I2CMDR)



14.2.5.1 7-Bit Addressing Format

In the 7-bit addressing format (see Figure 14-7), the first byte after a START condition (S) consists of a 7-bit slave address followed by a R/$\overline{W}$ bit. R/$\overline{W}$ determines the direction of the data:

- R/$\overline{W}$ = 0: The master writes (transmits) data to the addressed slave.
- R/$\overline{W}$ = 1: The master reads (receives) data from the slave.

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/$\overline{W}$ 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.

To select the 7-bit addressing format, write 0 to the expanded address enable (XA) bit of I2CMDR, and make sure the free data format mode is off (FDF = 0 in I2CMDR).

14.2.5.2 10-Bit Addressing Format

The 10-bit addressing format (see Figure 14-8) 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/$\overline{W}$ = 0 (write). 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 Philips Semiconductors I2C-bus specification.

To select the 10-bit addressing format, write 1 to the XA bit of I2CMDR and make sure the free data format mode is off (FDF = 0 in I2CMDR).

14.2.5.3 Free Data Format

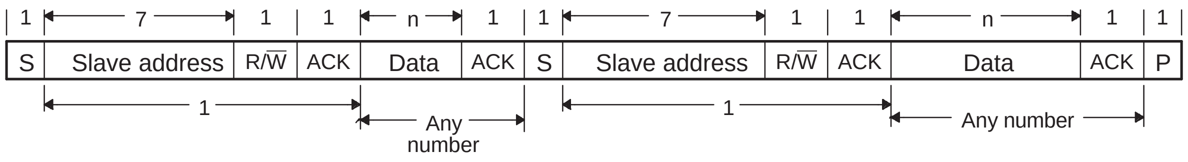
In this format (see Figure 14-9), the first byte after a START condition (S) is a data byte. An ACK bit is inserted after each data byte, which can be from 1 to 8 bits, depending on the BC field of I2CMDR. No address or data-direction bit is sent. Therefore, the transmitter and the receiver must both support the free data format, and the direction of the data must be constant throughout the transfer.

To select the free data format, write 1 to the free data format (FDF) bit of I2CMDR. The free data format is not supported in the digital loopback mode (DLB = 1 in I2CMDR).

14.2.5.4 Using a Repeated START Condition

At the end of each data byte, the master can drive another START condition. Using this capability, a master can communicate with multiple slave addresses without having to give up control of the bus by driving a STOP condition. The length of a data byte can be from 1 to 8 bits and is selected with the BC field of I2CMDR. The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. Figure 14-10 shows a repeated START condition in the 7-bit addressing format.

Figure 14-10. Repeated START Condition (in This Case, 7-Bit Addressing Format)



NOTE: In Figure 14-10, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

14.2.6 *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 14-2 summarizes the various ways you can tell the I2C module to send a NACK bit.

Table 14-2. Ways to Generate a NACK Bit

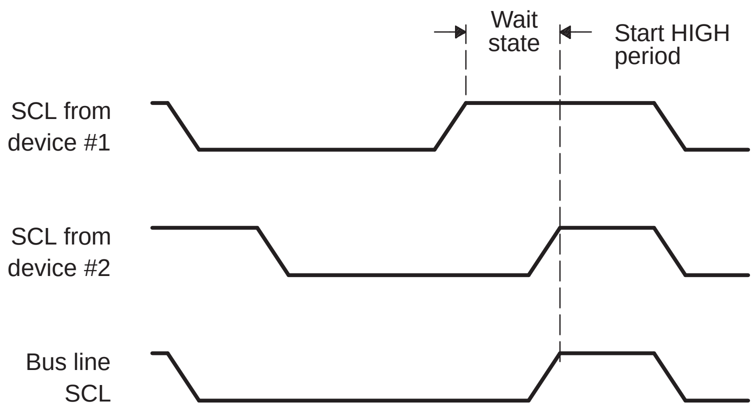
I2C Module Condition	NACK Bit Generation Options
Slave-receiver modes	• Allow an overrun condition (RSFULL = 1 in I2CSTR) • Reset the module (IRS = 0 in I2CMDR) • Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive
Master-receiver mode AND Repeat mode (RM = 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

14.2.7 *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 14-11 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 14-11. Synchronization of Two I2C Clock Generators During Arbitration



14.2.8 Arbitration

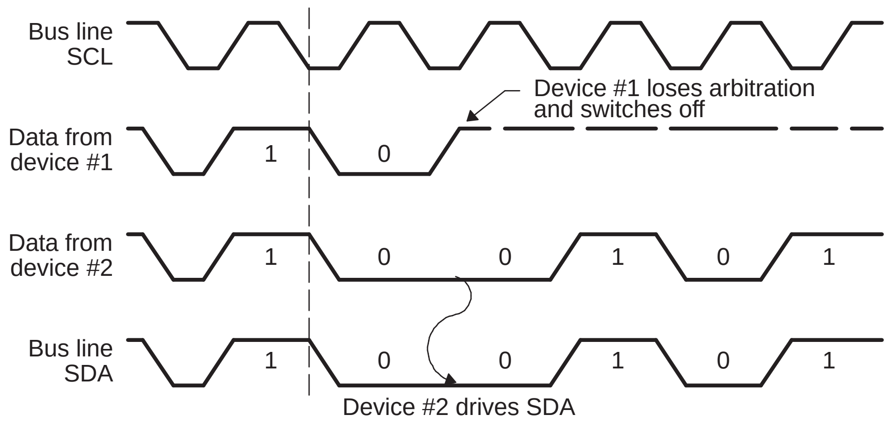
If two or more master-transmitters attempt to start a transmission on the same bus at approximately the same time, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. Figure 14-12 illustrates the arbitration procedure between two devices. The first master-transmitter, which release the SDA line high, is overruled by another master-transmitter that drives 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 (AL) 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 14-12. Arbitration Procedure Between Two Master-Transmitters



14.3 Interrupt Requests Generated by the I2C Module

The I2C module can generate seven types of basic interrupt requests, which are described in Section 14.3.1. Two of these can tell the CPU when to write transmit data and when to read receive data. If you want the FIFOs to handle transmit and receive data, you can also use the FIFO interrupts described in Section 14.3.2. The basic I2C interrupts are combined to form PIE Group 8, Interrupt 1 (I2CINT1A_ISR), and the FIFO interrupts are combined to form PIE Group 8, Interrupt 2 (I2CINT2A_ISR).

14.3.1 Basic I2C Interrupt Requests

The I2C module generates the interrupt requests described in Table 14-3. As shown in Figure 14-13, 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.

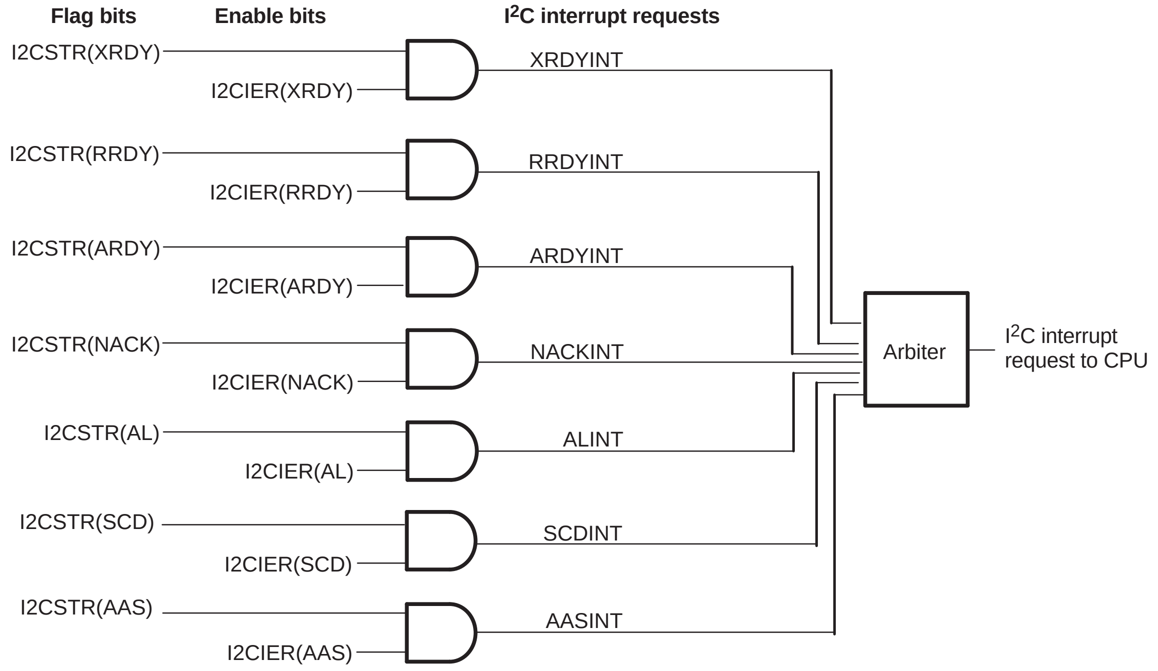
After the CPU reads I2CISRC, the following events occur:

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 14-3. Descriptions of the Basic I2C Interrupt Requests

I2C Interrupt Request	Interrupt Source
XRDYINT	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). 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.
RRDYINT	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. 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.
ARDYINT	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. The specific events that generate ARDYINT are the same events that set the ARDY bit of I2CSTR. As an alternative to using ARDYINT, the CPU can poll the ARDY bit.
NACKINT	No-acknowledgment condition: The I2C module is configured as a master-transmitter and did not received acknowledgment from the slave-receiver. As an alternative to using NACKINT, the CPU can poll the NACK bit of I2CSTR.
ALINT	Arbitration-lost condition: The I2C module has lost an arbitration contest with another master-transmitter. As an alternative to using ALINT, the CPU can poll the AL bit of I2CSTR.
SCDINT	Stop condition detected: A STOP condition was detected on the I2C bus. As an alternative to using SCDINT, the CPU can poll the SCD bit of the status register, I2CSTR.
AASINT	Addressed as slave condition: The I2C has been addressed as a slave device by another master on the I2C bus. As an alternative to using AASINT, the CPU can poll the AAS bit of the status register, I2CSTR.

Figure 14-13. Enable Paths of the I2C Interrupt Requests



14.3.2 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. 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.

14.4 Resetting/Disabling the I2C Module

You can reset/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 $\overline{\text{XRS}}$ 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.

IRS must be 0 while you configure/reconfigure the I2C module. Forcing IRS to 0 can be used to save power and to clear error conditions.

14.5 I2C Module Registers

Table 14-4 lists the I2C module registers. All but the receive and transmit shift registers (I2CRSR and I2CXSR) are accessible to the CPU.

Table 14-4. I2C Module Registers

Name	Address	Description
I2COAR	0x7900	I2C own address register
I2CIER	0x7901	I2C interrupt enable register
I2CSTR	0x7902	I2C status register
I2CCLKL	0x7903	I2C clock low-time divider register
I2CCLKH	0x7904	I2C clock high-time divider register
I2CCNT	0x7905	I2C data count register
I2CDRR	0x7906	I2C data receive register
I2CSAR	0x7907	I2C slave address register
I2CDXR	0x7908	I2C data transmit register
I2CMDR	0x7909	I2C mode register
I2CISRC	0x790A	I2C interrupt source register
I2CEMDR	0x790B	I2C extended mode register
I2CPSC	0x790C	I2C prescaler register
I2CFFTX	0x7920	I2C FIFO transmit register
I2CFFRX	0x7921	I2C FIFO receive register
I2CRSR	-	I2C receive shift register (not accessible to the CPU)
I2CXSR	-	I2C transmit shift register (not accessible to the CPU)

NOTE: To use the I2C module the system clock to the module must be enabled by setting the appropriate bit in the PCLKR0 register. See *TMS320x2806x Piccolo MCU System Control and Interrupts Reference Guide.*

14.5.1 I2C Mode Register (I2CMDR)

The I2C mode register (I2CMDR) is a 16-bit register that contains the control bits of the I2C module. The bit fields of I2CMDR are shown in Figure 14-14 and described in Table 14-5.

Figure 14-14. I2C Mode Register (I2CMDR)

15	14	13	12	11	10	9	8
NACKMOD	FREE	STT	Reserved	STP	MST	TRX	XA
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		0
RM	DLB	IRS	STB	FDF	BC		
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 14-5. I2C Mode Register (I2CMDR) Field Descriptions

Bit	Field	Value	Description
15	NACKMOD		NACK mode bit. This bit is only applicable when the I2C module is acting as a receiver.
		0	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.
			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.
		1	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.
			Important: To send a NACK bit in the next acknowledge cycle, you must set NACKMOD before the rising edge of the last data bit.
14	FREE		This bit controls the action taken by the I2C module when a debugger breakpoint is encountered.
		0	When I2C module is master:
			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.
			When I2C module is slave:
			A breakpoint forces the I2C module to stop when the current transmission/reception is complete.
		1	The I2C module runs free; that is, it continues to operate when a breakpoint occurs.
13	STT		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 14-6). 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.
		0	In the master mode, STT is automatically cleared after the START condition has been generated.
		1	In the master mode, setting STT to 1 causes the I2C module to generate a START condition on the I2C-bus.
12	Reserved		This reserved bit location is always read as a 0. A value written to this bit has no effect.
11	STP		STOP condition bit (only applicable when the I2C module is a master). In the master mode, the RM, STT, and STP bits determine when the I2C module starts and stops data transmissions (see Table 14-6). 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. 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 ater the I2CSTR[SCD] bit is set. If the STOP bit is not checked prior to initiating a new message, the I2C module could become confused. To avoid disrupting the I2C state machine, the user must wait until this bit is clear before initiating a new message.
		0	STP is automatically cleared after the STOP condition has been generated.
		1	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.

Table 14-5. I2C Mode Register (I2CMDR) Field Descriptions (continued)

Bit	Field	Value	Description
10	MST		Master mode bit. 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.
		0	Slave mode. The I2C module is a slave and receives the serial clock from the master.
		1	Master mode. The I2C module is a master and generates the serial clock on the SCL pin.
9	TRX		Transmitter mode bit. When relevant, TRX selects whether the I2C module is in the transmitter mode or the receiver mode. Table 14-7 summarizes when TRX is used and when it is a don't care.
		0	Receiver mode. The I2C module is a receiver and receives data on the SDA pin.
		1	Transmitter mode. The I2C module is a transmitter and transmits data on the SDA pin.
8	XA		Expanded address enable bit.
		0	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).
		1	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).
7	RM		Repeat mode bit (only applicable when the I2C module is a master-transmitter). The RM, STT, and STP bits determine when the I2C module starts and stops data transmissions (see Table 14-6).
		0	Nonrepeat mode. The value in the data count register (I2CCNT) determines how many bytes are received/transmitted by the I2C module.
		1	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.
6	DLB		Digital loopback mode bit. The effects of this bit are shown in Figure 14-15.
		0	Digital loopback mode is disabled.
		1	Digital loopback mode is enabled. For proper operation in this mode, the MST bit must be 1. In the digital loopback mode, data transmitted out of I2CDXR is received in I2CDRR after n device cycles by an internal path, where: n = ((I2C input clock frequency/module clock frequency) × 8) The transmit clock is also the receive clock. The address transmitted on the SDA pin is the address in I2COAR. Note: The free data format (FDF = 1) is not supported in the digital loopback mode.
5	IRS		I2C module reset bit.
		0	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.
		1	The I2C module is enabled. This has the effect of releasing the I2C bus if the I2C peripheral is holding it.
4	STB		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.
		0	The I2C module is not in the START byte mode.
		1	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: 1. A START condition 2. A START byte (0000 0001b) 3. A dummy acknowledge clock pulse 4. A repeated START condition Then, as normal, the I2C module sends the slave address that is in I2CSAR.
3	FDF		Free data format mode bit.
		0	Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit.
		1	Free data format mode is enabled. Transfers have the free data (no address) format described in Section 14.2.5. The free data format is not supported in the digital loopback mode (DLB=1).

Table 14-5. I2C Mode Register (I2CMDR) Field Descriptions (continued)

Bit	Field	Value	Description
2-0	BC		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.
		000	8 bits per data byte
		001	1 bit per data byte
		010	2 bits per data byte
		011	3 bits per data byte
		100	4 bits per data byte
		101	5 bits per data byte
		110	6 bits per data byte
		111	7 bits per data byte

Table 14-6. Master-Transmitter/Receiver Bus Activity Defined by the RM, STT, and STP Bits of I2CMDR

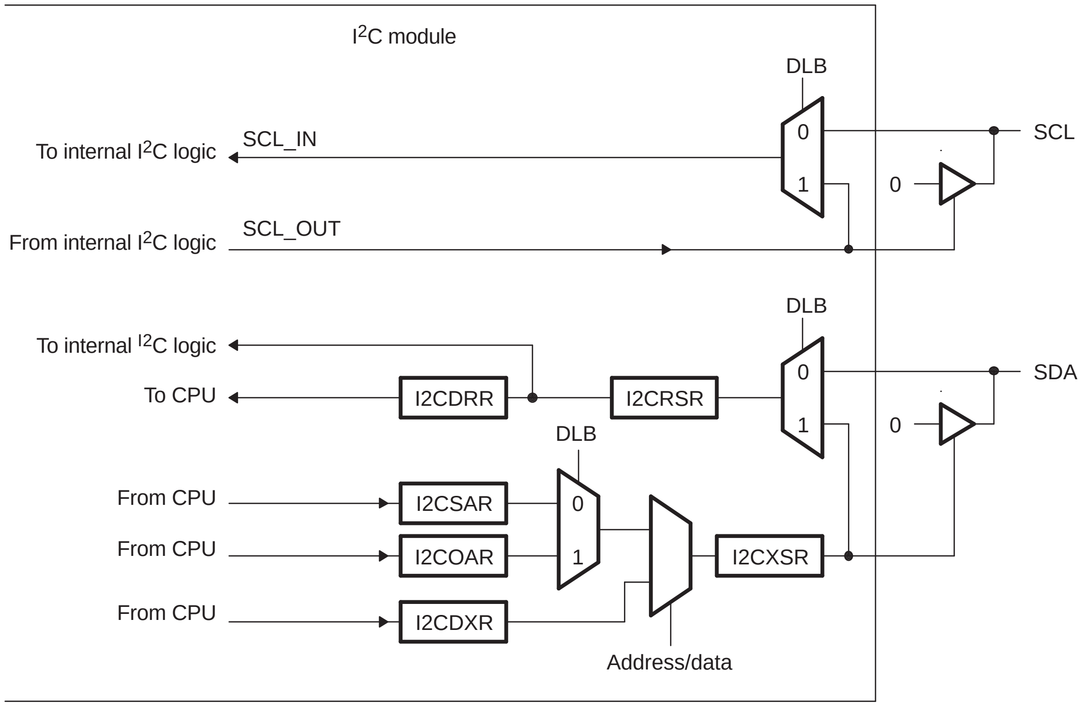
RM	STT	STP	Bus Activity[1]	Description
0	0	0	None	No activity
0	0	1	P	STOP condition
0	1	0	S-A-D..(n)..D.	START condition, slave address, n data bytes (n = value in I2CCNT)
0	1	1	S-A-D..(n)..D-P	START condition, slave address, n data bytes, STOP condition (n = value in I2CCNT)
1	0	0	None	No activity
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;

Table 14-7. 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.

Figure 14-15. Pin Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit



14.5.2 *I2C Extended Mode Register (I2CEMDR)*

The I2C extended mode register is shown in Figure 14-16 and described in Table 14-8.

Figure 14-16. I2C Extended Mode Register (I2CEMDR)

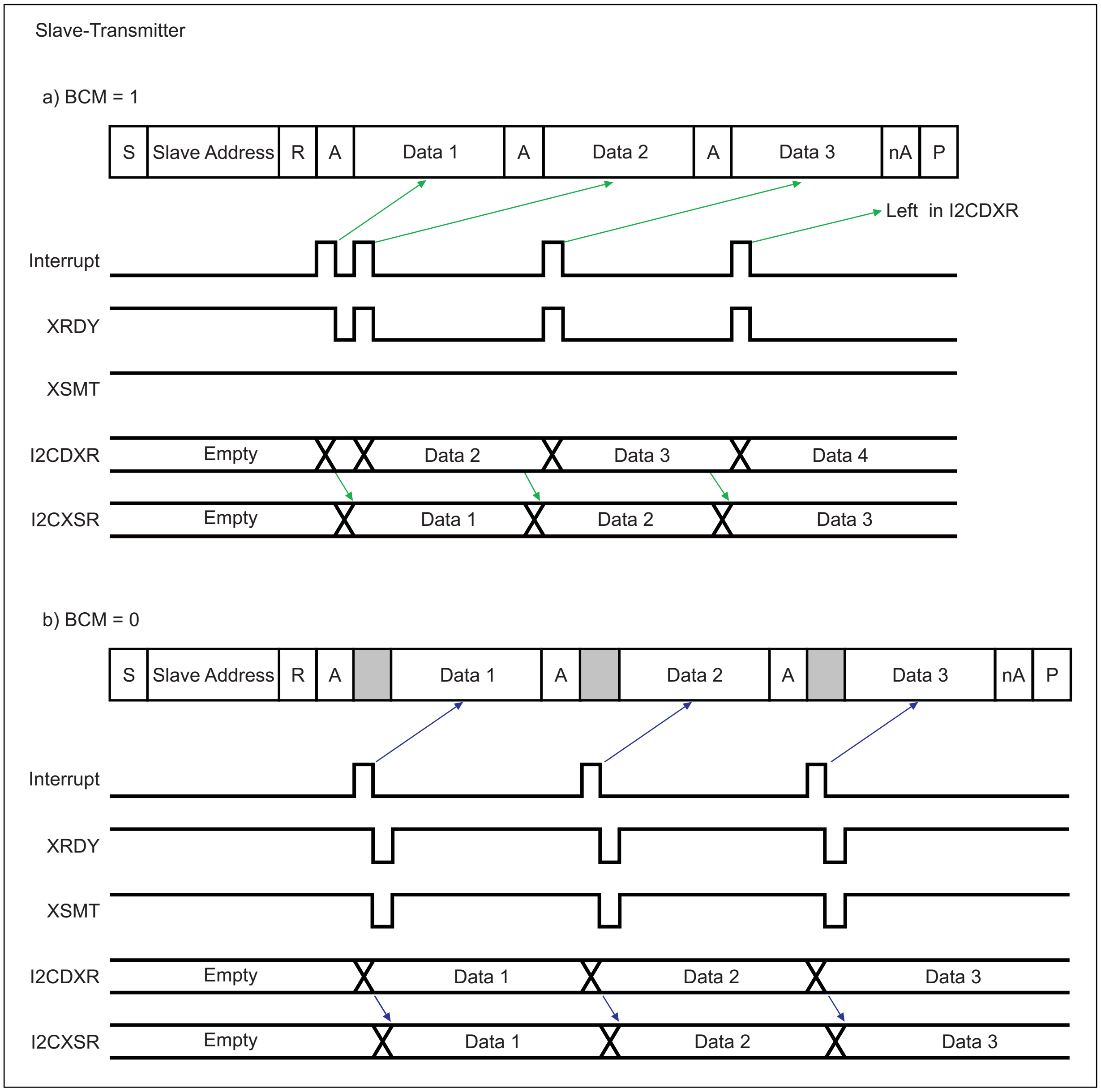
15 1	0
Reserved	BCM
R-0	R/W-1

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

Table 14-8. I2C Extended Mode Register (I2CEMDR) Field Descriptions

Bit	Field	Value	Description
15-1	Reserved		Any writes to these bit(s) must always have a value of 0.
0	BCM		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. See Figure 14-17 for details

Figure 14-17. BCM Bit, Slave Transmitter Mode



14.5.3 I2C Interrupt Enable Register (I2CIER)

I2CIER is used by the CPU to individually enable or disable I2C interrupt requests. The bits of I2CIER are shown and described in Figure 14-18 and Table 14-9, respectively.

Figure 14-18. I2C Interrupt Enable Register (I2CIER)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved	AAS	SCD	XRDY	RRDY	ARDY	NACK	AL
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 14-9. I2C Interrupt Enable Register (I2CIER) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
6	AAS		Addressed as slave interrupt enable bit
		0	Interrupt request disabled
		1	Interrupt request enabled
5	SCD		Stop condition detected interrupt enable bit
		0	Interrupt request disabled
		1	Interrupt request enabled
4	XRDY		Transmit-data-ready interrupt enable bit. This bit should not be set when using FIFO mode.
		0	Interrupt request disabled
		1	Interrupt request enabled
3	RRDY		Receive-data-ready interrupt enable bit. This bit should not be set when using FIFO mode.
		0	Interrupt request disabled
		1	Interrupt request enabled
2	ARDY		Register-access-ready interrupt enable bit
		0	Interrupt request disabled
		1	Interrupt request enabled
1	NACK		No-acknowledgment interrupt enable bit
		0	Interrupt request disabled
		1	Interrupt request enabled
0	AL		Arbitration-lost interrupt enable bit
		0	Interrupt request disabled
		1	Interrupt request enabled

14.5.4 I2C Status Register (I2CSTR)

The I2C status register (I2CSTR) is a 16-bit register used to determine which interrupt has occurred and to read status information. The bits of I2CSTR are shown and described in Figure 14-19 and Table 14-10, respectively.

Figure 14-19. I2C Status Register (I2CSTR)

15	14	13	12	11	10	9	8
Reserved	SDIR	NACKSNT	BB	RSFULL	XSMT	AAS	AD0
R-0	R/W1C-0	R/W1C-0	R-0	R-0	R-1	R-0	R-0

7	6	5	4	3	2	1	0
Reserved		SCD	XRDY	RRDY	ARDY	NACK	AL
R-0		R/W1C-0	R-1	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0

LEGEND: R/W = Read/Write; W1C = Write 1 to clear (writing 0 has no effect); R = Read only; -*n* = value after reset

Table 14-10. I2C Status Register (I2CSTR) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	This reserved bit location is always read as zeros. A value written to this bit has no effect.
14	SDIR		Slave direction bit
		0	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.
		1	I2C is addressed as a slave transmitter.
13	NACKSNT		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 Section 14.5.1).
		0	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).
		1	NACK sent: A no-acknowledge bit was sent during the acknowledge cycle on the I2C-bus.
12	BB		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.
		0	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.
		1	Bus busy: The I2C module has received or transmitted a START bit on the bus.
11	RSFULL		Receive shift register full bit. 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.
		0	No overrun detected. RSFULL is cleared by any one of the following events: • I2CDRR is read is read by the CPU. Emulator reads of the I2CDRR do not affect this bit. • The I2C module is reset.
		1	Overrun detected
10	XSMT		Transmit shift register empty bit. 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.
		0	Underflow detected (empty)
		1	No underflow detected (not empty). XSMT is set by one of the following events: • Data is written to I2CDXR. • The I2C module is reset

Table 14-10. I2C Status Register (I2CSTR) Field Descriptions (continued)

Bit	Field	Value	Description
9	AAS		Addressed-as-slave bit
		0	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.
		1	The I2C module has recognized its own slave address or an address of all zeros (general call). The AAS bit is also set if the first byte has been received in the free data format (FDF = 1 in I2CMDR).
8	AD0		Address 0 bits
		0	AD0 has been cleared by a START or STOP condition.
		1	An address of all zeros (general call) is detected.
7-6	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
5	SCD		Stop condition detected bit. 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.
		0	STOP condition not detected since SCD was last cleared. SCD is cleared by any one of the following events: • I2CISRC is read by the CPU when it contains the value 110b (stop condition detected). Emulator reads of the I2CISRC do not affect this bit. • SCD is manually cleared. To clear this bit, write a 1 to it. • The I2C module is reset.
		1	A STOP condition has been detected on the I2C bus.
4	XRDY		Transmit-data-ready interrupt flag bit. 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 (see Section 14.3.1). When in FIFO mode, use TXFFINT instead.
		0	I2CDXR not ready. XRDY is cleared when data is written to I2CDXR.
		1	I2CDXR ready: Data has been copied from I2CDXR to I2CXSR. XRDY is also forced to 1 when the I2C module is reset.
3	RRDY		Receive-data-ready interrupt flag bit. 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 (see Section 14.3.1). When in FIFO mode, use RXFFINT instead.
		0	I2CDRR not ready. RRDY is cleared by any one of the following events: • I2CDRR is read by the CPU. Emulator reads of the I2CDRR do not affect this bit. • RRDY is manually cleared. To clear this bit, write a 1 to it. • The I2C module is reset.
		1	I2CDRR ready: Data has been copied from I2CRSR to I2CDRR.
2	ARDY		Register-access-ready interrupt flag bit (only applicable when the I2C module is in the master mode). 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 (see Section 14.3.1)
		0	The registers are not ready to be accessed. ARDY is cleared by any one of the following events: • The I2C module starts using the current register contents. • ARDY is manually cleared. To clear this bit, write a 1 to it. • The I2C module is reset.
		1	The registers are ready to be accessed. 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). In the repeat mode (RM = 1): ARDY is set at the end of each byte transmitted from I2CDXR.

Table 14-10. I2C Status Register (I2CSTR) Field Descriptions (continued)

Bit	Field	Value	Description
1	NACK		No-acknowledgment interrupt flag bit. NACK applies when the I2C module is a transmitter (master or slave). NACK indicates whether the I2C module has detected an acknowledge bit (ACK) or a no-acknowledge bit (NACK) from the receiver. The CPU can poll NACK or use the NACK interrupt request (see Section 14.3.1).
		0	ACK received/NACK not received. This bit is cleared by any one of the following events: • An acknowledge bit (ACK) has been sent by the receiver. • NACK is manually cleared. To clear this bit, write a 1 to it. • The CPU reads the interrupt source register (I2CISRC) and the register contains the code for a NACK interrupt. Emulator reads of the I2CISRC do not affect this bit. • The I2C module is reset.
		1	NACK bit received. The hardware detects that a no-acknowledge (NACK) bit has been received. Note: While the I2C module performs a general call transfer, NACK is 1, even if one or more slaves send acknowledgment.
0	AL		Arbitration-lost interrupt flag bit (only applicable when the I2C module is a master-transmitter). AL primarily indicates when the I2C module has lost an arbitration contest with another master-transmitter. The CPU can poll AL or use the AL interrupt request (see Section 14.3.1)
		0	Arbitration not lost. AL is cleared by any one of the following events: • AL is manually cleared. To clear this bit, write a 1 to it. • The CPU reads the interrupt source register (I2CISRC) and the register contains the code for an AL interrupt. Emulator reads of the I2CISRC do not affect this bit. • The I2C module is reset.
		1	Arbitration lost. AL is set by any one of the following events: • The I2C module senses that it has lost an arbitration with two or more competing transmitters that started a transmission almost simultaneously. • The I2C module attempts to start a transfer while the BB (bus busy) bit is set to 1. When AL becomes 1, the MST and STP bits of I2CMDR are cleared, and the I2C module becomes a slave-receiver.

The I2C peripheral cannot detect a START or STOP condition when it is in reset, i.e. the IRS bit is set to 0. Therefore, the BB bit will remain in the state it was at when the peripheral was placed in reset. The BB bit will stay in that state until the I2C peripheral is taken out of reset, i.e. the IRS bit is set to 1, and a START or STOP condition is detected on the I2C bus.

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 certain period to detect the actual bus status before starting the first data transfer. Set this 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.

14.5.5 I2C Interrupt Source Register (I2CISRC)

The I2C interrupt source register (I2CISRC) is a 16-bit register used by the CPU to determine which event generated the I2C interrupt. For more information about these events, see the descriptions of the I2C interrupt requests in Table 14-3.

Figure 14-20. I2C Interrupt Source Register (I2CISRC)

15 - 12	11 - 8	7 - 3	2 - 0
Reserved	Reserved	Reserved	INTCODE
R-0	R/W-0	R-0	R-0

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

Table 14-11. I2C Interrupt Source Register (I2CISRC) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
11-8	Reserved		These reserved bit locations should always be written as zeros.
7-3	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
2-0	INTCODE		Interrupt code bits. The binary code in INTCODE indicates the event that generated an I2C interrupt.
		000	None
		001	Arbitration lost
		010	No-acknowledgment condition detected
		011	Registers ready to be accessed
		100	Receive data ready
		101	Transmit data ready
		110	Stop condition detected
		111	Addressed as slave
			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.
			Emulator reads will not affect the state of this field or of the status bits in the I2CSTR register.

14.5.6 I2C Prescaler Register (I2CPSC)

The I2C prescaler register (I2CPSC) is a 16-bit register (see Figure 14-21) 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. Table 14-12 lists the bit descriptions. For more details about the module clock, see Section 14.1.3.

IPSC must be initialized while the I2C module is in reset (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.

Figure 14-21. I2C Prescaler Register (I2CPSC)

15 - 8	7 - 0
Reserved	IPSC
R-0	R/W-0

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

Table 14-12. I2C Prescaler Register (I2CPSC) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
7-0	IPSC		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 I2CMDR).

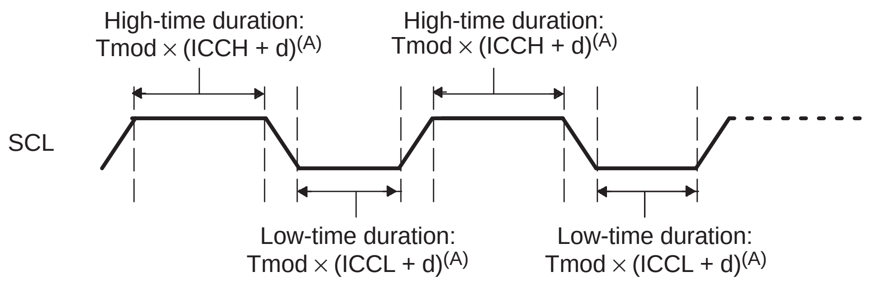
NOTE: To meet all of the I2C protocol timing specifications, the module clock must be configured between 7-12 MHz.

14.5.7 I2C Clock Divider Registers (I2CCLKL and I2CCLKH)

As explained in Section 14.1.3, when the I2C module is a master, the module clock is divided down for use as the master clock on the SCL pin. As shown in Figure 14-22, the shape of the master clock depends on two divide-down values:

- ICCL in I2CCLKL (summarized by Figure 14-23 and Table 14-13). For each master clock cycle, ICCL determines the amount of time the signal is low.
- ICCH in I2CCLKH (summarized by Figure 14-24 and Table 14-14). For each master clock cycle, ICCH determines the amount of time the signal is high.

Figure 14-22. The Roles of the Clock Divide-Down Values (ICCL and ICCH)



A As described in Section 14.5.7.1, Tmod is the module clock period, and d is 5, 6, or 7.

Figure 14-23. I2C Clock Low-Time Divider Register (I2CCLKL)

15 ... 0
ICCL
R/W-0

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

Table 14-13. I2C Clock Low-Time Divider Register (I2CCLKL) Field Description

Bit	Field	Value	Description
15-0	ICCL		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 5, 6, or 7 as described in Section 14.5.7.1. Note: These bits must be set to a non-zero value for proper I2C clock operation.

Figure 14-24. I2C Clock High-Time Divider Register (I2CCLKH)

15 ... 0
ICCH
R/W-0

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

Table 14-14. I2C Clock High-Time Divider Register (I2CCLKH) Field Description

Bit	Field	Value	Description
15-0	ICCH		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 (ICCH + d). d is 5, 6, or 7 as described in Section 14.5.7.1. Note: These bits must be set to a non-zero value for proper I2C clock operation.

14.5.7.1 Formula for the Master Clock Period

The period of the master clock (Tmst) is a multiple of the period of the module clock (Tmod):

$$T_{mst} = T_{mod} \times [(\mathrm{ICCL} + d) + (\mathrm{ICCH} + d)]$$

$$T_{mst} = \frac{(\mathrm{IPSC} + 1)\ [(\mathrm{ICCL} + d) + (\mathrm{ICCH} + d)]}{\mathrm{I2C\ input\ clock\ frequency}}$$

where d depends on the divide-down value IPSC, as shown in Table 14-15. IPSC is described in Section 14.5.6.

Table 14-15. Dependency of Delay d on the Divide-Down Value IPSC

IPSC	d
0	7
1	6
Greater than 1	5

14.5.8 I2C Slave Address Register (I2CSAR)

The I2C slave address register (I2CSAR) is a register for storing the next slave address that will be transmitted by the I2C module when it is a master. It is a 16-bit register with the format shown in Figure 14-25. As described in Table 14-16, 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 14-25. I2C Slave Address Register (I2CSAR)

15 - 10	9 - 0
Reserved	SAR
R-0	R/W-3FFh

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

Table 14-16. I2C Slave Address Register (I2CSAR) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
9-0	SAR		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.

14.5.9 I2C Own Address Register (I2COAR)

The I2C own address register (I2COAR) is a 16-bit register. Figure 14-26 shows the format of I2COAR, and Table 14-17 describes its bit fields. 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 14-26. I2C Own Address Register (I2COAR)

15 10	9 0
Reserved	OAR
R-0	R/W-0

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

Table 14-17. I2C Own Address Register (I2COAR) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
9-0	OAR		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.

14.5.10 I2C Data Count Register (I2CCNT)

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 bits of I2CCNT are shown and described in Figure 14-27 and Table 14-18, respectively.

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 14-27. I2C Data Count Register (I2CCNT)

15 0
ICDC
R/W-0

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

Table 14-18. I2C Data Count Register (I2CCNT) Field Descriptions

Bit	Field	Value	Description
15-0	ICDC		Data count value. ICDC indicates the number of data bytes to transfer or receive. The value in I2CCNT is a don't care when the RM bit in I2CMDR is set to 1.
		0000h	The start value loaded to the internal data counter is 65536.
		0001h-FFFFh	The start value loaded to internal data counter is 1-65535.

14.5.11 I2C Data Receive Register (I2CDRR)

I2CDRR (see Figure 14-28 and Table 14-19) 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 I2CMDR. 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 14-28. I2C Data Receive Register (I2CDRR)

15 – 8	7 – 0
Reserved	DATA
R-0	R-0

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

Table 14-19. I2C Data Receive Register (I2CDRR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
7-0	DATA		Receive data

14.5.12 I2C Data Transmit Register (I2CDXR)

The CPU writes transmit data to I2CDXR (see Figure 14-29 and Table 14-20). 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 14-29. I2C Data Transmit Register (I2CDXR)

15 – 8	7 – 0
Reserved	DATA
R-0	R/W-0

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

Table 14-20. I2C Data Transmit Register (I2CDXR) Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		These reserved bit locations are always read as zeros. A value written to this field has no effect.
7-0	DATA		Transmit data

14.5.13 I2C Transmit FIFO Register (I2CFFTX)

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. The bit fields are shown in Figure 14-30 and described in Table 14-21.

Figure 14-30. I2C Transmit FIFO Register (I2CFFTX)

15	14	13	12	11	10	9	8
Reserved	I2CFFEN	TXFFRST	TXFFST4	TXFFST3	TXFFST2	TXFFST1	TXFFST0
R-0	R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL4	TXFFIL3	TXFFIL2	TXFFIL1	TXFFIL0
R-0	R/W1C-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 14-21. I2C Transmit FIFO Register (I2CFFTX) Field Descriptions

Bit	Field	Value	Description
15	Reserved		Reserved. Reads will return a 0, writes have no effect.
14	I2CFFEN		I2C FIFO mode enable bit. This bit must be enabled for either the transmit or the receive FIFO to operate correctly.
		0	Disable the I2C FIFO mode.
		1	Enable the I2C FIFO mode.
13	TXFFRST		I2C transmit FIFO reset bit.
		0	Reset the transmit FIFO pointer to 0000 and hold the transmit FIFO in the reset state.
		1	Enable the transmit FIFO operation.
12-8	TXFFST4-0		Contains the status of the transmit FIFO:
		00xxx	Transmit FIFO contains xxx 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.
7	TXFFINT		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.
		0	Transmit FIFO interrupt condition has not occurred.
		1	Transmit FIFO interrupt condition has occurred.
6	TXFFINTCLR		Transmit FIFO interrupt flag clear bit.
		0	Writes of zeros have no effect. Reads return a 0.
		1	Writing a 1 to this bit clears the TXFFINT flag.
5	TXFFIENA		Transmit FIFO interrupt enable bit.
		0	Disabled. TXFFINT flag does not generate an interrupt when set.
		1	Enabled. TXFFINT flag does generate an interrupt when set.
4-0	TXFFIL4-0		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 these devices has a 4-level transmit FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels. TXFFIL4 and TXFFIL3 are tied to zero.

14.5.14 I2C Receive FIFO Register (I2CFFRX)

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. The bit fields are shown in Figure 14-31 and described in Table 14-22.

Figure 14-31. I2C Receive FIFO Register (I2CFFRX)

15	14	13	12	11	10	9	8
Reserved		RXFFRST	RXFFST4	RXFFST3	RXFFST2	RXFFST1	RXFFST0
R-0		R/W-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL4	RXFFIL3	RXFFIL2	RXFFIL1	RXFFIL0
R-0	R/W1C-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 14-22. I2C Receive FIFO Register (I2CFFRX) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved		Reserved. Reads will return a 0, writes have no effect.
13	RXFFRST		I2C receive FIFO reset bit
		0	Reset the receive FIFO pointer to 0000 and hold the receive FIFO in the reset state.
		1	Enable the receive FIFO operation.
12-8	RXFFST4-0		Contains the status of the receive FIFO:
		00xxx	Receive FIFO contains xxx bytes
		00000	Receive FIFO is empty.
7	RXFFINT		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.
		0	Receive FIFO interrupt condition has not occurred.
		1	Receive FIFO interrupt condition has occurred.
6	RXFFINTCLR		Receive FIFO interrupt flag clear bit.
		0	Writes of zeros have no effect. Reads return a zero.
		1	Writing a 1 to this bit clears the RXFFINT flag.
5	RXFFIENA		Receive FIFO interrupt enable bit.
		0	Disabled. RXFFINT flag does not generate an interrupt when set.
		1	Enabled. RXFFINT flag does generate an interrupt when set.
4-0	RXFFIL4-0		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 these devices has a 4-level receive FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels. RXFFIL4 and RXFFIL3 are tied to zero.

Multichannel Buffered Serial Port (McBSP)

This chapter describes the multichannel buffered serial port (McBSP). This device provides one high-speed multichannel buffered serial port (McBSP) that allows direct interface to codecs and other devices in a system.

Topic **Page**

15.1 Overview

The McBSP consists of a data-flow path and a control path connected to external devices by six pins as shown in Figure 15-1.

Data is communicated to devices interfaced with the McBSP via the data transmit (DX) pin for transmission and via the data receive (DR) pin for reception. Control information in the form of clocking and frame synchronization is communicated via the following pins: CLKX (transmit clock), CLKR (receive clock), FSX (transmit frame synchronization), and FSR (receive frame synchronization).

The CPU and the DMA controller communicate with the McBSP through 16-bit-wide registers accessible via the internal peripheral bus. The CPU or the DMA controller writes the data to be transmitted to the data transmit registers (DXR1, DXR2). Data written to the DXRs is shifted out to DX via the transmit shift registers (XSR1, XSR2). Similarly, receive data on the DR pin is shifted into the receive shift registers (RSR1, RSR2) and copied into the receive buffer registers (RBR1, RBR2). The contents of the RBRs is then copied to the DRRs, which can be read by the CPU or the DMA controller. This allows simultaneous movement of internal and external data communications.

DRR2, RBR2, RSR2, DXR2, and XSR2 are not used (written, read, or shifted) if the serial word length is 8 bits, 12 bits, or 16 bits. For larger word lengths, these registers are needed to hold the most significant bits.

The frame and clock loop-back is implemented at chip level to enable CLKX and FSX to drive CLKR and FSR. If the loop-back is enabled, the CLKR and FSR get their signals from the CLKX and FSX pads; instead of the CLKR and FSR pins.

15.1.1 Features of the McBSP

The McBSP features:

- Full-duplex communication
- Double-buffered transmission and triple-buffered reception, allowing a continuous data stream
- Independent clocking and framing for reception and transmission
- The capability to send interrupts to the CPU and to send DMA events to the DMA controller
- 128 channels for transmission and reception
- Multichannel selection modes that enable or disable block transfers in each of the channels
- Direct interface to industry-standard codecs, analog interface chips (AICs), and other serially connected A/D and D/A devices
- Support for external generation of clock signals and frame-synchronization signals
- A programmable sample rate generator for internal generation and control of clock signals and frame-synchronization signals
- Programmable polarity for frame-synchronization pulses and clock signals
- Direct interface to:
  - T1/E1 framers
  - IOM-2 compliant devices
  - AC97-compliant devices (the necessary multiphase frame capability is provided)
  - I2S compliant devices
  - SPI devices
- A wide selection of data sizes: 8, 12, 16, 20, 24, and 32 bits

NOTE: A value of the chosen data size is referred to as a *serial word* or *word* throughout the McBSP documentation. Elsewhere, *word* is used to describe a 16-bit value.

- μ-law and A-law companding
- The option of transmitting/receiving 8-bit data with the LSB first
- Status bits for flagging exception/error conditions

- ABIS mode is not supported.

15.1.2 McBSP Pins/Signals

Table 15-1 describes the McBSP interface pins and some internal signals.

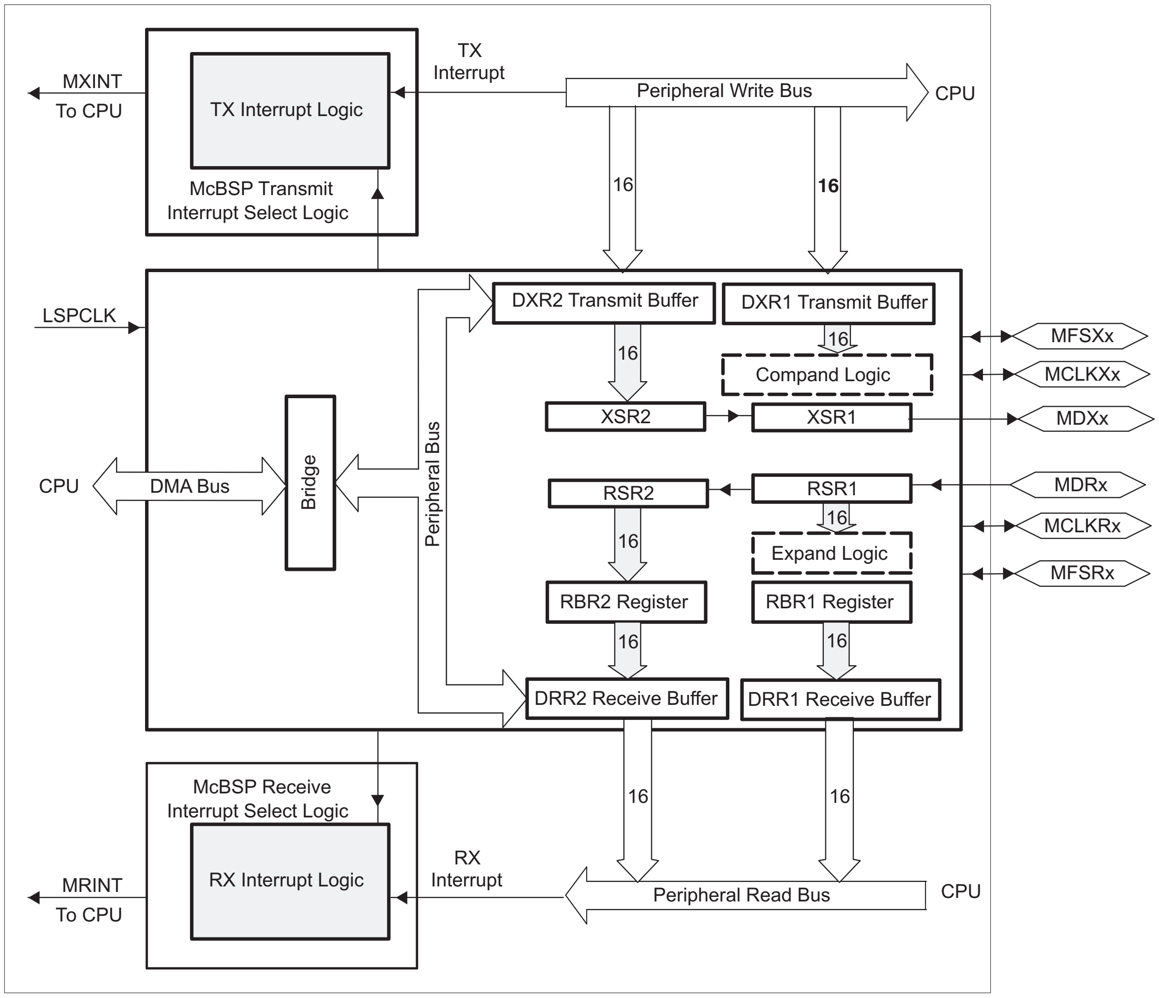
Table 15-1. McBSP Interface Pins/Signals

McBSP-A Pin	Type	Description
MCLKRA	I/O	Supplying or reflecting the receive clock; supplying the input clock of the sample rate generator
MCLKXA	I/O	Supplying or reflecting the transmit clock; supplying the input clock of the sample rate generator
MDRA	I	Serial data receive pin
MDXA	O	Serial data transmit pin
MFSRA	I/O	Supplying or reflecting the receive frame-sync signal; controlling sample rate generator synchronization for the case when GSYNC = 1 (see Section 15.4.3)
MFSXA	I/O	Supplying or reflecting the transmit frame-sync signal
CPU Interrupt Signals		
MRINT		Receive interrupt to CPU
MXINT		Transmit interrupt to CPU
DMA Events		
REVT		Receive synchronization event to DMA
XEVT		Transmit synchronization event to DMA

15.1.2.1 McBSP Generic Block Diagram

The McBSP consists of a data-flow path and a control path connected to external devices by six pins as shown in Figure 15-1. The figure and the text in this section use generic pin names.

Figure 15-1. Conceptual Block Diagram of the McBSP



A Not available in all devices. See the device-specific data sheet

15.1.3 McBSP Operation

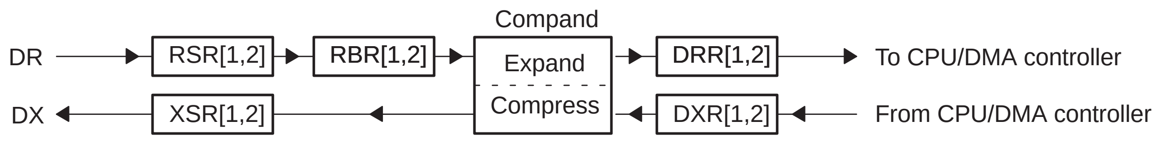
This section addresses the following topics:

- Data transfer process
- Companding (compressing and expanding) data
- Clocking and framing data
- Frame phases
- McBSP reception
- McBSP transmission
- Interrupts and DMA events generated by McBSPs

15.1.4 Data Transfer Process of McBSP

Figure 15-2 shows a diagram of the McBSP data transfer paths. The McBSP receive operation is triple-buffered, and transmit operation is double-buffered. The use of registers varies, depending on whether the defined length of each serial word is 16 bits.

Figure 15-2. McBSP Data Transfer Paths



15.1.4.1 Data Transfer Process for Word Length of 8, 12, or 16 Bits

If the word length is 16 bits or smaller, only one 16-bit register is needed at each stage of the data transfer paths. The registers DRR2, RBR2, RSR2, DXR2, and XSR2 are not used (written, read, or shifted).

Receive data arrives on the DR pin and is shifted into receive shift register 1 (RSR1). Once a full word is received, the content of RSR1 is copied to receive buffer register 1 (RBR1) if RBR1 is not full with previous data. RBR1 is then copied to data receive register 1 (DRR1), unless the previous content of DRR1 has not been read by the CPU or the DMA controller. If the companding feature of the McBSP is implemented, the required word length is 8 bits and receive data is expanded into the appropriate format before being passed from RBR1 to DRR1. For more details about reception, see Section 15.3.5.

Transmit data is written by the CPU or the DMA controller to data transmit register 1 (DXR1). If there is no previous data in transmit shift register (XSR1), the value in DXR1 is copied to XSR1; otherwise, DXR1 is copied to XSR1 when the last bit of the previous data is shifted out on the DX pin. If selected, the companding module compresses 16-bit data into the appropriate 8-bit format before passing it to XSR1. After transmit frame synchronization, the transmitter begins shifting bits from XSR1 to the DX pin. For more details about transmission, see Section 15.3.6.

15.1.4.2 Data Transfer Process for Word Length of 20, 24, or 32 Bits

If the word length is larger than 16 bits, two 16-bit registers are needed at each stage of the data transfer paths. The registers DRR2, RBR2, RSR2, DXR2, and XSR2 are needed to hold the most significant bits.

Receive data arrives on the DR pin and is shifted first into RSR2 and then into RSR1. Once the full word is received, the contents of RSR2 and RSR1 are copied to RBR2 and RBR1, respectively, if RBR1 is not full. Then the contents of RBR2 and RBR1 are copied to DRR2 and DRR1, respectively, unless the previous content of DRR1 has not been read by the CPU or the DMA controller. The CPU or the DMA controller must read data from DRR2 first and then from DRR1. When DRR1 is read, the next RBR-to-DRR copy occurs. For more details about reception, see Section 15.3.5.

For transmission, the CPU or the DMA controller must write data to DXR2 first and then to DXR1. When new data arrives in DXR1, if there is no previous data in XSR1, the contents of DXR2 and DXR1 are copied to XSR2 and XSR1, respectively; otherwise, the contents of the DXRs are copied to the XSRs when the last bit of the previous data is shifted out on the DX pin. After transmit frame synchronization, the transmitter begins shifting bits from the XSRs to the DX pin. For more details about transmission, see Section 15.3.6.

15.1.5 Companding (Compressing and Expanding) Data

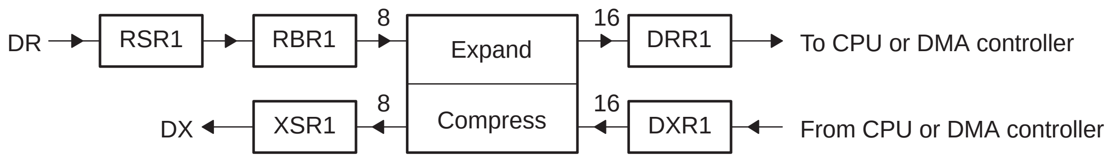
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM are part of the CCITT G.711 recommendation.

A-law and μ-law allow 13 bits and 14 bits of dynamic range, respectively. Any values outside this range are set to the most positive or most negative value. Thus, for companding to work best, the data transferred to and from the McBSP via the CPU or DMA controller must be at least 16 bits wide.

The µ-law and A-law formats both encode data into 8-bit code words. Companded data is always 8 bits wide; the appropriate word length bits (RWDLEN1, RWDLEN2, XWDLEN1, XWDLEN2) must therefore be set to 0, indicating an 8-bit wide serial data stream. If companding is enabled and either of the frame phases does not have an 8-bit word length, companding continues as if the word length is 8 bits.

Figure 15-3 illustrates the companding processes. When companding is chosen for the transmitter, compression occurs during the process of copying data from DXR1 to XSR1. The transmit data is encoded according to the specified companding law (A-law or µ-law). When companding is chosen for the receiver, expansion occurs during the process of copying data from RBR1 to DRR1. The receive data is decoded to twos-complement format.

Figure 15-3. Companding Processes



15.1.5.1 Companding Formats

For reception, the 8-bit compressed data in RBR1 is expanded to left-justified 16-bit data in DRR1. The receive sign-extension and justification mode specified in RJUST is ignored when companding is used.

For transmission using µ-law compression, the 14 data bits must be left-justified in DXR1 and that the remaining two low-order bits are filled with 0s as shown in Figure 15-4.

Figure 15-4. µ-Law Transmit Data Companding Format

	15-2	1-0
µ-law format in DXR1	Value	00

For transmission using A-law compression, the 13 data bits must be left-justified in DXR1, with the remaining three low-order bits filled with 0s as shown in Figure 15-5.

Figure 15-5. A-Law Transmit Data Companding Format

	15-3	2-0
A-law format in DXR1	Value	000

15.1.5.2 Capability to Compand Internal Data

If the McBSP is otherwise unused (the serial port transmit and receive sections are reset), the companding hardware can compand internal data. This can be used to:

- Convert linear to the appropriate µ-law or A-law format
- Convert µ-law or A-law to the linear format
- Observe the quantization effects in companding by transmitting linear data and compressing and re-expanding this data. This is useful only if both XCOMPAND and RCOMPAND enable the same companding format.

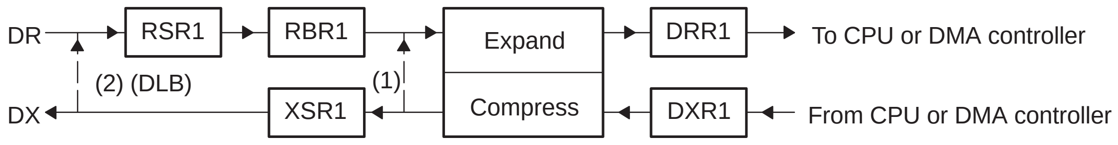
Figure 15-6 shows two methods by which the McBSP can compand internal data. Data paths for these two methods are used to indicate:

- When both the transmit and receive sections of the serial port are reset, DRR1 and DXR1 are connected internally through the companding logic. Values from DXR1 are compressed, as selected by XCOMPAND, and then expanded, as selected by RCOMPAND. RRDY and XRDY bits are not set. However, data is available in DRR1 within four CPU clocks after being written to DXR1.

  The advantage of this method is its speed. The disadvantage is that there is no synchronization available to the CPU and DMA to control the flow. DRR1 and DXR1 are internally connected if the (X/R)COMPAND bits are set to 10b or 11b (compand using µ-law or A-law).

- The McBSP is enabled in digital loopback mode with companding appropriately enabled by RCOMPAND and XCOMPAND. Receive and transmit interrupts (RINT when RINTM = 0 and XINT when XINTM = 0) or synchronization events (REVT and XEVT) allow synchronization of the CPU or DMA to these conversions, respectively. Here, the time for this companding depends on the serial bit rate selected.

Figure 15-6. Two Methods by Which the McBSP Can Compand Internal Data



15.1.5.3 Reversing Bit Order: Option to Transfer LSB First

Generally, the McBSP transmits or receives all data with the most significant bit (MSB) first. However, certain 8-bit data protocols (that do not use companded data) require the least significant bit (LSB) to be transferred first. If you set XCOMPAND = 01b in XCR2, the bit ordering of 8-bit words is reversed (LSB first) before being sent from the serial port. If you set RCOMPAND = 01b in RCR2, the bit ordering of 8-bit words is reversed during reception. Similar to companding, this feature is enabled only if the appropriate word length bits are set to 0, indicating that 8-bit words are to be transferred serially. If either phase of the frame does not have an 8-bit word length, the McBSP assumes the word length is eight bits, and LSB-first ordering is done.

15.2 Clocking and Framing Data

This section explains basic concepts and terminology important for understanding how McBSP data transfers are timed and delimited.

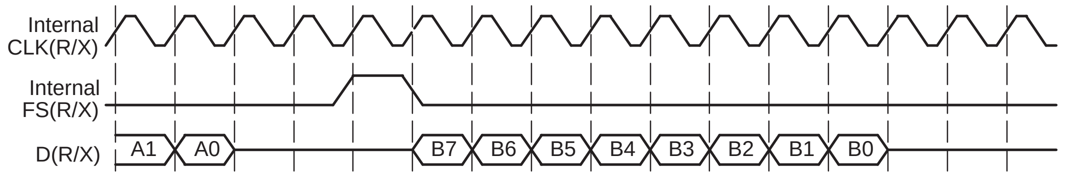
15.2.1 Clocking

Data is shifted one bit at a time from the DR pin to the RSR(s) or from the XSR(s) to the DX pin. The time for each bit transfer is controlled by the rising or falling edge of a clock signal.

The receive clock signal (CLKR) controls bit transfers from the DR pin to the RSR(s). The transmit clock signal (CLKX) controls bit transfers from the XSR(s) to the DX pin. CLKR or CLKX can be derived from a pin at the boundary of the McBSP or derived from inside the McBSP. The polarities of CLKR and CLKX are programmable.

In the example in Figure 15-7, the clock signal controls the timing of each bit transfer on the pin.

Figure 15-7. Example - Clock Signal Control of Bit Transfer Timing



NOTE: The McBSP cannot operate at a frequency faster than ½ the LSPCLK frequency. When driving CLKX or CLKR at the pin, choose an appropriate input clock frequency. When using the internal sample rate generator for CLKX and/or CLKR, choose an appropriate input clock frequency and divide down value (CLKDV) (i.e., be certain that CLKX or CLKR ≤ LSPCLK/2).

15.2.2 Serial Words

Bits traveling between a shift register (RSR or XSR) and a data pin (DR or DX) are transferred in a group called a serial word. You can define how many bits are in a word.

Bits coming in on the DR pin are held in RSR until RSR holds a full serial word. Only then is the word passed to RBR (and ultimately to the DRR).

During transmission, XSR does not accept new data from DXR until a full serial word has been passed from XSR to the DX pin.

In the example in Figure 15-7, an 8-bit word size was defined (see bits 7 through 0 of word B being transferred).

15.2.3 Frames and Frame Synchronization

One or more words are transferred in a group called a frame. You can define how many words are in a frame.

All of the words in a frame are sent in a continuous stream. However, there can be pauses between frame transfers. The McBSP uses frame-synchronization signals to determine when each frame is received/transmitted. When a pulse occurs on a frame-synchronization signal, the McBSP begins receiving/transmitting a frame of data. When the next pulse occurs, the McBSP receives/transmits the next frame, and so on.

Pulses on the receive frame-synchronization (FSR) signal initiate frame transfers on DR. Pulses on the transmit frame-sync (FSX) signal initiate frame transfers on DX. FSR or FSX can be derived from a pin at the boundary of the McBSP or derived from inside the McBSP.

In the example in Figure 15-7, a one-word frame is transferred when a frame-synchronization pulse occurs.

In McBSP operation, the inactive-to-active transition of the frame-synchronization signal indicates the start of the next frame. For this reason, the frame-synchronization signal may be high for an arbitrary number of clock cycles. Only after the signal is recognized to have gone inactive, and then active again, does the next frame synchronization occur.

15.2.4 Generating Transmit and Receive Interrupts

The McBSP can send receive and transmit interrupts to the CPU to indicate specific events in the McBSP. To facilitate detection of frame synchronization, these interrupts can be sent in response to frame-synchronization pulses. Set the appropriate interrupt mode bits to 10b (for reception, RINTM = 10b; for transmission, XINTM = 10b).

15.2.4.1 Detecting Frame-Synchronization Pulses, Even in Reset State

Unlike other serial port interrupt modes, this mode can operate while the associated portion of the serial port is in reset (such as activating RINT when the receiver is in reset). In this case, FSRM/FSXM and FSRP/FSXP still select the appropriate source and polarity of frame synchronization. Thus, even when the serial port is in the reset state, these signals are synchronized to the CPU clock and then sent to the CPU in the form of RINT and XINT at the point at which they feed the receiver and transmitter of the serial port. Consequently, a new frame-synchronization pulse can be detected, and after this occurs the CPU can take the serial port out of reset safely.

15.2.5 Ignoring Frame-Synchronization Pulses

The McBSP can be configured to ignore transmit and/or receive frame-synchronization pulses. To have the receiver or transmitter recognize frame-synchronization pulses, clear the appropriate frame-synchronization ignore bit (RFIG = 0 for the receiver, XFIG = 0 for the transmitter). To have the receiver or transmitter ignore frame-synchronization pulses until the desired frame length or number of words is reached, set the appropriate frame-synchronization ignore bit (RFIG = 1 for the receiver, XFIG = 1 for the transmitter). For more details on unexpected frame-synchronization pulses, see one of the following topics:

- *Unexpected Receive Frame-Synchronization Pulse* (see Section 15.5.3)
- *Unexpected Transmit Frame-Synchronization Pulse* (see Section 15.5.5)

You can also use the frame-synchronization ignore function for data packing (for more details, see Section 15.11.2).

15.2.6 Frame Frequency

The frame frequency is determined by the period between frame-synchronization pulses and is defined as shown by Example 1.

Equation 1: McBSP Frame Frequency

$$\text{Frame Frequency} = \frac{\text{Clock Frequency}}{\text{Number of Clock Cycles Between Frame- Sync Pulses}}$$

The frame frequency can be increased by decreasing the time between frame-synchronization pulses (limited only by the number of bits per frame). As the frame transmit frequency increases, the inactivity period between the data packets for adjacent transfers decreases to zero.

15.2.7 Maximum Frame Frequency

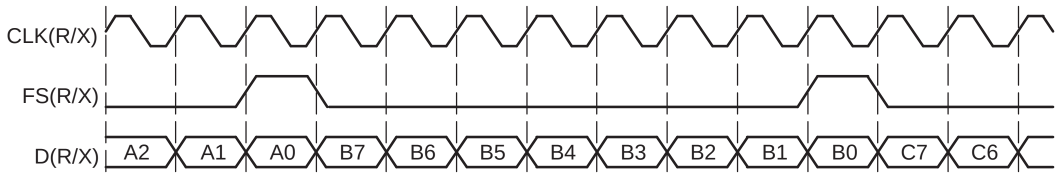
The minimum number of clock cycles between frame synchronization pulses is equal to the number of bits transferred per frame. The maximum frame frequency is defined as shown by Example 2.

Equation 2: McBSP Maximum Frame Frequency

$$\text{Maximum Frame Frequency} = \frac{\text{Clock Frequency}}{\text{Number of Bits Per Frame}}$$

Figure 15-8 shows the McBSP operating at maximum packet frequency. At maximum packet frequency, the data bits in consecutive packets are transmitted contiguously with no inactivity between bits.

Figure 15-8. McBSP Operating at Maximum Packet Frequency



If there is a 1-bit data delay as shown in this figure, the frame-synchronization pulse overlaps the last bit transmitted in the previous frame. Effectively, this permits a continuous stream of data, making frame-synchronization pulses redundant. Theoretically, only an initial frame-synchronization pulse is required to initiate a multipacket transfer.

The McBSP supports operation of the serial port in this fashion by ignoring the successive frame-synchronization pulses. Data is clocked into the receiver or clocked out of the transmitter during every clock cycle.

NOTE: For XDATDLY = 0 (0-bit data delay), the first bit of data is transmitted asynchronously to the internal transmit clock signal (CLKX). For more details, see Section 15.9.12, *Set the Transmit Data Delay*.

15.3 Frame Phases

The McBSP allows you to configure each frame to contain one or two phases. The number of words and the number of bits per word can be specified differently for each of the two phases of a frame, allowing greater flexibility in structuring data transfers. For example, you might define a frame as consisting of one phase containing two words of 16 bits each, followed by a second phase consisting of 10 words of 8 bits each. This configuration permits you to compose frames for custom applications or, in general, to maximize the efficiency of data transfers.

15.3.1 Number of Phases, Words, and Bits Per Frame

Table 15-2 shows which bit-fields in the receive control registers (RCR1 and RCR2) and in the transmit control registers (XCR1 and XCR2) determine the number of phases per frame, the number of words per frame, and number of bits per word for each phase, for the receiver and transmitter. The maximum number of words per frame is 128 for a single-phase frame and 256 for a dual-phase frame. The number of bits per word can be 8, 12, 16, 20, 24, or 32 bits.

Table 15-2. Register Bits That Determine the Number of Phases, Words, and Bits

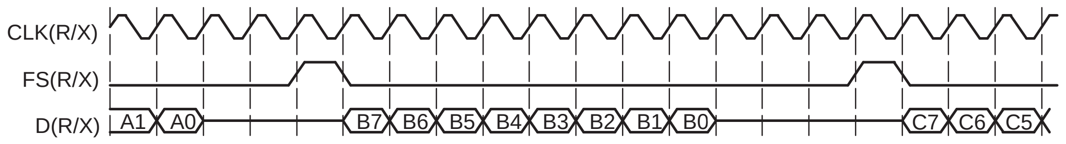
Operation	Number of Phases	Words per Frame Set With	Bits per Word Set With
Reception	1 (RPHASE = 0)	RFRLEN1	RWDLEN1
Reception	2 (RPHASE = 1)	RFRLEN1 and RFRLEN2	RWDLEN1 for phase 1
			RWDLEN2 for phase 2
Transmission	1 (XPHASE = 0)	XFRLEN1	XWDLEN1
Transmission	2 (XPHASE = 1)	XFRLEN1 and XFRLEN2	XWDLEN1 for phase 1
			XWDLEN2 for phase 2

15.3.2 Single-Phase Frame Example

Figure 15-9 shows an example of a single-phase data frame containing one 8-bit word. Because the transfer is configured for one data bit delay, the data on the DX and DR pins are available one clock cycle after FS(R/X) goes active. The figure makes the following assumptions:

- (R/X)PHASE = 0: Single-phase frame
- (R/X)FRLEN1 = 0b: 1 word per frame
- (R/X)WDLEN1 = 000b: 8-bit word length
- (R/X)FRLEN2 and (R/X)WDLEN2 are ignored
- CLK(X/R)P = 0: Receive data clocked on falling edge; transmit data clocked on rising edge
- FS(R/X)P = 0: Active-high frame-synchronization signals
- (R/X)DATDLY = 01b: 1-bit data delay

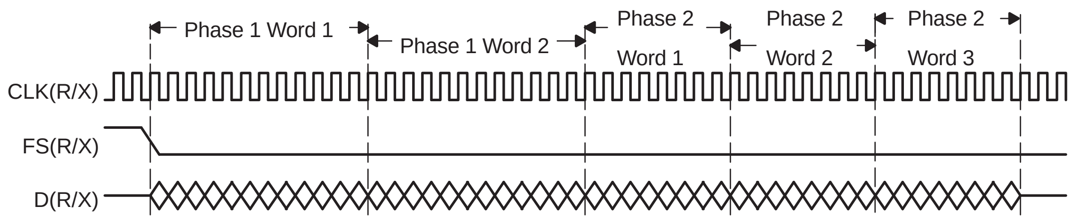
Figure 15-9. Single-Phase Frame for a McBSP Data Transfer



15.3.3 Dual-Phase Frame Example

Figure 15-10 shows an example of a frame where the first phase consists of two words of 12 bits each, followed by a second phase of three words of 8 bits each. The entire bit stream in the frame is contiguous. There are no gaps either between words or between phases.

Figure 15-10. Dual-Phase Frame for a McBSP Data Transfer

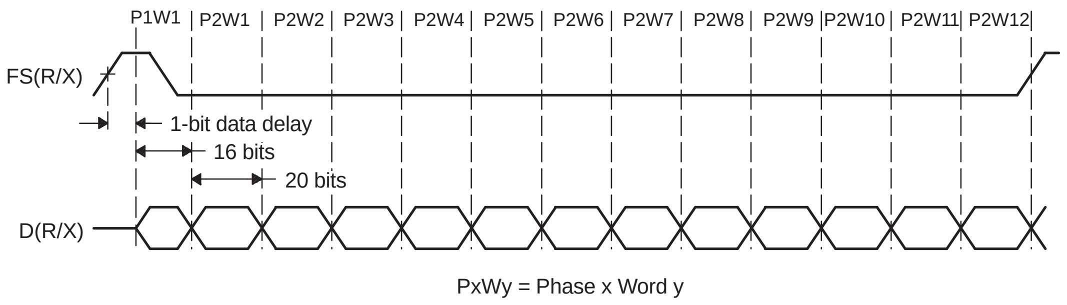


A XRDY gets asserted once per phase. So, if there are 2 phases, XRDY gets asserted twice (once per phase).

15.3.4 Implementing the AC97 Standard With a Dual-Phase Frame

Figure 15-11 shows an example of the Audio Codec '97 (AC97) standard, which uses the dual-phase frame feature. Notice that words, not individual bits, are shown on the D(R/X) signal. The first phase (P1) consists of a single 16-bit word. The second phase (P2) consists of twelve 20-bit words. The phase configurations are listed after the figure.

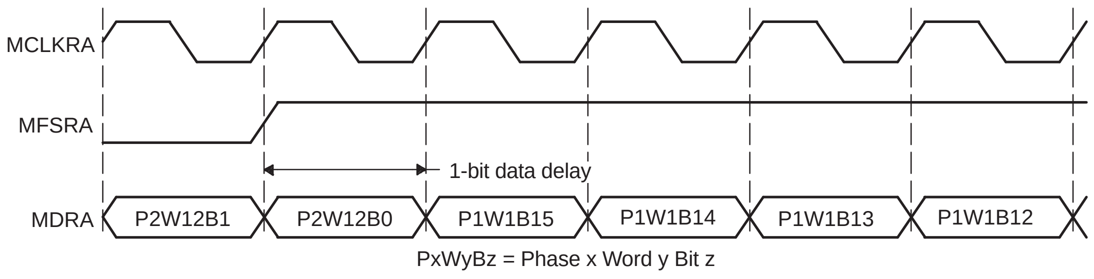
Figure 15-11. Implementing the AC97 Standard With a Dual-Phase Frame



- (R/X)PHASE = 1: Dual-phase frame
- (R/X)FRLEN1 = 0000000b: 1 word in phase 1
- (R/X)WDLEN1 = 010b: 16 bits per word in phase 1
- (R/X)FRLEN2 = 0001011b: 12 words in phase 2
- (R/X)WDLEN2 = 011b: 20 bits per word in phase 2
- CLKRP/CLKXP= 0: Receive data sampled on falling edge of internal CLKR / transmit data clocked on rising edge of internal CLKX
- FSRP/FSXP = 0: Active-high frame-sync signal
- (R/X)DATDLY = 01b: Data delay of 1 clock cycle (1-bit data delay)

Figure 15-12 shows the timing of an AC97-standard data transfer near frame synchronization. In this figure, individual bits are shown on D(R/X). Specifically, the figure shows the last two bits of phase 2 of one frame and the first four bits of phase 1 of the next frame. Regardless of the data delay, data transfers can occur without gaps. The first bit of the second frame (P1W1B15) immediately follows the last bit of the first frame (P2W12B0). Because a 1-bit data delay has been chosen, the transition on the frame-sync signal can occur when P2W12B0 is transferred.

Figure 15-12. Timing of an AC97-Standard Data Transfer Near Frame Synchronization

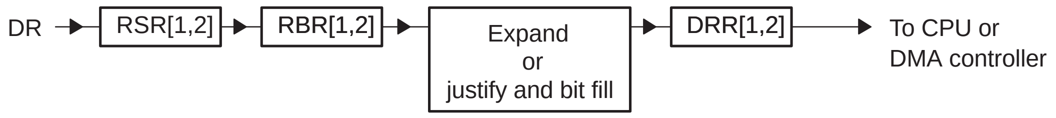


15.3.5 McBSP Reception

This section explains the fundamental process of reception in the McBSP. For details about how to program the McBSP receiver, see *Receiver Configuration* in Section 15.8.

Figure 15-13 and Figure 15-14 show how reception occurs in the McBSP. Figure 15-13 shows the physical path for the data. Figure 15-14 is a timing diagram showing signal activity for one possible reception scenario. A description of the process follows the figures.

Figure 15-13. McBSP Reception Physical Data Path

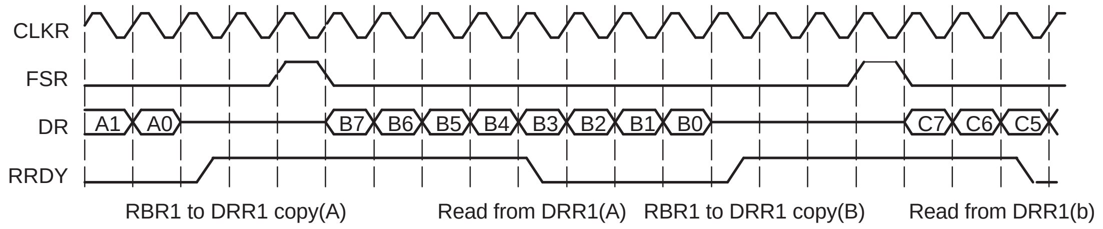


A RSR[1,2]: Receive shift registers 1 and 2

B RBR[1,2]: Receive buffer registers 1 and 2

C DRR[1,2]: Data receive registers 1 and 2

Figure 15-14. McBSP Reception Signal Activity



A CLKR: Internal receive clock

B FSR: Internal receive frame-synchronization signal

C DR: Data on DR pin

D RRDY: Status of receiver ready bit (high is 1)

The following process describes how data travels from the DR pin to the CPU or to the DMA controller:

1. The McBSP waits for a receive frame-synchronization pulse on internal FSR.
2. When the pulse arrives, the McBSP inserts the appropriate data delay that is selected with the RDATDLY bits of RCR2.

   In the preceding timing diagram (Figure 15-14), a 1-bit data delay is selected.
3. The McBSP accepts data bits on the DR pin and shifts them into the receive shift register(s).

   If the word length is 16 bits or smaller, only RSR1 is used. If the word length is larger than 16 bits, RSR2 and RSR1 are used and RSR2 contains the most significant bits. For details on choosing a word length, see Section 15.8.8, *Set the Receive Word Length(s)*.
4. When a full word is received, the McBSP copies the contents of the receive shift register(s) to the receive buffer register(s), provided that RBR1 is not full with previous data.

   If the word length is 16 bits or smaller, only RBR1 is used. If the word length is larger than 16 bits, RBR2 and RBR1 are used and RBR2 contains the most significant bits.
5. The McBSP copies the contents of the receive buffer register(s) into the data receive register(s), provided that DRR1 is not full with previous data. When DRR1 receives new data, the receiver ready bit (RRDY) is set in SPCR1. This indicates that received data is ready to be read by the CPU or the DMA controller.

   If the word length is 16 bits or smaller, only DRR1 is used. If the word length is larger than 16 bits, DRR2 and DRR1 are used and DRR2 contains the most significant bits.

   If companding is used during the copy (RCOMPAND = 10b or 11b in RCR2), the 8-bit compressed data in RBR1 is expanded to a left-justified 16-bit value in DRR1. If companding is disabled, the data copied from RBR[1,2] to DRR[1,2] is justified and bit filled according to the RJUST bits.
6. The CPU or the DMA controller reads the data from the data receive register(s). When DRR1 is read, RRDY is cleared and the next RBR-to-DRR copy is initiated.

NOTE: If both DRRs are required (word length larger than 16 bits), the CPU or the DMA controller must read from DRR2 first and then from DRR1. As soon as DRR1 is read, the next RBR-to-DRR copy is initiated. If DRR2 is not read first, the data in DRR2 is lost.

When activity is not properly timed, errors can occur. See the following topics for more details:

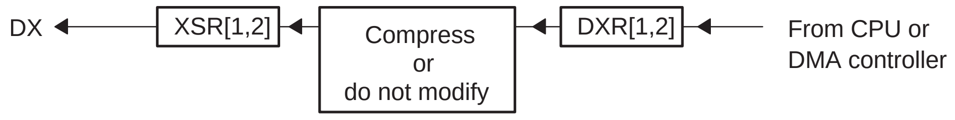
- *Overrun in the Receiver* (see Section 15.5.2)
- *Unexpected Receive Frame-Synchronization Pulse* (see Section 15.5.3)

15.3.6 McBSP Transmission

This section explains the fundamental process of transmission in the McBSP. For details about how to program the McBSP transmitter, see Section 15.9, *Transmitter Configuration*.

Figure 15-15 and Figure 15-16 show how transmission occurs in the McBSP. Figure 15-15 shows the physical path for the data. Figure 15-16 is a timing diagram showing signal activity for one possible transmission scenario. A description of the process follows the figures.

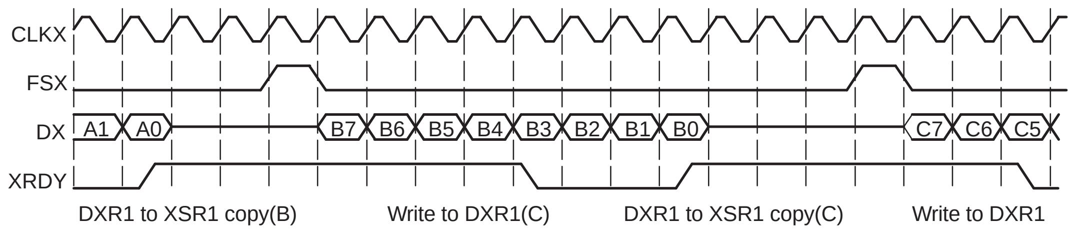
Figure 15-15. McBSP Transmission Physical Data Path



A XSR[1,2]: Transmit shift registers 1 and 2

B DXR[1,2]: Data transmit registers 1 and 2

Figure 15-16. McBSP Transmission Signal Activity



A CLKX: Internal transmit clock

B FSX: Internal transmit frame-synchronization signal

C DX: Data on DX pin

D XRDY: Status of transmitter ready bit (high is 1)

1. The CPU or the DMA controller writes data to the data transmit register(s). When DXR1 is loaded, the transmitter ready bit (XRDY) is cleared in SPCR2 to indicate that the transmitter is not ready for new data.

   If the word length is 16 bits or smaller, only DXR1 is used. If the word length is larger than 16 bits, DXR2 and DXR1 are used and DXR2 contains the most significant bits. For details on choosing a word length, see Section 15.9.8, *Set the Transmit Word Length(s)*.

   NOTE: If both DXRs are needed (word length larger than 16 bits), the CPU or the DMA controller must load DXR2 first and then load DXR1. As soon as DXR1 is loaded, the contents of both DXRs are copied to the transmit shift registers (XSRs), as described in the next step. If DXR2 is not loaded first, the previous content of DXR2 is passed to the XSR2.

2. When new data arrives in DXR1, the McBSP copies the content of the data transmit register(s) to the transmit shift register(s). In addition, the transmit ready bit (XRDY) is set. This indicates that the transmitter is ready to accept new data from the CPU or the DMA controller.

   If the word length is 16 bits or smaller, only XSR1 is used. If the word length is larger than 16 bits, XSR2 and XSR1 are used and XSR2 contains the most significant bits.

   If companding is used during the transfer (XCOMPAND = 10b or 11b in XCR2), the McBSP compresses the 16-bit data in DXR1 to 8-bit data in the μ-law or A-law format in XSR1. If companding is disabled, the McBSP passes data from the DXR(s) to the XSR(s) without modification.
3. The McBSP waits for a transmit frame-synchronization pulse on internal FSX.
4. When the pulse arrives, the McBSP inserts the appropriate data delay that is selected with the XDATDLY bits of XCR2.

In the preceding timing diagram (Figure 15-16), a 1-bit data delay is selected.

5. The McBSP shifts data bits from the transmit shift register(s) to the DX pin.

When activity is not properly timed, errors can occur. See the following topics for more details:

- *Overwrite in the Transmitter* (Section 15.5.4)
- *Underflow in the Transmitter* (Section 15.5.4.3)
- *Unexpected Transmit Frame-Synchronization Pulse* (Section 15.5.5)

15.3.7 Interrupts and DMA Events Generated by a McBSP

The McBSP sends notification of important events to the CPU and the DMA controller via the internal signals shown in Table 15-3.

Table 15-3. Interrupts and DMA Events Generated by a McBSP

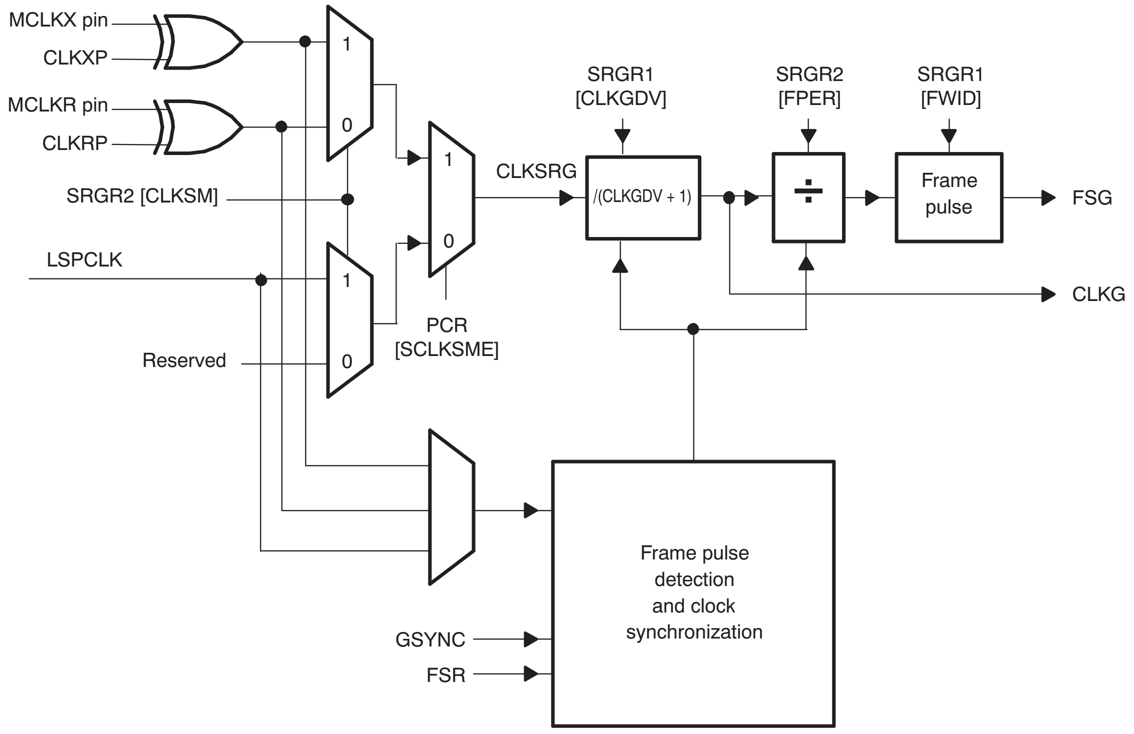
Internal Signal	Description
RINT	Receive interrupt
	The McBSP can send a receive interrupt request to CPU based upon a selected condition in the receiver of the McBSP (a condition selected by the RINTM bits of SPCR1).
XINT	Transmit interrupt
	The McBSP can send a transmit interrupt request to CPU based upon a selected condition in the transmitter of the McBSP (a condition selected by the XINTM bits of SPCR2).
REVT	Receive synchronization event
	An REVT signal is sent to the DMA when data has been received in the data receive registers (DRRs).
XEVT	Transmit synchronization event
	An XEVT signal is sent to the DMA when the data transmit registers (DXRs) are ready to accept the next serial word for transmission.

15.4 McBSP Sample Rate Generator

Each McBSP contains a sample rate generator (SRG) that can be programmed to generate an internal data clock (CLKG) and an internal frame-synchronization signal (FSG). CLKG can be used for bit shifting on the data receive (DR) pin and/or the data transmit (DX) pin. FSG can be used to initiate frame transfers on DR and/or DX. Figure 15-17 is a conceptual block diagram of the sample rate generator.

15.4.1 Block Diagram

Figure 15-17. Conceptual Block Diagram of the Sample Rate Generator



The source clock for the sample rate generator (labeled CLKSRG in the diagram) can be supplied by the LSPCLK, or by an external pin (MCLKX or MCLKR). The source is selected with the SCLKME bit of PCR and the CLKSM bit of SRGR2. If a pin is used, the polarity of the incoming signal can be inverted with the appropriate polarity bit (CLKXP of PCR or CLKRP of PCR).

The sample rate generator has a three-stage clock divider that gives CLKG and FSG programmability. The three stages provide:

- Clock divide-down. The source clock is divided according to the CLKGDV bits of SRGR1 to produce CLKG.
- Frame period divide-down. CLKG is divided according to the FPER bits of SRGR2 to control the period from the start of a frame-pulse to the start of the next pulse.
- Frame-synchronization pulse-width countdown. CLKG cycles are counted according to the FWID bits of SRGR1 to control the width of each frame-synchronization pulse.

NOTE: The McBSP cannot operate at a frequency faster than ½ the source clock frequency. Choose an input clock frequency and a CLKGDV value such that CLKG is less than or equal to ½ the source clock frequency.

In addition to the three-stage clock divider, the sample rate generator has a frame-synchronization pulse detection and clock synchronization module that allows synchronization of the clock divide down with an incoming frame-synchronization pulse on the FSR pin. This feature is enabled or disabled with the GSYNC bit of SRGR2.

For details on getting the sample rate generator ready for operation, see Section 15.4.4, *Reset and Initialization Procedure*.

15.4.1.1 Clock Generation in the Sample Rate Generator

The sample rate generator can produce a clock signal (CLKG) for use by the receiver, the transmitter, or both. Use of the sample rate generator to drive clocking is controlled by the clock mode bits (CLKRM and CLKXM) in the pin control register (PCR). When a clock mode bit is set to 1 (CLKRM = 1 for reception, CLKXM = 1 for transmission), the corresponding data clock (CLKR for reception, CLKX for transmission) is driven by the internal sample rate generator output clock (CLKG).

The effects of CLKRM = 1 and CLKXM = 1 on the McBSP are partially affected by the use of the digital loopback mode and the clock stop (SPI) mode, respectively, as described in Table 15-4. The digital loopback mode (described in Section 15.8.4) is selected with the DLB bit of SPCR1. The clock stop mode (described in Section 15.7.2) is selected with the CLKSTP bits of SPCR1.

When using the sample rate generator as a clock source, make sure the sample rate generator is enabled (GRST = 1).

Table 15-4. Effects of DLB and CLKSTP on Clock Modes

Mode Bit Settings		Effect
CLKRM = 1	DLB = 0 (Digital loopback mode disabled)	CLKR is an output pin driven by the sample rate generator output clock (CLKG).
	DLB = 1 (Digital loopback mode enabled)	CLKR is an output pin driven by internal CLKX. The source for CLKX depends on the CLKXM bit.
CLKXM = 1	CLKSTP = 00b or 01b (Clock stop (SPI) mode disabled)	CLKX is an output pin driven by the sample rate generator output clock (CLKG).
	CLKSTP = 10b or 11b (Clock stop (SPI) mode enabled)	The McBSP is a master in an SPI system. Internal CLKX drives internal CLKR and the shift clocks of any SPI-compliant slave devices in the system. CLKX is driven by the internal sample rate generator.

15.4.1.2 Choosing an Input Clock

The sample rate generator must be driven by an input clock signal from one of the three sources selectable with the SCLKME bit of PCR and the CLKSM bit of SRGR2 (see Table 15-5). When CLKSM = 1, the minimum divide down value in CLKGDV bits is 1. CLKGDV is described in Section 15.4.1.4.

Table 15-5. Choosing an Input Clock for the Sample Rate Generator with the SCLKME and CLKSM Bits

SCLKME	CLKSM	Input Clock for Sample Rate Generator
0	0	Reserved
0	1	LSPCLK
1	0	Signal on MCLKR pin
1	1	Signal on MCLKX pin

15.4.1.3 Choosing a Polarity for the Input Clock

As shown in Figure 15-18, when the input clock is received from a pin, you can choose the polarity of the input clock. The rising edge of CLKSRG generates CLKG and FSG, but you can determine which edge of the input clock causes a rising edge on CLKSRG. The polarity options and their effects are described in Table 15-6.

Figure 15-18. Possible Inputs to the Sample Rate Generator and the Polarity Bits

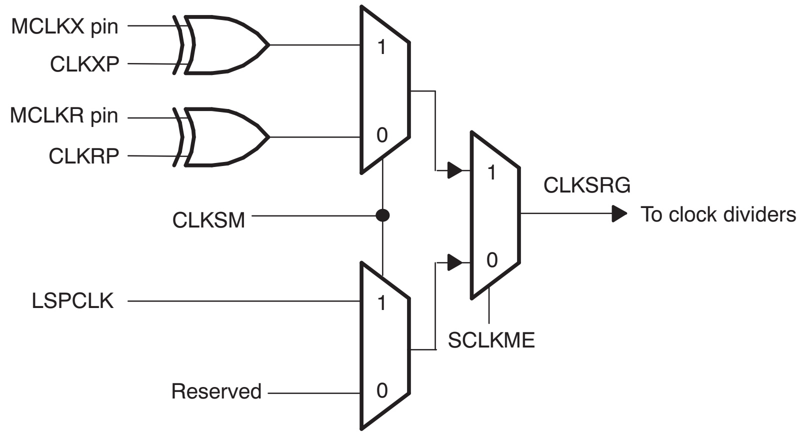


Table 15-6. Polarity Options for the Input to the Sample Rate Generator

Input Clock	Polarity Option	Effect
LSPCLK	Always positive polarity	Rising edge of CPU clock generates transitions on CLKG and FSG.
Signal on MCLKR pin	CLKRP = 0 in PCR	Falling edge on MCLKR pin generates transitions on CLKG and FSG.
	CLKRP = 1 in PCR	Rising edge on MCLKR pin generates transitions on CLKG and FSG.
Signal on MCLKX pin	CLKXP = 0 in PCR	Rising edge on MCLKX pin generates transitions on CLKG and FSG.
	CLKXP = 1 in PCR	Falling edge on MCLKX pin generates transitions on CLKG and FSG.

15.4.1.4 Choosing a Frequency for the Output Clock (CLKG)

The input clock (LSPCLK or external clock) can be divided down by a programmable value to drive CLKG. Regardless of the source to the sample rate generator, the rising edge of CLKSRG (see Figure 15-17) generates CLKG and FSG.

The first divider stage of the sample rate generator creates the output clock from the input clock. This divider stage uses a counter that is preloaded with the divide down value in the CLKGDV bits of SRGR1. The output of this stage is the data clock (CLKG). CLKG has the frequency represented by Example 3.

Equation 3: CLKG Frequency

$$\text{CLKG frequency} = \frac{\text{Input clock frequency}}{(\text{CLKGDV} + 1)}$$

Thus, the input clock frequency is divided by a value between 1 and 256. When CLKGDV is odd or equal to 0, the CLKG duty cycle is 50%. When CLKGDV is an even value, 2p, representing an odd divide down, the high-state duration is p+1 cycles and the low-state duration is p cycles.

15.4.1.5 Keeping CLKG Synchronized to External MCLKR

When the MCLKR pin is used to drive the sample rate generator (see Section 15.4.1.2), the GSYNC bit in SRGR2 and the FSR pin can be used to configure the timing of the output clock (CLKG) relative to the input clock. Note that this feature is available only when the MCLKR pin is used to feed the external clock.

GSYNC = 1 ensures that the McBSP and an external device are dividing down the input clock with the same phase relationship. If GSYNC = 1, an inactive-to-active transition on the FSR pin triggers a resynchronization of CLKG and generation of FSG.

For more details about synchronization, see Section 15.4.3.

15.4.2 Frame Synchronization Generation in the Sample Rate Generator

The sample rate generator can produce a frame-synchronization signal (FSG) for use by the receiver, the transmitter, or both.

If you want the receiver to use FSG for frame synchronization, make sure FSRM = 1. (When FSRM = 0, receive frame synchronization is supplied via the FSR pin.)

If you want the transmitter to use FSG for frame synchronization, you must set:

- FSXM = 1 in PCR: This indicates that transmit frame synchronization is supplied by the McBSP itself rather than from the FSX pin.
- FSGM = 1 in SRGR2: This indicates that when FSXM = 1, transmit frame synchronization is supplied by the sample rate generator. (When FSGM = 0 and FSXM = 1, the transmitter uses frame-synchronization pulses generated every time data is transferred from DXR[1,2] to XSR[1,2].)

In either case, the sample rate generator must be enabled (GRST = 1) and the frame-synchronization logic in the sample rate generator must be enabled (FRST = 0).

15.4.2.1 Choosing the Width of the Frame-Synchronization Pulse on FSG

Each pulse on FSG has a programmable width. You program the FWID bits of SRGR1, and the resulting pulse width is (FWID + 1) CLKG cycles, where CLKG is the output clock of the sample rate generator.

15.4.2.2 Controlling the Period Between the Starting Edges of Frame-Synchronization Pulses on FSG

You can control the amount of time from the starting edge of one FSG pulse to the starting edge of the next FSG pulse. This period is controlled in one of two ways, depending on the configuration of the sample rate generator:

- If the sample rate generator is using an external input clock and GSYNC = 1 in SRGR2, FSG pulses in response to an inactive-to-active transition on the FSR pin. Thus, the frame-synchronization period is controlled by an external device.
- Otherwise, you program the FPER bits of SRGR2, and the resulting frame-synchronization period is (FPER + 1) CLKG cycles, where CLKG is the output clock of the sample rate generator.

15.4.2.3 Keeping FSG Synchronized to an External Clock

When an external signal is selected to drive the sample rate generator (see Section 15.4.1.2 on page Section 15.4.1.2), the GSYNC bit of SRGR2 and the FSR pin can be used to configure the timing of FSG pulses.

GSYNC = 1 ensures that the McBSP and an external device are dividing down the input clock with the same phase relationship. If GSYNC = 1, an inactive-to-active transition on the FSR pin triggers a resynchronization of CLKG and generation of FSG.

See Section 15.4.3 for more details about synchronization.

15.4.3 Synchronizing Sample Rate Generator Outputs to an External Clock

The sample rate generator can produce a clock signal (CLKG) and a frame-synchronization signal (FSG) based on an input clock signal that is either the CPU clock signal or a signal at the MCLKR or MCLKX pin. When an external clock is selected to drive the sample rate generator, the GSYNC bit of SRGR2 and the FSR pin can be used to control the timing of CLKG and the pulsing of FSG relative to the chosen input clock.

Make GSYNC = 1 when you want the McBSP and an external device to divide down the input clock with the same phase relationship. If GSYNC = 1:

- An inactive-to-active transition on the FSR pin triggers a resynchronization of CLKG and a pulsing of FSG.
- CLKG always begins with a high state after synchronization.
- FSR is always detected at the same edge of the input clock signal that generates CLKG, no matter how long the FSR pulse is.

- The FPER bits of SRGR2 are ignored because the frame-synchronization period on FSG is determined by the arrival of the next frame-synchronization pulse on the FSR pin.

If GSYNC = 0, CLKG runs freely and is not resynchronized, and the frame-synchronization period on FSG is determined by FPER.

15.4.3.1 Operating the Transmitter Synchronously with the Receiver

When GSYNC = 1, the transmitter can operate synchronously with the receiver, provided that:

- FSX is programmed to be driven by FSG (FSGM = 1 in SRGR2 and FSXM = 1 in PCR). If the input FSR has appropriate timing so that it can be sampled by the falling edge of CLKG, it can be used, instead, by setting FSXM = 0 and connecting FSR to FSX externally.
- The sample rate generator clock drives the transmit and receive clocking (CLKRM = CLKXM = 1 in PCR).

15.4.3.2 Synchronization Examples

Figure 15-19 and Figure 15-20 show the clock and frame-synchronization operation with various polarities of CLKR and FSR. These figures assume FWID = 0 in SRGR1, for an FSG pulse that is one CLKG cycle wide. The FPER bits of SRGR2 are not programmed; the period from the start of a frame-synchronization pulse to the start of the next pulse is determined by the arrival of the next inactive-to-active transition on the FSR pin. Each of the figures shows what happens to CLKG when it is initially synchronized and GSYNC = 1, and when it is not initially synchronized and GSYNC = 1. The second figure has a slower CLKG frequency (it has a larger divide-down value in the CLKGDV bits of SRGR1).

Figure 15-19. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 1

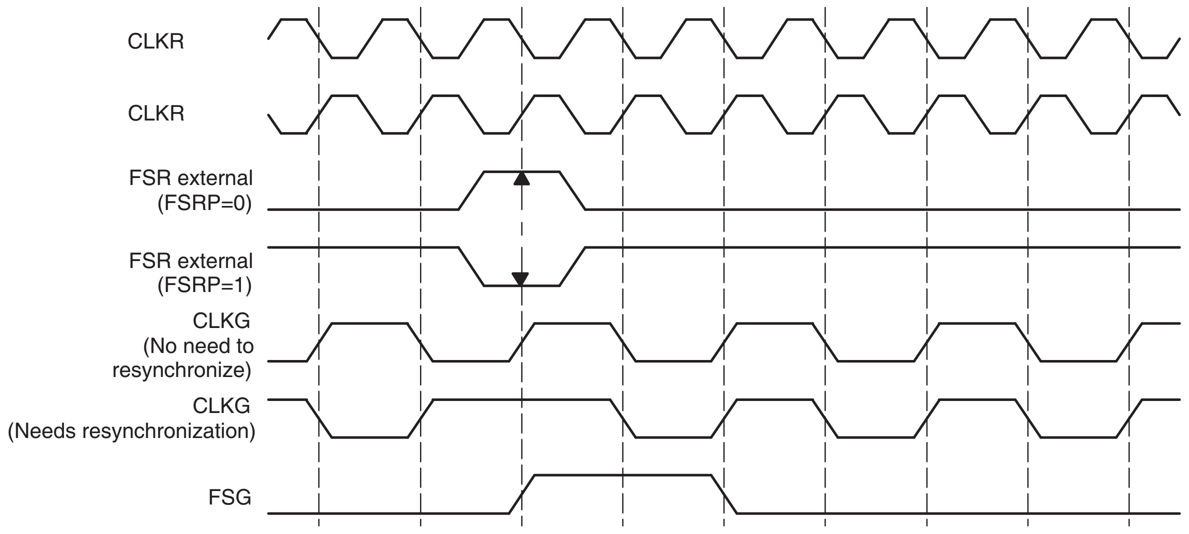
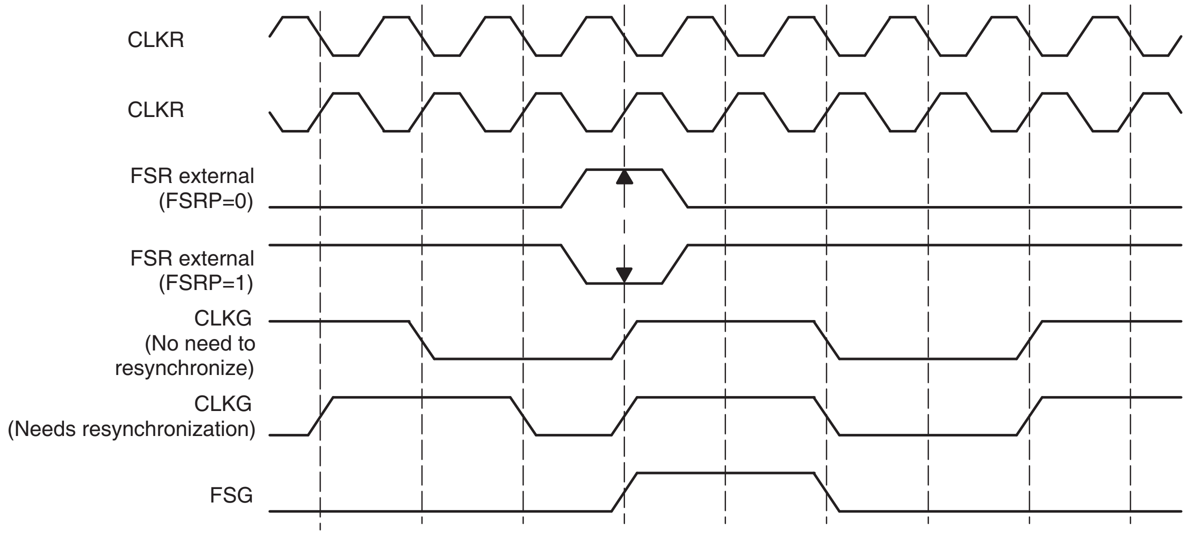

CLKR
CLKR
FSR external (FSRP=0)
FSR external (FSRP=1)
CLKG (No need to resynchronize)
CLKG (Needs resynchronization)
FSG

Figure 15-20. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 3


CLKR
CLKR
FSR external (FSRP=0)
FSR external (FSRP=1)
CLKG (No need to resynchronize)
CLKG (Needs resynchronization)
FSG

15.4.4 Reset and Initialization Procedure for the Sample Rate Generator

To reset and initialize the sample rate generator:

Step 1. Place the McBSP/sample rate generator in reset.

During a DSP reset, the sample rate generator, the receiver, and the transmitter reset bits (GRST, RRST, and XRST) are automatically forced to 0. Otherwise, during normal operation, the sample rate generator can be reset by making GRST = 0 in SPCR2, provided that CLKG and/or FSG is not used by any portion of the McBSP. Depending on your system you may also want to reset the receiver (RRST = 0 in SPCR1) and reset the transmitter (XRST = 0 in SPCR2).

If GRST = 0 due to a device reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven inactive-low. If GRST = 0 due to program code, CLKG and FSG are driven low (inactive).

Step 2. Program the registers that affect the sample rate generator.

Program the sample rate generator registers (SRGR1 and SRGR2) as required for your application. If necessary, other control registers can be loaded with desired values, provided the respective portion of the McBSP (the receiver or transmitter) is in reset.

After the sample rate generator registers are programmed, wait 2 CLKSRG cycles. This ensures proper synchronization internally.

Step 3. Enable the sample rate generator (take it out of reset).

In SPCR2, make GRST = 1 to enable the sample rate generator.

After the sample rate generator is enabled, wait two CLKG cycles for the sample rate generator logic to stabilize.

On the next rising edge of CLKSRG, CLKG transitions to 1 and starts clocking with a frequency equal to Example 4.

Table 15-7. Input Clock Selection for Sample Rate Generator

SCLKME	CLKSM	Input Clock for Sample Rate Generator
0	0	Reserved
0	1	LSPCLK
1	0	Signal on MCLKR pin
1	1	Signal on MCLKX pin

Step 4. If necessary, enable the receiver and/or the transmitter.

If necessary, remove the receiver and/or transmitter from reset by setting RRST and/or XRST = 1.

Step 5. If necessary, enable the frame-synchronization logic of the sample rate generator.

After the required data acquisition setup is done (DXR[1,2] is loaded with data), set GRST = 1 in SPCR2 if an internally generated frame-synchronization pulse is required. FSG is generated with an active-high edge after the programmed number of CLKG clocks (FPER + 1) have elapsed.

Equation 4: CLKG Frequency

$$\text{CLKG frequency} = \frac{\text{Input clock frequency}}{(\text{CLKGDV} + 1)}$$

where the input clock is selected with the SCLKME bit of PCR and the CLKSM bit of SRGR2 in one of the configurations shown in Table 15-7.

15.5 McBSP Exception/Error Conditions

This section describes exception/error conditions and how to handle them.

15.5.1 Types of Errors

There are five serial port events that can constitute a system error:

- Receiver overrun (RFULL = 1)

  This occurs when DRR1 has not been read since the last RBR-to-DRR copy. Consequently, the receiver does not copy a new word from the RBR(s) to the DRR(s) and the RSR(s) are now full with another new word shifted in from DR. Therefore, RFULL = 1 indicates an error condition wherein any new data that can arrive at this time on DR replaces the contents of the RSR(s), and the previous word is lost. The RSRs continue to be overwritten as long as new data arrives on DR and DRR1 is not read. For more details about overrun in the receiver, see Section 15.5.2.
- Unexpected receive frame-synchronization pulse (RSYNCERR = 1)

  This occurs during reception when RFIG = 0 and an unexpected frame-synchronization pulse occurs. An unexpected frame-synchronization pulse is one that begins the next frame transfer before all the bits of the current frame have been received. Such a pulse causes data reception to abort and restart. If new data has been copied into the RBR(s) from the RSR(s) since the last RBR-to-DRR copy, this new data in the RBR(s) is lost. This is because no RBR-to-DRR copy occurs; the reception has been restarted. For more details about receive frame-synchronization errors, see Section 15.5.3.
- Transmitter data overwrite

  This occurs when the CPU or DMA controller overwrites data in the DXR(s) before the data is copied to the XSR(s). The overwritten data never reaches the DX pin. For more details about overwrite in the transmitter, see Section 15.5.4.
- Transmitter underflow ($\overline{\text{XEMPTY}}$ = 0)

  If a new frame-synchronization signal arrives before new data is loaded into DXR1, the previous data in the DXR(s) is sent again. This procedure continues for every new frame-synchronization pulse that arrives until DXR1 is loaded with new data. For more details about underflow in the transmitter, see Section 15.5.4.3.
- Unexpected transmit frame-synchronization pulse (XSYNCERR = 1)

  This occurs during transmission when XFIG = 0 and an unexpected frame-synchronization pulse occurs. An unexpected frame-synchronization pulse is one that begins the next frame transfer before all the bits of the current frame have been transferred. Such a pulse causes the current data transmission to abort and restart. If new data has been written to the DXR(s) since the last DXR-to-XSR copy, the current value in the XSR(s) is lost. For more details about transmit frame-synchronization errors, see Section 15.5.5.

15.5.2 Overrun in the Receiver

RFULL = 1 in SPCR1 indicates that the receiver has experienced overrun and is in an error condition. RFULL is set when all of the following conditions are met:

1. DRR1 has not been read since the last RBR-to-DRR copy (RRDY = 1).
2. RBR1 is full and an RBR-to-DRR copy has not occurred.
3. RSR1 is full and an RSR1-to-RBR copy has not occurred.

As described in the Section 15.3.5, *McBSP Reception*, data arriving on DR is continuously shifted into RSR1 (for word length of 16 bits or smaller) or RSR2 and RSR1 (for word length larger than 16 bits). Once a complete word is shifted into the RSR(s), an RSR-to-RBR copy can occur only if the previous data in RBR1 has been copied to DRR1. The RRDY bit is set when new data arrives in DRR1 and is cleared when that data is read from DRR1. Until RRDY = 0, the next RBR-to-DRR copy does not take place, and the data is held in the RSR(s). New data arriving on the DR pin is shifted into RSR(s), and the previous content of the RSR(s) is lost.

You can prevent the loss of data if DRR1 is read no later than 2.5 cycles before the end of the third word is shifted into the RSR1.

NOTE: If both DRRs are needed (word length larger than 16 bits), the CPU or the DMA controller must read from DRR2 first and then from DRR1. As soon as DRR1 is read, the next RBR-to-DRR copy is initiated. If DRR2 is not read first, the data in DRR2 is lost.

After the receiver starts running from reset, a minimum of three words must be received before RFULL is set. Either of the following events clears the RFULL bit and allows subsequent transfers to be read properly:

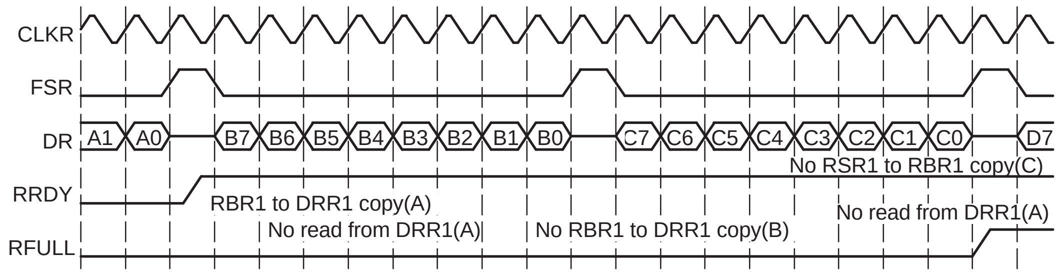
- The CPU or DMA controller reads DRR1.
- The receiver is reset individually (RRST = 0) or as part of a device reset.

Another frame-synchronization pulse is required to restart the receiver.

15.5.2.1 Example of Overrun Condition

Figure 15-21 shows the receive overrun condition. Because serial word A is not read from DRR1 before serial word B arrives in RBR1, B is not transferred to DRR1 yet. Another new word ©) arrives and RSR1 is full with this data. DRR1 is finally read, but not earlier than 2.5 cycles before the end of word C. Therefore, new data (D) overwrites word C in RSR1. If DRR1 is not read in time, the next word can overwrite D.

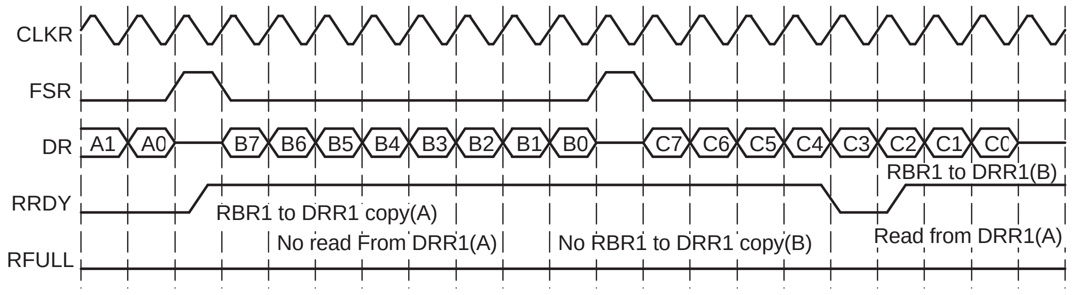
Figure 15-21. Overrun in the McBSP Receiver



15.5.2.2 Example of Preventing Overrun Condition

Figure 15-22 shows the case where RFULL is set, but the overrun condition is prevented by a read from DRR1 at least 2.5 cycles before the next serial word ©) is completely shifted into RSR1. This ensures that an RBR1-to-DRR1 copy of word B occurs before receiver attempts to transfer word C from RSR1 to RBR1.

Figure 15-22. Overrun Prevented in the McBSP Receiver



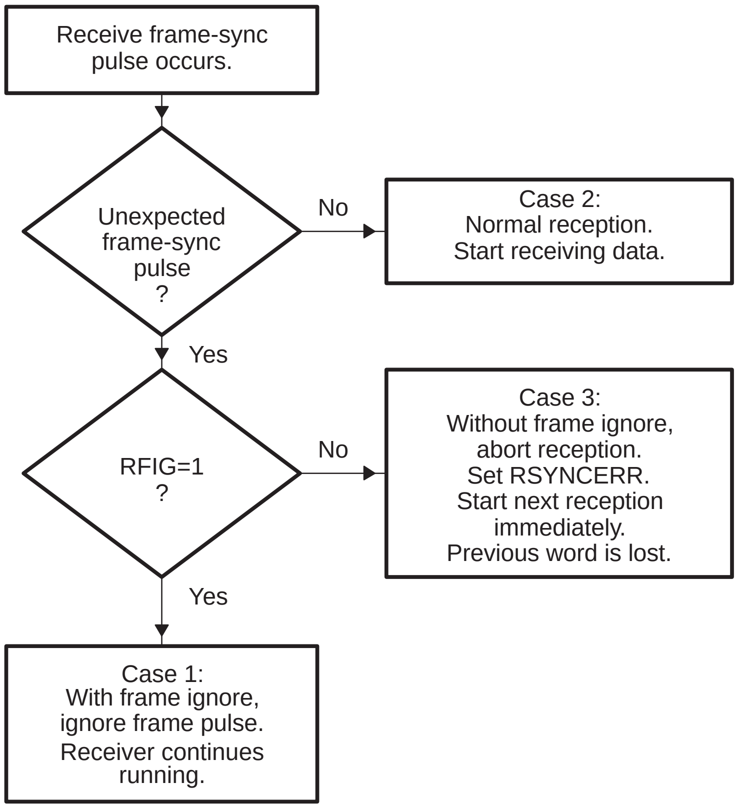
15.5.3 Unexpected Receive Frame-Synchronization Pulse

Section 15.5.3.1 shows how the McBSP responds to any receive frame-synchronization pulses, including an unexpected pulse. Section 15.5.3.2 and Section 15.5.3.3 show an examples of a frame-synchronization error and an example of how to prevent such an error, respectively.

15.5.3.1 Possible Responses to Receive Frame-Synchronization Pulses

Figure 15-23 shows the decision tree that the receiver uses to handle all incoming frame-synchronization pulses. The figure assumes that the receiver has been started (RRST = 1 in SPCR1). Case 3 in the figure is the case in which an error occurs.

Figure 15-23. Possible Responses to Receive Frame-Synchronization Pulses


Receive frame-sync pulse occurs.
Unexpected frame-sync pulse ?
No
Case 2: Normal reception. Start receiving data.
Yes
RFIG=1 ?
No
Case 3: Without frame ignore, abort reception. Set RSYNCERR. Start next reception immediately. Previous word is lost.
Yes
Case 1: With frame ignore, ignore frame pulse. Receiver continues running.

Any one of three cases can occur:

- Case 1: Unexpected internal FSR pulses with RFIG = 1 in RCR2. Receive frame-synchronization pulses are ignored, and the reception continues.
- Case 2: Normal serial port reception. Reception continues normally because the frame-synchronization

pulse is not unexpected. There are three possible reasons why a receive operation might *not* be in progress when the pulse occurs:

- The FSR pulse is the first after the receiver is enabled (RRST = 1 in SPCR1).
- The FSR pulse is the first after DRR[1,2] is read, clearing a receiver full (RFULL = 1 in SPCR1) condition.
- The serial port is in the interpacket intervals. The programmed data delay for reception (programmed with the RDATDLY bits in RCR2) may start during these interpacket intervals for the first bit of the next word to be received. Thus, at maximum frame frequency, frame synchronization can still be received 0 to 2 clock cycles before the first bit of the synchronized frame.

• Case 3: Unexpected receive frame synchronization with RFIG = 0 (frame-synchronization pulses not ignored). Unexpected frame-synchronization pulses can originate from an external source or from the internal sample rate generator.

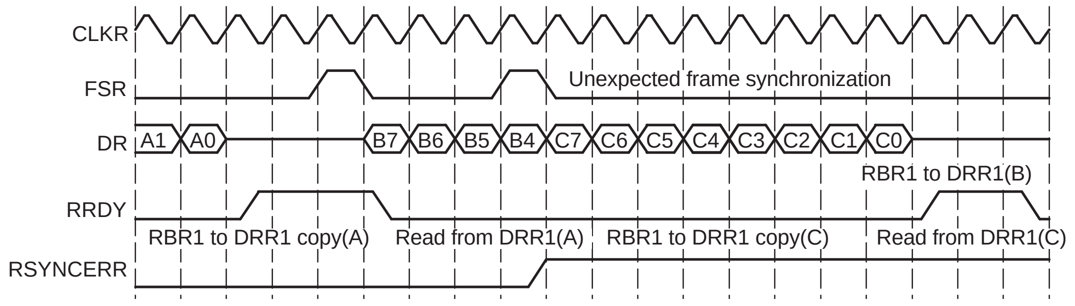
If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully received, this pulse is treated as an unexpected frame-synchronization pulse, and the receiver sets the receive frame-synchronization error bit (RSYNCERR) in SPCR1. RSYNCERR can be cleared only by a receiver reset or by a write of 0 to this bit.

If you want the McBSP to notify the CPU of receive frame-synchronization errors, you can set a special receive interrupt mode with the RINTM bits of SPCR1. When RINTM = 11b, the McBSP sends a receive interrupt (RINT) request to the CPU each time that RSYNCERR is set.

15.5.3.2 Example of Unexpected Receive Frame-Synchronization Pulse

Figure 15-24 shows an unexpected receive frame-synchronization pulse during normal operation of the serial port, with time intervals between data packets. When the unexpected frame-synchronization pulse occurs, the RSYNCERR bit is set, the reception of data B is aborted, and the reception of data C begins. In addition, if RINTM = 11b, the McBSP sends a receive interrupt (RINT) request to the CPU.

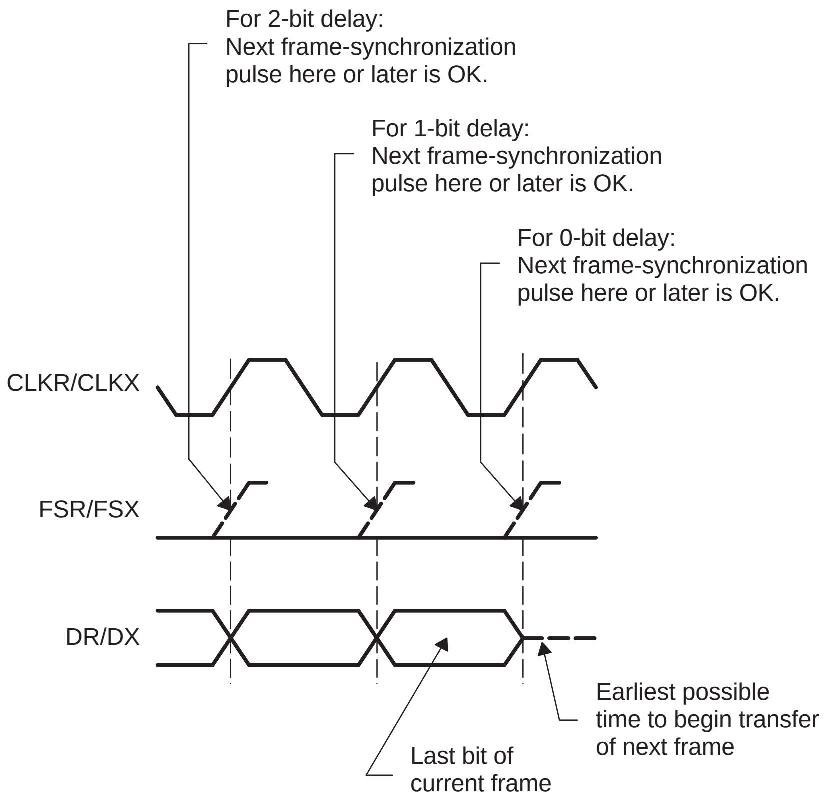
Figure 15-24. An Unexpected Frame-Synchronization Pulse During a McBSP Reception



15.5.3.3 Preventing Unexpected Receive Frame-Synchronization Pulses

Each frame transfer can be delayed by 0, 1, or 2 MCLKR cycles, depending on the value in the RDATDLY bits of RCR2. For each possible data delay, Figure 15-25 shows when a new frame-synchronization pulse on FSR can safely occur relative to the last bit of the current frame.

Figure 15-25. Proper Positioning of Frame-Synchronization Pulses


For 2-bit delay:
Next frame-synchronization
pulse here or later is OK.
For 1-bit delay:
Next frame-synchronization
pulse here or later is OK.
For 0-bit delay:
Next frame-synchronization
pulse here or later is OK.
CLKR/CLKX
FSR/FSX
DR/DX
Earliest possible
time to begin transfer
of next frame
Last bit of
current frame

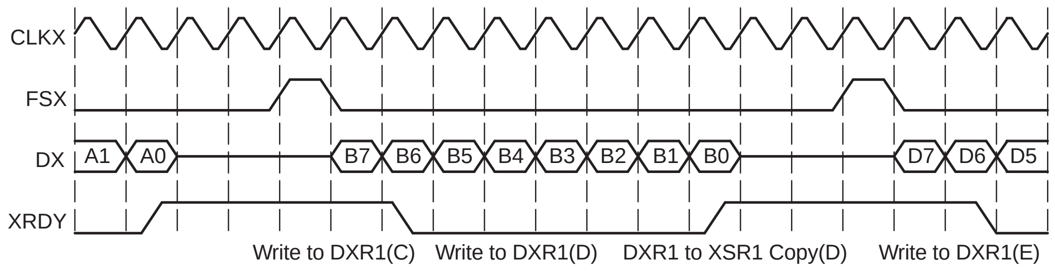
15.5.4 Overwrite in the Transmitter

As described in the section on McBSP transmission (page Section 15.3.6), the transmitter must copy the data previously written to the DXR(s) by the CPU or DMA controller into the XSR(s) and then shift each bit from the XSR(s) to the DX pin. If new data is written to the DXR(s) before the previous data is copied to the XSR(s), the previous data in the DXR(s) is overwritten and thus lost.

15.5.4.1 Example of Overwrite Condition

Figure 15-26 shows what happens if the data in DXR1 is overwritten before being transmitted. Initially, DXR1 is loaded with data C. A subsequent write to DXR1 overwrites C with D before C is copied to XSR1. Thus, C is never transmitted on DX.

Figure 15-26. Data in the McBSP Transmitter Overwritten and Thus Not Transmitted



15.5.4.2 Preventing Overwrites

You can prevent CPU overwrites by making the CPU:

- Poll for XRDY = 1 in SPCR2 before writing to the DXR(s). XRDY is set when data is copied from DXR1 to XSR1 and is cleared when new data is written to DXR1.

- Wait for a transmit interrupt (XINT) before writing to the DXR(s). When XINTM = 00b in SPCR2, the transmitter sends XINT to the CPU each time XRDY is set.

You can prevent DMA overwrites by synchronizing DMA transfers to the transmit synchronization event XEVT. The transmitter sends an XEVT signal each time XRDY is set.

15.5.4.3 Underflow in the Transmitter

The McBSP indicates a transmitter empty (or underflow) condition by clearing the $\overline{\text{XEMPTY}}$ bit in SPCR2. Either of the following events activates $\overline{\text{XEMPTY}}$ ($\overline{\text{XEMPTY}}$ = 0):

- DXR1 has not been loaded since the last DXR-to-XSR copy, and all bits of the data word in the XSR(s) have been shifted out on the DX pin.
- The transmitter is reset (by forcing XRST = 0 in SPCR2, or by a device reset) and is then restarted.

In the underflow condition, the transmitter continues to transmit the old data that is in the DXR(s) for every new transmit frame-synchronization signal until a new value is loaded into DXR1 by the CPU or the DMA controller.

NOTE: If both DXRs are needed (word length larger than 16 bits), the CPU or the DMA controller must load DXR2 first and then load DXR1. As soon as DXR1 is loaded, the contents of both DXRs are copied to the transmit shift registers (XSRs). If DXR2 is not loaded first, the previous content of DXR2 is passed to the XSR2.

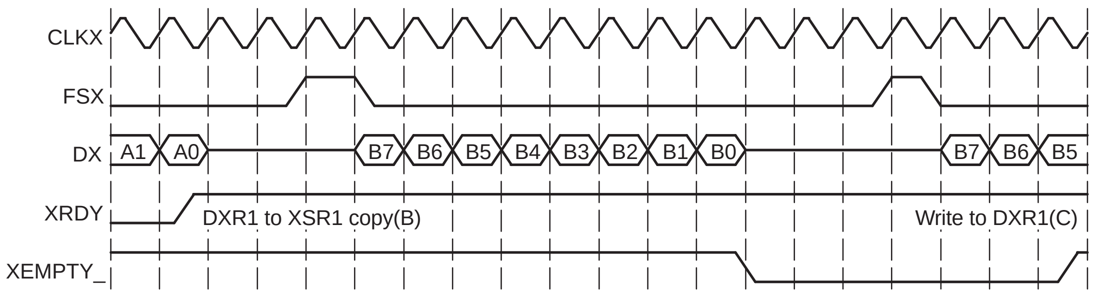
XEMPTY is deactivated (XEMPTY = 1) when a new word in DXR1 is transferred to XSR1. If FSXM = 1 in PCR and FSGM = 0 in SRGR2, the transmitter generates a single internal FSX pulse in response to a DXR-to-XSR copy. Otherwise, the transmitter waits for the next frame-synchronization pulse before sending out the next frame on DX.

When the transmitter is taken out of reset (XRST = 1), it is in a transmitter ready (XRDY = 1 in SPCR2) and transmitter empty (XEMPTY = 0) state. If DXR1 is loaded by the CPU or the DMA controller before internal FSX goes active high, a valid DXR-to-XSR transfer occurs. This allows for the first word of the first frame to be valid even before the transmit frame-synchronization pulse is generated or detected. Alternatively, if a transmit frame-synchronization pulse is detected before DXR1 is loaded, zeros are output on DX.

15.5.4.3.1 Example of the Underflow Condition

Figure 15-27 shows an underflow condition. After B is transmitted, DXR1 is not reloaded before the subsequent frame-synchronization pulse. Thus, B is again transmitted on DX.

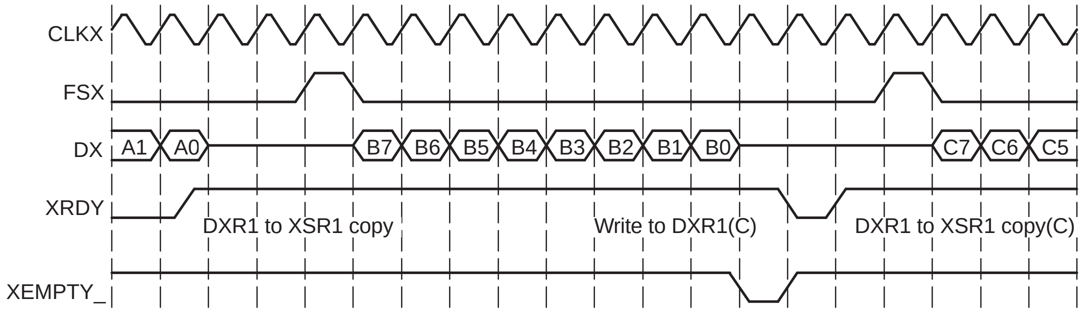
Figure 15-27. Underflow During McBSP Transmission



15.5.4.3.2 Example of Preventing Underflow Condition

Figure 15-28 shows the case of writing to DXR1 just before an underflow condition would otherwise occur. After B is transmitted, C is written to DXR1 before the next frame-synchronization pulse. As a result, there is no underflow; B is not transmitted twice.

Figure 15-28. Underflow Prevented in the McBSP Transmitter



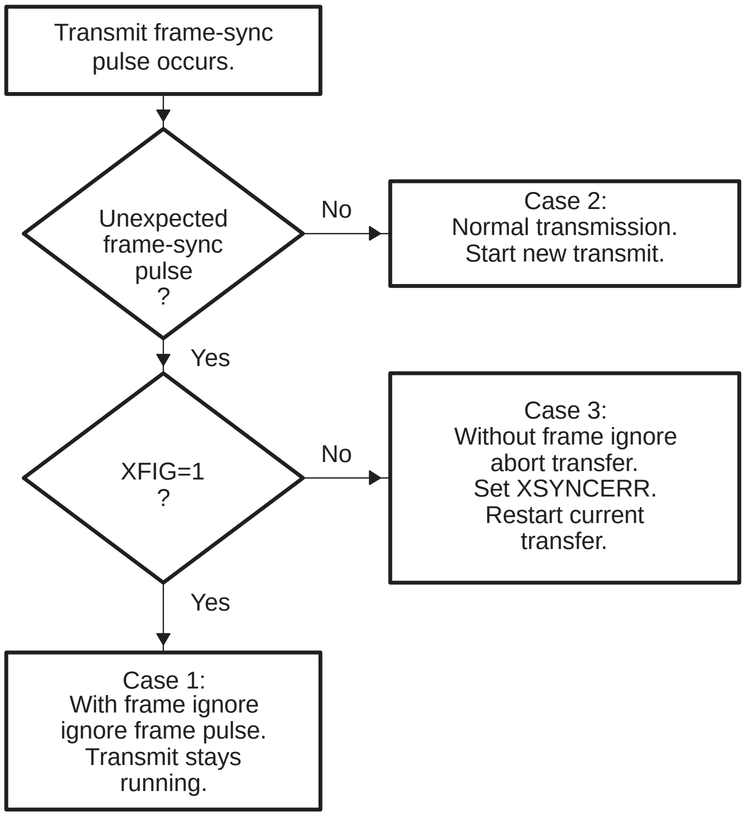
15.5.5 Unexpected Transmit Frame-Synchronization Pulse

Section 15.5.5.1 shows how the McBSP responds to any transmit frame-synchronization pulses, including an unexpected pulse. Section 15.5.5.2 and Section 15.5.5.3 show examples of a frame-synchronization error and an example of how to prevent such an error, respectively.

15.5.5.1 Possible Responses to Transmit Frame-Synchronization Pulses

Figure 15-29 shows the decision tree that the transmitter uses to handle all incoming frame-synchronization pulses. The figure assumes that the transmitter has been started (XRST = 1 in SPCR2). Case 3 in the figure is the case in which an error occurs.

Figure 15-29. Possible Responses to Transmit Frame-Synchronization Pulses


Transmit frame-sync pulse occurs.
Unexpected frame-sync pulse ?
No
Case 2: Normal transmission. Start new transmit.
Yes
XFIG=1 ?
No
Case 3: Without frame ignore abort transfer. Set XSYNCERR. Restart current transfer.
Yes
Case 1: With frame ignore ignore frame pulse. Transmit stays running.

Any one of three cases can occur:

- Case 1: Unexpected internal FSX pulses with XFIG = 1 in XCR2. Transmit frame-synchronization pulses are ignored, and the transmission continues.
- Case 2: Normal serial port transmission. Transmission continues normally because the frame-

synchronization pulse is not unexpected. There are two possible reasons why a transmit operations might *not* be in progress when the pulse occurs:

This FSX pulse is the first after the transmitter is enabled (XRST = 1).

The serial port is in the interpacket intervals. The programmed data delay for transmission (programmed with the XDATDLY bits of XCR2) may start during these interpacket intervals before the first bit of the previous word is transmitted. Thus, at maximum packet frequency, frame synchronization can still be received 0 to 2 clock cycles before the first bit of the synchronized frame.

- Case 3: Unexpected transmit frame synchronization with XFIG = 0 (frame-synchronization pulses not ignored). Unexpected frame-synchronization pulses can originate from an external source or from the internal sample rate generator.

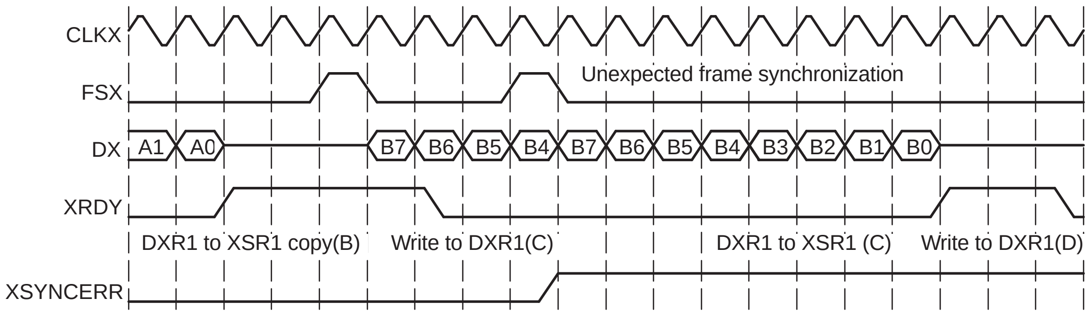
  If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully transmitted, this pulse is treated as an unexpected frame-synchronization pulse, and the transmitter sets the transmit frame-synchronization error bit (XSYNCERR) in SPCR2. XSYNCERR can be cleared only by a transmitter reset or by a write of 0 to this bit.

  If you want the McBSP to notify the CPU of frame-synchronization errors, you can set a special transmit interrupt mode with the XINTM bits of SPCR2. When XINTM = 11b, the McBSP sends a transmit interrupt (XINT) request to the CPU each time that XSYNCERR is set.

15.5.5.2 Example of Unexpected Transmit Frame-Synchronization Pulse

Figure 15-30 shows an unexpected transmit frame-synchronization pulse during normal operation of the serial port with intervals between the data packets. When the unexpected frame-synchronization pulse occurs, the XSYNCERR bit is set and the transmission of data B is restarted because no new data has been passed to XSR1 yet. In addition, if XINTM = 11b, the McBSP sends a transmit interrupt (XINT) request to the CPU.

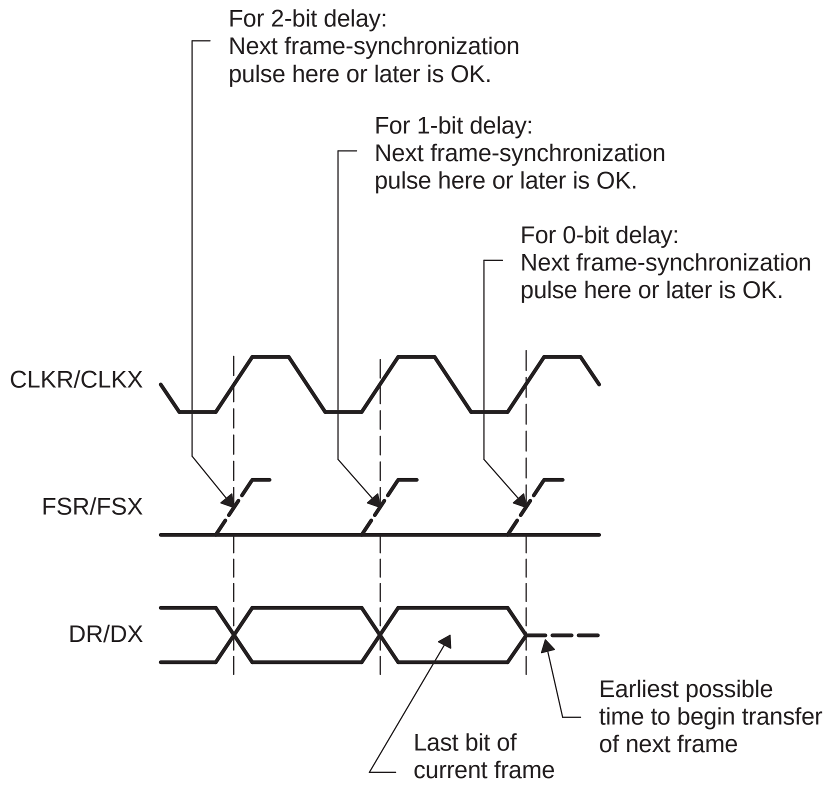
Figure 15-30. An Unexpected Frame-Synchronization Pulse During a McBSP Transmission



15.5.5.3 Preventing Unexpected Transmit Frame-Synchronization Pulses

Each frame transfer can be delayed by 0, 1, or 2 CLKX cycles, depending on the value in the XDATDLY bits of XCR2. For each possible data delay, Figure 15-31 shows when a new frame-synchronization pulse on FSX can safely occur relative to the last bit of the current frame.

Figure 15-31. Proper Positioning of Frame-Synchronization Pulses


For 2-bit delay:
Next frame-synchronization pulse here or later is OK.
For 1-bit delay:
Next frame-synchronization pulse here or later is OK.
For 0-bit delay:
Next frame-synchronization pulse here or later is OK.
CLKR/CLKX
FSR/FSX
DR/DX
Earliest possible time to begin transfer of next frame
Last bit of current frame

15.6 Multichannel Selection Modes

This section discusses the multichannel selection modes for the McBSP.

15.6.1 Channels, Blocks, and Partitions

A McBSP channel is a time slot for shifting in/out the bits of one serial word. Each McBSP supports up to 128 channels for reception and 128 channels for transmission.

In the receiver and in the transmitter, the 128 available channels are divided into eight blocks that each contain 16 contiguous channels (see Table 15-8 through Table 15-10) :

- It is possible to have two receive partitions (A & B) and 8 transmit partitions (A – H).
- McBSP can transmit/receive on selected channels.
- Each channel partition has a dedicated channel-enable register. Each bit controls whether data flow is allowed or prevented in one of the channels assigned to that partition.
- There are three transmit multichannel modes and one receive multichannel mode.

Table 15-8. Block - Channel Assignment

Block	Channels
0	0 -15
1	16 - 31
2	32 - 47
3	48 - 63
4	64 - 79
5	80 - 95
6	96 - 111
7	112 - 127

The blocks are assigned to partitions according to the selected partition mode. In the two-partition mode (described in Section 15.6.4), you assign one even-numbered block (0, 2, 4, or 6) to partition A and one odd-numbered block (1, 3, 5, or 7) to partition B. In the 8-partition mode (described in Section 15.6.5), blocks 0 through 7 are automatically assigned to partitions, A through H, respectively.

Table 15-9. 2-Partition Mode

Partition	Blocks
A	0 or 2 or 4 or 6
B	1 or 3 or 5 or 7

Table 15-10. 8-Partition mode

Partition	Blocks	Channels
A	0	0 -15
B	1	16 - 31
C	2	32 - 47
D	3	48 - 63
E	4	64 - 79
F	5	80 - 95
G	6	96 - 111
H	7	112 - 127

The number of partitions for reception and the number of partitions for transmission are independent. For example, it is possible to use two receive partitions (A and B) and eight transmit partitions (A-H).

15.6.2 Multichannel Selection

When a McBSP uses a time-division multiplexed (TDM) data stream while communicating with other McBSPs or serial devices, the McBSP may need to receive and/or transmit on only a few channels. To save memory and bus bandwidth, you can use a multichannel selection mode to prevent data flow in some of the channels.

Each channel partition has a dedicated channel enable register. If the appropriate multichannel selection mode is on, each bit in the register controls whether data flow is allowed or prevented in one of the channels that is assigned to that partition.

The McBSP has one receive multichannel selection mode (described in Section 15.6.6) and three transmit multichannel selection modes (described in Section 15.6.7).

15.6.3 Configuring a Frame for Multichannel Selection

Before you enable a multichannel selection mode, make sure you properly configure the data frame:

- Select a single-phase frame (RPHASE/XPHASE = 0). Each frame represents a TDM data stream.
- Set a frame length (in RFRLEN1/XFRLEN1) that includes the highest-numbered channel to be used. For example, if you plan to use channels 0, 15, and 39 for reception, the receive frame length must be at least 40 (RFRLEN1 = 39). If XFRLEN1 = 39 in this case, the receiver creates 40 time slots per frame but only receives data during time slots 0, 15, and 39 of each frame.

15.6.4 Using Two Partitions

For multichannel selection operation in the receiver and/or the transmitter, you can use two partitions or eight partitions (described in Section 15.6.5). If you choose the 2-partition mode (RMCME = 0 for reception, XMCME = 0 for transmission), McBSP channels are activated using an alternating scheme. In response to a frame-synchronization pulse, the receiver or transmitter begins with the channels in partition A and then alternates between partitions B and A until the complete frame has been transferred. When the next frame-synchronization pulse occurs, the next frame is transferred beginning with the channels in partition A.

15.6.4.1 Assigning Blocks to Partitions A and B

For reception, any two of the eight receive-channel blocks can be assigned to receive partitions A and B, which means up to 32 receive channels can be enabled at any given point in time. Similarly, any two of the eight transmit-channel blocks (up 32 enabled transmit channels) can be assigned to transmit partitions A and B.

For reception:

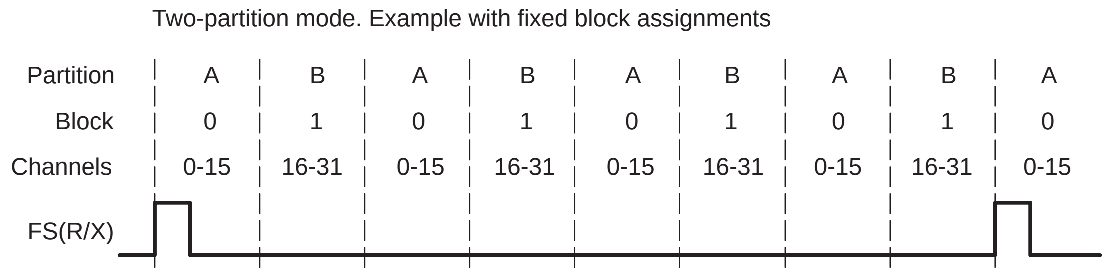
- Assign an even-numbered channel block (0, 2, 4, or 6) to receive partition A by writing to the RPABLK bits. In the receive multichannel selection mode (described in Section 15.6.6), the channels in this partition are controlled by receive channel enable register A (RCERA).
- Assign an odd-numbered block (1, 3, 5, or 7) to receive partition B with the RPBBLK bits. In the receive multichannel selection mode, the channels in this partition are controlled by receive channel enable register B (RCERB).

For transmission:

- Assign an even-numbered channel block (0, 2, 4, or 6) to transmit partition A by writing to the XPABLK bits. In one of the transmit multichannel selection modes (described in Section 15.6.7), the channels in this partition are controlled by transmit channel enable register A (XCERA).
- Assign an odd-numbered block (1, 3, 5, or 7) to transmit partition B with the XPBBLK bits. In one of the transmit multichannel selection modes, the channels in this partition are controlled by transmit channel enable register B (XCERB).

Figure 15-32 shows an example of alternating between the channels of partition A and the channels of partition B. Channels 0-15 have been assigned to partition A, and channels 16-31 have been assigned to partition B. In response to a frame-synchronization pulse, the McBSP begins a frame transfer with partition A and then alternates between partitions B and A until the complete frame is transferred.

Figure 15-32. Alternating Between the Channels of Partition A and the Channels of Partition B



As explained in Section 15.6.4.2, you can dynamically change which blocks of channels are assigned to the partitions.

15.6.4.2 Reassigning Blocks During Reception/Transmission

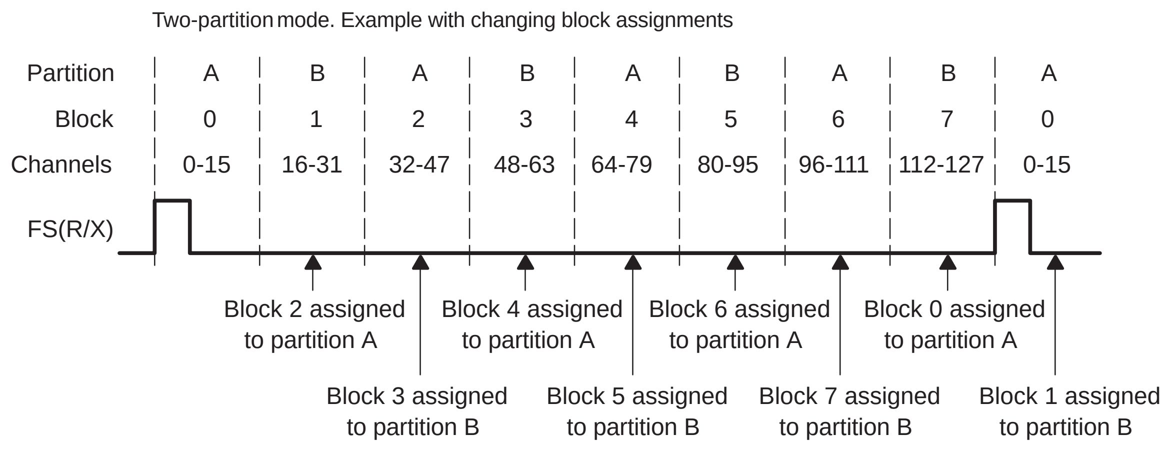
If you want to use more than 32 channels, you can change which channel blocks are assigned to partitions A and B during the course of a data transfer. However, these changes must be carefully timed. While a partition is being transferred, its associated block assignment bits cannot be modified and its associated channel enable register cannot be modified. For example, if block 3 is being transferred and block 3 is assigned to partition A, you can modify neither (R/X)PABLK to assign different channels to partition A nor (R/X)CERA to change the channel configuration for partition A.

Several features of the McBSP help you time the reassignment:

- The block of channels currently involved in reception/transmission (the current block) is reflected in the RCBLK/XCBLK bits. Your program can poll these bits to determine which partition is active. When a partition is not active, it is safe to change its block assignment and channel configuration.
- At the end of every block (at the boundary of two partitions), an interrupt can be sent to the CPU. In response to the interrupt, the CPU can then check the RCBLK/XCBLK bits and update the inactive partition. See Section 15.6.7.3, *Using Interrupts Between Block Transfers*.

Figure 15-33 shows an example of reassigning channels throughout a data transfer. In response to a frame-synchronization pulse, the McBSP alternates between partitions A and B. Whenever partition B is active, the CPU changes the block assignment for partition A. Whenever partition A is active, the CPU changes the block assignment for partition B.

Figure 15-33. Reassigning Channel Blocks Throughout a McBSP Data Transfer


Two-partition mode. Example with changing block assignments
Partition
A
B
A
B
A
B
A
B
A
Block
0
1
2
3
4
5
6
7
0
Channels
0-15
16-31
32-47
48-63
64-79
80-95
96-111
112-127
0-15
FS(R/X)
Block 2 assigned to partition A
Block 4 assigned to partition A
Block 6 assigned to partition A
Block 0 assigned to partition A
Block 3 assigned to partition B
Block 5 assigned to partition B
Block 7 assigned to partition B
Block 1 assigned to partition B

15.6.5 Using Eight Partitions

For multichannel selection operation in the receiver and/or the transmitter, you can use eight partitions or two partitions (described in Section 15.6.4). If you choose the 8-partition mode (RMCME = 1 for reception, XMCME = 1 for transmission), McBSP channels are activated in the following order: A, B, C, D, E, F, G, H. In response to a frame-synchronization pulse, the receiver or transmitter begins with the channels in partition A and then continues with the other partitions in order until the complete frame has been transferred. When the next frame-synchronization pulse occurs, the next frame is transferred, beginning with the channels in partition A.

In the 8-partition mode, the (R/X)PABLK and (R/X)PBBLK bits are ignored and the 16-channel blocks are assigned to the partitions as shown in Table 15-11 and Table 15-12. These assignments cannot be changed. The tables also show the registers used to control the channels in the partitions.

Table 15-11. Receive Channel Assignment and Control With Eight Receive Partitions

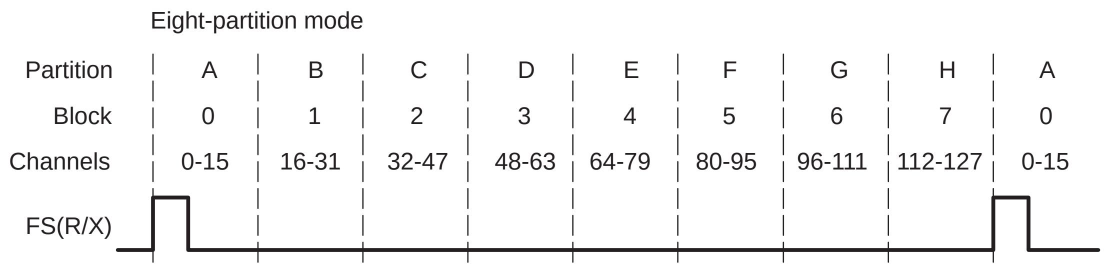
Receive Partition	Assigned Block of Receive Channels	Register Used For Channel Control
A	Block 0: channels 0 through 15	RCERA
B	Block 1: channels 16 through 31	RCERB
C	Block 2: channels 32 through 47	RCERC
D	Block 3: channels 48 through 63	RCERD
E	Block 4: channels 64 through 79	RCERE
F	Block 5: channels 80 through 95	RCERF
G	Block 6: channels 96 through 111	RCERG
H	Block 7: channels 112 through 127	RCERH

Table 15-12. Transmit Channel Assignment and Control When Eight Transmit Partitions Are Used

Transmit Partition	Assigned Block of Transmit Channels	Register Used For Channel Control
A	Block 0: channels 0 through 15	XCERA
B	Block 1: channels 16 through 31	XCERB
C	Block 2: channels 32 through 47	XCERC
D	Block 3: channels 48 through 63	XCERD
E	Block 4: channels 64 through 79	XCERE
F	Block 5: channels 80 through 95	XCERF
G	Block 6: channels 96 through 111	XCERG
H	Block 7: channels 112 through 127	XCERH

Figure 15-34 shows an example of the McBSP using the 8-partition mode. In response to a frame-synchronization pulse, the McBSP begins a frame transfer with partition A and then activates B, C, D, E, F, G, and H to complete a 128-word frame.

Figure 15-34. McBSP Data Transfer in the 8-Partition Mode



15.6.6 Receive Multichannel Selection Mode

The RMCM bit of MCR1 determines whether all channels or only selected channels are enabled for reception. When RMCM = 0, all 128 receive channels are enabled and cannot be disabled. When RMCM = 1, the receive multichannel selection mode is enabled. In this mode:

- Channels can be individually enabled or disabled. The only channels enabled are those selected in the appropriate receive channel enable registers (RCERs). The way channels are assigned to the RCERs depends on the number of receive channel partitions (2 or 8), as defined by the RMCME bit of MCR1.
- If a receive channel is disabled, any bits received in that channel are passed only as far as the receive buffer register(s) (RBR(s)). The receiver does not copy the content of the RBR(s) to the DRR(s), and as a result, does not set the receiver ready bit (RRDY). Therefore, no DMA synchronization event (REVT) is generated and, if the receiver interrupt mode depends on RRDY (RINTM = 00b), no interrupt is generated.

As an example of how the McBSP behaves in the receive multichannel selection mode, suppose you enable only channels 0, 15, and 39 and that the frame length is 40. The McBSP:

1. Accepts bits shifted in from the DR pin in channel 0
2. Ignores bits received in channels 1-14
3. Accepts bits shifted in from the DR pin in channel 15
4. Ignores bits received in channels 16-38
5. Accepts bits shifted in from the DR pin in channel 39

15.6.7 Transmit Multichannel Selection Modes

The XMCM bits of XCR2 determine whether all channels or only selected channels are enabled and unmasked for transmission. More details on enabling and masking are in Section 15.6.7.1. The McBSP has three transmit multichannel selection modes (XMCM = 01b, XMCM = 10b, and XMCM = 11b), which are described in the following table.

Table 15-13. Selecting a Transmit Multichannel Selection Mode With the XMCM Bits

XMCM	Transmit Multichannel Selection Mode
00b	No transmit multichannel selection mode is on. All channels are enabled and unmasked. No channels can be disabled or masked.
01b	All channels are disabled unless they are selected in the appropriate transmit channel enable registers (XCERs). If enabled, a channel in this mode is also unmasked. The XMCME bit of MCR2 determines whether 32 channels or 128 channels are selectable in XCERs.
10b	All channels are enabled, but they are masked unless they are selected in the appropriate transmit channel enable registers (XCERs). The XMCME bit of MCR2 determines whether 32 channels or 128 channels are selectable in XCERs.
11b	This mode is used for symmetric transmission and reception. All channels are disabled for transmission unless they are enabled for reception in the appropriate receive channel enable registers (RCERs). Once enabled, they are masked unless they are also selected in the appropriate transmit channel enable registers (XCERs). The XMCME bit of MCR2 determines whether 32 channels or 128 channels are selectable in RCERs and XCERs.

As an example of how the McBSP behaves in a transmit multichannel selection mode, suppose that XMCM = 01b (all channels disabled unless individually enabled) and that you have enabled only channels 0, 15, and 39. Suppose also that the frame length is 40. The McBSP:…

1. Shifts data to the DX pin in channel 0
2. Places the DX pin in the high impedance state in channels 1-14
3. Shifts data to the DX pin in channel 15
4. Places the DX pin in the high impedance state in channels 16-38
5. Shifts data to the DX pin in channel 39

15.6.7.1 Disabling/Enabling Versus Masking/Unmasking

For transmission, a channel can be:

- Enabled and unmasked (transmission can begin and can be completed)
- Enabled but masked (transmission can begin but cannot be completed)
- Disabled (transmission cannot occur)

The following definitions explain the channel control options:

Enabled channel	A channel that can begin transmission by passing data from the data transmit register(s) (DXR(s)) to the transmit shift registers (XSR(s)).
Masked channel	A channel that cannot complete transmission. The DX pin is held in the high impedance state; data cannot be shifted out on the DX pin. In systems where symmetric transmit and receive provides software benefits, this feature allows transmit channels to be disabled on a shared serial bus. A similar feature is not needed for reception because multiple receptions cannot cause serial bus contention.
Disabled channel	A channel that is not enabled. A disabled channel is also masked. Because no DXR-to-XSR copy occurs, the XRDY bit of SPCR2 is not set. Therefore, no DMA synchronization event (XEVT) is generated, and if the transmit interrupt mode depends on XRDY (XINTM = 00b in SPCR2), no interrupt is generated. The XEMPTY bit of SPCR2 is not affected.
Unmasked channel	A channel that is not masked. Data in the XSR(s) is shifted out on the DX pin.

15.6.7.2 Activity on McBSP Pins for Different Values of XMCM

Figure 15-35 shows the activity on the McBSP pins for the various XMCM values. In all cases, the transmit frame is configured as follows:

- XPHASE = 0: Single-phase frame (required for multichannel selection modes)
- XFRLEN1 = 0000011b: 4 words per frame

- XWDLEN1 = 000b: 8 bits per word
- XMCME = 0: 2-partition mode (only partitions A and B used)

In the case where XMCM = 11b, transmission and reception are symmetric, which means the corresponding bits for the receiver (RPHASE, RFRLEN1, RWDLEN1, and RMCME) must have the same values as XPHASE, XFRLEN1, and XWDLEN1, respectively.

In the figure, the arrows showing where the various events occur are only sample indications. Wherever possible, there is a time window in which these events can occur.

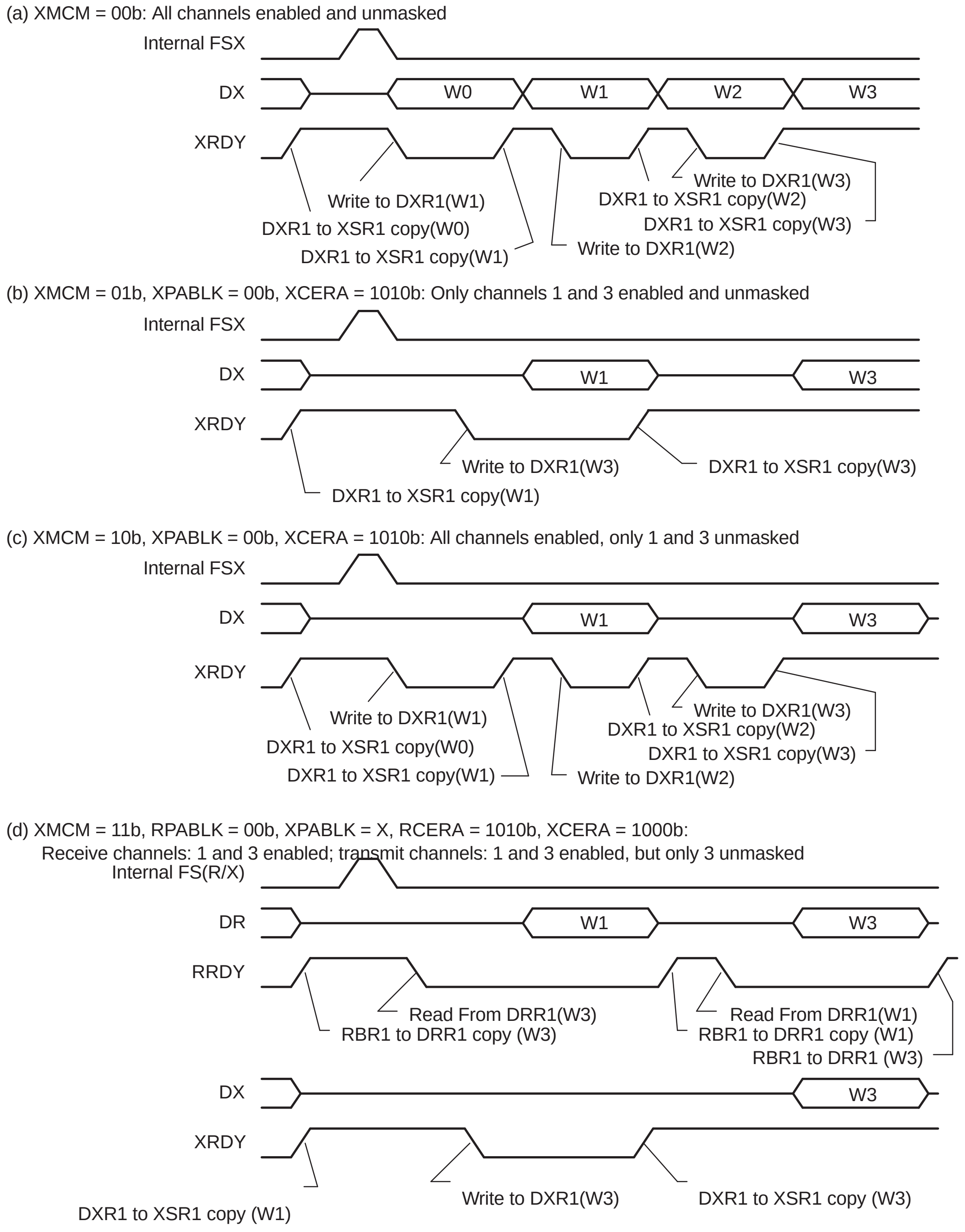
15.6.7.3 Using Interrupts Between Block Transfers

When a multichannel selection mode is used, an interrupt request can be sent to the CPU at the end of every 16-channel block (at the boundary between partitions and at the end of the frame). In the receive multichannel selection mode, a receive interrupt (RINT) request is generated at the end of each block transfer if RINTM = 01b. In any of the transmit multichannel selection modes, a transmit interrupt (XINT) request is generated at the end of each block transfer if XINTM = 01b. When RINTM/XINTM = 01b, no interrupt is generated unless a multichannel selection mode is on.

These interrupt pulses are active high and last for two CPU clock cycles.

This type of interrupt is especially helpful if you are using the two-partition mode (described in Section 15.6.4) and you want to know when you can assign a different block of channels to partition A or B.

Figure 15-35. Activity on McBSP Pins for the Possible Values of XMCM



15.7 SPI Operation Using the Clock Stop Mode

This section explains how to use the McBSP in SPI mode.

15.7.1 SPI Protocol

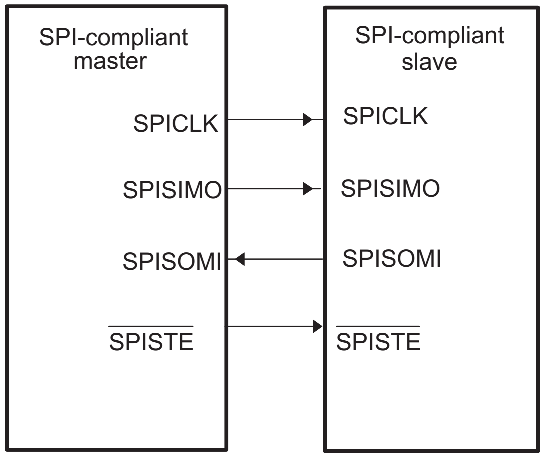
The SPI protocol is a master-slave configuration with one master device and one or more slave devices. The interface consists of the following four signals:

- Serial data input (also referred to as slave out/master in, or SOMI)

- Serial data output (also referred to as slave in/master out, or SIMO)
- Shift-clock (also referred to as SPICLK)
- Slave-enable signal (also referred to as $\overline{\text{SPISTE}}$)

A typical SPI interface with a single slave device is shown in Figure 15-36.

Figure 15-36. Typical SPI Interface



The master device controls the flow of communication by providing shift-clock and slave-enable signals. The slave-enable signal is an optional active-low signal that enables the serial data input and output of the slave device (device not sending out the clock).

In the absence of a dedicated slave-enable signal, communication between the master and slave is determined by the presence or absence of an active shift-clock. When the McBSP is operating in SPI master mode and the $\overline{\text{SPISTE}}$ signal is not used by the slave SPI port, the slave device must remain enabled at all times, and multiple slaves cannot be used.

15.7.2 Clock Stop Mode

The clock stop mode of the McBSP provides compatibility with the SPI protocol. When the McBSP is configured in clock stop mode, the transmitter and receiver are internally synchronized so that the McBSP functions as an SPI master or slave device. The transmit clock signal (CLKX) corresponds to the serial clock signal (SPICLK) of the SPI protocol, while the transmit frame-synchronization signal (FSX) is used as the slave-enable signal ($\overline{\text{SPISTE}}$).

The receive clock signal (MCLKR) and receive frame-synchronization signal (FSR) are not used in the clock stop mode because these signals are internally connected to their transmit counterparts, CLKX and FSX.

15.7.3 Bits Used to Enable and Configure the Clock Stop Mode

The bits required to configure the McBSP as an SPI device are introduced in Table 15-14. Table 15-15 shows how the various combinations of the CLKSTP bit and the polarity bits CLKXP and CLKRP create four possible clock stop mode configurations. The timing diagrams in Section 15.7.4 show the effects of CLKSTP, CLKXP, and CLKRP.

Table 15-14. Bits Used to Enable and Configure the Clock Stop Mode

Bit Field	Description
CLKSTP bits of SPCR1	Use these bits to enable the clock stop mode and to select one of two timing variations. (See also Table 15-15.)
CLKXP bit of PCR	This bit determines the polarity of the CLKX signal. (See also Table 15-15.)
CLKRP bit of PCR	This bit determines the polarity of the MCLKR signal. (See also Table 15-15.)
CLKXM bit of PCR	This bit determines whether CLKX is an input signal (McBSP as slave) or an output signal (McBSP as master).

Table 15-14. Bits Used to Enable and Configure the Clock Stop Mode (continued)

Bit Field	Description
XPHASE bit of XCR2	You must use a single-phase transmit frame (XPHASE = 0).
RPHASE bit of RCR2	You must use a single-phase receive frame (RPHASE = 0).
XFRLEN1 bits of XCR1	You must use a transmit frame length of 1 serial word (XFRLEN1 = 0).
RFRLEN1 bits of RCR1	You must use a receive frame length of 1 serial word (RFRLEN1 = 0).
XWDLEN1 bits of XCR1	The XWDLEN1 bits determine the transmit packet length. XWDLEN1 must be equal to RWDLEN1 because in the clock stop mode. The McBSP transmit and receive circuits are synchronized to a single clock.
RWDLEN1 bits of RCR1	The RWDLEN1 bits determine the receive packet length. RWDLEN1 must be equal to XWDLEN1 because in the clock stop mode. The McBSP transmit and receive circuits are synchronized to a single clock.

Table 15-15. Effects of CLKSTP, CLKXP, and CLKRP on the Clock Scheme

Bit Settings	Clock Scheme

CLKSTP = 00b or 01b
CLKXP = 0 or 1
CLKRP = 0 or 1
	Clock stop mode disabled. Clock enabled for non-SPI mode.

CLKSTP = 10b
CLKXP = 0
CLKRP = 0
	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of MCLKR.

CLKSTP = 11b
CLKXP = 0
CLKRP = 1
	Low inactive state with delay: The McBSP transmits data one-half cycle ahead of the rising edge of CLKX and receives data on the rising edge of MCLKR.

CLKSTP = 10b
CLKXP = 1
CLKRP = 0
	High inactive state without delay: The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of MCLKR.

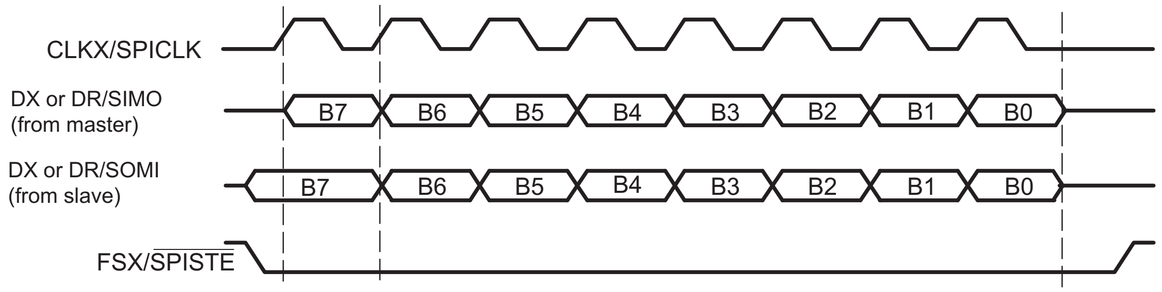
CLKSTP = 11b
CLKXP = 1
CLKRP = 1
	High inactive state with delay: The McBSP transmits data one-half cycle ahead of the falling edge of CLKX and receives data on the falling edge of MCLKR.

15.7.4 Clock Stop Mode Timing Diagrams

The timing diagrams for the four possible clock stop mode configurations are shown here. Notice that the frame-synchronization signal used in clock stop mode is active throughout the entire transmission as a slave-enable signal. Although the timing diagrams show 8-bit transfers, the packet length can be set to 8, 12, 16, 20, 24, or 32 bits per packet. The receive packet length is selected with the RWDLEN1 bits of RCR1, and the transmit packet length is selected with the XWDLEN1 bits of XCR1. For clock stop mode, the values of RWDLEN1 and XWDLEN1 must be the same because the McBSP transmit and receive circuits are synchronized to a single clock.

NOTE: Even if multiple words are consecutively transferred, the CLKX signal is always stopped and the FSX signal returns to the inactive state after a packet transfer. When consecutive packet transfers are performed, this leads to a minimum idle time of two bit-periods between each packet transfer.

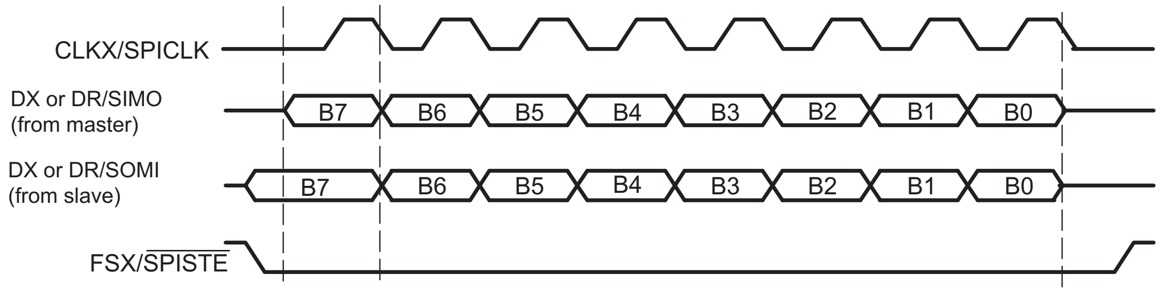
Figure 15-37. SPI Transfer With CLKSTP = 10b (No Clock Delay), CLKXP = 0, and CLKRP = 0



A If the McBSP is the SPI master (CLKXM = 1), SIMO = DX. If the McBSP is the SPI slave (CLKXM = 0), SIMO = DR.

B If the McBSP is the SPI master (CLKXM = 1), SOMI = DR. If the McBSP is the SPI slave (CLKXM = 0), SOMI = DX.

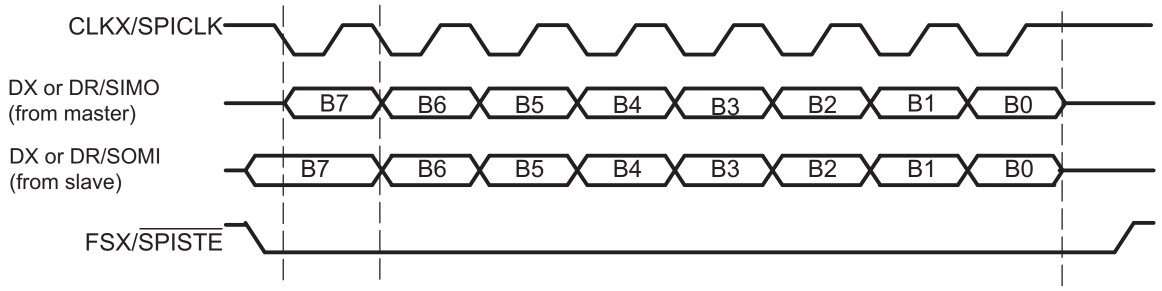
Figure 15-38. SPI Transfer With CLKSTP = 11b (Clock Delay), CLKXP = 0, CLKRP = 1



A If the McBSP is the SPI master (CLKXM = 1), SIMO = DX. If the McBSP is the SPI slave (CLKXM = 0), SIMO = DR.

B If the McBSP is the SPI master (CLKXM = 1), SOMI = DR. If the McBSP is the SPI slave (CLKXM = 0), SOMI = DX.

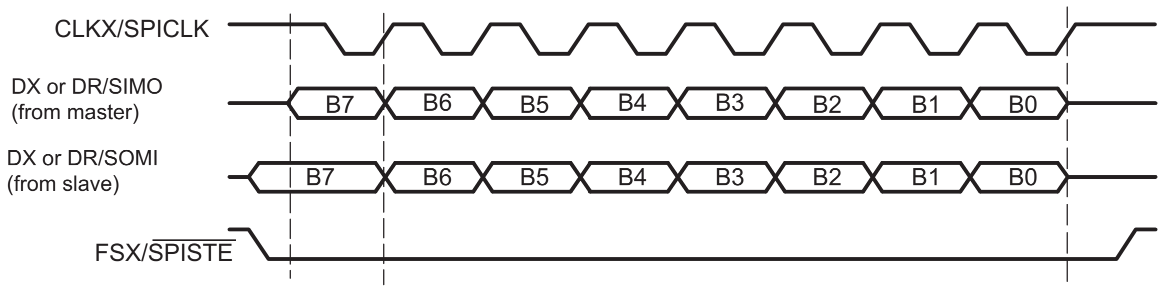
Figure 15-39. SPI Transfer With CLKSTP = 10b (No Clock Delay), CLKXP = 1, and CLKRP = 0



A If the McBSP is the SPI master (CLKXM = 1), SIMO = DX. If the McBSP is the SPI slave (CLKXM = 0), SIMO = DR.

B If the McBSP is the SPI master (CLKXM = 1), SOMI = DR. If the McBSP is the SPI slave (CLKXM = 0), SOMI = DX.

Figure 15-40. SPI Transfer With CLKSTP = 11b (Clock Delay), CLKXP = 1, CLKRP = 1



A If the McBSP is the SPI master (CLKXM = 1), SIMO=DX. If the McBSP is the SPI slave (CLKXM = 0), SIMO = DR.

B If the McBSP is the SPI master (CLKXM = 1), SOMI=DR. If the McBSP is the SPI slave (CLKXM = 0), SOMI = DX.

15.7.5 Procedure for Configuring a McBSP for SPI Operation

To configure the McBSP for SPI master or slave operation:

Step 1. Place the transmitter and receiver in reset.

Clear the transmitter reset bit (XRST = 0) in SPCR2 to reset the transmitter. Clear the receiver reset bit (RRST = 0) in SPCR1 to reset the receiver.

Step 2. Place the sample rate generator in reset.

Clear the sample rate generator reset bit (GRST = 0) in SPCR2 to reset the sample rate generator.

Step 3. Program registers that affect SPI operation.

Program the appropriate McBSP registers to configure the McBSP for proper operation as an SPI master or an SPI slave. For a list of important bits settings, see one of the following topics:

- *McBSP as the SPI Master* (Section 15.7.6)
- *McBSP as an SPI Slave* (Section 15.7.7)

Step 4. Enable the sample rate generator.

To release the sample rate generator from reset, set the sample rate generator reset bit (GRST = 1) in SPCR2.

Make sure that during the write to SPCR2, you only modify GRST. Otherwise, you modify the McBSP configuration you selected in the previous step.

Step 5. Enable the transmitter and receiver.

After the sample rate generator is released from reset, wait two sample rate generator clock periods for the McBSP logic to stabilize.

If the CPU services the McBSP transmit and receive buffers, then you can immediately enable the transmitter (XRST = 1 in SPCR2) and enable the receiver (RRST = 1 in SPCR1).

If the DMA controller services the McBSP transmit and receive buffers, then you must first configure the DMA controller (this includes enabling the channels that service the McBSP buffers). When the DMA controller is ready, make XRST = 1 and RRST = 1.

In either case, make sure you only change XRST and RRST when you write to SPCR2 and SPCR1. Otherwise, you modify the bit settings you selected earlier in this procedure.

After the transmitter and receiver are released from reset, wait two sample rate generator clock periods for the McBSP logic to stabilize.

Step 6. If necessary, enable the frame-synchronization logic of the sample rate generator.

After the required data acquisition setup is done (DXR[1,2] is loaded with data), set FRST = 1 if an internally generated frame-synchronization pulse is required (that is, if the McBSP is the SPI master).

15.7.6 McBSP as the SPI Master

An SPI interface with the McBSP used as the master is shown in Figure 15-41. When the McBSP is configured as a master, the transmit output signal (DX) is used as the SIMO signal of the SPI protocol and the receive input signal (DR) is used as the SOMI signal.

The register bit values required to configure the McBSP as a master are listed in Table 15-16. After the table are more details about the configuration requirements.

Figure 15-41. SPI Interface with McBSP Used as Master

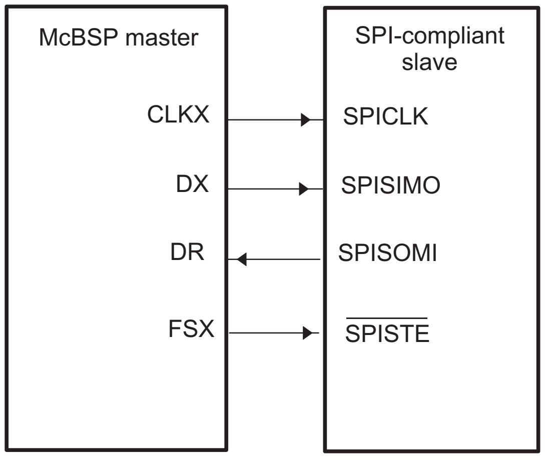


Table 15-16. Bit Values Required to Configure the McBSP as an SPI Master

Required Bit Setting	Description
CLKSTP = 10b or 11b	The clock stop mode (without or with a clock delay) is selected.
CLKXP = 0 or 1	The polarity of CLKX as seen on the MCLKX pin is positive (CLKXP = 0) or negative (CLKXP = 1).
CLKRP = 0 or 1	The polarity of MCLKR as seen on the MCLKR pin is positive (CLKRP = 0) or negative (CLKRP = 1).
CLKXM = 1	The MCLKX pin is an output pin driven by the internal sample rate generator. Because CLKSTP is equal to 10b or 11b, MCLKR is driven internally by CLKX.
SCLKME = 0	The clock generated by the sample rate generator (CLKG) is derived from the CPU clock.
CLKSM = 1	
CLKGDV is a value from 1 to 255	CLKGDV defines the divide down value for CLKG.
FSXM = 1	The FSX pin is an output pin driven according to the FSGM bit. (See theTMS320F28335 Applications and Media Processor Data Manual (SPRS224) for more information.
FSGM = 0	The transmitter drives a frame-synchronization pulse on the FSX pin every time data is transferred from DXR1 to XSR1.
FSXP = 1	The FSX pin is active low.
XDATDLY = 01b	This setting provides the correct setup time on the FSX signal.
RDATDLY = 01b	

When the McBSP functions as the SPI master, it controls the transmission of data by producing the serial clock signal. The clock signal on the MCLKX pin is enabled only during packet transfers. When packets are not being transferred, the MCLKX pin remains high or low depending on the polarity used.

For SPI master operation, the MCLKX pin must be configured as an output. The sample rate generator is then used to derive the CLKX signal from the CPU clock. The clock stop mode internally connects the MCLKX pin to the MCLKR signal so that no external signal connection is required on the MCLKR pin and both the transmit and receive circuits are clocked by the master clock (CLKX).

The data delay parameters of the McBSP (XDATDLY and RDATDLY) must be set to 1 for proper SPI master operation. A data delay value of 0 or 2 is undefined in the clock stop mode.

The McBSP can also provide a slave-enable signal ($\overline{\text{SPISTE}}$) on the FSX pin. If a slave-enable signal is required, the FSX pin must be configured as an output and the transmitter must be configured so that a frame-synchronization pulse is generated automatically each time a packet is transmitted (FSGM = 0). The polarity of the FSX pin is programmable high or low; however, in most cases the pin must be configured active low.

When the McBSP is configured as described for SPI-master operation, the bit fields for frame-synchronization pulse width (FWID) and frame-synchronization period (FPER) are overridden, and custom frame-synchronization waveforms are not allowed. To see the resulting waveform produced on the FSX pin, see the timing diagrams in Section 15.7.4. The signal becomes active before the first bit of a packet transfer, and remains active until the last bit of the packet is transferred. After the packet transfer is complete, the FSX signal returns to the inactive state.

15.7.7 McBSP as an SPI Slave

An SPI interface with the McBSP used as a slave is shown in Figure 15-42. When the McBSP is configured as a slave, DX is used as the SOMI signal and DR is used as the SIMO signal.

The register bit values required to configure the McBSP as a slave are listed in Table 15-17. Following the table are more details about configuration requirements.

Figure 15-42. SPI Interface With McBSP Used as Slave

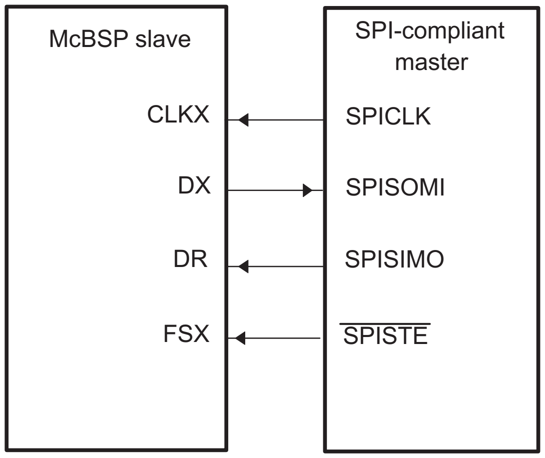


Table 15-17. Bit Values Required to Configure the McBSP as an SPI Slave

Required Bit Setting	Description
CLKSTP = 10b or 11b	The clock stop mode (without or with a clock delay) is selected.
CLKXP = 0 or 1	The polarity of CLKX as seen on the MCLKX pin is positive (CLKXP = 0) or negative (CLKXP = 1).
CLKRP = 0 or 1	The polarity of MCLKR as seen on the MCLKR pin is positive (CLKRP = 0) or negative (CLKRP = 1).
CLKXM = 0	The MCLKX pin is an input pin, so that it can be driven by the SPI master. Because CLKSTP = 10b or 11b, MCLKR is driven internally by CLKX.

SCLKME = 0
CLKSM = 1
	The clock generated by the sample rate generator (CLKG) is derived from the CPU clock. (The sample rate generator is used to synchronize the McBSP logic with the externally-generated master clock.)

CLKGDV = 1	The sample rate generator divides the CPU clock before generating CLKG.
FSXM = 0	The FSX pin is an input pin, so that it can be driven by the SPI master.
FSXP = 1	The FSX pin is active low.

XDATDLY = 00b
RDATDLY = 00b
	These bits must be 0s for SPI slave operation.

When the McBSP is used as an SPI slave, the master clock and slave-enable signals are generated externally by a master device. Accordingly, the CLKX and FSX pins must be configured as inputs. The MCLKX pin is internally connected to the MCLKR signal, so that both the transmit and receive circuits of the McBSP are clocked by the external master clock. The FSX pin is also internally connected to the FSR signal, and no external signal connections are required on the MCLKR and FSR pins.

Although the CLKX signal is generated externally by the master and is asynchronous to the McBSP, the sample rate generator of the McBSP must be enabled for proper SPI slave operation. The sample rate generator must be programmed to its maximum rate of half the CPU clock rate. The internal sample rate clock is then used to synchronize the McBSP logic to the external master clock and slave-enable signals.

The McBSP requires an active edge of the slave-enable signal on the FSX input for each transfer. This means that the master device must assert the slave-enable signal at the beginning of each transfer, and deassert the signal after the completion of each packet transfer; the slave-enable signal cannot remain active between transfers. Unlike the standard SPI, this pin cannot be tied low all the time.

The data delay parameters of the McBSP must be set to 0 for proper SPI slave operation. A value of 1 or 2 is undefined in the clock stop mode.

15.8 Receiver Configuration

To configure the McBSP receiver, perform the following procedure:

1. Place the McBSP/receiver in reset (see Section 15.8.2).
2. Program McBSP registers for the desired receiver operation (see Section 15.8.1).
3. Take the receiver out of reset (see Section 15.8.2).

15.8.1 Programming the McBSP Registers for the Desired Receiver Operation

The following is a list of important tasks to be performed when you are configuring the McBSP receiver. Each task corresponds to one or more McBSP register bit fields.

- Global behavior:
  - Set the receiver pins to operate as McBSP pins.
  - Enable/disable the digital loopback mode.
  - Enable/disable the clock stop mode.
  - Enable/disable the receive multichannel selection mode.
- Data behavior:
  - Choose 1 or 2 phases for the receive frame.
  - Set the receive word length(s).
  - Set the receive frame length.
  - Enable/disable the receive frame-synchronization ignore function.
  - Set the receive companding mode.
  - Set the receive data delay.
  - Set the receive sign-extension and justification mode.
  - Set the receive interrupt mode.
- Frame-synchronization behavior:
  - Set the receive frame-synchronization mode.
  - Set the receive frame-synchronization polarity.
  - Set the sample rate generator (SRG) frame-synchronization period and pulse width.
- Clock behavior:
  - Set the receive clock mode.
  - Set the receive clock polarity.
  - Set the SRG clock divide-down value.
  - Set the SRG clock synchronization mode.
  - Set the SRG clock mode (choose an input clock).
  - Set the SRG input clock polarity.

15.8.2 Resetting and Enabling the Receiver

The first step of the receiver configuration procedure is to reset the receiver, and the last step is to enable the receiver (to take it out of reset). Table 15-18 describes the bits used for both of these steps.

Table 15-18. Register Bits Used to Reset or Enable the McBSP Receiver Field Descriptions

Register	Bit	Field	Value	Description
SPCR2	7	FRST		Frame-synchronization logic reset
			0	Frame-synchronization logic is reset. The sample rate generator does not generate frame-synchronization signal FSG, even if GRST = 1.
			1	If GRST = 1, frame-synchronization signal FSG is generated after (FPER + 1) number of CLKG clock cycles; all frame counters are loaded with their programmed values.
SPCR2	6	GRST		Sample rate generator reset
			0	Sample rate generator is reset. If GRST = 0 due to a DSP reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven low (inactive). If GRST = 0 due to program code, CLKG and FSG are both driven low (inactive).
			1	Sample rate generator is enabled. CLKG is driven according to the configuration programmed in the sample rate generator registers (SRGR[1,2]). If FRST = 1, the generator also generates the frame-synchronization signal FSG as programmed in the sample rate generator registers.
SPCR1	0	RRST		Receiver reset
			0	The serial port receiver is disabled and in the reset state.
			1	The serial port receiver is enabled.

15.8.2.1 Reset Considerations

The serial port can be reset in the following two ways:

1. The DSP reset ($\overline{\text{XRS}}$ signal driven low) places the receiver, transmitter, and sample rate generator in reset. When the device reset is removed ($\overline{\text{XRS}}$ signal released), GRST = FRST = RRST = XRST = 0 keep the entire serial port in the reset state, provided the McBSP clock is turned on.
2. The serial port transmitter and receiver can be reset directly using the RRST and XRST bits in the serial port control registers. The sample rate generator can be reset directly using the GRST bit in SPCR2.

Table 15-19 shows the state of McBSP pins when the serial port is reset due to a device reset and a direct receiver/transmitter reset.

For more details about McBSP reset conditions and effects, see Section 15.10.2, *Resetting and Initializing a McBSP*.

Table 15-19. Reset State of Each McBSP Pin

Pin	Possible State(s)	State Forced By Device Reset	State Forced By Receiver Reset (RRST = 0 and GRST = 1)
MDRx	I	GPIO Input	Input
MCLKRx	I/O/Z	GPIO Input	Known state if input; MCLKR running if output
MFSRx	I/O/Z	GPIO Input	Known state if input; FSRP inactive state if output
			Transmitter reset (XRST = 0 and GRST = 1)
MDXx	O/Z	GPIO Input	Low impedance after transmit bit clock provided
MCLKXx	I/O/Z	GPIO Input	Known state if input; CLKX running if output
MFSXx	I/O/Z	GPIO Input	Known state if input; FSXP inactive state if output

15.8.3 Set the Receiver Pins to Operate as McBSP Pins

To configure a pin for its McBSP function , you should configure the bits of the GPxMUXn register appropriately. In addition to this, bits 12 and 13 of the PCR register must be set to 0. These bits are defined as reserved.

15.8.4 Enable/Disable the Digital Loopback Mode

The DLB bit determines whether the digital loopback mode is on. DLB is described in Table 15-20.

Table 15-20. Register Bit Used to Enable/Disable the Digital Loopback Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR1	15	DLB	Digital loopback mode		R/W	0
			DLB = 0	Digital loopback mode is disabled.		
			DLB = 1	Digital loopback mode is enabled.		

15.8.4.1 Digital Loopback Mode

In the digital loopback mode, the receive signals are connected internally through multiplexers to the corresponding transmit signals, as shown in Table 15-21. This mode allows testing of serial port code with a single DSP device; the McBSP receives the data it transmits.

Table 15-21. Receive Signals Connected to Transmit Signals in Digital Loopback Mode

This Receive Signal	Is Fed Internally by This Transmit Signal
MDR (receive data)	MDX (transmit data)
MFSR (receive frame synchronization)	MFSX (transmit frame synchronization)
MCLKR (receive clock)	MCLKX (transmit clock)

15.8.5 Enable/Disable the Clock Stop Mode

The CLKSTP bits determine whether the clock stop mode is on. CLKSTP is described in Table 15-22.

Table 15-22. Register Bits Used to Enable/Disable the Clock Stop Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR1	12-11	CLKSTP	Clock stop mode		R/W	00
			CLKSTP = 0Xb	Clock stop mode disabled; normal clocking for non-SPI mode		
			CLKSTP = 10b	Clock stop mode enabled, without clock delay		
			CLKSTP = 11b	Clock stop mode enabled, with clock delay		

15.8.5.1 Clock Stop Mode

The clock stop mode supports the SPI master-slave protocol. If you do not plan to use the SPI protocol, you can clear CLKSTP to disable the clock stop mode.

In the clock stop mode, the clock stops at the end of each data transfer. At the beginning of each data transfer, the clock starts immediately (CLKSTP = 10b) or after a half-cycle delay (CLKSTP = 11b). The CLKXP bit determines whether the starting edge of the clock on the MCLKX pin is rising or falling. The CLKRP bit determines whether receive data is sampled on the rising or falling edge of the clock shown on the MCLKR pin.

Table 15-23 summarizes the impact of CLKSTP, CLKXP, and CLKRP on serial port operation. In the clock stop mode, the receive clock is tied internally to the transmit clock, and the receive frame-synchronization signal is tied internally to the transmit frame-synchronization signal.

Table 15-23. Effects of CLKSTP, CLKXP, and CLKRP on the Clock Scheme

Bit Settings	Clock Scheme

CLKSTP = 00b or 01b
CLKXP = 0 or 1
CLKRP = 0 or 1
	Clock stop mode disabled. Clock enabled for non-SPI mode.

CLKSTP = 10b
CLKXP = 0
CLKRP = 0
	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of MCLKR.

CLKSTP = 11b
CLKXP = 0
CLKRP = 1
	Low inactive state with delay: The McBSP transmits data one-half cycle ahead of the rising edge of CLKX and receives data on the rising edge of MCLKR.

CLKSTP = 10b
CLKXP = 1
CLKRP = 0
	High inactive state without delay: The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of MCLKR.

CLKSTP = 11b
CLKXP = 1
CLKRP = 1
	High inactive state with delay: The McBSP transmits data one-half cycle ahead of the falling edge of CLKX and receives data on the falling edge of MCLKR.

15.8.6 Enable/Disable the Receive Multichannel Selection Mode

The RMCM bit determines whether the receive multichannel selection mode is on. RMCM is described in Table 15-24. For more details, see Section 15.6.6, *Receive Multichannel Selection Mode*.

Table 15-24. Register Bit Used to Enable/Disable the Receive Multichannel Selection Mode

Register	Bit	Name	Function		Type	Reset Value
MCR1	0	RMCM	Receive multichannel selection mode		R/W	0
			RMCM = 0	The mode is disabled. All 128 channels are enabled.		
			RMCM = 1	The mode is enabled. Channels can be individually enabled or disabled. The only channels enabled are those selected in the appropriate receive channel enable registers (RCERs). The way channels are assigned to the RCERs depends on the number of receive channel partitions (2 or 8), as defined by the RMCME bit.		

15.8.7 Choose One or Two Phases for the Receive Frame

The RPHASE bit (see Table 15-25) determines whether the receive data frame has one or two phases.

Table 15-25. Register Bit Used to Choose One or Two Phases for the Receive Frame

Register	Bit	Name	Function		Type	Reset Value
RCR2	15	RPHASE	Receive phase number		R/W	0
			Specifies whether the receive frame has 1 or 2 phases.			
			RPHASE = 0	Single-phase frame		
			RPHASE = 1	Dual-phase frame		

15.8.8 Set the Receive Word Length(s)

The RWDLEN1 and RWDLEN2 bit fields (see Table 15-26) determine how many bits are in each serial word in phase 1 and in phase 2, respectively, of the receive data frame.

Table 15-26. Register Bits Used to Set the Receive Word Length(s)

Register	Bit	Name	Function		Type	Reset Value
RCR1	7-5	RWDLEN1	Receive word length 1		R/W	000
			Specifies the length of every serial word in phase 1 of the receive frame.			
			RWDLEN1 = 000	8 bits		
			RWDLEN1 = 001	12 bits		
			RWDLEN1 = 010	16 bits		
			RWDLEN1 = 011	20 bits		
			RWDLEN1 = 100	24 bits		
			RWDLEN1 = 101	32 bits		
			RWDLEN1 = 11X	Reserved		
RCR2	7-5	RWDLEN2	Receive word length 2		R/W	000
			If a dual-phase frame is selected, RWDLEN2 specifies the length of every serial word in phase 2 of the frame.			
			RWDLEN2 = 000	8 bits		
			RWDLEN2 = 001	12 bits		
			RWDLEN2 = 010	16 bits		
			RWDLEN2 = 011	20 bits		
			RWDLEN2 = 100	24 bits		
			RWDLEN2 = 101	32 bits		
			RWDLEN2 = 11X	Reserved		

15.8.8.1 Word Length Bits

Each frame can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, RWDLEN1 selects the length for every serial word received in the frame. If a dual-phase frame is selected, RWDLEN1 determines the length of the serial words in phase 1 of the frame and RWDLEN2 determines the word length in phase 2 of the frame.

15.8.9 Set the Receive Frame Length

The RFRLEN1 and RFRLEN2 bit fields (see Table 15-27) determine how many serial words are in phase 1 and in phase 2, respectively, of the receive data frame.

Table 15-27. Register Bits Used to Set the Receive Frame Length

Register	Bit	Name	Function		Type	Reset Value
RCR1	14-8	RFRLEN1	Receive frame length 1		R/W	000 0000
			(RFRLEN1 + 1) is the number of serial words in phase 1 of the receive frame.			
			RFRLEN1 = 000 0000	1 word in phase 1		
			RFRLEN1 = 000 0001	2 words in phase 1		
			|	|		
			|	|		
			RFRLEN1 = 111 1111	128 words in phase 1		

Table 15-27. Register Bits Used to Set the Receive Frame Length (continued)

Register	Bit	Name	Function		Type	Reset Value
RCR2	14-8	RFRLEN2	Receive frame length 2		R/W	000 0000
			If a dual-phase frame is selected, (RFRLEN2 + 1) is the number of serial words in phase 2 of the receive frame.			
			RFRLEN2 = 000 0000	1 word in phase 2		
			RFRLEN2 = 000 0001	2 words in phase 2		
			|	|		
			|	|		
			RFRLEN2 = 111 1111	128 words in phase 2		

15.8.9.1 Selected Frame Length

The receive frame length is the number of serial words in the receive frame. Each frame can have one or two phases, depending on value that you load into the RPHASE bit.

If a single-phase frame is selected (RPHASE = 0), the frame length is equal to the length of phase 1. If a dual-phase frame is selected (RPHASE = 1), the frame length is the length of phase 1 plus the length of phase 2.

The 7-bit RFRLEN fields allow up to 128 words per phase. See Table 15-28 for a summary of how to calculate the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization pulse.

Program the RFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For the example, if you want a phase length of 128 words in phase 1, load 127 into RFRLEN1.

Table 15-28. How to Calculate the Length of the Receive Frame

RPHASE	RFRLEN1	RFRLEN2	Frame Length
0	0 ≤ RFRLEN1 ≤ 127	Don't care	(RFRLEN1 + 1) words
1	0 ≤ RFRLEN1 ≤ 127	0 ≤ RFRLEN2 ≤ 127	(RFRLEN1 + 1) + (RFRLEN2 + 1) words

15.8.10 Enable/Disable the Receive Frame-Synchronization Ignore Function

The RFIG bit (see Table 15-29) controls the receive frame-synchronization ignore function.

Table 15-29. Register Bit Used to Enable/Disable the Receive Frame-Synchronization Ignore Function

Register	Bit	Name	Function		Type	Reset Value
RCR2	2	RFIG	Receive frame-synchronization ignore		R/W	0
			RFIG = 0	An unexpected receive frame-synchronization pulse causes the McBSP to restart the frame transfer.		
			RFIG = 1	The McBSP ignores unexpected receive frame-synchronization pulses.		

15.8.10.1 Unexpected Frame-Synchronization Pulses and the Frame-Synchronization Ignore Function

If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully received, this pulse is treated as an unexpected frame-synchronization pulse.

When RFIG = 1, reception continues, ignoring the unexpected frame-synchronization pulses.

When RFIG = 0, an unexpected FSR pulse causes the McBSP to discard the contents of RSR[1,2] in favor of the new incoming data. Therefore, if RFIG = 0 and an unexpected frame-synchronization pulse occurs, the serial port:

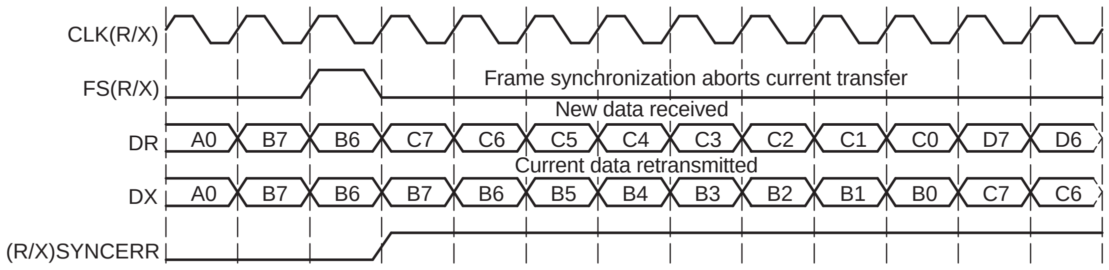
1. Aborts the current data transfer
2. Sets RSYNCERR in SPCR1 to 1
3. Begins the transfer of a new data word

For more details about the frame-synchronization error condition, see Section 15.5.3, *Unexpected Receive Frame-Synchronization Pulse*.

15.8.10.2 Examples of Effects of RFIG

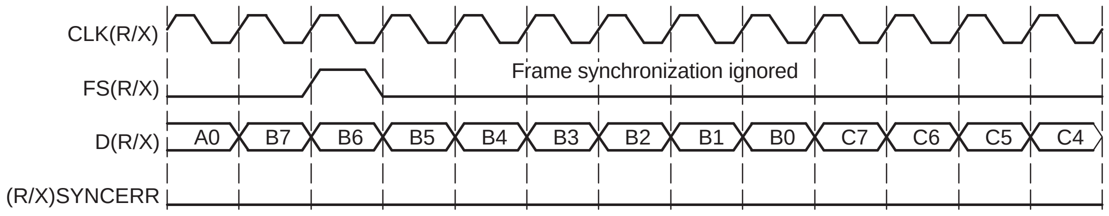
Figure 15-43 shows an example in which word B is interrupted by an unexpected frame-synchronization pulse when (R/X)FIG = 0. In the case of reception, the reception of B is aborted (B is lost), and a new data word © in this example) is received after the appropriate data delay. This condition is a receive synchronization error, which sets the RSYNCERR bit.

Figure 15-43. Unexpected Frame-Synchronization Pulse With (R/X)FIG = 0


CLK(R/X)
FS(R/X)
Frame synchronization aborts current transfer
New data received
DR
A0 B7 B6 C7 C6 C5 C4 C3 C2 C1 C0 D7 D6
Current data retransmitted
DX
A0 B7 B6 B7 B6 B5 B4 B3 B2 B1 B0 C7 C6
(R/X)SYNCERR

In contrast with Figure 15-43, Figure 15-44 shows McBSP operation when unexpected frame-synchronization signals are ignored (when (R/X)FIG = 1). Here, the transfer of word B is not affected by an unexpected pulse.

Figure 15-44. Unexpected Frame-Synchronization Pulse With (R/X)FIG = 1


CLK(R/X)
FS(R/X)
Frame synchronization ignored
D(R/X)
A0 B7 B6 B5 B4 B3 B2 B1 B0 C7 C6 C5 C4
(R/X)SYNCERR

15.8.11 Set the Receive Companding Mode

The RCOMPAND bits (see Table 15-30) determine whether companding or another data transfer option is chosen for McBSP reception.

Table 15-30. Register Bits Used to Set the Receive Companding Mode

Register	Bit	Name	Function		Type	Reset Value
RCR2	4-3	RCOMPAND	Receive companding mode		R/W	00
			Modes other than 00b are enabled only when the appropriate RWDLEN is 000b, indicating 8-bit data.			
			RCOMPAND = 00	No companding, any size data, MSB received first		
			RCOMPAND = 01	No companding, 8-bit data, LSB received first (for details, see Section 15.8.11.4).		
			RCOMPAND = 10	μ-law companding, 8-bit data, MSB received first		
			RCOMPAND = 11	A-law companding, 8-bit data, MSB received first		

15.8.11.1 Companding

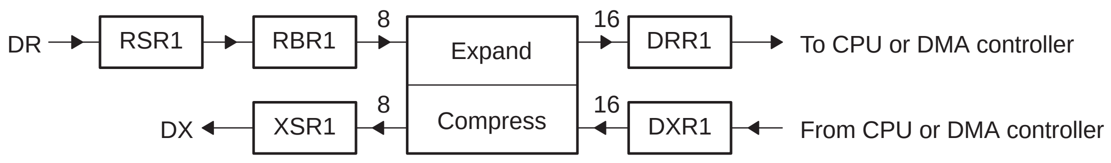
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either µ-law or A-law format. The companding standard employed in the United States and Japan is µ-law. The European companding standard is referred to as A-law. The specifications for µ-law and A-law log PCM are part of the CCITT G.711 recommendation.

A-law and µ-law allow 13 bits and 14 bits of dynamic range, respectively. Any values outside this range are set to the most positive or most negative value. Thus, for companding to work best, the data transferred to and from the McBSP via the CPU or DMA controller must be at least 16 bits wide.

The µ-law and A-law formats both encode data into 8-bit code words. Companded data is always 8 bits wide; the appropriate word length bits (RWDLEN1, RWDLEN2, XWDLEN1, XWDLEN2) must therefore be set to 0, indicating an 8-bit wide serial data stream. If companding is enabled and either of the frame phases does not have an 8-bit word length, companding continues as if the word length is 8 bits.

Figure 15-45 illustrates the companding processes. When companding is chosen for the transmitter, compression occurs during the process of copying data from DXR1 to XSR1. The transmit data is encoded according to the specified companding law (A-law or µ-law). When companding is chosen for the receiver, expansion occurs during the process of copying data from RBR1 to DRR1. The receive data is decoded to 2's-complement format.

Figure 15-45. Companding Processes for Reception and for Transmission



15.8.11.2 Format of Expanded Data

For reception, the 8-bit compressed data in RBR1 is expanded to left-justified 16-bit data in DRR1. The RJUST bit of SPCR1 is ignored when companding is used.

15.8.11.3 Companding Internal Data

If the McBSP is otherwise unused (the serial port transmit and receive sections are reset), the companding hardware can compand internal data. See Section 15.1.5.2, *Capability to Compand Internal Data*.

15.8.11.4 Option to Receive LSB First

Normally, the McBSP transmits or receives all data with the most significant bit (MSB) first. However, certain 8-bit data protocols (that do not use companded data) require the least significant bit (LSB) to be transferred first. If you set RCOMPAND = 01b in RCR2, the bit ordering of 8-bit words is reversed during reception. Similar to companding, this feature is enabled only if the appropriate word length bits are set to 0, indicating that 8-bit words are to be transferred serially. If either phase of the frame does not have an 8-bit word length, the McBSP assumes the word length is eight bits and LSB-first ordering is done.

15.8.12 Set the Receive Data Delay

The RDATDLY bits (see Table 15-31) determine the length of the data delay for the receive frame.

Table 15-31. Register Bits Used to Set the Receive Data Delay

Register	Bit	Name	Function		Type	Reset Value
RCR2	1-0	RDATDLY	Receive data delay		R/W	00
			RDATDLY = 00	0-bit data delay		
			RDATDLY = 01	1-bit data delay		
			RDATDLY = 10	2-bit data delay		

Table 15-31. Register Bits Used to Set the Receive Data Delay (continued)

Register	Bit	Name	Function		Type	Reset Value
			RDATDLY = 11	Reserved		

15.8.12.1 Data Delay

The start of a frame is defined by the first clock cycle in which frame synchronization is found to be active. The beginning of actual data reception or transmission with respect to the start of the frame can be delayed if required. This delay is called data delay.

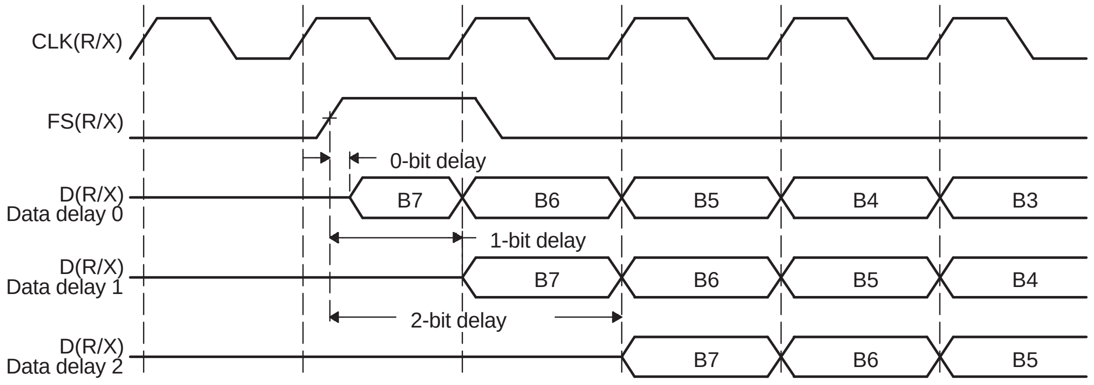
RDATDLY specifies the data delay for reception. The range of programmable data delay is zero to two bit-clocks (RDATDLY = 00b-10b), as described in Table 15-31 and shown in Figure 15-46. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on. Typically a 1-bit delay is selected, because data often follows a 1-cycle active frame-synchronization pulse.

15.8.12.2 0-Bit Data Delay

Normally, a frame-synchronization pulse is detected or sampled with respect to an edge of internal serial clock CLK(R/X). Thus, on the following cycle or later (depending on the data delay value), data may be received or transmitted. However, in the case of 0-bit data delay, the data must be ready for reception and/or transmission on the same serial clock cycle.

For reception, this problem is solved because receive data is sampled on the first falling edge of MCLKR where an active-high internal FSR is detected. However, data transmission must begin on the rising edge of the internal CLKX clock that generated the frame synchronization. Therefore, the first data bit is assumed to be present in XSR1, and thus on DX. The transmitter then asynchronously detects the frame-synchronization signal (FSX) going active high and immediately starts driving the first bit to be transmitted on the DX pin.

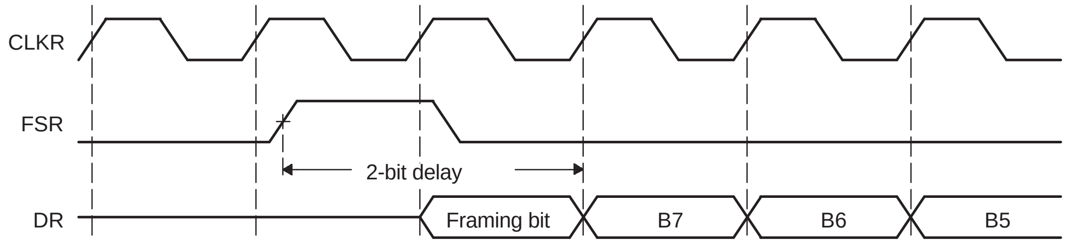
Figure 15-46. Range of Programmable Data Delay



15.8.12.3 2-Bit Data Delay

A data delay of two bit periods allows the serial port to interface to different types of T1 framing devices where the data stream is preceded by a framing bit. During reception of such a stream with data delay of two bits (framing bit appears after a 1-bit delay and data appears after a 2-bit delay), the serial port essentially discards the framing bit from the data stream, as shown in Figure 15-47. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on.

Figure 15-47. 2-Bit Data Delay Used to Skip a Framing Bit



15.8.13 Set the Receive Sign-Extension and Justification Mode

The RJUST bits (see Table 15-32) determine whether data received by the McBSP is sign-extended and how it is justified.

Table 15-32. Register Bits Used to Set the Receive Sign-Extension and Justification Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR1	14-13	RJUST	Receive sign-extension and justification mode		R/W	00
			RJUST = 00	Right justify data and zero fill MSBs in DRR[1,2]		
			RJUST = 01	Right justify data and sign extend it into the MSBs in DRR[1,2]		
			RJUST = 10	Left justify data and zero fill LSBs in DRR[1,2]		
			RJUST = 11	Reserved		

15.8.13.1 Sign-Extension and the Justification

RJUST in SPCR1 selects whether data in RBR[1,2] is right- or left-justified (with respect to the MSB) in DRR[1,2] and whether unused bits in DRR[1,2] are filled with zeros or with sign bits.

Table 15-33 and Table 15-34 show the effects of various RJUST values. The first table shows the effect on an example 12-bit receive-data value ABCh. The second table shows the effect on an example 20-bit receive-data value ABCDEh.

Table 15-33. Example: Use of RJUST Field With 12-Bit Data Value ABCh

RJUST	Justification	Extension	Value in DRR2	Value in DRR1
00b	Right	Zero fill MSBs	0000h	0ABCh
01b	Right	Sign extend data into MSBs	FFFFh	FABCh
10b	Left	Zero fill LSBs	0000h	ABC0h
11b	Reserved	Reserved	Reserved	Reserved

Table 15-34. Example: Use of RJUST Field With 20-Bit Data Value ABCDEh

RJUST	Justification	Extension	Value in DRR2	Value in DRR1
00b	Right	Zero fill MSBs	000Ah	BCDEh
01b	Right	Sign extend data into MSBs	FFFAh	BCDEh
10b	Left	Zero fill LSBs	ABCDh	E000h
11b	Reserved	Reserved	Reserved	Reserved

15.8.14 Set the Receive Interrupt Mode

The RINTM bits (see Table 15-35) determine which event generates a receive interrupt request to the CPU.

The receive interrupt (RINT) informs the CPU of changes to the serial port status. Four options exist for configuring this interrupt. The options are set by the receive interrupt mode bits, RINTM, in SPCR1.

Table 15-35. Register Bits Used to Set the Receive Interrupt Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR1	5-4	RINTM	Receive interrupt mode		R/W	00
			RINTM = 00	RINT generated when RRDY changes from 0 to 1. Interrupt on every serial word by tracking the RRDY bit in SPCR1. Regardless of the value of RINTM, RRDY can be read to detect the RRDY = 1 condition.		
			RINTM = 01	RINT generated by an end-of-block or end-of-frame condition in the receive multichannel selection mode. In the multichannel selection mode, interrupt after every 16-channel block boundary has been crossed within a frame and at the end of the frame. For details, see Section 15.6.7.3, *Using Interrupts Between Block Transfers*. In any other serial transfer case, this setting is not applicable and, therefore, no interrupts are generated.		
			RINTM = 10	RINT generated by a new receive frame-synchronization pulse. Interrupt on detection of receive frame-synchronization pulses. This generates an interrupt even when the receiver is in its reset state. This is done by synchronizing the incoming frame-synchronization pulse to the CPU clock and sending it to the CPU via RINT.		
			RINTM = 11	RINT generated when RSYNCERR is set. Interrupt on frame-synchronization error. Regardless of the value of RINTM, RSYNCERR can be read to detect this condition. For information on using RSYNCERR, see Section 15.5.3, *Unexpected Receive Frame-Synchronization Pulse*.		

15.8.15 Set the Receive Frame-Synchronization Mode

The bits described in Table 15-36 determine the source for receive frame synchronization and the function of the FSR pin.

15.8.15.1 Receive Frame-Synchronization Modes

Table 15-37 shows how you can select various sources to provide the receive frame-synchronization signal and the effect on the FSR pin. The polarity of the signal on the FSR pin is determined by the FSRP bit.

In digital loopback mode (DLB = 1), the transmit frame-synchronization signal is used as the receive frame-synchronization signal.

Also in the clock stop mode, the internal receive clock signal (MCLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.

Table 15-36. Register Bits Used to Set the Receive Frame Synchronization Mode

Register	Bit	Name	Function		Type	Reset Value
PCR	10	FSRM	Receive frame-synchronization mode		R/W	0
			FSRM = 0	Receive frame synchronization is supplied by an external source via the FSR pin.		
			FSRM = 1	Receive frame synchronization is supplied by the sample rate generator. FSR is an output pin reflecting internal FSR, except when GSYNC = 1 in SRGR2.		

Table 15-36. Register Bits Used to Set the Receive Frame Synchronization Mode (continued)

Register	Bit	Name	Function		Type	Reset Value
SRGR2	15	GSYNC	Sample rate generator clock synchronization mode		R/W	0
			If the sample rate generator creates a frame-synchronization signal (FSG) that is derived from an external input clock, the GSYNC bit determines whether FSG is kept synchronized with pulses on the FSR pin.			
			GSYNC = 0	No clock synchronization is used: CLKG oscillates without adjustment, and FSG pulses every (FPER + 1) CLKG cycles.		
			GSYNC = 1	Clock synchronization is used. When a pulse is detected on the FSR pin: • CLKG is adjusted as necessary so that it is synchronized with the input clock on the MCLKR pin. • FSG pulses FSG only pulses in response to a pulse on the FSR pin. The frame-synchronization period defined in FPER is ignored. For more details, see Section 15.4.3, *Synchronizing Sample Rate Generator Outputs to an External Clock*.		
SPCR1	15	DLB	Digital loopback mode		R/W	0
			DLB = 0	Digital loopback mode is disabled.		
			DLB = 1	Digital loopback mode is enabled. The receive signals, including the receive frame-synchronization signal, are connected internally through multiplexers to the corresponding transmit signals.		
SPCR1	12-11	CLKSTP	Clock stop mode		R/W	00
			CLKSTP = 0Xb	Clock stop mode disabled; normal clocking for non-SPI mode.		
			CLKSTP = 10b	Clock stop mode enabled without clock delay. The internal receive clock signal (MCLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.		
			CLKSTP = 11b	Clock stop mode enabled with clock delay. The internal receive clock signal (MCLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.		

Table 15-37. Select Sources to Provide the Receive Frame-Synchronization Signal and the Effect on the FSR Pin

DLB	FSRM	GSYNC	Source of Receive Frame Synchronization	FSR Pin Status
0	0	0 or 1	An external frame-synchronization signal enters the McBSP through the FSR pin. The signal is then inverted as determined by FSRP before being used as internal FSR.	Input
0	1	0	Internal FSR is driven by the sample rate generator frame-synchronization signal (FSG).	Output. FSG is inverted as determined by FSRP before being driven out on the FSR pin.
0	1	1	Internal FSR is driven by the sample rate generator frame-synchronization signal (FSG).	Input. The external frame-synchronization input on the FSR pin is used to synchronize CLKG and generate FSG pulses.
1	0	0	Internal FSX drives internal FSR.	High impedance

Table 15-37. Select Sources to Provide the Receive Frame-Synchronization Signal and the Effect on the FSR Pin (continued)

DLB	FSRM	GSYNC	Source of Receive Frame Synchronization	FSR Pin Status
1	0 or 1	1	Internal FSX drives internal FSR.	Input. If the sample rate generator is running, external FSR is used to synchronize CLKG and generate FSG pulses.
1	1	0	Internal FSX drives internal FSR.	Output. Receive (same as transmit) frame synchronization is inverted as determined by FSRP before being driven out on the FSR pin.

15.8.16 Set the Receive Frame-Synchronization Polarity

The FSRP bit (see Table 15-38) determines whether frame-synchronization pulses are active high or active low on the FSR pin.

Table 15-38. Register Bit Used to Set Receive Frame-Synchronization Polarity

Register	Bit	Name	Function		Type	Reset Value
PCR	2	FSRP	Receive frame-synchronization polarity		R/W	0
			FSRP = 0	Frame-synchronization pulse FSR is active high.		
			FSRP = 1	Frame-synchronization pulse FSR is active low.		

15.8.16.1 Frame-Synchronization Pulses, Clock Signals, and Their Polarities

Receive frame-synchronization pulses can be generated internally by the sample rate generator (see Section 15.4.2) or driven by an external source. The source of frame synchronization is selected by programming the mode bit, FSRM, in PCR. FSR is also affected by the GSYNC bit in SRGR2. For information about the effects of FSRM and GSYNC, see Section 15.8.15, *Set the Receive Frame-Synchronization Mode*. Similarly, receive clocks can be selected to be inputs or outputs by programming the mode bit, CLKRM, in the PCR (see Section 15.8.17, *Set the Receive Clock Mode*).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-synchronization pulses), the McBSP detects them on the internal falling edge of clock, internal MCLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal MCLKR. These internal clock signals are either derived from an external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of the internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

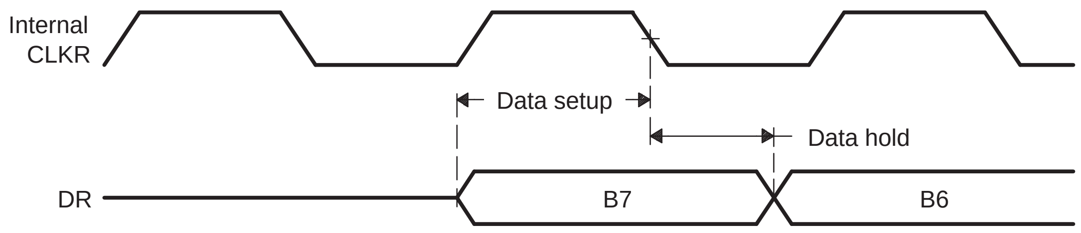
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, MCLKR, and CLKX signals, respectively. All frame-synchronization signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP), and FSRP = FSXP = 1, the external active-low frame-synchronization signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active-high frame-synchronization signals are inverted, if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin.

On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1 and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1, and internal clocking selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the MCLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. The receive data is always sampled on the falling edge of internal MCLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and MCLKR is an input pin), the external rising-edge triggered input clock on MCLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1 and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the MCLKR pin.

MCLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold of data around this edge. Figure 15-48 shows how data clocked by an external serial device using a rising edge can be sampled by the McBSP receiver on the falling edge of the same clock.

Figure 15-48. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



Set the SRG Frame-Synchronization Period and Pulse Width.

15.8.16.2 Frame-Synchronization Period and the Frame-Synchronization Pulse Width

The sample rate generator can produce a clock signal, CLKG, and a frame-synchronization signal, FSG. If the sample rate generator is supplying receive or transmit frame synchronization, you must program the bit fields FPER and FWID.

On FSG, the period from the start of a frame-synchronization pulse to the start of the next pulse is (FPER + 1) CLKG cycles. The 12 bits of FPER allow a frame-synchronization period of 1 to 4096 CLKG cycles, which allows up to 4096 data bits per frame. When GSYNC = 1, FPER is a don't care value.

Each pulse on FSG has a width of (FWID + 1) CLKG cycles. The eight bits of FWID allow a pulse width of 1 to 256 CLKG cycles. It is recommended that FWID be programmed to a value less than the programmed word length.

The values in FPER and FWID are loaded into separate down-counters. The 12-bit FPER counter counts down the generated clock cycles from the programmed value (4095 maximum) to 0. The 8-bit FWID counter counts down from the programmed value (255 maximum) to 0. Table 15-39 shows settings for FPER and FWID.

Figure 15-49 shows a frame-synchronization period of 16 CLKG periods (FPER = 15 or 00001111b) and a frame-synchronization pulse with an active width of 2 CLKG periods (FWID = 1).

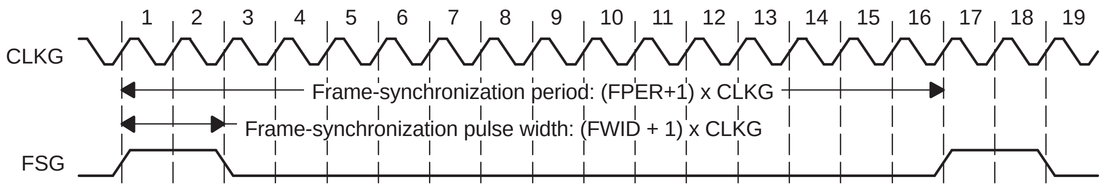
Table 15-39. Register Bits Used to Set the SRG Frame-Synchronization Period and Pulse Width

Register	Bit	Name	Function	Type	Reset Value
SRGR2	11-0	FPER	Sample rate generator frame-synchronization period For the frame-synchronization signal FSG, (FPER + 1) determines the period from the start of a frame-synchronization pulse to the start of the next frame-synchronization pulse. Range for (FPER + 1): 1 to 4096 CLKG cycles	R/W	0000 0000 0000
SRGR1	15-8	FWID	Sample rate generator frame-synchronization pulse width This field plus 1 determines the width of each frame-synchronization pulse on FSG.	R/W	0000 0000

Table 15-39. Register Bits Used to Set the SRG Frame-Synchronization Period and Pulse Width (continued)

Register	Bit	Name	Function	Type	Reset Value
			Range for (FWID + 1):		
			1 to 256 CLKG cycles		

Figure 15-49. Frame of Period 16 CLKG Periods and Active Width of 2 CLKG Periods


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
CLKG
Frame-synchronization period: (FPER+1) x CLKG
Frame-synchronization pulse width: (FWID + 1) x CLKG
FSG

When the sample rate generator comes out of reset, FSG is in its inactive state. Then, when GRST = 1 and FSGM = 1, a frame-synchronization pulse is generated. The frame width value (FWID + 1) is counted down on every CLKG cycle until it reaches 0, at which time FSG goes low. At the same time, the frame period value (FPER + 1) is also counting down. When this value reaches 0, FSG goes high, indicating a new frame.

15.8.17 Set the Receive Clock Mode

Table 15-40 shows the settings for bits used to set receive clock mode.

Table 15-40. Register Bits Used to Set the Receive Clock Mode

Register	Bit	Name	Function		Type	Reset Value
PCR	8	CLKRM	Receive clock mode		R/W	0
			Case 1: Digital loopback mode not set (DLB = 0) in SPCR1.			
			CLKRM = 0	The MCLKR pin is an input pin that supplies the internal receive clock (MCLKR).		
			CLKRM = 1	Internal MCLKR is driven by the sample rate generator of the McBSP. The MCLKR pin is an output pin that reflects internal MCLKR.		
			Case 2: Digital loopback mode set (DLB = 1) in SPCR1.			
			CLKRM = 0	The MCLKR pin is in the high impedance state. The internal receive clock (MCLKR) is driven by the internal transmit clock (CLKX). Internal CLKX is derived according to the CLKXM bit of PCR.		
			CLKRM = 1	Internal MCLKR is driven by internal CLKX. The MCLKR pin is an output pin that reflects internal MCLKR. Internal CLKX is derived according to the CLKXM bit of PCR.		
SPCR1	15	DLB	Digital loopback mode		R/W	00
			DLB = 0	Digital loopback mode is disabled.		
			DLB = 1	Digital loopback mode is enabled. The receive signals, including the receive frame-synchronization signal, are connected internally through multiplexers to the corresponding transmit signals.		

Table 15-40. Register Bits Used to Set the Receive Clock Mode (continued)

Register	Bit	Name	Function		Type	Reset Value
SPCR1	12-11	CLKSTP	Clock stop mode		R/W	00
			CLKSTP = 0Xb	Clock stop mode disabled; normal clocking for non-SPI mode.		
			CLKSTP = 10b	Clock stop mode enabled without clock delay. The internal receive clock signal (MCLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.		
			CLKSTP = 11b	Clock stop mode enabled with clock delay. The internal receive clock signal (MCLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.		

15.8.17.1 Selecting a Source for the Receive Clock and a Data Direction for the MCLKR Pin

Table 15-41 shows how you can select various sources to provide the receive clock signal and affect the MCLKR pin. The polarity of the signal on the MCLKR pin is determined by the CLKRP bit.

In the digital loopback mode (DLB = 1), the transmit clock signal is used as the receive clock signal.

Also, in the clock stop mode, the internal receive clock signal (MCLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.

Table 15-41. Receive Clock Signal Source Selection

DLB in SPCR1	CLKRM in PCR	Source of Receive Clock	MCLKR Pin Status
0	0	The MCLKR pin is an input driven by an external clock. The external clock signal is inverted as determined by CLKRP before being used.	Input
0	1	The sample rate generator clock (CLKG) drives internal MCLKR.	Output. CLKG, inverted as determined by CLKRP, is driven out on the MCLKR pin.
1	0	Internal CLKX drives internal MCLKR. To configure CLKX, see Section 15.9.18, *Set the Transmit Clock Mode*.	High impedance
1	1	Internal CLKX drives internal MCLKR. To configure CLKX, see Section 15.9.18, *Set the Transmit Clock Mode*.	Output. Internal MCLKR (same as internal CLKX) is inverted as determined by CLKRP before being driven out on the MCLKR pin.

15.8.18 Set the Receive Clock Polarity

Table 15-42. Register Bit Used to Set Receive Clock Polarity

Register	Bit	Name	Function		Type	Reset Value
PCR	0	CLKRP	Receive clock polarity		R/W	0
			CLKRP = 0	Receive data sampled on falling edge of MCLKR		
			CLKRP = 1	Receive data sampled on rising edge of MCLKR		

15.8.18.1 Frame Synchronization Pulses, Clock Signals, and Their Polarities

Receive frame-synchronization pulses can be generated internally by the sample rate generator (see Section 15.4.2) or driven by an external source. The source of frame synchronization is selected by programming the mode bit, FSRM, in PCR. FSR is also affected by the GSYNC bit in SRGR2. For information about the effects of FSRM and GSYNC, see Section 15.8.15, *Set the Receive Frame-Synchronization Mode*. Similarly, receive clocks can be selected to be inputs or outputs by programming the mode bit, CLKRM, in the PCR (see Section 15.8.17, *Set the Receive Clock Mode*).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-synchronization pulses), the McBSP detects them on the internal falling edge of clock, internal MCLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal MCLKR. These internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

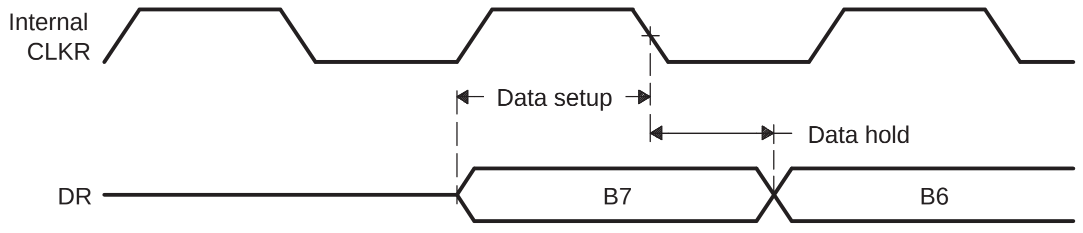
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, MCLKR, and CLKX signals, respectively. All frame-synchronization signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP) and FSRP = FSXP = 1, the external active-low frame-synchronization signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active-high frame-synchronization signals are inverted, if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin.

On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1 and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1 and internal clocking is selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the MCLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. The receive data is always sampled on the falling edge of internal MCLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and MCLKR is an input pin), the external rising-edge triggered input clock on MCLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1 and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the MCLKR pin.

CLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold of data around this edge. Figure 15-50 shows how data clocked by an external serial device using a rising edge can be sampled by the McBSP receiver on the falling edge of the same clock.

Figure 15-50. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



15.8.19 Set the SRG Clock Divide-Down Value

Table 15-43. Register Bits Used to Set the Sample Rate Generator (SRG) Clock Divide-Down Value

Register	Bit	Name	Function	Type	Reset Value
SRGR1	7-0	CLKGDV	Sample rate generator clock divide-down value The input clock of the sample rate generator is divided by (CLKGDV + 1) to generate the required sample rate generator clock frequency. The default value of CLKGDV is 1 (divide input clock by 2).	R/W	0000 0001

15.8.19.1 Sample Rate Generator Clock Divider

The first divider stage generates the serial data bit clock from the input clock. This divider stage utilizes a counter, preloaded by CLKGDV, that contains the divide ratio value.

The output of the first divider stage is the data bit clock, which is output as CLKG and which serves as the input for the second and third stages of the divider.

CLKG has a frequency equal to 1/(CLKGDV + 1) of sample rate generator input clock. Thus, the sample generator input clock frequency is divided by a value between 1 and 256. When CLKGDV is odd or equal to 0, the CLKG duty cycle is 50%. When CLKGDV is an even value, 2p, representing an odd divide-down, the high-state duration is p + 1 cycles and the low-state duration is p cycles.

15.8.20 Set the SRG Clock Synchronization Mode

For more details on using the clock synchronization feature, see Section 15.4.3, *Synchronizing Sample Rate Generator Outputs to an External Clock*.

Table 15-44. Register Bit Used to Set the SRG Clock Synchronization Mode

Register	Bit	Name	Function		Type	Reset Value
SRGR2	15	GSYNC	Sample rate generator clock synchronization GSYNC is used only when the input clock source for the sample rate generator is external—on the MCLKR or MCLKX pin.		R/W	0
			GSYNC = 0	The sample rate generator clock (CLKG) is free running. CLKG oscillates without adjustment, and FSG pulses every (FPER + 1) CLKG cycles.		
			GSYNC = 1	Clock synchronization is performed. When a pulse is detected on the FSR pin: • CLKG is adjusted as necessary so that it is synchronized with the input clock on the MCLKR or MCLKX pin. • FSG pulses. FSG only pulses in response to a pulse on the FSR pin. The frame-synchronization period defined in FPER is ignored.		

15.8.21 Set the SRG Clock Mode (Choose an Input Clock)

Table 15-45. Register Bits Used to Set the SRG Clock Mode (Choose an Input Clock)

Register	Bit	Name	Function		Type	Reset Value
PCR	7	SCLKME	Sample rate generator clock mode		R/W	0
SRGR2	13	CLKSM			R/W	1
			SCLKME = 0 CLKSM = 0	Reserved		

Table 15-45. Register Bits Used to Set the SRG Clock Mode (Choose an Input Clock) (continued)

Register	Bit	Name	Function		Type	Reset Value
			SCLKME = 0	Sample rate generator clock derived from LSPCLK (default)		
			CLKSM = 1			
			SCLKME = 1	Sample rate generator clock derived from MCLKR pin		
			CLKSM = 0			
			SCLKME = 1	Sample rate generator clock derived from MCLKX pin		
			CLKSM = 1			

15.8.21.1 SRG Clock Mode

The sample rate generator can produce a clock signal (CLKG) for use by the receiver, the transmitter, or both, but CLKG is derived from an input clock. Table 15-45 shows the four possible sources of the input clock. For more details on generating CLKG, see Section 15.4.1.1, *Clock Generation in the Sample Rate Generator*.

15.8.22 Set the SRG Input Clock Polarity

Table 15-46. Register Bits Used to Set the SRG Input Clock Polarity

Register	Bit	Name	Function		Type	Reset Value
PCR	1	CLKXP	MCLKX pin polarity		R/W	0
			CLKXP determines the input clock polarity when the MCLKX pin supplies the input clock (SCLKME = 1 and CLKSM = 1).			
			CLKXP = 0	Rising edge on MCLKX pin generates transitions on CLKG and FSG.		
			CLKXP = 1	Falling edge on MCLKX pin generates transitions on CLKG and FSG.		
PCR	0	CLKRP	MCLKR pin polarity		R/W	0
			CLKRP determines the input clock polarity when the MCLKR pin supplies the input clock (SCLKME = 1 and CLKSM = 0).			
			CLKRP = 0	Falling edge on MCLKR pin generates transitions on CLKG and FSG.		
			CLKRP = 1	Rising edge on MCLKR pin generates transitions on CLKG and FSG.		

15.8.22.1 Using CLKXP/CLKRP to Choose an Input Clock Polarity

The sample rate generator can produce a clock signal (CLKG) and a frame-synchronization signal (FSG) for use by the receiver, the transmitter, or both. To produce CLKG and FSG, the sample rate generator must be driven by an input clock signal derived from the CPU clock or from an external clock on the CLKX or MCLKR pin. If you use a pin, choose a polarity for that pin by using the appropriate polarity bit (CLKXP for the MCLKX pin, CLKRP for the MCLKR pin). The polarity determines whether the rising or falling edge of the input clock generates transitions on CLKG and FSG.

15.9 Transmitter Configuration

To configure the McBSP transmitter, perform the following procedure:

1. Place the McBSP/transmitter in reset (see Section 15.9.2).
2. Program the McBSP registers for the desired transmitter operation (see Section 15.9.1).
3. Take the transmitter out of reset (see Section 15.9.2).

15.9.1 Programming the McBSP Registers for the Desired Transmitter Operation

The following is a list of important tasks to be performed when you are configuring the McBSP transmitter. Each task corresponds to one or more McBSP register bit fields.

- Global behavior:
  – Set the transmitter pins to operate as McBSP pins.
  – Enable/disable the digital loopback mode.
  – Enable/disable the clock stop mode.
  – Enable/disable transmit multichannel selection.
- Data behavior:
  – Choose 1 or 2 phases for the transmit frame.
  – Set the transmit word length(s).
  – Set the transmit frame length.
  – Enable/disable the transmit frame-synchronization ignore function.
  – Set the transmit companding mode.
  – Set the transmit data delay.

- Set the transmit DXENA mode.
- Set the transmit interrupt mode.

- Frame-synchronization behavior:
  - Set the transmit frame-synchronization mode.
  - Set the transmit frame-synchronization polarity.
  - Set the SRG frame-synchronization period and pulse width.
- Clock behavior:
  - Set the transmit clock mode.
  - Set the transmit clock polarity.
  - Set the SRG clock divide-down value.
  - Set the SRG clock synchronization mode.
  - Set the SRG clock mode (choose an input clock).
  - Set the SRG input clock polarity.

15.9.2 Resetting and Enabling the Transmitter

The first step of the transmitter configuration procedure is to reset the transmitter, and the last step is to enable the transmitter (to take it out of reset). Table 15-47 describes the bits used for both of these steps.

Table 15-47. Register Bits Used to Place Transmitter in Reset Field Descriptions

Register	Bit	Field	Value	Description
SPCR2	7	FRST		Frame-synchronization logic reset
			0	Frame-synchronization logic is reset. The sample rate generator does not generate frame-synchronization signal FSG, even if GRST = 1.
			1	Frame-synchronization is enabled. If GRST = 1, frame-synchronization signal FSG is generated after (FPER + 1) number of CLKG clock cycles; all frame counters are loaded with their programmed values.
SPCR2	6	GRST		Sample rate generator reset
			0	Sample rate generator is reset. If GRST = 0 due to a device reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven low (inactive). If GRST = 0 due to program code, CLKG and FSG are both driven low (inactive).
			1	Sample rate generator is enabled. CLKG is driven according to the configuration programmed in the sample rate generator registers (SRGR[1,2]). If FRST = 1, the generator also generates the frame-synchronization signal FSG as programmed in the sample rate generator registers.
SPCR2	0	XRST		Transmitter reset
			0	The serial port transmitter is disabled and in the reset state.
			1	The serial port transmitter is enabled.

15.9.2.1 Reset Considerations

The serial port can be reset in the following two ways:

1. A DSP reset ($\overline{\text{XRS}}$ signal driven low) places the receiver, transmitter, and sample rate generator in reset. When the device reset is removed, GRST = FRST = RRST = XRST = 0, keeping the entire serial port in the reset state.
2. The serial port transmitter and receiver can be reset directly using the RRST and XRST bits in the serial port control registers. The sample rate generator can be reset directly using the GRST bit in SPCR2.
3. When using the DMA, the order in which McBSP events must occur is important. DMA channel and peripheral interrupts must be configured prior to releasing the McBSP transmitter from reset.

   The reason for this is that an XRDY is fired when XRST = 1. The XRDY signals the DMA to start copying data from the buffer into the transmit register. If the McBSP transmitter is released from reset before the DMA channel and peripheral interrupts are configured, the XRDY signals before the DMA channel can receive the signal; therefore, the DMA does not move the data from the buffer to the transmit register. The DMA PERINTFLG is edge-sensitive and will fail to recognize the XRDY, which is continuously high.

For more details about McBSP reset conditions and effects, see Section 15.10.2, *Resetting and Initializing a McBSP*.

15.9.3 Set the Transmitter Pins to Operate as McBSP Pins

To configure a pin for its McBSP function , you should configure the bits of the GPxMUXn register appropriately. In addition to this, bits 12 and 13 of the PCR register must be set to 0. These bits are defined as reserved.

15.9.4 Enable/Disable the Digital Loopback Mode

The DLB bit determines whether the digital loopback mode is on. DLB is described in Table 15-48.

Table 15-48. Register Bit Used to Enable/Disable the Digital Loopback Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR1	15	DLB	Digital loopback mode		R/W	0
			DLB = 0	Digital loopback mode is disabled.		
			DLB = 1	Digital loopback mode is enabled.		

15.9.4.1 Digital Loopback Mode

In the digital loopback mode, the receive signals are connected internally through multiplexers to the corresponding transmit signals, as shown in Table 15-49. This mode allows testing of serial port code with a single DSP device; the McBSP receives the data it transmits.

Table 15-49. Receive Signals Connected to Transmit Signals in Digital Loopback Mode

This Receive Signal	Is Fed Internally by This Transmit Signal
DR (receive data)	DX (transmit data)
FSR (receive frame synchronization)	FSX (transmit frame synchronization)
MCLKR (receive clock)	CLKX (transmit clock)

15.9.5 Enable/Disable the Clock Stop Mode

The CLKSTP bits determine whether the clock stop mode is on. CLKSTP is described in Table 15-50.

Table 15-50. Register Bits Used to Enable/Disable the Clock Stop Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR1	12-11	CLKSTP	Clock stop mode		R/W	00
			CLKSTP = 0Xb	Clock stop mode disabled; normal clocking for non-SPI mode.		
			CLKSTP = 10b	Clock stop mode enabled without clock delay		
			CLKSTP = 11b	Clock stop mode enabled with clock delay		

15.9.5.1 Clock Stop Mode

The clock stop mode supports the SPI master-slave protocol. If you do not plan to use the SPI protocol, you can clear CLKSTP to disable the clock stop mode.

In the clock stop mode, the clock stops at the end of each data transfer. At the beginning of each data transfer, the clock starts immediately (CLKSTP = 10b) or after a half-cycle delay (CLKSTP = 11b). The CLKXP bit determines whether the starting edge of the clock on the MCLKX pin is rising or falling. The CLKRP bit determines whether receive data is sampled on the rising or falling edge of the clock shown on the MCLKR pin.

Table 15-51 summarizes the impact of CLKSTP, CLKXP, and CLKRP on serial port operation. In the clock stop mode, the receive clock is tied internally to the transmit clock, and the receive frame-synchronization signal is tied internally to the transmit frame-synchronization signal.

Table 15-51. Effects of CLKSTP, CLKXP, and CLKRP on the Clock Scheme

Bit Settings	Clock Scheme

CLKSTP = 00b or 01b
CLKXP = 0 or 1
CLKRP = 0 or 1
	Clock stop mode disabled. Clock enabled for non-SPI mode.

CLKSTP = 10b
CLKXP = 0
CLKRP = 0
	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of MCLKR.

CLKSTP = 11b
CLKXP = 0
CLKRP = 1
	Low inactive state with delay: The McBSP transmits data one-half cycle ahead of the rising edge of CLKX and receives data on the rising edge of MCLKR.

CLKSTP = 10b
CLKXP = 1
CLKRP = 0
	High inactive state without delay: The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of MCLKR.

CLKSTP = 11b
CLKXP = 1
CLKRP = 1
	High inactive state with delay: The McBSP transmits data one-half cycle ahead of the falling edge of CLKX and receives data on the falling edge of MCLKR.

15.9.6 Enable/Disable Transmit Multichannel Selection

For more details, see Section 15.6.7, *Transmit Multichannel Selection Modes*.

Table 15-52. Register Bits Used to Enable/Disable Transmit Multichannel Selection

Register	Bit	Name	Function		Type	Reset Value
MCR2	1-0	XMCM	Transmit multichannel selection		R/W	00
			XMCM = 00b	No transmit multichannel selection mode is on. All channels are enabled and unmasked. No channels can be disabled or masked.		
			XMCM = 01b	All channels are disabled unless they are selected in the appropriate transmit channel enable registers (XCERs). If enabled, a channel in this mode is also unmasked.		
				The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.		
			XMCM = 10b	All channels are enabled, but they are masked unless they are selected in the appropriate transmit channel enable registers (XCERs).		
				The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.		
			XMCM = 11b	This mode is used for symmetric transmission and reception.		
				All channels are disabled for transmission unless they are enabled for reception in the appropriate receive channel enable registers (RCERs). Once enabled, they are masked unless they are also selected in the appropriate transmit channel enable registers (XCERs).		
				The XMCME bit determines whether 32 channels or 128 channels are selectable in RCERs and XCERs.		

15.9.7 Choose One or Two Phases for the Transmit Frame

Table 15-53. Register Bit Used to Choose 1 or 2 Phases for the Transmit Frame

Register	Bit	Name	Function		Type	Reset Value
XCR2	15	XPHASE	Transmit phase number		R/W	0
			Specifies whether the transmit frame has 1 or 2 phases.			
			XPHASE = 0	Single-phase frame		
			XPHASE = 1	Dual-phase frame		

15.9.8 Set the Transmit Word Length(s)

Table 15-54. Register Bits Used to Set the Transmit Word Length(s)

Register	Bit	Name	Function		Type	Reset Value
XCR1	7-5	XWDLEN1	Transmit word length of frame phase 1		R/W	000
			XWDLEN1 = 000b	8 bits		
			XWDLEN1 = 001b	12 bits		
			XWDLEN1 = 010b	16 bits		
			XWDLEN1 = 011b	20 bits		
			XWDLEN1 = 100b	24 bits		
			XWDLEN1 = 101b	32 bits		
			XWDLEN1 = 11Xb	Reserved		
XCR2	7-5	XWDLEN2	Transmit word length of frame phase 2		R/W	000
			XWDLEN2 = 000b	8 bits		
			XWDLEN2 = 001b	12 bits		
			XWDLEN2 = 010b	16 bits		
			XWDLEN2 = 011b	20 bits		
			XWDLEN2 = 100b	24 bits		
			XWDLEN2 = 101b	32 bits		
			XWDLEN2 = 11Xb	Reserved		

15.9.8.1 Word Length Bits

Each frame can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, XWDLEN1 selects the length for every serial word transmitted in the frame. If a dual-phase frame is selected, XWDLEN1 determines the length of the serial words in phase 1 of the frame, and XWDLEN2 determines the word length in phase 2 of the frame.

15.9.9 Set the Transmit Frame Length

Table 15-55. Register Bits Used to Set the Transmit Frame Length

Register	Bit	Name	Function		Type	Reset Value
XCR1	14-8	XFRLEN1	Transmit frame length 1		R/W	000 0000
			(XFRLEN1 + 1) is the number of serial words in phase 1 of the transmit frame.			
			XFRLEN1 = 000 0000	1 word in phase 1		
			XFRLEN1 = 000 0001	2 words in phase 1		
			|	|		
			|	|		
			XFRLEN1 = 111 1111	128 words in phase 1		
XCR2	14-8	XFRLEN2	Transmit frame length 2		R/W	000 0000
			If a dual-phase frame is selected, (XFRLEN2 + 1) is the number of serial words in phase 2 of the transmit frame.			
			XFRLEN2 = 000 0000	1 word in phase 2		
			XFRLEN2 = 000 0001	2 words in phase 2		
			|	|		
			|	|		
			XFRLEN2 = 111 1111	128 words in phase 2		

15.9.9.1 Selected Frame Length

The transmit frame length is the number of serial words in the transmit frame. Each frame can have one or two phases, depending on the value that you load into the XPHASE bit.

If a single-phase frame is selected (XPHASE = 0), the frame length is equal to the length of phase 1. If a dual-phase frame is selected (XPHASE = 1), the frame length is the length of phase 1 plus the length of phase 2.

The 7-bit XFRLEN fields allow up to 128 words per phase. See Table 15-56 for a summary of how to calculate the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization pulse.

NOTE: Program the XFRLEN fields with [*w minus 1*], where w represents the number of words per phase. For example, if you want a phase length of 128 words in phase 1, load 127 into XFRLEN1.

Table 15-56. How to Calculate Frame Length

XPHASE	XFRLEN1	XFRLEN2	Frame Length
0	0 ≤ XFRLEN1 ≤ 127	Don't care	(XFRLEN1 + 1) words
1	0 ≤ XFRLEN1 ≤ 127	0 ≤ XFRLEN2 ≤ 127	(XFRLEN1 + 1) + (XFRLEN2 + 1) words

15.9.10 Enable/Disable the Transmit Frame-Synchronization Ignore Function

Table 15-57. Register Bit Used to Enable/Disable the Transmit Frame-Synchronization Ignore Function

Register	Bit	Name	Function		Type	Reset Value
XCR2	2	XFIG	Transmit frame-synchronization ignore		R/W	0
			XFIG = 0	An unexpected transmit frame-synchronization pulse causes the McBSP to restart the frame transfer.		
			XFIG = 1	The McBSP ignores unexpected transmit frame-synchronization pulses.		

15.9.10.1 Unexpected Frame-Synchronization Pulses and Frame-Synchronization Ignore

If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully transmitted, this pulse is treated as an unexpected frame-synchronization pulse.

When XFIG = 1, normal transmission continues with unexpected frame-synchronization signals ignored.

When XFIG = 0 and an unexpected frame-synchronization pulse occurs, the serial port:

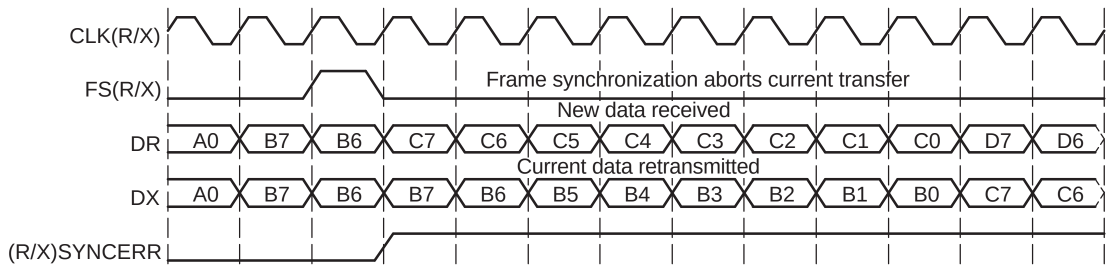
1. Aborts the present transmission
2. Sets XSYNCERR to 1 in SPCR2
3. Reinitiates transmission of the current word that was aborted

For more details about the frame-synchronization error condition, see Section 15.5.5, *Unexpected Transmit Frame-Synchronization Pulse*.

15.9.10.2 Examples Showing the Effects of XFIG

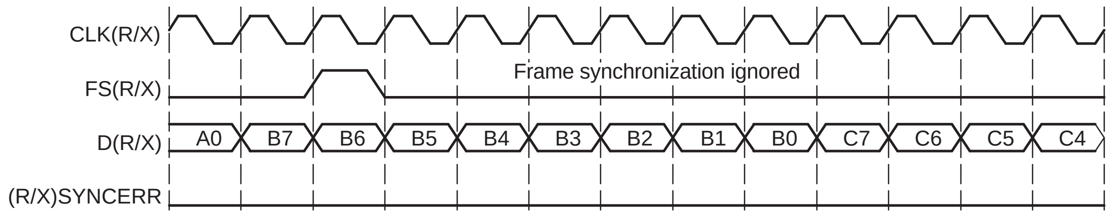
Figure 15-51 shows an example in which word B is interrupted by an unexpected frame-synchronization pulse when (R/X)FIG = 0. In the case of transmission, the transmission of B is aborted (B is lost). This condition is a transmit synchronization error, which sets the XSYNCERR bit. No new data has been written to DXR[1,2]; therefore, the McBSP transmits B again.

Figure 15-51. Unexpected Frame-Synchronization Pulse With (R/X) FIG = 0


CLK(R/X)
FS(R/X)
Frame synchronization aborts current transfer
New data received
DR
A0 B7 B6 C7 C6 C5 C4 C3 C2 C1 C0 D7 D6
Current data retransmitted
DX
A0 B7 B6 B7 B6 B5 B4 B3 B2 B1 B0 C7 C6
(R/X)SYNCERR

In contrast with Figure 15-51, Figure 15-52 shows McBSP operation when unexpected frame-synchronization signals are ignored (when (R/X)FIG = 1). Here, the transfer of word B is not affected by an unexpected frame-synchronization pulse.

Figure 15-52. Unexpected Frame-Synchronization Pulse With (R/X) FIG = 1


CLK(R/X)
FS(R/X)
Frame synchronization ignored
D(R/X)
A0 B7 B6 B5 B4 B3 B2 B1 B0 C7 C6 C5 C4
(R/X)SYNCERR

15.9.11 Set the Transmit Companding Mode

Table 15-58. Register Bits Used to Set the Transmit Companding Mode

Register	Bit	Name	Function		Type	Reset Value
XCR2	4-3	XCOMPAND	Transmit companding mode		R/W	00
			Modes other than 00b are enabled only when the appropriate XWDLEN is 000b, indicating 8-bit data.			
			XCOMPAND = 00b	No companding, any size data, MSB transmitted first		
			XCOMPAND = 01b	No companding, 8-bit data, LSB transmitted first (for details, see Section 15.8.11.4, *Option to Receive LSB First*)		
			XCOMPAND = 10b	μ-law companding, 8-bit data, MSB transmitted first		
			XCOMPAND = 11b	A-law companding, 8-bit data, MSB transmitted first		

15.9.11.1 Companding

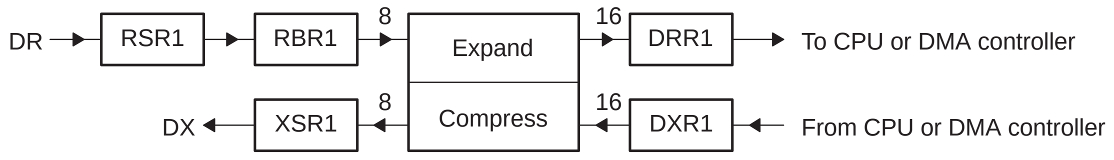
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM are part of the CCITT G.711 recommendation.

A-law and μ-law allow 13 bits and 14 bits of dynamic range, respectively. Any values outside this range are set to the most positive or most negative value. Thus, for companding to work best, the data transferred to and from the McBSP via the CPU or DMA controller must be at least 16 bits wide.

The μ-law and A-law formats both encode data into 8-bit code words. Companded data is always 8 bits wide; the appropriate word length bits (RWDLEN1, RWDLEN2, XWDLEN1, XWDLEN2) must therefore be set to 0, indicating an 8-bit wide serial data stream. If companding is enabled and either of the frame phases does not have an 8-bit word length, companding continues as if the word length is 8 bits.

Figure 15-53 illustrates the companding processes. When companding is chosen for the transmitter, compression occurs during the process of copying data from DXR1 to XSR1. The transmit data is encoded according to the specified companding law (A-law or μ-law). When companding is chosen for the receiver, expansion occurs during the process of copying data from RBR1 to DRR1. The receive data is decoded to twos-complement format.

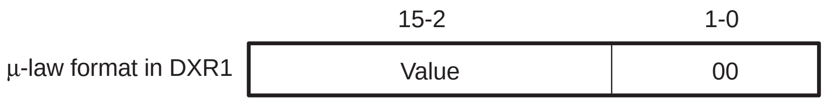
Figure 15-53. Companding Processes for Reception and for Transmission



15.9.11.2 Format for Data To Be Compressed

For transmission using μ-law compression, make sure the 14 data bits are left-justified in DXR1, with the remaining two low-order bits filled with 0s as shown in Figure 15-54.

Figure 15-54. μ-Law Transmit Data Companding Format



For transmission using A-law compression, make sure the 13 data bits are left-justified in DXR1, with the remaining three low-order bits filled with 0s as shown in Figure 15-55.

Figure 15-55. A-Law Transmit Data Companding Format

	15-3	2-0
A-law format in DXR1	Value	000

15.9.11.3 Capability to Compand Internal Data

If the McBSP is otherwise unused (the serial port transmit and receive sections are reset), the companding hardware can compand internal data. See Section 15.1.5.2, *Capability to Compand Internal Data*.

15.9.11.4 Option to Transmit LSB First

Normally, the McBSP transmit or receives all data with the most significant bit (MSB) first. However, certain 8-bit data protocols (that do not use companded data) require the least significant bit (LSB) to be transferred first. If you set XCOMPAND = 01b in XCR2, the bit ordering of 8-bit words is reversed (LSB first) before being sent from the serial port. Similar to companding, this feature is enabled only if the appropriate word length bits are set to 0, indicating that 8-bit words are to be transferred serially. If either phase of the frame does not have an 8-bit word length, the McBSP assumes the word length is eight bits and LSB-first ordering is done.

15.9.12 Set the Transmit Data Delay

Table 15-59. Register Bits Used to Set the Transmit Data Delay

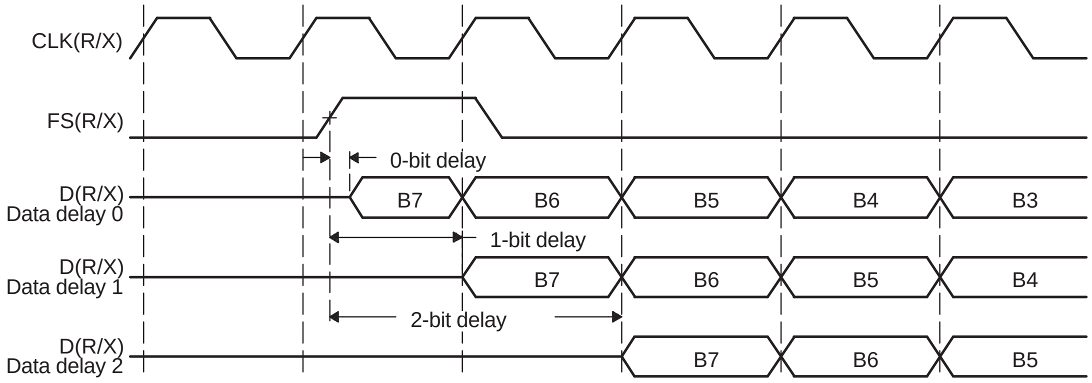
Register	Bit	Name	Function		Type	Reset Value
XCR2	1-0	XDATDLY	Transmitter data delay		R/W	00
			XDATDLY = 00	0-bit data delay		
			XDATDLY = 01	1-bit data delay		
			XDATDLY = 10	2-bit data delay		
			XDATDLY = 11	Reserved		

15.9.12.1 Data Delay

The start of a frame is defined by the first clock cycle in which frame synchronization is found to be active. The beginning of actual data reception or transmission with respect to the start of the frame can be delayed if necessary. This delay is called data delay.

XDATDLY specifies the data delay for transmission. The range of programmable data delay is zero to two bit-clocks (XDATDLY = 00b-10b), as described in Table 15-59 and Figure 15-56. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on. Typically a 1-bit delay is selected, because data often follows a 1-cycle active frame-synchronization pulse.

Figure 15-56. Range of Programmable Data Delay



15.9.12.2 0-Bit Data Delay

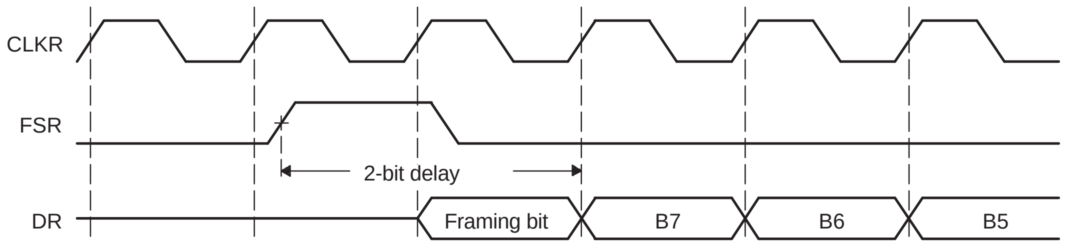
Normally, a frame-synchronization pulse is detected or sampled with respect to an edge of serial clock internal CLK(R/X). Thus, on the following cycle or later (depending on the data delay value), data can be received or transmitted. However, in the case of 0-bit data delay, the data must be ready for reception and/or transmission on the same serial clock cycle.

For reception this problem is solved because receive data is sampled on the first falling edge of MCLKR where an active-high internal FSR is detected. However, data transmission must begin on the rising edge of the internal CLKX clock that generated the frame synchronization. Therefore, the first data bit is assumed to be present in XSR1, and thus DX. The transmitter then asynchronously detects the frame synchronization, FSX, going active high and immediately starts driving the first bit to be transmitted on the DX pin.

15.9.12.3 2-Bit Data Delay

A data delay of two bit-periods allows the serial port to interface to different types of T1 framing devices where the data stream is preceded by a framing bit. During reception of such a stream with data delay of two bits (framing bit appears after a 1-bit delay and data appears after a 2-bit delay), the serial port essentially discards the framing bit from the data stream, as shown in the following figure. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on.

Figure 15-57. 2-Bit Data Delay Used to Skip a Framing Bit



15.9.13 Set the Transmit DXENA Mode

Table 15-60. Register Bit Used to Set the Transmit DXENA (DX Delay Enabler) Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR1	7	DXENA	DX delay enabler mode		R/W	0
			DXENA = 0	DX delay enabler is off.		
			DXENA = 1	DX delay enabler is on.		

15.9.13.1 DXENA Mode

The DXENA bit controls the delay enabler on the DX pin. Set DXENA to enable an extra delay for turn-on time. This bit does not control the data itself, so only the first bit is delayed.

If you tie together the DX pins of multiple McBSPs, make sure DXENA = 1 to avoid having more than one McBSP transmit on the data line at one time.

15.9.14 Set the Transmit Interrupt Mode

The transmitter interrupt (XINT) signals the CPU of changes to the serial port status. Four options exist for configuring this interrupt. The options are set by the transmit interrupt mode bits, XINTM, in SPCR2.

Table 15-61. Register Bits Used to Set the Transmit Interrupt Mode

Register	Bit	Name	Function		Type	Reset Value
SPCR2	5-4	XINTM	Transmit interrupt mode		R/W	00
			XINTM = 00	XINT generated when XRDY changes from 0 to 1.		
			XINTM = 01	XINT generated by an end-of-block or end-of-frame condition in a transmit multichannel selection mode. In any of the transmit multichannel selection modes, interrupt after every 16-channel block boundary has been crossed within a frame and at the end of the frame. For details, see Section 15.6.7.3, *Using Interrupts Between Block Transfers*. In any other serial transfer case, this setting is not applicable and, therefore, no interrupts are generated.		
			XINTM = 10	XINT generated by a new transmit frame-synchronization pulse. Interrupt on detection of each transmit frame-synchronization pulse. This generates an interrupt even when the transmitter is in its reset state. This is done by synchronizing the incoming frame-synchronization pulse to the CPU clock and sending it to the CPU via XINT.		
			XINTM = 11	XINT generated when XSYNCERR is set. Interrupt on frame-synchronization error. Regardless of the value of XINTM, XSYNCERR can be read to detect this condition. For more information on using XSYNCERR, see Section 15.5.5, *Unexpected Transmit Frame-Synchronization Pulse*.		

15.9.15 Set the Transmit Frame-Synchronization Mode

Table 15-62. Register Bits Used to Set the Transmit Frame-Synchronization Mode

Register	Bit	Name	Function		Type	Reset Value
PCR	11	FSXM	Transmit frame-synchronization mode		R/W	0
			FSXM = 0	Transmit frame synchronization is supplied by an external source via the FSX pin.		
			FSXM = 1	Transmit frame synchronization is supplied by the McBSP, as determined by the FSGM bit of SRGR2.		
SRGR2	12	FSGM	Sample rate generator transmit frame-synchronization mode		R/W	0
			Used when FSXM = 1 in PCR.			
			FSGM = 0	The McBSP generates a transmit frame-synchronization pulse when the content of DXR[1,2] is copied to XSR[1,2].		
			FSGM = 1	The transmitter uses frame-synchronization pulses generated by the sample rate generator. Program the FWID bits to set the width of each pulse. Program the FPER bits to set the frame-synchronization period.		

15.9.15.1 Transmit Frame-Synchronization Modes

Table 15-63 shows how FSXM and FSGM select the source of transmit frame-synchronization pulses. The three choices are:

- External frame-synchronization input
- Sample rate generator frame-synchronization signal (FSG)
- Internal signal that indicates a DXR-to-XSR copy has been made

Table 15-63 also shows the effect of each bit setting on the FSX pin. The polarity of the signal on the FSX pin is determined by the FSXP bit.

Table 15-63. How FSXM and FSGM Select the Source of Transmit Frame-Synchronization Pulses

FSXM	FSGM	Source of Transmit Frame Synchronization	FSX Pin Status
0	0 or 1	An external frame-synchronization signal enters the McBSP through the FSX pin. The signal is then inverted by FSXP before being used as internal FSX.	Input
1	1	Internal FSX is driven by the sample rate generator frame-synchronization signal (FSG).	Output. FSG is inverted by FSXP before being driven out on FSX pin.
1	0	A DXR-to-XSR copy causes the McBSP to generate a transmit frame-synchronization pulse that is 1 cycle wide.	Output. The generated frame-synchronization pulse is inverted as determined by FSXP before being driven out on FSX pin.

15.9.15.2 Other Considerations

If the sample rate generator creates a frame-synchronization signal (FSG) that is derived from an external input clock, the GSYNC bit determines whether FSG is kept synchronized with pulses on the FSR pin. For more details, see Section 15.4.3, *Synchronizing Sample Rate Generator Outputs to an External Clock*.

In the clock stop mode (CLKSTP = 10b or 11b), the McBSP can act as a master or as a slave in the SPI protocol. If the McBSP is a master and must provide a slave-enable signal ($\overline{\text{SPISTE}}$) on the FSX pin, make sure that FSXM = 1 and FSGM = 0 so that FSX is an output and is driven active for the duration of each transmission. If the McBSP is a slave, make sure that FSXM = 0 so that the McBSP can receive the slave-enable signal on the FSX pin.

15.9.16 Set the Transmit Frame-Synchronization Polarity

Table 15-64. Register Bit Used to Set Transmit Frame-Synchronization Polarity

Register	Bit	Name	Function		Type	Reset Value
PCR	3	FSXP	Transmit frame-synchronization polarity		R/W	0
			FSXP = 0	Frame-synchronization pulse FSX is active high.		
			FSXP = 1	Frame-synchronization pulse FSX is active low.		

15.9.16.1 Frame Synchronization Pulses, Clock Signals, and Their Polarities

Transmit frame-synchronization pulses can be generated internally by the sample rate generator (see Section 15.4.2) or driven by an external source. The source of frame synchronization is selected by programming the mode bit, FSXM, in PCR. FSX is also affected by the FSGM bit in SRGR2. For information about the effects of FSXM and FSGM, see Section 15.9.15, *Set the Transmit Frame-Synchronization Mode*). Similarly, transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLKXM, in the PCR (see Section 15.9.18, *Set the Transmit Clock Mode*).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-synchronization pulses), the McBSP detects them on the internal falling edge of clock, internal MCLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal MCLKR. These internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

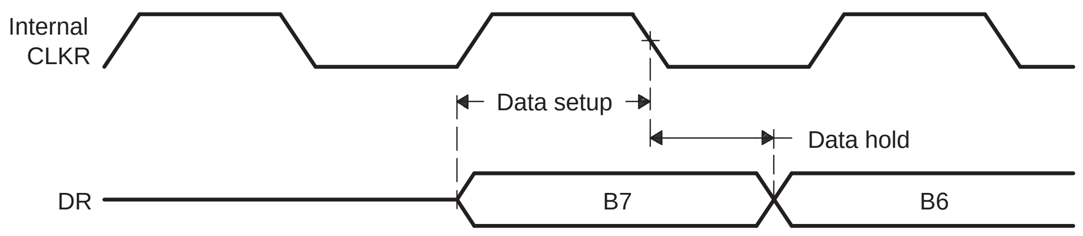
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, MCLKR, and CLKX signals, respectively. All frame-synchronization signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP) and FSRP = FSXP = 1, the external active-low frame-synchronization signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected and the polarity bit FS(R/X)P = 1, the internal active-high frame-synchronization signals are inverted before being sent to the FS(R/X) pin.

On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1 and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1, and internal clocking selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the MCLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. The receive data is always sampled on the falling edge of internal MCLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and MCLKR is an input pin), the external rising-edge triggered input clock on MCLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1 and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the MCLKR pin.

CLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold of data around this edge. Figure 15-58 shows how data clocked by an external serial device using a rising edge can be sampled by the McBSP receiver on the falling edge of the same clock.

Figure 15-58. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



15.9.17 Set the SRG Frame-Synchronization Period and Pulse Width

Table 15-65. Register Bits Used to Set SRG Frame-Synchronization Period and Pulse Width

Register	Bit	Name	Function	Type	Reset Value
SRGR2	11-0	FPER	Sample rate generator frame-synchronization period For the frame-synchronization signal FSG, (FPER + 1) determines the period from the start of a frame-synchronization pulse to the start of the next frame-synchronization pulse. Range for (FPER + 1): 1 to 4096 CLKG cycles.	R/W	0000 0000 0000
SRGR1	15-8	FWID	Sample rate generator frame-synchronization pulse width This field plus 1 determines the width of each frame-synchronization pulse on FSG. Range for (FWID + 1): 1 to 256 CLKG cycles.	R/W	0000 0000

15.9.17.1 Frame-Synchronization Period and Frame-Synchronization Pulse Width

The sample rate generator can produce a clock signal, CLKG, and a frame-synchronization signal, FSG. If the sample rate generator is supplying receive or transmit frame synchronization, you must program the bit fields FPER and FWID.

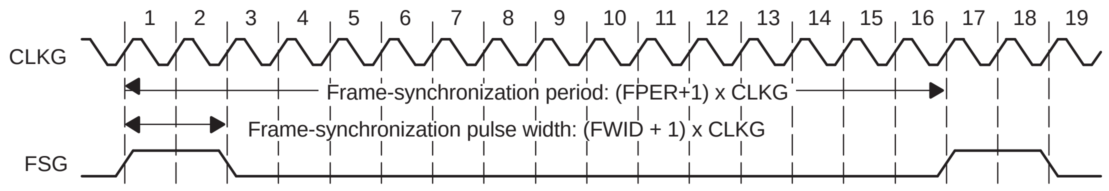
On FSG, the period from the start of a frame-synchronization pulse to the start of the next pulse is (FPER + 1) CLKG cycles. The 12 bits of FPER allow a frame-synchronization period of 1 to 4096 CLKG cycles, which allows up to 4096 data bits per frame. When GSYNC = 1, FPER is a don't care value.

Each pulse on FSG has a width of (FWID + 1) CLKG cycles. The eight bits of FWID allow a pulse width of 1 to 256 CLKG cycles. It is recommended that FWID be programmed to a value less than the programmed word length.

The values in FPER and FWID are loaded into separate down-counters. The 12-bit FPER counter counts down the generated clock cycles from the programmed value (4095 maximum) to 0. The 8-bit FWID counter counts down from the programmed value (255 maximum) to 0.

Figure 15-59 shows a frame-synchronization period of 16 CLKG periods (FPER = 15 or 00001111b) and a frame-synchronization pulse with an active width of 2 CLKG periods (FWID = 1).

Figure 15-59. Frame of Period 16 CLKG Periods and Active Width of 2 CLKG Periods


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
CLKG
Frame-synchronization period: (FPER+1) x CLKG
Frame-synchronization pulse width: (FWID + 1) x CLKG
FSG

When the sample rate generator comes out of reset, FSG is in its inactive state. Then, when GRST = 1 and FSGM = 1, a frame-synchronization pulse is generated. The frame width value (FWID + 1) is counted down on every CLKG cycle until it reaches 0, at which time FSG goes low. At the same time, the frame period value (FPER + 1) is also counting down. When this value reaches 0, FSG goes high, indicating a new frame.

15.9.18 Set the Transmit Clock Mode

Table 15-66. Register Bit Used to Set the Transmit Clock Mode

Register	Bit	Name	Function		Type	Reset Value
PCR	9	CLKXM	Transmit clock mode		R/W	0
			CLKXM = 0	The transmitter gets its clock signal from an external source via the MCLKX pin.		
			CLKXM = 1	The MCLKX pin is an output pin driven by the sample rate generator of the McBSP.		

15.9.18.1 Selecting a Source for the Transmit Clock and a Data Direction for the MCLKX pin

Table 15-67 shows how the CLKXM bit selects the transmit clock and the corresponding status of the MCLKX pin. The polarity of the signal on the MCLKX pin is determined by the CLKXP bit.

Table 15-67. How the CLKXM Bit Selects the Transmit Clock and the Corresponding Status of the MCLKX pin

CLKXM in PCR	Source of Transmit Clock	MCLKX pin Status
0	Internal CLKX is driven by an external clock on the MCLKX pin. CLKX is inverted as determined by CLKXP before being used.	Input
1	Internal CLKX is driven by the sample rate generator clock, CLKG.	Output. CLKG, inverted as determined by CLKXP, is driven out on CLKX.

15.9.18.2 Other Considerations

If the sample rate generator creates a clock signal (CLKG) that is derived from an external input clock, the GSYNC bit determines whether CLKG is kept synchronized with pulses on the FSR pin. For more details, see Section 15.4.3, *Synchronizing Sample Rate Generator Outputs to an External Clock*.

In the clock stop mode (CLKSTP = 10b or 11b), the McBSP can act as a master or as a slave in the SPI protocol. If the McBSP is a master, make sure that CLKXM = 1 so that CLKX is an output to supply the master clock to any slave devices. If the McBSP is a slave, make sure that CLKXM = 0 so that CLKX is an input to accept the master clock signal.

15.9.19 Set the Transmit Clock Polarity

Table 15-68. Register Bit Used to Set Transmit Clock Polarity

Register	Bit	Name	Function		Type	Reset Value
PCR	1	CLKXP	Transmit clock polarity		R/W	0
			CLKXP = 0	Transmit data sampled on rising edge of CLKX.		
			CLKXP = 1	Transmit data sampled on falling edge of CLKX.		

15.9.19.1 Frame Synchronization Pulses, Clock Signals, and Their Polarities

Transmit frame-synchronization pulses can be either generated internally by the sample rate generator (see Section 15.4.2) or driven by an external source. The source of frame synchronization is selected by programming the mode bit, FSXM, in PCR. FSX is also affected by the FSGM bit in SRGR2. For information about the effects of FSXM and FSGM, see Section 15.9.15, *Set the Transmit Frame-Synchronization Mode*). Similarly, transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLKXM, in the PCR (see Section 15.9.18, *Set the Transmit Clock Mode*).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-synchronization pulses), the McBSP detects them on the internal falling edge of clock, internal MCLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal MCLKR. These internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, MCLKR, and CLKX signals, respectively. All frame-synchronization signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP), and FSRP = FSXP = 1, the external active-low frame-synchronization signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active-high frame-synchronization signals are inverted, if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin.

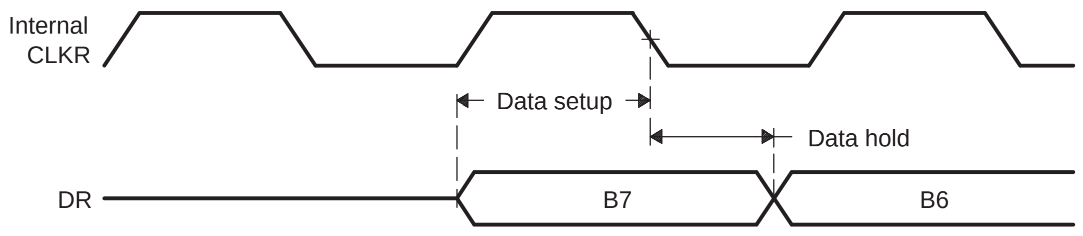
On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1 and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1 and internal clocking is selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the MCLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. The receive data is always sampled on the falling edge of internal MCLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and CLKR is an input pin), the external rising-edge triggered input clock on CLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1 and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the MCLKR pin.

CLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold of data around this edge (see Figure 15-58).

Figure 15-60 shows how data clocked by an external serial device using a rising edge can be sampled by the McBSP receiver on the falling edge of the same clock.

Figure 15-60. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



15.10 Emulation and Reset Considerations

This section covers the following topics:

- How to program McBSP response to a breakpoint in the high-level language debugger (see Section 15.10.1)
- How to reset and initialize the various parts of the McBSP (see Section 15.10.2)

15.10.1 McBSP Emulation Mode

FREE and SOFT are special emulation bits in SPCR2 that determine the state of the McBSP when a breakpoint is encountered in the high-level language debugger. If FREE = 1, the clock continues to run upon a software breakpoint and data is still shifted out. When FREE = 1, the SOFT bit is a *don't care.*

If FREE = 0, the SOFT bit takes effect. If SOFT = 0 when breakpoint occurs, the clock stops immediately, aborting a transmission. If SOFT = 1 and a breakpoint occurs while transmission is in progress, the transmission continues until completion of the transfer and then the clock halts. These options are listed in Table 15-69.

The McBSP receiver functions in a similar fashion. If a mode other than the immediate stop mode (SOFT = FREE = 0) is chosen, the receiver continues running and an overrun error is possible.

Table 15-69. McBSP Emulation Modes Selectable with FREE and SOFT Bits of SPCR2

FREE	SOFT	McBSP Emulation Mode
0	0	Immediate stop mode (reset condition)
		The transmitter or receiver stops immediately in response to a breakpoint.
0	1	Soft stop mode
		When a breakpoint occurs, the transmitter stops after completion of the current word. The receiver is not affected.
1	0 or 1	Free run mode
		The transmitter and receiver continue to run when a breakpoint occurs.

15.10.2 Resetting and Initializing McBSP

15.10.2.1 McBSP Pin States: DSP Reset Versus Receiver/Transmitter Reset

Table 15-70 shows the state of McBSP pins when the serial port is reset due to direct receiver or transmitter reset on the device.

Table 15-70. Reset State of Each McBSP Pin

Pin	Possible State(s)[(1)]	State Forced by Device Reset	State Forced by Receiver/Transmitter Reset
Receiver reset (RRST = 0 and GRST = 1)			
MDRx	I	GPIO-input	Input
MCLKRx	I/O/Z	GPIO-input	Known state if input; MCLKR running if output
MFSRx	I/O/Z	GPIO-input	Known state if input; FSRP inactive state if output
Transmitter reset (XRST = 0 and GRST = 1)			
MDXx	O/Z	GPIO Input	High impedance
MCLKXx	I/O/Z	GPIO-input	Known state if input; CLKX running if output
MFSXx	I/O/Z	GPIO-input	Known state if input; FSXP inactive state if output

(1) In Possible State(s) column, I = Input, O = Output, Z = High impedance. In the 28x family, at device reset, all I/Os default to GPIO function and generally as inputs.

15.10.2.2 Device Reset, McBSP Reset, and Sample Rate Generator Reset

When the McBSP is reset in either of the above two ways, the machine is reset to its initial state, including reset of all counters and status bits. The receive status bits include RFULL, RRDY, and RSYNCERR. The transmit status bits include XEMPTY, XRDY, and XSYNCERR.

- Device reset. When the whole DSP is reset ($\overline{\text{XRS}}$ signal is driven low), all McBSP pins are in GPIO mode. When the device is pulled out of reset, the clock to the McBSP modules remains disabled.
- McBSP reset. When the receiver and transmitter reset bits, RRST and XRST, are loaded with 0s, the respective portions of the McBSP are reset and activity in the corresponding section of the serial port stops. Input-only pins such as MDRx, and all other pins that are configured as inputs are in a known state. The MFSRx and MFSXx pins are driven to their inactive state if they are not outputs. If the MCLKR and MCLKX pins are programmed as outputs, they are driven by CLKG, provided that GRST = 1. Lastly, the MDXx pin is in the high-impedance state when the transmitter and/or the device is reset.

  During normal operation, the sample rate generator is reset if the GRST bit is cleared. GRST must be 0 only when neither the transmitter nor the receiver is using the sample rate generator. In this case, the internal sample rate generator clock (CLKG) and its frame-synchronization signal (FSG) are driven inactive low.

  When the sample rate generator is not in the reset state (GRST = 1), pins MFSRx and MFSXx are in an inactive state when RRST = 0 and XRST = 0, respectively, even if they are outputs driven by FSG. This ensures that when only one portion of the McBSP is in reset, the other portion can continue operation when GRST = 1 and its frame synchronization is driven by FSG.
- Sample rate generator reset. The sample rate generator is reset when GRST is loaded with 0.

  When neither the transmitter nor the receiver is fed by CLKG and FSG, you can reset the sample rate generator by clearing GRST. In this case, CLKG and FSG are driven inactive low. If you then set GRST, CLKG starts and runs as programmed. Later, if GRST = 1, FSG pulses active high after the programmed number of CLKG cycles has elapsed.

15.10.2.3 McBSP Initialization Procedure

The serial port initialization procedure is as follows:

1. Make XRST = RRST = GRST = 0 in SPCR[1,2]. If coming out of a device reset, this step is not required.
2. While the serial port is in the reset state, program only the McBSP configuration registers (not the data registers) as required.
3. Wait for two clock cycles. This ensures proper internal synchronization.
4. Set up data acquisition as required (such as writing to DXR[1,2]).
5. Make XRST = RRST = 1 to enable the serial port. Make sure that as you set these reset bits, you do not modify any of the other bits in SPCR1 and SPCR2. Otherwise, you change the configuration you selected in step 2.
6. Set FRST = 1, if internally generated frame synchronization is required.
7. Wait two clock cycles for the receiver and transmitter to become active.

Alternatively, on either write (step 1 or 5), the transmitter and receiver can be placed in or taken out of reset individually by modifying the desired bit.

The above procedure for reset/initialization can be applied in general when the receiver or transmitter must be reset during its normal operation and when the sample rate generator is not used for either operation.

NOTE:

1. The necessary duration of the active-low period of XRST or RRST is at least two MCLKR/CLKX cycles.
2. The appropriate bits in serial port configuration registers SPCR[1,2], PCR, RCR[1,2], XCR[1,2], and SRGR[1,2] must only be modified when the affected portion of the serial port is in its reset state.
3. In most cases, the data transmit registers (DXR[1,2]) must be loaded by the CPU or by the DMA controller only when the transmitter is enabled (XRST = 1). An exception to this rule is when these registers are used for companding internal data (see Section 15.1.5.2, *Capability to Compand Internal Data*).
4. The bits of the channel control registers—MCR[1,2], RCER[A-H], XCER[A-H]—can be modified at any time as long as they are not being used by the current reception/transmission in a multichannel selection mode.

15.10.2.4 Resetting the Transmitter While the Receiver is Running

Example 5 shows values in the control registers that reset and configure the transmitter while the receiver is running.

Equation 5: Resetting and Configuring McBSP Transmitter While McBSP Receiver Running

```
SPCR1 = 0001h SPCR2 = 0030h
; The receiver is running with the receive interrupt (RINT) triggered by the
; receiver ready bit (RRDY). The transmitter is in its reset state
. The transmit interrupt (XINT) will be triggered by the transmit frame-sync
; error bit (XSYNCERR). PCR = 0900h
; Transmit frame synchronization is generated internally according to the
; FSGM bit of SRGR2.
; The transmit clock is driven by an external source.
; The receive clock continues to be driven by sample rate generator. The input clock
; of the sample rate generator is supplied by the CPU clock SRGR1 = 0001h SRGR2 = 2000h
; The CPU clock is the input clock for the sample rate generator. The sample
; rate generator divides the CPU clock by 2 to generate its output clock (CLKG).
; Transmit frame synchronization is tied to the automatic copying of data from
; the DXR(s) to the XSR(s). XCR1 = 0740h XCR2 = 8321h
; The transmit frame has two phases. Phase 1 has eight 16-bit words. Phase 2
; has four 12-bit words. There is 1-bit data delay between the start of a
; frame-sync pulse and the first data bit
; transmitted. SPCR2 = 0031h
; The transmitter is taken out of reset.
```

15.11 Data Packing Examples

This section shows two ways to implement data packing in the McBSP.

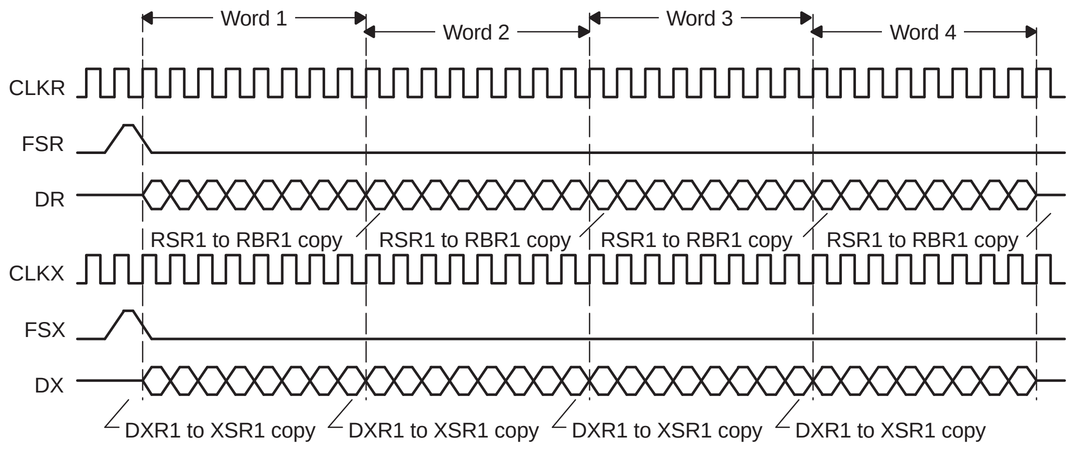
15.11.1 Data Packing Using Frame Length and Word Length

Frame length and word length can be manipulated to effectively pack data. For example, consider a situation where four 8-bit words are transferred in a single-phase frame as shown in Figure 15-61. In this case:

- (R/X)PHASE = 0: Single-phase frame
- (R/X)FRLEN1 = 0000011b: 4-word frame
- (R/X)WDLEN1 = 000b: 8-bit words

Four 8-bit data words are transferred to and from the McBSP by the CPU or by the DMA controller. Thus, four reads from DRR1 and four writes to DXR1 are necessary for each frame.

Figure 15-61. Four 8-Bit Data Words Transferred To/From the McBSP



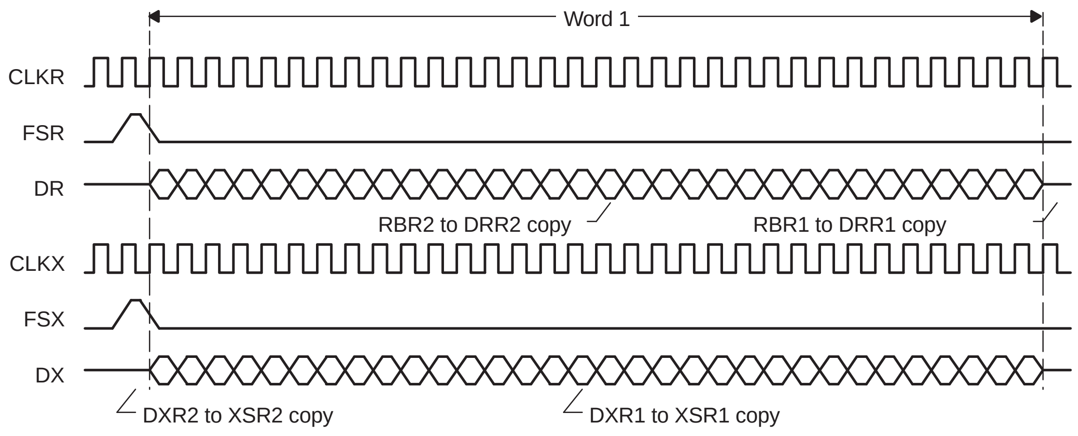
This data can also be treated as a single-phase frame consisting of one 32-bit data word, as shown in Figure 15-62. In this case:

- (R/X)PHASE = 0: Single-phase frame
- (R/X)FRLEN1 = 0000000b: 1-word frame
- (R/X)WDLEN1 = 101b: 32-bit word

Two 16-bit data words are transferred to and from the McBSP by the CPU or DMA controller. Thus, two reads, from DRR2 and DRR1, and two writes, to DXR2 and DXR1, are necessary for each frame. This results in only half the number of transfers compared to the previous case. This manipulation reduces the percentage of bus time required for serial port data movement.

NOTE: When the word length is larger than 16 bits, make sure you access DRR2/DXR2 before you access DRR1/DXR1. McBSP activity is tied to accesses of DRR1/DXR1. During the reception of 24-bit or 32-bit words, read DRR2 and then read DRR1. Otherwise, the next RBR[1,2]-to-DRR[1,2] copy occurs before DRR2 is read. Similarly, during the transmission of 24-bit or 32-bit words, write to DXR2 and then write to DXR1. Otherwise, the next DXR[1,2]-to-XSR[1,2] copy occurs before DXR2 is loaded with new data.

Figure 15-62. One 32-Bit Data Word Transferred To/From the McBSP



15.11.2 Data Packing Using Word Length and the Frame-Synchronization Ignore Function

When there are multiple words per frame, you can implement data packing by increasing the word length (defining a serial word with more bits) and by ignoring frame-synchronization pulses. First, consider Figure 15-63, which shows the McBSP operating at the maximum packet frequency. Here, each frame only has a single 8-bit word. Notice the frame-synchronization pulse that initiates each frame transfer for reception and for transmission. For reception, this configuration requires one read operation for each word. For transmission, this configuration requires one write operation for each word.

Figure 15-63. 8-Bit Data Words Transferred at Maximum Packet Frequency

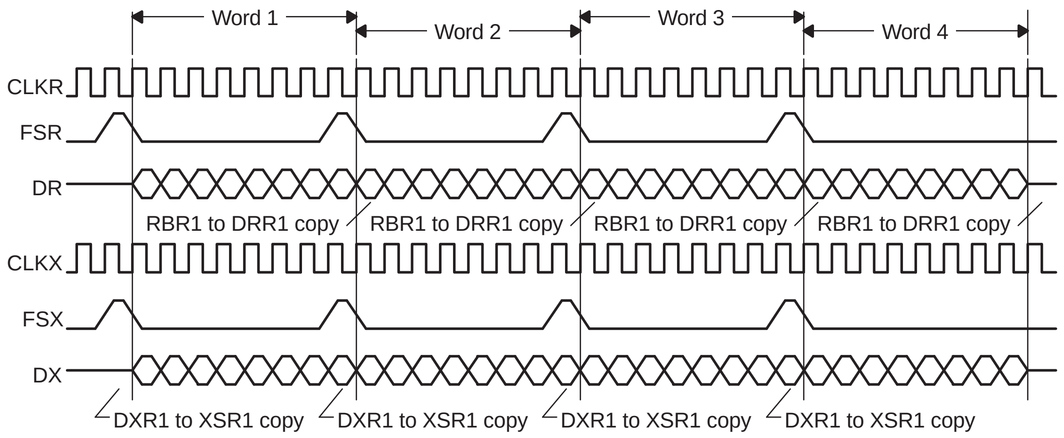
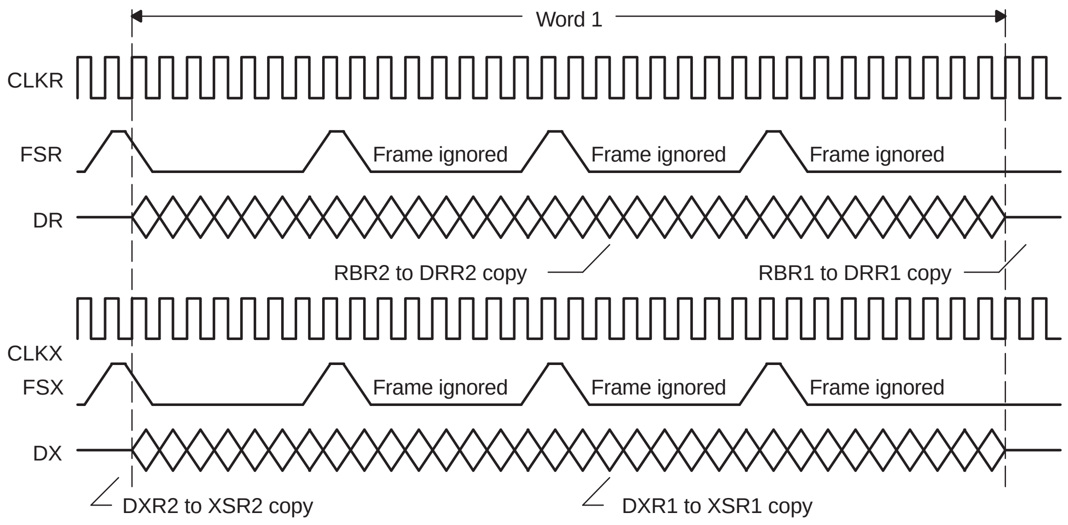


Figure 15-64 shows the McBSP configured to treat this data stream as a continuous 32-bit word. In this example, the McBSP responds to an initial frame-synchronization pulse. However, (R/X)FIG = 1 so that the McBSP ignores subsequent pulses. Only two read transfers or two write transfers are needed every 32 bits. This configuration effectively reduces the required bus bandwidth to half the bandwidth needed to transfer four 8-bit words.

Figure 15-64. Configuring the Data Stream of Figure 15-63 as a Continuous 32-Bit Word



15.12 McBSP Registers

This section describes the McBSP registers.

Table 15-71 shows the registers accessible on each McBSP. Section 15.12.2 through Section 15.12.11 describe the register bits.

15.12.1 Register Summary

Table 15-71. McBSP Register Summary

Name	McBSP-A Address	McBSP-B Address	Type	Reset Value	Description
Data Registers, Receive, Transmit					
DRR2	0x5000	0x5040	R	0x0000	McBSP Data Receive Register 2
DRR1	0x5001	0x5041	R	0x0000	McBSP Data Receive Register 1
DXR2	0x5002	0x5042	W	0x0000	McBSP Data Transmit Register 2
DXR1	0x5003	0x5043	W	0x0000	McBSP Data Transmit Register 1
McBSP Control Registers					
SPCR2	0x5004	0x5044	R/W	0x0000	McBSP Serial Port Control Register 2
SPCR1	0x5005	0x5045	R/W	0x0000	McBSP Serial Port Control Register 1
RCR2	0x5006	0x5046	R/W	0x0000	McBSP Receive Control Register 2
RCR1	0x5007	0x5047	R/W	0x0000	McBSP Receive Control Register 1
XCR2	0x5008	0x5048	R/W	0x0000	McBSP Transmit Control Register 2
XCR1	0x5009	0x5049	R/W	0x0000	McBSP Transmit Control Register 1
SRGR2	0x500A	0x504A	R/W	0x0000	McBSP Sample Rate Generator Register 2
SRGR1	0x500B	0x504B	R/W	0x0000	McBSP Sample Rate Generator Register 1
Multichannel Control Registers					
MCR2	0x500C	0x504C	R/W	0x0000	McBSP Multichannel Register 2
MCR1	0x500D	0x504D	R/W	0x0000	McBSP Multichannel Register 1
RCERA	0x500E	0x504E	R/W	0x0000	McBSP Receive Channel Enable Register Partition A
RCERB	0x500F	0x504F	R/W	0x0000	McBSP Receive Channel Enable Register Partition B
XCERA	0x5010	0x5050	R/W	0x0000	McBSP Transmit Channel Enable Register Partition A
XCERB	0x5011	0x5051	R/W	0x0000	McBSP Transmit Channel Enable Register Partition B
PCR	0x5012	0x5052	R/W	0x0000	McBSP Pin Control Register
RCERC	0x5013	0x5053	R/W	0x0000	McBSP Receive Channel Enable Register Partition C
RCERD	0x5014	0x5054	R/W	0x0000	McBSP Receive Channel Enable Register Partition D
XCERC	0x5015	0x5055	R/W	0x0000	McBSP Transmit Channel Enable Register Partition C
XCERD	0x5016	0x5056	R/W	0x0000	McBSP Transmit Channel Enable Register Partition D
RCERE	0x5017	0x5057	R/W	0x0000	McBSP Receive Channel Enable Register Partition E
RCERF	0x5018	0x5058	R/W	0x0000	McBSP Receive Channel Enable Register Partition F
XCERE	0x5019	0x5059	R/W	0x0000	McBSP Transmit Channel Enable Register Partition E
XCERF	0x501A	0x505A	R/W	0x0000	McBSP Transmit Channel Enable Register Partition F
RCERG	0x501B	0x505B	R/W	0x0000	McBSP Receive Channel Enable Register Partition G
RCERH	0x501C	0x505C	R/W	0x0000	McBSP Receive Channel Enable Register Partition H
XCERG	0x501D	0x505D	R/W	0x0000	McBSP Transmit Channel Enable Register Partition G
XCERH	0x501E	0x505E	R/W	0x0000	McBSP Transmit Channel Enable Register Partition H
MFFINT	0x5023	0x5063	R/W	0x0000	McBSP Interrupt Enable Register

15.12.2 Data Receive Registers (DRR[1,2])

The CPU or the DMA controller reads received data from one or both of the data receive registers (see Figure 15-65). If the serial word length is 16 bits or smaller, only DRR1 is used. If the serial length is larger than 16 bits, both DRR1 and DRR2 are used and DRR2 holds the most significant bits. Each frame of receive data in the McBSP can have one phase or two phases, each with its own serial word length.

Figure 15-65. Data Receive Registers (DRR2 and DRR1)

DDR2

15 … 0
High part of receive data (for 20-, 24- or 32-bit data)
R/W-0

DDR1

15 … 0
Receive data (for 8-, 12-, or 16-bit data) or low part of receive data (for 20-, 24- or 32-bit data)
R/W-0

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

15.12.2.1 Data Travel From Data Receive Pins to the Registers

If the serial word length is 16 bits or smaller, receive data on the MDRx pin is shifted into receive shift register 1 (RSR1) and then copied into receive buffer register 1 (RBR1). The content of RBR1 is then copied to DRR1, which can be read by the CPU or by the DMA controller. The RSRs and RBRs are not accessible to the user.

If the serial word length is larger than 16 bits, receive data on the MDRx pin is shifted into both of the receive shift registers (RSR2, RSR1) and then copied into both of the receive buffer registers (RBR2, RBR1). The content of the RBRs is then copied into both of the DRRs, which can be read by the CPU or by the DMA controller.

If companding is used during the copy from RBR1 to DRR1 (RCOMPAND = 10b or 11b), the 8-bit compressed data in RBR1 is expanded to a left-justified 16-bit value in DRR1. If companding is disabled, the data copied from RBR[1,2] to DRR[1,2] is justified and bit filled according to the RJUST bits.

15.12.3 Data Transmit Registers (DXR[1,2])

For transmission, the CPU or the DMA controller writes data to one or both of the data transmit registers (see Figure 15-66). If the serial word length is 16 bits or smaller, only DXR1 is used. If the word length is larger than 16 bits, both DXR1 and DXR2 are used and DXR2 holds the most significant bits. Each frame of transmit data in the McBSP can have one phase or two phases, each with its own serial word length.

Figure 15-66. Data Transmit Registers (DXR2 and DXR1)

DXR2

15 … 0
High part of transmit data (for 20-, 24- or 32-bit data)
R/W-0

DXR1

15 … 0
Transmit data (for 8-, 12-, or 16-bit data) or low part of receive data (for 20-, 24- or 32-bit data)
R/W-0

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

15.12.3.1 Data Travel From Registers to Data Transmit (DX) Pins

If the serial word length is 16 bits or fewer, data written to DXR1 is copied to transmit shift register 1 (XSR1). From XSR1, the data is shifted onto the DX pin one bit at a time. The XSRs are not accessible.

If the serial word length is more than 16 bits, data written to DXR1 and DXR2 is copied to both transmit shift registers (XSR2, XSR1). From the XSRs, the data is shifted onto the DX pin one bit at a time.

If companding is used during the transfer from DXR1 to XSR1 (XCOMPAND = 10b or 11b), the McBSP compresses the 16-bit data in DXR1 to 8-bit data in the μ-law or A-law format in XSR1. If companding is disabled, the McBSP passes data from the DXR(s) to the XSR(s) without modification.

15.12.4 Serial Port Control Registers (SPCR[1,2])

Each McBSP has two serial port control registers, SPCR1 (Table 15-72) and SPCR2 (Table 15-73). These registers enable you to:

- Control various McBSP modes: digital loopback mode (DLB), sign-extension and justification mode for reception (RJUST), clock stop mode (CLKSTP), interrupt modes (RINTM and XINTM), emulation mode (FREE and SOFT)
- Turn on and off the DX-pin delay enabler (DXENA)
- Check the status of receive and transmit operations (RSYNCERR, XSYNCERR, RFULL, XEMPTY, RRDY, XRDY)
- Reset portions of the McBSP (RRST, XRST, FRST, GRST)

15.12.4.1 Serial Port Control 1 Register (SPCR1)

The serial port control 1 register (SPCR1) is shown in Figure 15-67 and described in Table 15-72.

Figure 15-67. Serial Port Control 1 Register (SPCR1)

15	14	13	12	11	10		8
DLB	RJUST		CLKSTP		Reserved		
R/W-0	R/W-0		R/W-0		R-0		

7	6	5	4	3	2	1	0
DXENA	Reserved	RINTM		RSYNCERR	RFULL	RRDY	RRST
R/W-0	R/W-0	R/W-0		R/W-0	R-0	R-0	R/W-0

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

Table 15-72. Serial Port Control 1 Register (SPCR1) Field Descriptions

Bit	Field	Value	Description
15	DLB		Digital loopback mode bit. DLB disables or enables the digital loopback mode of the McBSP:
		0	Disabled
			Internal DR is supplied by the MDRx pin. Internal FSR and internal MCLKR can be supplied by their respective pins or by the sample rate generator, depending on the mode bits FSRM and CLKRM.
			Internal DX is supplied by the MDXx pin. Internal FSX and internal CLKX are supplied by their respective pins or are generated internally, depending on the mode bits FSXM and CLKXM.
		1	Enabled
			Internal receive signals are supplied by internal transmit signals:
			MDRx connected to MDXx
			MFSRx connected to MFSXx
			MCLKR connected to MCLKXx
			This mode allows you to test serial port code with a single DSP. The McBSP transmitter directly supplies data, frame synchronization, and clocking to the McBSP receiver.
14-13	RJUST	0-3h	Receive sign-extension and justification mode bits. During reception, RJUST determines how data is justified and bit filled before being passed to the data receive registers (DRR1, DRR2).
			RJUST is ignored if you enable a companding mode with the RCOMPAND bits. In a companding mode, the 8-bit compressed data in RBR1 is expanded to left-justified 16-bit data in DRR1.
			For more details about the effects of RJUST, see Section 15.8.13, *Set the Receive Sign-Extension and Justification Mode*
		0	Right justify the data and zero fill the MSBs.
		1h	Right justify the data and sign-extend the data into the MSBs.
		2h	Left justify the data and zero fill the LSBs.
		3h	Reserved (do not use)

Table 15-72. Serial Port Control 1 Register (SPCR1) Field Descriptions (continued)

Bit	Field	Value	Description
12-11	CLKSTP	0-3h	Clock stop mode bits. CLKSTP allows you to use the clock stop mode to support the SPI master-slave protocol. If you will not be using the SPI protocol, you can clear CLKSTP to disable the clock stop mode.
			In the clock stop mode, the clock stops at the end of each data transfer. At the beginning of each data transfer, the clock starts immediately (CLKSTP = 10b) or after a half-cycle delay (CLKSTP = 11b).
			For more details, see Section 15.8.5, *Enable/Disable the Clock Stop*.
		0-1h	Clock stop mode is disabled.
		2h	Clock stop mode, without clock delay
		3h	Clock stop mode, with half-cycle clock delay
10-8	Reserved	0	Reserved bits (not available for your use). They are read-only bits and return 0s when read.
7	DXENA		DX delay enabler mode bit. DXENA controls the delay enabler for the DX pin. The enabler creates an extra delay for turn-on time (for the length of the delay, see the device-specific data sheet). For more details about the effects of DXENA, see Section 15.9.13, *Set the Transmit DXENA Mode*.
		0	DX delay enabler off
		1	DX delay enabler on
6	Reserved	0	Reserved
5-4	RINTM	0-3h	Receive interrupt mode bits. RINTM determines which event in the McBSP receiver generates a receive interrupt (RINT) request. If RINT is properly enabled inside the CPU, the CPU services the interrupt request; otherwise, the CPU ignores the request.
		0	The McBSP sends a receive interrupt (RINT) request to the CPU when the RRDY bit changes from 0 to 1, indicating that receive data is ready to be read (the content of RBR[1,2] has been copied to DRR[1,2]):
			Regardless of the value of RINTM, you can check RRDY to determine whether a word transfer is complete.
			The McBSP sends a RINT request to the CPU when 16 enabled bits have been received on the DR pin.
		1h	In the multichannel selection mode, the McBSP sends a RINT request to the CPU after every 16-channel block is received in a frame.
			Outside of the multichannel selection mode, no interrupt request is sent.
		2h	The McBSP sends a RINT request to the CPU when each receive frame-synchronization pulse is detected. The interrupt request is sent even if the receiver is in its reset state.
		3h	The McBSP sends a RINT request to the CPU when the RSYNCERR bit is set, indicating a receive frame-synchronization error.
			Regardless of the value of RINTM, you can check RSYNCERR to determine whether a receive frame-synchronization error occurred.
3	RSYNCERR		Receive frame-sync error bit. RSYNCERR is set when a receive frame-sync error is detected by the McBSP. If RINTM = 11b, the McBSP sends a receive interrupt (RINT) request to the CPU when RSYNCERR is set. The flag remains set until you write a 0 to it or reset the receiver.
		0	No error
		1	Receive frame-synchronization error. For more details about this error, see Section 15.5.3, *Unexpected Receive Frame-Synchronization Pulse*.
2	RFULL		Receiver full bit. RFULL is set when the receiver is full with new data and the previously received data has not been read (receiver-full condition). For more details about this condition, see Section 15.5.2, *Overrun in the Receiver*.
		0	No receiver-full condition
		1	Receiver-full condition: RSR[1,2] and RBR[1,2] are full with new data, but the previous data in DRR[1,2] has not been read.

Table 15-72. Serial Port Control 1 Register (SPCR1) Field Descriptions (continued)

Bit	Field	Value	Description
1	RRDY		Receiver ready bit. RRDY is set when data is ready to be read from DRR[1,2]. Specifically, RRDY is set in response to a copy from RBR1 to DRR1. If the receive interrupt mode is RINTM = 00b, the McBSP sends a receive interrupt request to the CPU when RRDY changes from 0 to 1. Also, when RRDY changes from 0 to 1, the McBSP sends a receive synchronization event (REVT) signal to the DMA controller.
		0	Receiver not ready When the content of DRR1 is read, RRDY is automatically cleared.
		1	Receiver ready: New data can be read from DRR[1,2]. Important: If both DRRs are required (word length larger than 16 bits), the CPU or the DMA controller must read from DRR2 first and then from DRR1. As soon as DRR1 is read, the next RBR-to-DRR copy is initiated. If DRR2 is not read first, the data in DRR2 is lost.
0	RRST		Receiver reset bit. You can use RRST to take the McBSP receiver into and out of its reset state. This bit has a negative polarity; RRST = 0 indicates the reset state. To read about the effects of a receiver reset, see Section 15.10.2, *Resetting and Initializing a McBSP*.
		0	If you read a 0, the receiver is in its reset state. If you write a 0, you reset the receiver.
		1	If you read a 1, the receiver is enabled. If you write a 1, you enable the receiver by taking it out of its reset state.

15.12.4.2 Serial Port Control 2 Register (SPCR2)

The serial port control 2 register (SPCR2) is shown in Figure 15-68 and described in Table 15-73.

Figure 15-68. Serial Port Control 2 Register (SPCR2)

15–10	9	8
Reserved	FREE	SOFT
R-0	R/W-0	R/W-0

7	6	5–4	3	2	1	0
FRST	GRST	XINTM	XSYNCERR	XEMPTY	XRDY	XRST
R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R/W-0

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

Table 15-73. Serial Port Control 2 Register (SPCR2) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved bits (not available for your use). They are read-only bits and return 0s when read.
9	FREE		Free run bit. When a breakpoint is encountered in the high-level language debugger, FREE determines whether the McBSP transmit and receive clocks continue to run or whether they are affected as determined by the SOFT bit. When one of the clocks stops, the corresponding data transfer (transmission or reception) stops.
8	SOFT		Soft stop bit. When FREE = 0, SOFT determines the response of the McBSP transmit and receive clocks when a breakpoint is encountered in the high-level language debugger. When one of the clocks stops, the corresponding data transfer (transmission or reception) stops.
7	FRST		Frame-synchronization logic reset bit. The sample rate generator of the McBSP includes frame-synchronization logic to generate an internal frame-synchronization signal. You can use FRST to take the frame-synchronization logic into and out of its reset state. This bit has a negative polarity; FRST = 0 indicates the reset state.
		0	If you read a 0, the frame-synchronization logic is in its reset state. If you write a 0, you reset the frame-synchronization logic. In the reset state, the frame-synchronization logic does not generate a frame-synchronization signal (FSG).
		1	If you read a 1, the frame-synchronization logic is enabled. If you write a 1, you enable the frame-synchronization logic by taking it out of its reset state. When the frame-synchronization logic is enabled (FRST = 1) and the sample rate generator as a whole is enabled (GRST = 1), the frame-synchronization logic generates the frame-synchronization signal FSG as programmed.
6	GRST		Sample rate generator reset bit. You can use GRST to take the McBSP sample rate generator into and out of its reset state. This bit has a negative polarity; GRST = 0 indicates the reset state. To read about the effects of a sample rate generator reset, see Section 15.10.2, *Resetting and Initializing a McBSP*.
		0	If you read a 0, the sample rate generator is in its reset state. If you write a 0, you reset the sample rate generator. If GRST = 0 due to a reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven low (inactive). If GRST = 0 due to program code, CLKG and FSG are both driven low (inactive).
		1	If you read a 1, the sample rate generator is enabled. If you write a 1, you enable the sample rate generator by taking it out of its reset state. When enabled, the sample rate generator generates the clock signal CLKG as programmed in the sample rate generator registers. If FRST = 1, the generator also generates the frame-synchronization signal FSG as programmed in the sample rate generator registers.

Table 15-73. Serial Port Control 2 Register (SPCR2) Field Descriptions (continued)

Bit	Field	Value	Description
5-4	XINTM	0-3h	Transmit interrupt mode bits. XINTM determines which event in the McBSP transmitter generates a transmit interrupt (XINT) request. If XINT is properly enabled, the CPU services the interrupt request; otherwise, the CPU ignores the request.
		0	The McBSP sends a transmit interrupt (XINT) request to the CPU when the XRDY bit changes from 0 to 1, indicating that transmitter is ready to accept new data (the content of DXR[1,2] has been copied to XSR[1,2]).
			Regardless of the value of XINTM, you can check XRDY to determine whether a word transfer is complete.
			The McBSP sends an XINT request to the CPU when 16 enabled bits have been transmitted on the DX pin.
		1h	In the multichannel selection mode, the McBSP sends an XINT request to the CPU after every 16-channel block is transmitted in a frame.
			Outside of the multichannel selection mode, no interrupt request is sent.
		2h	The McBSP sends an XINT request to the CPU when each transmit frame-synchronization pulse is detected. The interrupt request is sent even if the transmitter is in its reset state.
		3h	The McBSP sends an XINT request to the CPU when the XSYNCERR bit is set, indicating a transmit frame-synchronization error.
			Regardless of the value of XINTM, you can check XSYNCERR to determine whether a transmit frame-synchronization error occurred.
3	XSYNCERR		Transmit frame-synchronization error bit. XSYNCERR is set when a transmit frame-synchronization error is detected by the McBSP. If XINTM = 11b, the McBSP sends a transmit interrupt (XINT) request to the CPU when XSYNCERR is set. The flag remains set until you write a 0 to it or reset the transmitter.
			If XINTM = 11b, writing a 1 to XSYNCERR triggers a transmit interrupt just as if a transmit frame-synchronization error occurred.
			For details about this error see Section 15.5.5, *Unexpected Transmit Frame-Synchronization Pulse*.
		0	No error
		1	Transmit frame-synchronization error
2	XEMPTY		Transmitter empty bit. XEMPTY is cleared when the transmitter is ready to send new data but no new data is available (transmitter-empty condition). This bit has a negative polarity; a transmitter-empty condition is indicated by XEMPTY = 0.
		0	Transmitter-empty condition
			Typically this indicates that all the bits of the current word have been transmitted but there is no new data in DXR1. XEMPTY is also cleared if the transmitter is reset and then restarted.
			For more details about this error condition, see Section 15.5.4.3, *Underflow in the Transmitter*.
		1	No transmitter-empty condition
1	XRDY		Transmitter ready bit. XRDY is set when the transmitter is ready to accept new data in DXR[1,2]. Specifically, XRDY is set in response to a copy from DXR1 to XSR1.
			If the transmit interrupt mode is XINTM = 00b, the McBSP sends a transmit interrupt (XINT) request to the CPU when XRDY changes from 0 to 1.
			Also, when XRDY changes from 0 to 1, the McBSP sends a transmit synchronization event (XEVT) signal to the DMA controller.
		0	Transmitter not ready
			When DXR1 is loaded, XRDY is automatically cleared.
		1	Transmitter ready: DXR[1,2] is ready to accept new data.
			If both DXRs are needed (word length larger than 16 bits), the CPU or the DMA controller must load DXR2 first and then load DXR1. As soon as DXR1 is loaded, the contents of both DXRs are copied to the transmit shift registers (XSRs), as described in the next step. If DXR2 is not loaded first, the previous content of DXR2 is passed to the XSR2.

Table 15-73. Serial Port Control 2 Register (SPCR2) Field Descriptions (continued)

Bit	Field	Value	Description
0	XRST		Transmitter reset bit. You can use XRST to take the McBSP transmitter into and out of its reset state. This bit has a negative polarity; XRST = 0 indicates the reset state.
			To read about the effects of a transmitter reset, see Section 15.10.2, *Resetting and Initializing a McBSP*.
		0	If you read a 0, the transmitter is in its reset state.
			If you write a 0, you reset the transmitter.
		1	If you read a 1, the transmitter is enabled.
			If you write a 1, you enable the transmitter by taking it out of its reset state.

15.12.5 Receive Control Registers (RCR[1, 2])

Each McBSP has two receive control registers, RCR1 (Table 15-74) and RCR2 (Table 15-76). These registers enable you to:

- Specify one or two phases for each frame of receive data (RPHASE)
- Define two parameters for phase 1 and (if necessary) phase 2: the serial word length (RWDLEN1, RWDLEN2) and the number of words (RFRLEN1, RFRLEN2)
- Choose a receive companding mode, if any (RCOMPAND)
- Enable or disable the receive frame-synchronization ignore function (RFIG)
- Choose a receive data delay (RDATDLY)

15.12.5.1 Receive Control Register 1 (RCR1)

The receive control register 1 (RCR1) is shown in Figure 15-69 and described in Table 15-74.

Figure 15-69. Receive Control Register 1 (RCR1)

15	14–8
Reserved	RFRLEN1
R-0	R/W-0

7–5	4–0
RWDLEN1	Reserved
R/W-0	R-0

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

Table 15-74. Receive Control Register 1 (RCR1) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved bits (not available for your use). They are read-only bits and return 0s when read.
14-8	RFRLEN1	0-7Fh	Receive frame length 1 (1 to 128 words). Each frame of receive data can have one or two phases, depending on value that you load into the RPHASE bit. If a single-phase frame is selected, RFRLEN1 in RCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, RFRLEN1 determines the number of serial words in phase 1 of the frame, and RFRLEN2 in RCR2 determines the number of words in phase 2 of the frame. The 7-bit RFRLEN fields allow up to 128 words per phase. See Table 15-75 for a summary of how you determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period.
			Program the RFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 1, load 127 into RFRLEN1.

Table 15-74. Receive Control Register 1 (RCR1) Field Descriptions (continued)

Bit	Field	Value	Description
7-5	RWDLEN1	0-7h	Receive word length 1. Each frame of receive data can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, RWDLEN1 in RCR1 selects the length for every serial word received in the frame. If a dual-phase frame is selected, RWDLEN1 determines the length of the serial words in phase 1 of the frame, and RWDLEN2 in RCR2 determines the word length in phase 2 of the frame.
		0	8 bits
		1h	12 bits
		2h	16 bits
		3h	20 bits
		4h	24 bits
		5h	32 bits
		6h-7h	Reserved (do not use)
4-0	Reserved	0	Reserved bits (not available for your use). They are read-only bits and return 0s when read.

Table 15-75. Frame Length Formula for Receive Control 1 Register (RCR1)

RPHASE	RFRLEN1	RFRLEN2	Frame Length
0	$0 \le$ RFRLEN1 ≤ 127	Not used	(RFRLEN1 + 1) words
1	$0 \le$ RFRLEN1 ≤ 127	$0 \le$ RFRLEN2 ≤ 127	(RFRLEN1 + 1) + (RFRLEN2 + 1) words

15.12.5.2 Receive Control Register 2 (RCR2)

The receive control register 2 (RCR2) is shown in Figure 15-70 and described in Table 15-76.

Figure 15-70. Receive Control Register 2 (RCR2)

15	14-8
RPHASE	RFRLEN2
R/W-0	R/W-0

7-5	4-3	2	1-0
RWDLEN2	RCOMPAND	RFIG	RDATDLY
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 15-76. Receive Control Register 2 (RCR2) Field Descriptions

Bit	Field	Value	Description
15	RPHASE		Receive phase number bit. RPHASE determines whether the receive frame has one phase or two phases. For each phase you can define the serial word length and the number of serial words in the phase. To set up phase 1, program RWDLEN1 (word length) and RFRLEN1 (number of words). To set up phase 2 (if there are two phases), program RWDLEN2 and RFRLEN2.
		0	Single-phase frame
			The receive frame has only one phase, phase 1.
		1	Dual-phase frame
			The receive frame has two phases, phase 1 and phase 2.
14-8		0-7Fh	Receive frame length 2 (1 to 128 words). Each frame of receive data can have one or two phases, depending on value that you load into the RPHASE bit. If a single-phase frame is selected, RFRLEN1 in RCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, RFRLEN1 determines the number of serial words in phase 1 of the frame, and RFRLEN2 in RCR2 determines the number of words in phase 2 of the frame. The 7-bit RFRLEN fields allow up to 128 words per phase. See Table 15-77 for a summary of how to determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period.
			Program the RFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 2, load 127 into RFRLEN2.

Table 15-76. Receive Control Register 2 (RCR2) Field Descriptions (continued)

Bit	Field	Value	Description
7-5	RWDLEN2	0-7h	Receive word length 2. Each frame of receive data can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, RWDLEN1 in RCR1 selects the length for every serial word received in the frame. If a dual-phase frame is selected, RWDLEN1 determines the length of the serial words in phase 1 of the frame, and RWDLEN2 in RCR2 determines the word length in phase 2 of the frame.
		0	8 bits
		1h	12 bits
		2h	16 bits
		3h	20 bits
		4h	24 bits
		5h	32 bits
		6h-7h	Reserved (do not use)
4-3	RCOMPAND	0-3h	Receive companding mode bits. Companding (COMpress and exPAND) hardware allows compression and expansion of data in either μ-law or A-law format. RCOMPAND allows you to choose one of the following companding modes for the McBSP receiver: For more details about these companding modes, see Section 15.1.5, *Companding (Compressing and Expanding) Data*.
		0	No companding, any size data, MSB received first
		1h	No companding, 8-bit data, LSB received first
		2h	μ-law companding, 8-bit data, MSB received first
		3h	A-law companding, 8-bit data, MSB received first
2	RFIG		Receive frame-synchronization ignore bit. If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully received, this pulse is treated as an unexpected frame-synchronization pulse. For more details about the frame-synchronization error condition, see Section 15.5.3, *Unexpected Receive Frame-Synchronization Pulse*. Setting RFIG causes the serial port to ignore unexpected frame-synchronization signals during reception. For more details on the effects of RFIG, see Section 15.8.10.1, *Enable/Disable the Receive Frame-Synchronization Ignore Function*.
		0	Frame-synchronization detect. An unexpected FSR pulse causes the receiver to discard the contents of RSR[1,2] in favor of the new incoming data. The receiver: 1. Aborts the current data transfer 2. Sets RSYNCERR in SPCR1 3. Begins the transfer of a new data word
		1	Frame-synchronization ignore. An unexpected FSR pulse is ignored. Reception continues uninterrupted.
1-0	RDATDLY	0-3h	Receive data delay bits. RDATDLY specifies a data delay of 0, 1, or 2 receive clock cycles after frame-synchronization and before the reception of the first bit of the frame. For more details, see Section 15.8.12, *Set the Receive Data Delay*.
		0	0-bit data delay
		1h	1-bit data delay
		2h	2-bit data delay
		3h	Reserved (do not use)

Table 15-77. Frame Length Formula for Receive Control 2 Register (RCR2)

RPHASE	RFRLEN1	RFRLEN2	Frame Length
0	0 ≤ RFRLEN1 ≤ 127	Not used	(RFRLEN1 + 1) words
1	0 ≤ RFRLEN1 ≤ 127	0 ≤ RFRLEN2 ≤ 127	(RFRLEN1 + 1) + (RFRLEN2 + 1) words

15.12.6 Transmit Control Registers (XCR1 and XCR2)

Each McBSP has two transmit control registers, XCR1 (Table 15-78) and XCR2 (Table 15-80). These registers enable you to:

- Specify one or two phases for each frame of transmit data (XPHASE)
- Define two parameters for phase 1 and (if necessary) phase 2: the serial word length (XWDLEN1, XWDLEN2) and the number of words (XFRLEN1, XFRLEN2)
- Choose a transmit companding mode, if any (XCOMPAND)
- Enable or disable the transmit frame-sync ignore function (XFIG)
- Choose a transmit data delay (XDATDLY)

15.12.6.1 Transmit Control 1 Register (XCR1)

The transmit control 1 register (XCR1) is shown in Figure 15-71 and described in Table 15-78.

Figure 15-71. Transmit Control 1 Register (XCR1)

15	14 8
Reserved	XFRLEN1
R-0	R/W-0

7 5	4 0
XWDLEN1	Reserved
R/W-0	R-0

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

Table 15-78. Transmit Control 1 Register (XCR1) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved bit. Read-only; returns 0 when read.
14-8	XFRLEN1	0-7Fh	Transmit frame length 1 (1 to 128 words). Each frame of transmit data can have one or two phases, depending on value that you load into the XPHASE bit. If a single-phase frame is selected, XFRLEN1 in XCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, XFRLEN1 determines the number of serial words in phase 1 of the frame and XFRLEN2 in XCR2 determines the number of words in phase 2 of the frame. The 7-bit XFRLEN fields allow up to 128 words per phase. See Table 15-79 for a summary of how you determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period. Program the XFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 1, load 127 into XFRLEN1.
7-5	XWDLEN1	0-3h	Transmit word length 1. Each frame of transmit data can have one or two phases, depending on the value that you load into the XPHASE bit. If a single-phase frame is selected, XWDLEN1 in XCR1 selects the length for every serial word transmitted in the frame. If a dual-phase frame is selected, XWDLEN1 determines the length of the serial words in phase 1 of the frame and XWDLEN2 in XCR2 determines the word length in phase 2 of the frame.
		0	8 bits
		1h	12 bits
		2h	16 bits
		3h	20 bits
		4h	24 bits
		5h	32 bits
		6h-7h	Reserved (do not use)
4-0	Reserved	0	Reserved bits. They are read-only bits and return 0s when read.

Table 15-79. Frame Length Formula for Transmit Control 1 Register (XCR1)

XPHASE	XFRLEN1	XFRLEN2	Frame Length
0	0 ≤ XFRLEN1 ≤ 127	Not used	(XFRLEN1 + 1) words
1	0 ≤ XFRLEN1 ≤ 127	0 ≤ XFRLEN2 ≤ 127	(XFRLEN1 + 1) + (XFRLEN2 + 1) words

15.12.6.2 Transmit Control 2 Register (XCR2)

The transmit control 2 register (XCR2) is shown in Figure 15-72 and described in Table 15-80.

Figure 15-72. Transmit Control 2 Register (XCR2)

15	14–8
XPHASE	XFRLEN2
R/W-0	R/W-0

7–5	4–3	2	1–0
XWDLEN2	XCOMPAND	XFIG	XDATDLY
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 15-80. Transmit Control 2 Register (XCR2) Field Descriptions

Bit	Field	Value	Description
15	XPHASE		Transmit phase number bit. XPHASE determines whether the transmit frame has one phase or two phases. For each phase you can define the serial word length and the number of serial words in the phase. To set up phase 1, program XWDLEN1 (word length) and XFRLEN1 (number of words). To set up phase 2 (if there are two phases), program XWDLEN2 and XFRLEN2.
		0	Single-phase frame
			The transmit frame has only one phase, phase 1.
		1	Dual-phase frame
			The transmit frame has two phases, phase 1 and phase 2.
14-8	XFRLEN2	0-7Fh	Transmit frame length 2 (1 to 128 words). Each frame of transmit data can have one or two phases, depending on value that you load into the XPHASE bit. If a single-phase frame is selected, XFRLEN1 in XCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, XFRLEN1 determines the number of serial words in phase 1 of the frame and XFRLEN2 in XCR2 determines the number of words in phase 2 of the frame. The 7-bit XFRLEN fields allow up to 128 words per phase. See Table 15-81 for a summary of how to determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period.
			Program the XFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 1, load 127 into XFRLEN1.
7-5	XWDLEN2	0-7h	Transmit word length 2. Each frame of transmit data can have one or two phases, depending on the value that you load into the XPHASE bit. If a single-phase frame is selected, XWDLEN1 in XCR1 selects the length for every serial word transmitted in the frame. If a dual-phase frame is selected, XWDLEN1 determines the length of the serial words in phase 1 of the frame and XWDLEN2 in XCR2 determines the word length in phase 2 of the frame.
		0	8 bits
		1h	12 bits
		2h	16 bits
		3h	20 bits
		4h	24 bits
		5h	32 bits
		6h-7h	Reserved (do not use)
4-3	XCOMPAN D	0-3h	Transmit companding mode bits. Companding (COMpress and exPAND) hardware allows compression and expansion of data in either μ-law or A-law format. For more details, see Section 15.1.5.
			XCOMPAND allows you to choose one of the following companding modes for the McBSP transmitter. For more details about these companding modes, see Section 15.1.5 .
		0	No companding, any size data, MSB transmitted first
		1h	No companding, 8-bit data, LSB transmitted first
		2h	μ-law companding, 8-bit data, MSB transmitted first
		3h	A-law companding, 8-bit data, MSB transmitted first

Table 15-80. Transmit Control 2 Register (XCR2) Field Descriptions (continued)

Bit	Field	Value	Description
2	XFIG		Transmit frame-synchronization ignore bit. If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully transmitted, this pulse is treated as an unexpected frame-synchronization pulse. For more details about the frame-synchronization error condition, see Section 15.5.5. Setting XFIG causes the serial port to ignore unexpected frame-synchronization pulses during transmission. For more details on the effects of XFIG, see Section 15.9.10.
		0	Frame-synchronization detect. An unexpected FSX pulse causes the transmitter to discard the content of XSR[1,2]. The transmitter: 1. Aborts the present transmission 2. Sets XSYNCERR in SPCR2 3. Begins a new transmission from DXR[1,2]. If new data was written to DXR[1,2] since the last DXR[1,2]-to-XSR[1,2] copy, the current value in XSR[1,2] is lost. Otherwise, the same data is transmitted.
		1	Frame-synchronization ignore. An unexpected FSX pulse is ignored. Transmission continues uninterrupted.
1-0	XDATDLY	0-3h	Transmit data delay bits. XDATDLY specifies a data delay of 0, 1, or 2 transmit clock cycles after frame synchronization and before the transmission of the first bit of the frame. For more details, see Section 15.9.12.
		0	0-bit data delay
		1h	1-bit data delay
		2h	2-bit data delay
		3h	Reserved (do not use)

Table 15-81. Frame Length Formula for Transmit Control 2 Register (XCR2)

XPHASE	XFRLEN1	XFRLEN2	Frame Length
0	0 ≤ XFRLEN1 ≤ 127	Not used	(XFRLEN1 + 1) words
1	0 ≤ XFRLEN1 ≤ 127	0 ≤ XFRLEN2 ≤ 127	(XFRLEN1 + 1) + (XFRLEN2 + 1) words

15.12.7 Sample Rate Generator Registers (SRGR1 and SRGR2)

Each McBSP has two sample rate generator registers, SRGR1 (Table 15-82) and SRGR2 (Table 15-83). The sample rate generator can generate a clock signal (CLKG) and a frame-synchronization signal (FSG). The registers SRGR1 and SRGR2 enable you to:

- Select the input clock source for the sample rate generator (CLKSM, in conjunction with the SCLKME bit of PCR)
- Divide down the frequency of CLKG (CLKGDV)
- Select whether internally-generated transmit frame-synchronization pulses are driven by FSG or by activity in the transmitter (FSGM).
- Specify the width of frame-synchronization pulses on FSG (FWID) and specify the period between those pulses (FPER)

When an external source (via the MCLKR or MCLKX pin) provides the input clock source for the sample rate generator:

- If the CLKX/MCLKR pin is used, the polarity of the input clock is selected with CLKXP/CLKRP of PCR.
- The GSYNC bit of SRGR2 allows you to make CLKG synchronized to an external frame-synchronization signal on the FSR pin, so that CLKG is kept in phase with the input clock.

15.12.7.1 Sample Rate Generator 1 Register (SRGR1)

The sample rate generator 1 register is shown in Figure 15-73 and described in Table 15-82.

Figure 15-73. Sample Rate Generator 1 Register (SRGR1)

15	8
FWID	
R/W-0	

7	0
CLKGDV	
R/W-1	

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

Table 15-82. Sample Rate Generator 1 Register (SRGR1) Field Descriptions

Bit	Field	Value	Description
15-8	FWID	0-FFh	Frame-synchronization pulse width bits for FSG The sample rate generator can produce a clock signal, CLKG, and a frame-synchronization signal, FSG. For frame-synchronization pulses on FSG, (FWID + 1) is the pulse width in CLKG cycles. The eight bits of FWID allow a pulse width of 1 to 256 CLKG cycles: $0 \le \text{FWID} \le 255$ $1 \le (\text{FWID} + 1) \le 256$ CLKG cycles The period between the frame-synchronization pulses on FSG is defined by the FPER bits.
7-0	CLKGDV	0-FFh	Divide-down value for CLKG. The sample rate generator can accept an input clock signal and divide it down according to CLKGDV to produce an output clock signal, CLKG. The frequency of CLKG is: CLKG frequency = (Input clock frequency)/ (CLKGDV + 1) The input clock is selected by the SCLKME and CLKSM bits: **SCLKME** / **CLKSM** / **Input Clock For Sample Rate Generator** 0 / 0 / Reserved 0 / 1 / LSPCLK 1 / 0 / Signal on MCLKR pin 1 / 1 / Signal on MCLKX pin

15.12.7.2 Sample Rate Generator 2 Register (SRGR2)

The sample rate generator 2 register (SRGR2) is shown in Figure 15-74 and described in Table 15-83.

Figure 15-74. Sample Rate Generator 2 Register (SRGR2)

15	14	13	12	11	8
GSYNC	Reserved	CLKSM	FSGM	FPER	
R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	

7	0
FPER	
R/W-0	

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

Table 15-83. Sample Rate Generator 2 Register (SRGR2) Field Descriptions

Bit	Field	Value	Description
15	GSYNC		Clock synchronization mode bit for CLKG. GSYNC is used only when the input clock source for the sample rate generator is external—on the MCLKR pin. When GSYNC = 1, the clock signal (CLKG) and the frame-synchronization signal (FSG) generated by the sample rate generator are made dependent on pulses on the FSR pin.
		0	No clock synchronization CLKG oscillates without adjustment, and FSG pulses every (FPER + 1) CLKG cycles.
		1	Clock synchronization • CLKG is adjusted as necessary so that it is synchronized with the input clock on the MCLKR pin. • FSG pulses. FSG only pulses in response to a pulse on the FSR pin. The frame-synchronization period defined in FPER is ignored. For more details, see Section 15.4.3, *Synchronizing Sample Rate Generator Outputs to an External Clock*.
14	Reserved		Reserved
13	CLKSM	0	Sample rate generator input clock mode bit. The sample rate generator can accept an input clock signal and divide it down according to CLKGDV to produce an output clock signal, CLKG. The frequency of CLKG is:) CLKG frequency = (input clock frequency)/ (CLKGDV + 1 CLKSM is used in conjunction with the SCLKME bit to determine the source for the input clock. A reset selects the CPU clock as the input clock and forces the CLKG frequency to ½ the LSPCLK frequency. The input clock for the sample rate generator is taken from the MCLKR pin, depending on the value of the SCLKME bit of PCR: **SCLKME** / **CLKSM** / **Input Clock For Sample Rate Generator** 0 / 0 / Reserved 1 / 0 / Signal on MCLKR pin
		1	The input clock for the sample rate generator is taken from the LSPCLK or from the MCLKX pin, depending on the value of the SCLKME bit of PCR: **SCLKME** / **CLKSM** / **Input Clock For Sample Rate Generator** 0 / 1 / LSPCLK 1 / 1 / Signal on MCLKX pin
12	FSGM		Sample rate generator transmit frame-synchronization mode bit. The transmitter can get frame synchronization from the FSX pin (FSXM = 0) or from inside the McBSP (FSXM = 1). When FSXM = 1, the FSGM bit determines how the McBSP supplies frame-synchronization pulses.
		0	If FSXM = 1, the McBSP generates a transmit frame-synchronization pulse when the content of DXR[1,2] is copied to XSR[1,2].
		1	If FSXM = 1, the transmitter uses frame-synchronization pulses generated by the sample rate generator. Program the FWID bits to set the width of each pulse. Program the FPER bits to set the period between pulses.
11-0	FPER	0-FFFh	Frame-synchronization period bits for FSG. The sample rate generator can produce a clock signal, CLKG, and a frame-synchronization signal, FSG. The period between frame-synchronization pulses on FSG is (FPER + 1) CLKG cycles. The 12 bits of FPER allow a frame-synchronization period of 1 to 4096 CLKG cycles: 0 ≤ FPER ≤ 4095 1 ≤ (FPER + 1) ≤ 4096 CLKG cycles The width of each frame-synchronization pulse on FSG is defined by the FWID bits.

15.12.8 Multichannel Control Registers (MCR[1,2])

Each McBSP has two multichannel control registers. MCR1 (Table 15-84) has control and status bits (with an R prefix) for multichannel selection operation in the receiver. MCR2 (Table 15-85) contains the same type of bits (bit with an X prefix) for the transmitter. These registers enable you to:

- Enable all channels or only selected channels for reception (RMCM)
- Choose which channels are enabled/disabled and masked/unmasked for transmission (XMCM)
- Specify whether two partitions (32 channels at a time) or eight partitions (128 channels at a time) can be used (RMCME for reception, XMCME for transmission)
- Assign blocks of 16 channels to partitions A and B when the 2-partition mode is selected (RPABLK and RPBBLK for reception, XPABLK and XPBBLK for transmission)
- Determine which block of 16 channels is currently involved in a data transfer (RCBLK for reception, XCBLK for transmission)

15.12.8.1 Multichannel Control 1 Register (MCR1)

The multichannel control 1 register (MCR1) is shown in Figure 15-75 and described in Table 15-84.

Figure 15-75. Multichannel Control 1 Register (MCR1)

15-10	9	8
Reserved	RMCME	RPBBLK
R-0	R/W-0	R/W-0

7	6-5	4-2	1	0
RPBBLK	RPABLK	RCBLK	Reserved	RMCM
R/W-0	R/W-0	R-0	R-0	R/W-0

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

Table 15-84. Multichannel Control 1 Register (MCR1) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved bits (not available for your use). They are read-only bits and return 0s when read.
9	RMCME		Receive multichannel partition mode bit. RMCME is only applicable if channels can be individually enabled or disabled for reception (RMCM = 1).
			RMCME determines whether only 32 channels or all 128 channels are to be individually selectable.
		0	2-partition mode
			Only partitions A and B are used. You can control up to 32 channels in the receive multichannel selection mode (RMCM = 1).
			Assign 16 channels to partition A with the RPABLK bits.
			Assign 16 channels to partition B with the RPBBLK bits.
			You control the channels with the appropriate receive channel enable registers:
			RCERA: Channels in partition A
			RCERB: Channels in partition B
		1	8-partition mode
			All partitions (A through H) are used. You can control up to 128 channels in the receive multichannel selection mode. You control the channels with the appropriate receive channel enable registers:
			RCERA: Channels 0 through 15
			RCERB: Channels 16 through 31
			RCERC: Channels 32 through 47
			RCERD: Channels 48 through 63
			RCERE: Channels 64 through 79
			RCERF: Channels 80 through 95
			RCERG: Channels 96 through 111
			RCERH: Channels 112 through 127

Table 15-84. Multichannel Control 1 Register (MCR1) Field Descriptions (continued)

Bit	Field	Value	Description
8-7	RPBBLK	0-3h	Receive partition B block bits RPBBLK is only applicable if channels can be individually enabled or disabled (RMCM = 1) and the 2-partition mode is selected (RMCME = 0). Under these conditions, the McBSP receiver can accept or ignore data in any of the 32 channels that are assigned to partitions A and B of the receiver. The 128 receive channels of the McBSP are divided equally among 8 blocks (0 through 7). When RPBBLK is applicable, use RPBBLK to assign one of the odd-numbered blocks (1, 3, 5, or 7) to partition B. Use the RPABLK bits to assign one of the even-numbered blocks (0, 2, 4, or 6) to partition A. If you want to use more than 32 channels, you can change block assignments dynamically. You can assign a new block to one partition while the receiver is handling activity in the other partition. For example, while the block in partition A is active, you can change which block is assigned to partition B. The RCBLK bits are regularly updated to indicate which block is active. When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive block bits (RPABLK and RPBBLK) rather than the transmit block bits (XPABLK and XPBBLK).
		0	Block 1: channels 16 through 31
		1h	Block 3: channels 48 through 63
		2h	Block 5: channels 80 through 95
		3h	Block 7: channels 112 through 127
6-5	RPABLK	0-3h	Receive partition A block bits RPABLK is only applicable if channels can be individually enabled or disabled (RMCM = 1) and the 2-partition mode is selected (RMCME = 0). Under these conditions, the McBSP receiver can accept or ignore data in any of the 32 channels that are assigned to partitions A and B of the receiver. See the description for RPBBLK (bits 8-7) for more information about assigning blocks to partitions A and B.
		0	Block 0: channels 0 through 15
		1h	Block 2: channels 32 through 47
		2h	Block 5: channels 64 through 79
		3h	Block 7: channels 96 through 111
4-2	RCBLK	0-7h	Receive current block indicator. RCBLK indicates which block fo 16 channels is involved in the current McBSP reception:
		0	Block 0: channels 0 through 15
		1h	Block 1: channels 16 through 31
		2h	Block 2: channels 32 through 47
		3h	Block 3: channels 48 through 63
		4h	Block 4: channels 64 through 79
		5h	Block 5: channels 80 through 95
		6h	Block 6: channels 96 through 111
		7h	Block 7: channels 112 through 127
1	Reserved	0	Reserved bits (not available for your use). They are read-only bits and return 0s when read.
0	RMCM		Receive multichannel selection mode bit. RMCM determines whether all channels or only selected channels are enabled for reception:
		0	All 128 channels are enabled.
		1	Multichanneled selection mode. Channels can be individually enabled or disabled. The only channels enabled are those selected in the appropriate receive channel enable registers (RCERs). The way channels are assigned to the RCERs depends on the number of receive channel partitions (2 or 8), as defined by the RMCME bit.

15.12.8.2 Multichannel Control 2 Register (MCR2)

The multichannel control 2 register (MCR2) is shown in Figure 15-76 and described in Table 15-85.

Figure 15-76. Multichannel Control 2 Register (MCR2)

15–10	9	8
Reserved	XMCME	XPBBLK
R-0	R/W-0	R/W-0

7	6–5	4–2	1–0
XPBBLK	XPABLK	XCBLK	XMCM
R/W-0	R/W-0	R-0	R/W-0

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

Table 15-85. Multichannel Control 2 Register (MCR2) Field Descriptions

Bit	Field	Value	Description
15-10	Reserved	0	Reserved bits (not available for your use). They are read-only bits and return 0s when read.
9	XMCME		Transmit multichannel partition mode bit. XMCME determines whether only 32 channels or all 128 channels are to be individually selectable. XMCME is only applicable if channels can be individually disabled/enabled or masked/unmasked for transmission (XMCM is nonzero).
		0	2-partition mode. Only partitions A and B are used. You can control up to 32 channels in the transmit multichannel selection mode selected with the XMCM bits.
			If XMCM = 01b or 10b, assign 16 channels to partition A with the XPABLK bits. Assign 16 channels to partition B with the XPBBLK bits.
			If XMCM = 11b(for symmetric transmission and reception), assign 16 channels to receive partition A with the RPABLK bits. Assign 16 channels to receive partition B with the RPBBLK bits.
			You control the channels with the appropriate transmit channel enable registers:
			XCERA: Channels in partition A
			XCERB: Channels in partition B
		1	8-partition mode. All partitions (A through H) are used. You can control up to 128 channels in the transmit multichannel selection mode selected with the XMCM bits.
			You control the channels with the appropriate transmit channel enable registers:
			XCERA: Channels 0 through 15
			XCERB: Channels 16 through 31
			XCERC: Channels 32 through 47
			XCERD: Channels 48 through 63
			XCERE: Channels 64 through 79
			XCERF: Channels 80 through 95
			XCERG: Channels 96 through 111
			XCERH: Channels 112 through 127

Table 15-85. Multichannel Control 2 Register (MCR2) Field Descriptions (continued)

Bit	Field	Value	Description
8-7	XPBBLK	0-3h	Transmit partition B block bits
			XPBBLK is only applicable if channels can be individually disabled/enabled and masked/unmasked (XMCM is nonzero) and the 2-partition mode is selected (XMCME = 0). Under these conditions, the McBSP transmitter can transmit or withhold data in any of the 32 channels that are assigned to partitions A and B of the transmitter.
			The 128 transmit channels of the McBSP are divided equally among 8 blocks (0 through 7). When XPBBLK is applicable, use XPBBLK to assign one of the odd-numbered blocks (1, 3, 5, or 7) to partition B, as shown in the following table. Use the XPABLK bit to assign one of the even-numbered blocks (0, 2, 4, or 6) to partition A.
			If you want to use more than 32 channels, you can change block assignments dynamically. You can assign a new block to one partition while the transmitter is handling activity in the other partition. For example, while the block in partition A is active, you can change which block is assigned to partition B. The XCBLK bits are regularly updated to indicate which block is active.
			When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive block bits (RPABLK and RPBBLK) rather than the transmit block bits (XPABLK and XPBBLK).
		0	Block 1: channels 16 through 31
		1h	Block 3: channels 48 through 63
		2h	Block 5: channels 80 through 95
		3h	Block 7: channels 112 through 127
6-5	XPABLK		Transmit partition A block bits. XPABLK is only applicable if channels can be individually disabled/enabled and masked/unmasked (XMCM is nonzero) and the 2-partition mode is selected (XMCME = 0). Under these conditions, the McBSP transmitter can transmit or withhold data in any of the 32 channels that are assigned to partitions A and B of the transmitter. See the description for XPBBLK (bits 8-7) for more information about assigning blocks to partitions A and B.
		0	Block 0: channels 0 through 15
		1h	Block 2: channels 32 through 47
		2h	Block 4: channels 64 through 79
		3h	Block 6: channels 96 through 111
4-2	XCBLK		Transmit current block indicator. XCBLK indicates which block of 16 channels is involved in the current McBSP transmission:
		0	Block 0: channels 0 through 15
		1h	Block 1: channels 16 through 31
		2h	Block 2: channels 32 through 47
		3h	Block 3: channels 48 through 63
		4h	Block 4: channels 64 through 79
		5h	Block 5: channels 80 through 95
		6h	Block 6: channels 96 through 111
		7h	Block 7: channels 112 through 127
1-0	XMCM	0-3h	Transmit multichannel selection mode bits. XMCM determines whether all channels or only selected channels are enabled and unmasked for transmission. For more details on how the channels are affected, see Section 15.6.7 *Transmit Multichannel Selection Modes*.
		0	No transmit multichannel selection mode is on. All channels are enabled and unmasked. No channels can be disabled or masked.
		1h	All channels are disabled unless they are selected in the appropriate transmit channel enable registers (XCERs). If enabled, a channel in this mode is also unmasked.
			The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.
		2h	All channels are enabled, but they are masked unless they are selected in the appropriate transmit channel enable registers (XCERs).
			The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.
		3h	This mode is used for symmetric transmission and reception.
			All channels are disabled for transmission unless they are enabled for reception in the appropriate receive channel enable registers (RCERs). Once enabled, they are masked unless they are also selected in the appropriate transmit channel enable registers (XCERs).
			The XMCME bit determines whether 32 channels or 128 channels are selectable in RCERs and XCERs.

15.12.9 Pin Control Register (PCR)

Each McBSP has one pin control register (PCR). Table 15-86 describes the bits of PCR. This register enables you to:

- Choose a frame-synchronization mode for the transmitter (FSXM) and for the receiver (FSRM)
- Choose a clock mode for transmitter (CLKXM) and for the receiver (CLKRM)
- Select the input clock source for the sample rate generator (SCLKME, in conjunction with the CLKSM bit of SRGR2)
- Choose whether frame-synchronization signals are active low or active high (FSXP for transmission, FSRP for reception)
- Specify whether data is sampled on the falling edge or the rising edge of the clock signals (CLKXP for transmission, CLKRP for reception)

The pin control register (PCR) is shown in Figure 15-77 and described in Table 15-86.

Figure 15-77. Pin Control Register (PCR)

15–12	11	10	9	8
Reserved	FSXM	FSRM	CLKXM	CLKRM
R-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6–4	3	2	1	0
SCLKME	Reserved	FSXP	FSRP	CLKXP	CLKRP
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; -*n* = value after reset

Table 15-86. Pin Control Register (PCR) Field Descriptions

Bit	Field	Value	Description
15:12	Reserved	0	Reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.
11	FSXM		Transmit frame-synchronization mode bit. FSXM determines whether transmit frame-synchronization pulses are supplied externally or internally. The polarity of the signal on the FSX pin is determined by the FSXP bit.
		0	Transmit frame synchronization is supplied by an external source via the FSX pin.
		1	Transmit frame synchronization is generated internally by the Sample Rate generator, as determined by the FSGM bit of SRGR2.
10	FSRM		Receive frame-synchronization mode bit. FSRM determines whether receive frame-synchronization pulses are supplied externally or internally. The polarity of the signal on the FSR pin is determined by the FSRP bit.
		0	Receive frame synchronization is supplied by an external source via the FSR pin.
		1	Receive frame synchronization is supplied by the sample rate generator. FSR is an output pin reflecting internal FSR, except when GSYNC = 1 in SRGR2.

Table 15-86. Pin Control Register (PCR) Field Descriptions (continued)

Bit	Field	Value	Description
9	CLKXM		Transmit clock mode bit. CLKXM determines whether the source for the transmit clock is external or internal, and whether the MCLKX pin is an input or an output. The polarity of the signal on the MCLKX pin is determined by the CLKXP bit.
			In the clock stop mode (CLKSTP = 10b or 11b), the McBSP can act as a master or as a slave in the SPI protocol. If the McBSP is a master, make sure that CLKX is an output. If the McBSP is a slave, make sure that CLKX is an input.
			Not in clock stop mode (CLKSTP = 00b or 01b):
		0	The transmitter gets its clock signal from an external source via the MCLKX pin.
		1	Internal CLKX is driven by the sample rate generator of the McBSP. The MCLKX pin is an output pin that reflects internal CLKX.
			In clock stop mode (CLKSTP = 10b or 11b):
		0	The McBSP is a slave in the SPI protocol. The internal transmit clock (CLKX) is driven by the SPI master via the MCLKX pin. The internal receive clock (MCLKR) is driven internally by CLKX, so that both the transmitter and the receiver are controlled by the external master clock.
		1	The McBSP is a master in the SPI protocol. The sample rate generator drives the internal transmit clock (CLKX). Internal CLKX is reflected on the MCLKX pin to drive the shift clock of the SPI-compliant slaves in the system. Internal CLKX also drives the internal receive clock (MCLKR), so that both the transmitter and the receiver are controlled by the internal master clock.
8	CLKRM		Receive clock mode bit. The role of CLKRM and the resulting effect on the MCLKR pin depend on whether the McBSP is in the digital loopback mode (DLB = 1).
			The polarity of the signal on the MCLKR pin is determined by the CLKRP bit.
			Not in digital loopback mode (DLB = 0):
		0	The MCLKR pin is an input pin that supplies the internal receive clock (MCLKR).
		1	Internal MCLKR is driven by the sample rate generator of the McBSP. The MCLKR pin is an output pin that reflects internal MCLKR.
			In digital loopback mode (DLB = 1):
		0	The MCLKR pin is in the high impedance state. The internal receive clock (MCLKR) is driven by the internal transmit clock (CLKX). CLKX is derived according to the CLKXM bit.
		1	Internal MCLKR is driven by internal CLKX. The MCLKR pin is an output pin that reflects internal MCLKR. CLKX is derived according to the CLKXM bit.
7	SCLKME		Sample rate generator input clock mode bit. The sample rate generator can produce a clock signal, CLKG. The frequency of CLKG is:
			CLKG freq. = (Input clock frequency) / (CLKGDV + 1)
			SCLKME is used in conjunction with the CLKSM bit to select the input clock.
			SCLKME / **CLKSM** / **Input Clock For Sample Rate Generator**
			0 / 0 / Reserved
			0 / 1 / LSPCLK
			The input clock for the sample rate generator is taken from the MCLKR pin or from the MCLKX pin, depending on the value of the CLKSM bit of SRGR2:
			SCLKME / **CLKSM** / **Input Clock For Sample Rate Generator**
			1 / 0 / Signal on MCLKR pin
			1 / 1 / Signal on MCLKX pin
6-4	Reserved		Reserved
3	FSXP		Transmit frame-synchronization polarity bit. FSXP determines the polarity of FSX as seen on the FSX pin.
		0	Transmit frame-synchronization pulses are active high.
		1	Transmit frame-synchronization pulses are active low.
2	FSRP		Receive frame-synchronization polarity bit. FSRP determines the polarity of FSR as seen on the FSR pin.
		0	Receive frame-synchronization pulses are active high.
		1	Receive frame-synchronization pulses are active low.

Table 15-86. Pin Control Register (PCR) Field Descriptions (continued)

Bit	Field	Value	Description
1	CLKXP		Transmit clock polarity bit. CLKXP determines the polarity of CLKX as seen on the MCLKX pin.
		0	Transmit data is sampled on the rising edge of CLKX.
		1	Transmit data is sampled on the falling edge of CLKX.
0	CLKRP		Receive clock polarity bit. CLKRP determines the polarity of CLKR as seen on the MCLKR pin.
		0	Receive data is sampled on the falling edge of MCLKR.
		1	Receive data is sampled on the rising edge of MCLKR.

Table 15-87. Pin Configuration

Pin	Selected as Output When …	Selected as Input When …
CLKX	CLKXM = 1	CLKXM = 0
FSX	FSXM = 1	FSXM = 0
CLKR	CLKRM = 1	CLKRM = 0
FSR	FSRM = 1	FSRM = 0

15.12.10 *Receive Channel Enable Registers (RCERA, RCERB, RCERC, RCERD, RCERE, RCERF, RCERG, RCERH)*

Each McBSP has eight receive channel enable registers of the format shown in Figure 15-78. There is one enable register for each of the receive partitions: A, B, C, D, E, F, G, and H. Table 15-88 provides a summary description that applies to any bit x of a receive channel enable register.

These memory-mapped registers are only used when the receiver is configured to allow individual enabling and disabling of the channels (RMCM = 1). For more details about the way these registers are used, see Section 15.12.10.1, *RCERs Used in the Receive Multichannel Selection Mode*.

The receive channel enable registers (RCERA...RCERH) are shown in Figure 15-78 and described in Table 15-88.

Figure 15-78. Receive Channel Enable Registers (RCERA...RCERH)

15	14	13	12	11	10	9	8
RCE15	RCE14	RCE13	RCE12	RCE11	RCE10	RCE9	RCE8
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
RCE7	RCE6	RCE5	RCE4	RCE3	RCE2	RCE1	RCE0
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 15-88. Receive Channel Enable Registers (RCERA...RCERH) Field Descriptions

Bit	Field	Value	Description
15-0	RCEx		Receive channel enable bit.
			For receive multichannel selection mode (RMCM = 1):
		0	Disable the channel that is mapped to RCEx.
		1	Enable the channel that is mapped to RCEx.

15.12.10.1 RCERs Used in the Receive Multichannel Selection Mode

For multichannel selection operation, the assignment of channels to the RCERs depends on whether 32 or 128 channels are individually selectable, as defined by the RMCME bit. For each of these two cases, Table 15-89 shows which block of channels is assigned to each of the RCERs used. For each RCER, the table shows which channel is assigned to each of the bits.

Table 15-89. Use of the Receive Channel Enable Registers

Number of Selectable Channels	Block Assignments		Channel Assignments	
	RCERx	Block Assigned	Bit in RCERx	Channel Assigned
32 (RMCME = 0)	RCERA	Channels n to (n + 15)	RCE0	Channel n
			RCE1	Channel (n + 1)
			RCE2	Channel (n + 2)
			:	:
		The block of channels is chosen with the RPABLK bits.	RCE15	Channel (n + 15)
	RCERB	Channels m to (m + 15)	RCE0	Channel m
			RCE1	Channel (m + 1)
			RCE2	Channel (m + 2)
			:	:
		The block of channels is chosen with the RPBBLK bits.	RCE15	Channel (m + 15)
128 (RMCME = 1)	RCERA	Block 0	RCE0	Channel 0
			RCE1	Channel 1
			RCE2	Channel 2
			:	:
			RCE15	Channel 15
	RCERB	Block 1	RCE0	Channel 16
			RCE1	Channel 17
			RCE2	Channel 18
			:	:
			RCE15	Channel 31
	RCERC	Block 2	RCE0	Channel 32
			RCE1	Channel 33
			RCE2	Channel 34
			:	:
			RCE15	Channel 47
	RCERD	Block 3	RCE0	Channel 48
			RCE1	Channel 49
			RCE2	Channel 50
			:	:
			RCE15	Channel 63
	RCERE	Block 4	RCE0	Channel 64
			RCE1	Channel 65
			RCE2	Channel 66
			:	:
			RCE15	Channel 79
	RCERF	Block 5	RCE0	Channel 80
			RCE1	Channel 81
			RCE2	Channel 82
			:	:
			RCE15	Channel 95
	RCERG	Block 6	RCE0	Channel 96
			RCE1	Channel 97
			RCE2	Channel 98
			:	:
			RCE15	Channel 111

Table 15-89. Use of the Receive Channel Enable Registers (continued)

Number of Selectable Channels	Block Assignments		Channel Assignments	
	RCERx	Block Assigned	Bit in RCERx	Channel Assigned
	RCERH	Block 7	RCE0	Channel 112
			RCE1	Channel 113
			RCE2	Channel 114
			:	:
			RCE15	Channel 127

15.12.11 Transmit Channel Enable Registers (XCERA, XCERB, XCERC, XCERD, XCERE, XCERF, XCERG, XCERH)

Each McBSP has eight transmit channel enable registers of the form shown in Figure 15-79. There is one for each of the transmit partitions: A, B, C, D, E, F, G, and H. Table 15-90 provides a summary description that applies to each bit XCEx of a transmit channel enable register.

The XCERs are only used when the transmitter is configured to allow individual disabling/enabling and masking/unmasking of the channels (XMCM is nonzero).

The transmit channel enable registers (XCERA...XCERH) are shown in Figure 15-79 and described in Table 15-90.

Figure 15-79. Transmit Channel Enable Registers (XCERA...XCERH)

15	14	13	12	11	10	9	8
XCE15	XCE14	XCE13	XCE12	XCE11	XCE10	XCE9	XCE8
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
XCE7	XCE6	XCE5	XCE4	XCE3	XCE2	XCE1	XCE0
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 15-90. Transmit Channel Enable Registers (XCERA...XCERH) Field Descriptions

Bit	Field	Value	Description
15-0	XCEx		Transmit channel enable bit. The role of this bit depends on which transmit multichannel selection mode is selected with the XMCM bits.
			For multichannel selection when XMCM = 01b (all channels disabled unless selected):
		0	Disable and mask the channel that is mapped to XCEx.
		1	Enable and unmask the channel that is mapped to XCEx.
			For multichannel selection when XMCM = 10b (all channels enabled but masked unless selected):
		0	Mask the channel that is mapped to XCEx.
		1	Unmask the channel that is mapped to XCEx.
			For multichannel selection when XMCM = 11b (all channels masked unless selected):
		0	Mask the channel that is mapped to XCEx. Even if the channel is enabled by the corresponding receive channel enable bit, this channel's data cannot appear on the DX pin.
		1	Unmask the channel that is mapped to XCEx. If the channel is also enabled by the corresponding receive channel enable bit, full transmission can occur.

15.12.11.1 XCERs Used in a Transmit Multichannel Selection Mode

For multichannel selection operation, the assignment of channels to the XCERs depends on whether 32 or 128 channels are individually selectable, as defined by the XMCME bit. These two cases are shown in Table 15-91. The table shows which block of channels is assigned to each XCER that is used. For each XCER, the table shows which channel is assigned to each of the bits.

NOTE: When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive channel enable registers (RCERs) to enable channels and uses the XCERs to unmask channels for transmission.

Table 15-91. Use of the Transmit Channel Enable Registers

Number of Selectable Channels	Block Assignments		Channel Assignments	
	XCERx	**Block Assigned**	**Bit in XCERx**	**Channel Assigned**
32 (XMCME = 0)	XCERA	Channels n to (n + 15)	XCE0	Channel n
			XCE1	Channel (n + 1)
			XCE2	Channel (n + 2)
			:	:
		When XMCM = 01b or 10b, the block of channels is chosen with the XPABLK bits. When XMCM = 11b, the block is chosen with the RPABLK bits.	XCE15	Channel (n + 15)
	XCERB	Channels m to (m + 15)	XCE0	Channel m
			XCE1	Channel (m + 1)
			XCE2	Channel (m + 2)
			:	:
		When XMCM = 01b or 10b, the block of channels is chosen with the XPBBLK bits. When XMCM = 11b, the block is chosen with the RPBBLK bits.	XCE15	Channel (m + 15)
128 (XMCME = 1)	XCERA	Block 0	XCE0	Channel 0
			XCE1	Channel 1
			XCE2	Channel 2
			:	:
			XCE15	Channel 15
	XCERB	Block 1	XCE0	Channel 16
			XCE1	Channel 17
			XCE2	Channel 18
			:	:
			XCE15	Channel 31
	XCERC	Block 2	XCE0	Channel 32
			XCE1	Channel 33
			XCE2	Channel 34
			:	:
			XCE15	Channel 47
	XCERD	Block 3	XCE0	Channel 48
			XCE1	Channel 49
			XCE2	Channel 50
			:	:
			XCE15	Channel 63

Table 15-91. Use of the Transmit Channel Enable Registers (continued)

Number of Selectable Channels	Block Assignments		Channel Assignments	
	XCERx	Block Assigned	Bit in XCERx	Channel Assigned
	XCERE	Block 4	XCE0	Channel 64
			XCE1	Channel 65
			XCE2	Channel 66
			:	:
			XCE15	Channel 79
	XCERF	Block 5	XCE0	Channel 80
			XCE1	Channel 81
			XCE2	Channel 82
			:	:
			XCE15	Channel 95
	XCERG	Block 6	XCE0	Channel 96
			XCE1	Channel 97
			XCE2	Channel 98
			:	:
			XCE15	Channel 111
	XCERH	Block 7	XCE0	Channel 112
			XCE1	Channel 113
			XCE2	Channel 114
			:	:
			XCE15	Channel 127

15.12.12 Interrupt Generation

McBSP registers can be programmed to receive and transmit data through DRR2/DRR1 and DXR2/DXR1 registers, respectively. The CPU can directly access these registers to move data from memory to these registers. Interrupt signals will be based on these register pair contents and its related flags.MRINT/MXINT will generate CPU interrupts for receive and transmit conditions.

15.12.12.1 McBSP Receive Interrupt Generation

In the McBSP module, data receive and error conditions generate two sets of interrupt signals. One set is used for the CPU and the other set is for DMA.

Figure 15-80. Receive Interrupt Generation

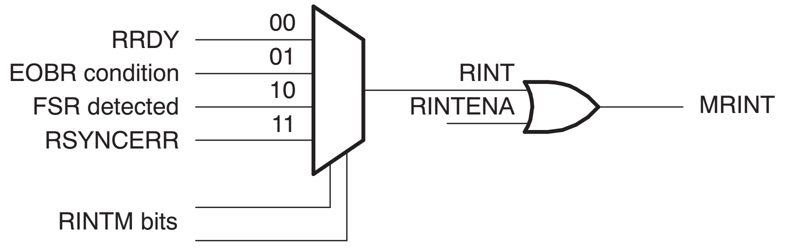


Table 15-92. Receive Interrupt Sources and Signals

McBSP Interrupt Signal	Interrupt Flags	Interrupt Enables in SPCR1	Interrupt Enables	Type of Interrupt	Interrupt Line
		RINTM Bits			

Table 15-92. Receive Interrupt Sources and Signals (continued)

McBSP Interrupt Signal	Interrupt Flags	Interrupt Enables in SPCR1	Interrupt Enables	Type of Interrupt	Interrupt Line
RINT	RRDY	0	RINTENA	Every word receive	MRINT
	EOBR	1	RINTENA	Every 16 channel block boundary	
	FSR	10	RINTENA	On every FSR	
	RSYNCERR	11	RINTENA	Frame sync error	

NOTE: Since X/RINT, X/REVTA, and X/RXFFINT share the same CPU interrupt, it is recommended that all applications use one of the above selections for interrupt generation. If multiple interrupt enables are selected at the same time, there is a likelihood of interrupts being masked or not recognized.

15.12.12.2 McBSP Transmit Interrupt Generation

McBSP module data transmit and error conditions generate two sets of interrupt signals. One set is used for the CPU and the other set is for DMA.

Figure 15-81. Transmit Interrupt Generation

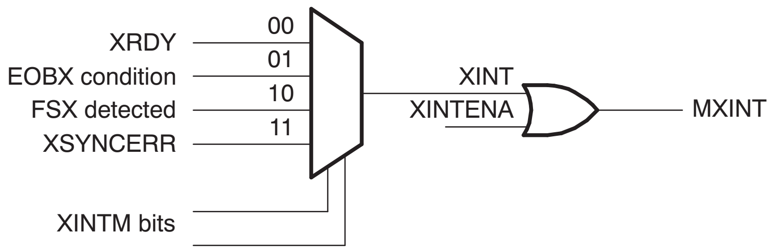


Table 15-93. Transmit Interrupt Sources and Signals

McBSP Interrupt Signal	Interrupt Flags	Interrupt Enables in SPCR1	Interrupt Enables	Type of Interrupt	Interrupt Line
		XINTM Bits			
XINT	XRDY	0	XINTENA	Every word receive	MXINT
	EOBX	1	XINTENA	Every 16-channel block boundary	
	FSX	10	XINTENA	On every FSX	
	XSYNCERR	11	XINTENA	Frame sync error	

15.12.12.3 Error Flags

The McBSP has several error flags both on receive and transmit channel. Table 15-94 explains the error flags and their meaning.

Table 15-94. Error Flags

Error Flags	Function
RFULL	Indicates DRR2/DRR1 are not read and RXR register is overwritten
RSYNCERR	Indicates unexpected frame-sync condition, current data reception will abort and restart. Use RINTM bit 11 for interrupt generation on this condition.
XSYNCERR	Indicates unexpected frame-sync condition, current data transmission will abort and restart. Use XINTM bit 11 for interrupt generation on this condition.

15.12.12.4 McBSP Interrupt Enable Register

Figure 15-82. McBSP Interrupt Enable Register (MFFINT)

15–8
Reserved
R-0

7–3	2	1	0
Reserved	RINT ENA	Reserved	XINT ENA
R-0	R/W-0	R-0	R/W-0

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

Table 15-95. McBSP Interrupt Enable Register (MFFINT) Field Descriptions

Bit	Field	Value	Description
15:3	Reserved		Reserved
2	RINT ENA		Enable for Receive Interrupt
		0	Receive interrupt on RRDY is disabled.
		1	Receive interrupt on RRDY is enabled.
1	Reserved		
0	XINT ENA		Enable for transmit Interrupt
		0	Transmit interrupt on XRDY is disabled.
		1	Transmit interrupt on XRDY is enabled.

15.12.12.5 McBSP Modes

McBSP, in its normal mode, communicates with various types of Codecs with variable word size. Apart from this mode, the McBSP uses time-division multiplexed (TDM) data stream while communicating with other McBSPs or serial devices. The multichannel mode provides flexibility while transmitting/receiving selected channels or all the channels in a TDM stream.

Table 15-96 provides a quick reference to McBSP mode selection.

Table 15-96. McBSP Mode Selection

		Register Bits Used for Mode Selection				
		MCR1 bit 9,0		MCR2 bit 9,1,0		
No.	McBSP Word Size	RMCME	RMCM	XMCME	XMCM	Mode and Function Description
						Normal Mode
1	8/12/16/20/24/32 bit words	0	0	0	0	All types of Codec interface will use this selection
						Multichannel Mode
2	8-bit words					2 Partition or 32-channel Mode
		0	1	0	1	All channels are disabled,unless selected in X/RCERA/B
		0	1	0	10	All channels are enabled,but masked unless selected in X/RCERA/B
		0	1	0	11	Symmetric transmit, receive
						8 Partition or 128 Channel Mode Transmit/ Receive
						Channels selected by X/RCERA to X/RCERH bits
						Multichannel Mode is ON
		1	1	1	1	All channels are disabled,unless selected in

Table 15-96. McBSP Mode Selection (continued)

		Register Bits Used for Mode Selection				
		MCR1 bit 9,0		MCR2 bit 9,1,0		
No.	McBSP Word Size	RMCME	RMCM	XMCME	XMCM	Mode and Function Description
						XCERs
		1	1	1	10	All channels are enabled,but masked unless selected in XCERs
		1	1	1	11	Symmetric transmit, receive
						Continuous Mode - Transmit
		1	0	1	0	**Multi-Channel Mode is OFF**
						All 128 channels are active and enabled

Enhanced Controller Area Network (eCAN)

The enhanced Controller Area Network (eCAN) module implemented in the C28x DSP is a full-CAN controller and is compatible with the CAN 2.0B standard (active). It uses established protocol to communicate serially with other controllers in electrically noisy environments. With 32 fully configurable mailboxes and time−stamping feature, the eCAN module provides a versatile and robust serial communication interface.

The eCAN module described in this reference guide is a Type 2 eCAN. Refer to theTMS320x28xx, 28xxx DSP Peripheral Reference Guide (SPRU566) for a list of other devices with a eCAN module of the same type, to determine the differences between types, and for a list of device-specific differences within a type. For a given CAN module, the same address space is used for the module registers in all 28xx /28xxx devices.

Topic **Page**

16.1 CAN Overview

Figure 16-1 shows the major blocks of the eCAN and the interface circuits.

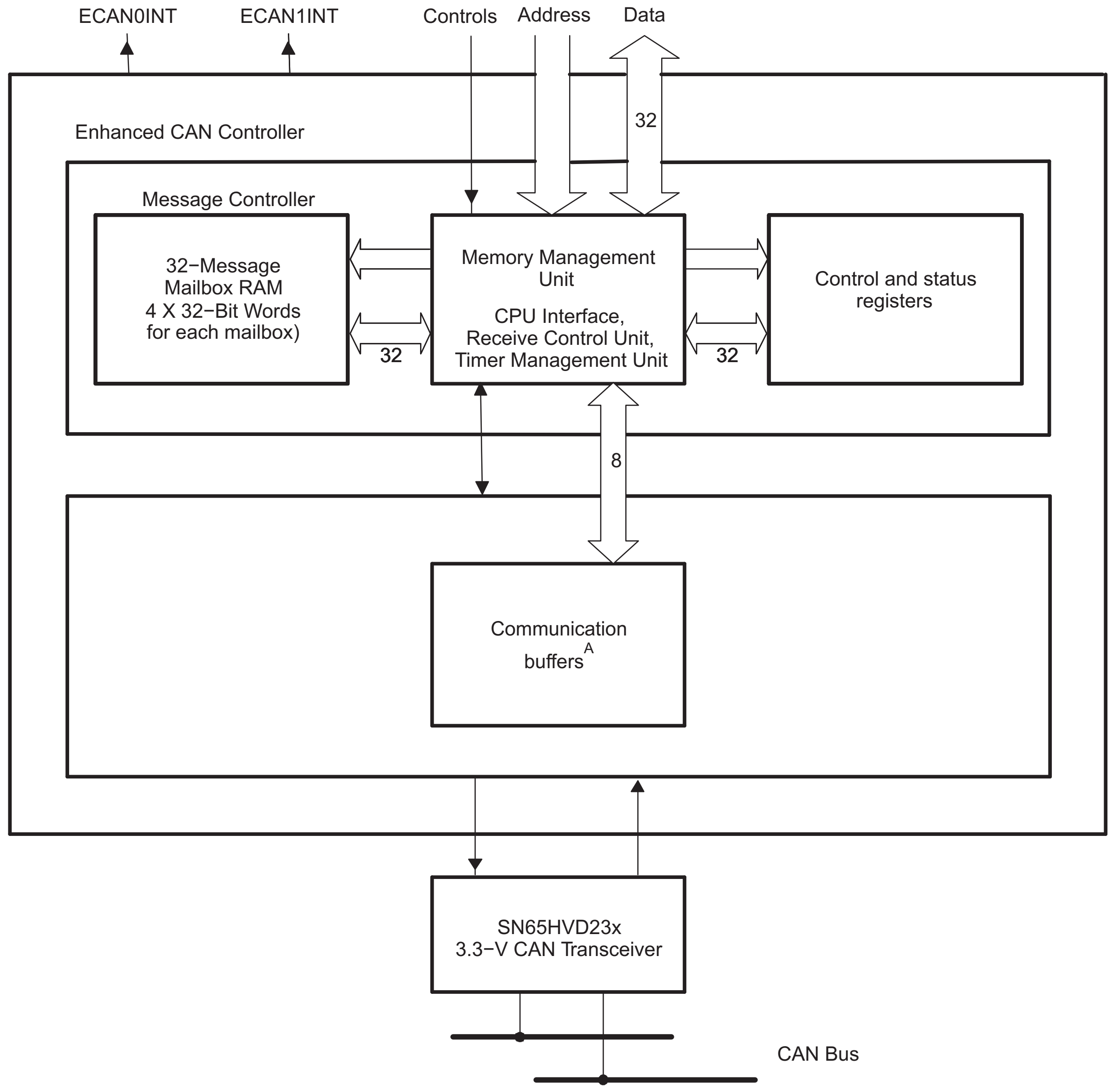
16.1.1 Features

The eCAN module has the following features:

- Fully compliant with CAN protocol, version 2.0B
- Supports data rates up to 1 Mbps
- Thirty-two mailboxes, each with the following properties:
  - Configurable as receive or transmit
  - Configurable with standard or extended identifier
  - Has a programmable acceptance filter mask
  - Supports data and remote frame
  - Supports 0 to 8 bytes of data
  - Uses a 32-bit time stamp on received and transmitted message
  - Protects against reception of new message
  - Allows dynamically programmable priority of transmit message
  - Employs a programmable interrupt scheme with two interrupt levels
  - Employs a programmable interrupt on transmission or reception time-out
- Low−power mode
- Programmable wake−up on bus activity
- Automatic reply to a remote request message
- Automatic retransmission of a frame in case of loss of arbitration or error
- 32-bit time-stamp counter synchronized by a specific message (communication in conjunction with mailbox 16)
- Self−test mode
  - Operates in a loopback mode receiving its own message. A “dummy” acknowledge is provided, thereby eliminating the need for another node to provide the acknowledge bit.

16.1.2 Block Diagram

Figure 16-1. eCAN Block Diagram and Interface Circuit


ECAN0INT
ECAN1INT
Controls
Address
Data
32
Enhanced CAN Controller
Message Controller
32−Message Mailbox RAM 4 X 32−Bit Words for each mailbox)
32
Memory Management Unit
CPU Interface, Receive Control Unit, Timer Management Unit
32
Control and status registers
8
Communication buffers A
SN65HVD23x 3.3−V CAN Transceiver
CAN Bus

A The communication buffers are transparent to the user and are not accessible by user code.

16.1.3 eCAN Compatibility With Other TI CAN Modules

The eCAN module is identical to the “High-end CAN Controller (HECC)” used in the TMS470™ series microcontrollers from Texas Instruments with some minor changes. The eCAN module features several enhancements (such as increased number of mailboxes with individual acceptance masks, time stamping, etc.) over the CAN module featured in 240x™ series of DSPs. For this reason, code written for 240x CAN modules cannot be directly ported to eCAN. However, eCAN follows the same register bit-layout structure and bit functionality as that of 240x CAN (for registers that exist in both devices) i.e., many registers and bits perform exactly identical functions across these two platforms. This makes code migration a relatively easy task, more so with code written in C language.

16.2 The CAN Network and Module

The controller area network (CAN) uses a serial multimaster communication protocol that efficiently supports distributed real-time control, with a very high level of security, and a communication rate of up to 1 Mbps. The CAN bus is ideal for applications operating in noisy and harsh environments, such as in the automotive and other industrial fields that require reliable communication.

Prioritized messages of up to eight bytes in data length can be sent on a multimaster serial bus using an arbitration protocol and an error-detection mechanism for a high level of data integrity.

16.2.1 CAN Protocol Overview

The CAN protocol supports four different frame types for communication:

- Data frames that carry data from a transmitter node to the receiver nodes
- Remote frames that are transmitted by a node to request the transmission of a data frame with the same identifier
- Error frames that are transmitted by any node on a bus-error detection
- Overload frames that provide an extra delay between the preceding and the succeeding data frames or remote frames.

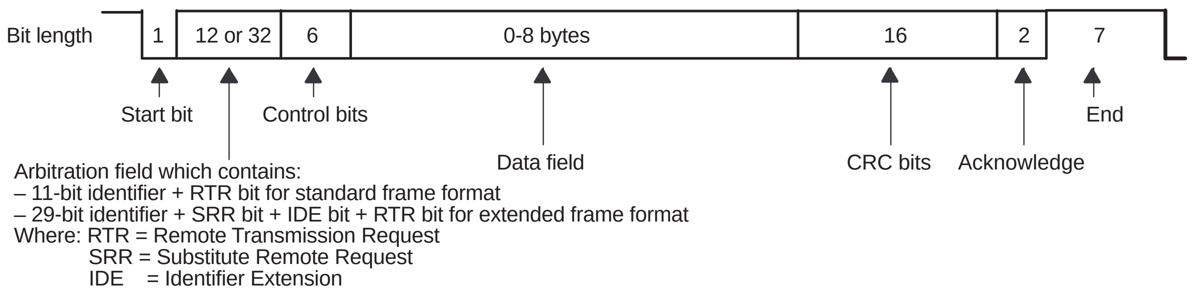
In addition, CAN specification version 2.0B defines two different formats that differ in the length of the identifier field: standard frames with an 11-bit identifier and extended frames with 29-bit identifier.

CAN standard data frames contain from 44 to 108 bits and CAN extended data frames contain 64 to 128 bits. Furthermore, up to 23 stuff bits can be inserted in a standard data frame, and up to 28 stuff bits in an extended data frame, depending on the data-stream coding. The overall maximum data frame length is then 131 bits for a standard frame and 156 bits for an extended frame.

The bit fields that make up standard/extended data frames, along with their position as shown in Figure 16-2 include the following:

- Start of frame
- Arbitration field containing the identifier and the type of message being sent
- Control field indicating the number of bytes being transmitted.
- Up to 8 bytes of data
- Cyclic redundancy check (CRC)
- Acknowledgment
- End-of-frame bits

Figure 16-2. CAN Data Frame

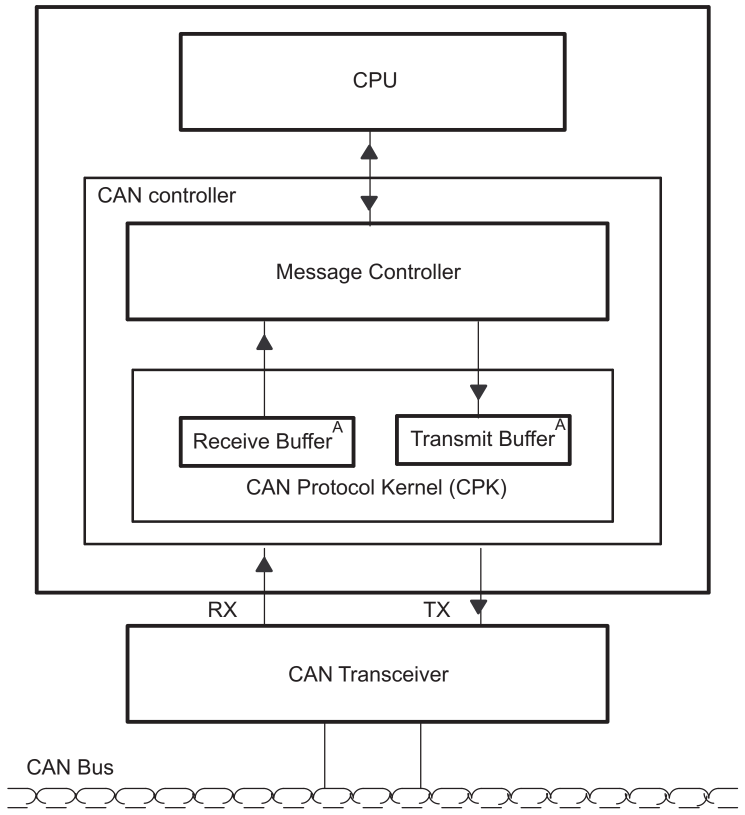

Bit length
1
12 or 32
6
0-8 bytes
16
2
7
Start bit
Control bits
End
Data field
CRC bits
Acknowledge
Arbitration field which contains:
– 11-bit identifier + RTR bit for standard frame format
– 29-bit identifier + SRR bit + IDE bit + RTR bit for extended frame format
Where: RTR = Remote Transmission Request
SRR = Substitute Remote Request
IDE = Identifier Extension

Note: Unless otherwise noted, numbers are amount of bits in field.

The eCAN controller provides the CPU with full functionality of the CAN protocol, version 2.0B. The CAN controller minimizes the CPU's load in communication overhead and enhances the CAN standard by providing additional features.

The architecture of eCAN module, shown in Figure 16-3, is composed of a CAN protocol kernel (CPK) and a message controller.

Figure 16-3. Architecture of the eCAN Module



A The receive and transmit buffers are transparent to the user and are not accessible by user code.

Two functions of the CPK are to decode all messages received on the CAN bus according to the CAN protocol and to transfer these messages into a receive buffer. Another CPK function is to transmit messages on the CAN bus according to the CAN protocol.

The message controller of a CAN controller is responsible for determining if any message received by the CPK must be preserved for the CPU use or be discarded. At the initialization phase, the CPU specifies to the message controller all message identifiers used by the application. The message controller is also responsible for sending the next message to transmit to the CPK according to the message's priority.

16.3 eCAN Controller Overview

The eCAN is a CAN controller with an internal 32-bit architecture.

The eCAN module consists of:

- The CAN protocol kernel (CPK)
- The message controller comprising:
  – The memory management unit (MMU), including the CPU interface and the receive control unit (acceptance filtering), and the timer management unit
  – Mailbox RAM enabling the storage of 32 messages
  – Control and status registers

After the reception of a valid message by the CPK, the receive control unit of the message controller determines if the received message must be stored into one of the 32 message objects of the mailbox RAM. The receive control unit checks the state, the identifier, and the mask of all message objects to determine the appropriate mailbox location. The received message is stored into the first mailbox passing the acceptance filtering. If the receive control unit could not find any mailbox to store the received message, the message is discarded.

A message is composed of an 11- or 29-bit identifier, a control field, and up to 8 bytes of data.

When a message must be transmitted, the message controller transfers the message into the transmit buffer of the CPK in order to start the message transmission at the next bus-idle state. When more than one message must be transmitted, the message with the highest priority that is ready to be transmitted is transferred into the CPK by the message controller. If two mailboxes have the same priority, then the mailbox with the higher number is transmitted first.

The timer management unit comprises a time-stamp counter and apposes a time stamp to all messages received or transmitted. It generates an interrupt when a message has not been received or transmitted during an allowed period of time (time-out). The time-stamping feature is available in eCAN mode only.

To initiate a data transfer, the transmission request bit (TRS.n) has to be set in the corresponding control register. The entire transmission procedure and possible error handling are then performed without any CPU involvement. If a mailbox has been configured to receive messages, the CPU easily reads its data registers using CPU read instructions. The mailbox may be configured to interrupt the CPU after every successful message transmission or reception.

16.3.1 Standard CAN Controller (SCC) Mode

The SCC Mode is a reduced functionality mode of the eCAN. Only 16 mailboxes (0 through 15) are available in this mode. The time stamping feature is not available and the number of acceptance masks available is reduced. This mode is selected by default. The SCC mode or the full featured eCAN mode is selected using the SCB bit (CANMC.13).

16.3.2 Memory Map

The eCAN module has two different address segments mapped in the memory. The first segment is used to access the control registers, the status registers, the acceptance masks, the time stamp, and the time-out of the message objects. The access to the control and status registers is limited to 32-bit wide accesses. The local acceptance masks, the time stamp registers, and the time-out registers can be accessed 8-bit, 16-bit and 32-bit wide. The second address segment is used to access the mailboxes. This memory range can be accessed 8-bit, 16-bit and 32-bit wide. Each of these two memory blocks, shown in Figure 16-4, uses 512 bytes of address space.

The message storage is implemented by a RAM that can be addressed by the CAN controller or the CPU. The CPU controls the CAN controller by modifying the various mailboxes in the RAM or the additional registers. The contents of the various storage elements are used to perform the functions of the acceptance filtering, message transmission, and interrupt handling.

The mailbox module in the eCAN provides 32 message mailboxes of 8-byte data length, a 29-bit identifier, and several control bits. Each mailbox can be configured as either transmit or receive. In the eCAN mode, each mailbox has its individual acceptance mask.

NOTE: LAMn, MOTSn and MOTOn registers and mailboxes not used in an application (disabled in the CANME register) may be used as general-purpose data memory by the CPU.

16.3.2.1 32-bit Access to Control and Status Registers

As indicated in Section 16.3.2, only 32-bit accesses are allowed to the Control and Status registers. 16-bit access to these registers could potentially corrupt the register contents or return false data. The DSP header files released by TI employs a shadow register structure that aids in 32-bit access. Following are a few examples of how the shadow register structure may be employed to perform 32-bit reads/writes:

Example 16-1. Modifying a bit in a register

```
 ECanaShadow.CANTIOC.all = ECanaRegs.CANTIOC.all; // Step 1 ECanaShadow.CANTIOC.bit.TXFUNC = 1; //
Step 2 ECanaRegs.CANTIOC.all = ECanaShadow.CANTIOC.all; // Step 3
```

Step 1: Perform a 32-bit read to copy the entire register to its shadow

Step 2: Modify the needed bit(s) in the shadow

Step 3: Perform a 32-bit write to copy the modified shadow to the original register.

NOTE: Some bits like TAn and RMPn are cleared by writing a 1 to it. Care should be taken not to clear bits inadvertently.

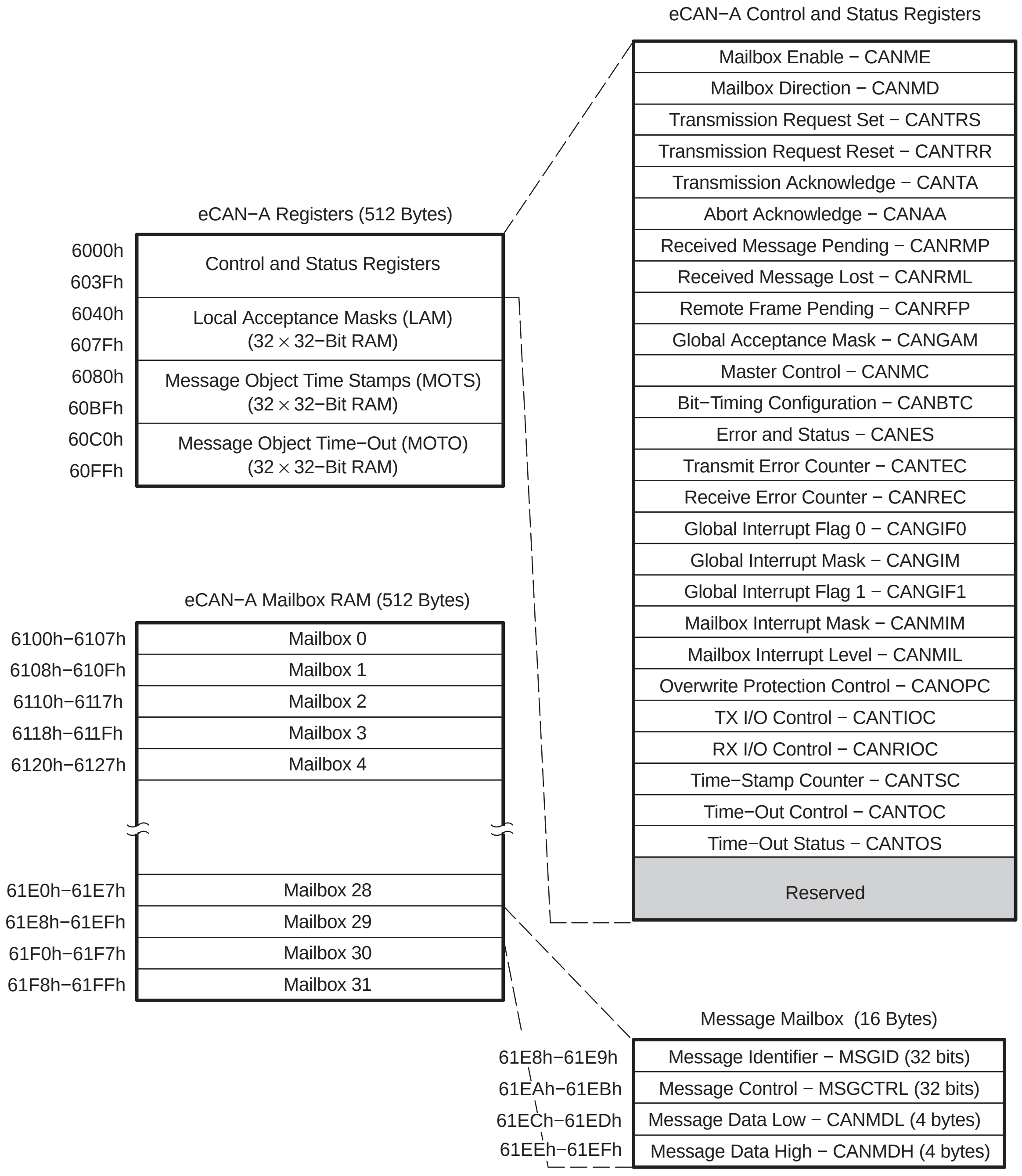
Example 16-2. Checking the value of a bit in a register

```
do { ECanaShadow.CANTA.all = ECanaRegs.CANTA.all; }while(ECanaShadow.CANTA.bit.TA25 == 0); // Wait
for TA5 bit to be set..
```

In the above example, the value of TA25 bit needs to be checked. This is done by first copying the entire CANTA register to its shadow (using a 32-bit read) and then checking the relevant bit, repeating this operation until that condition is satisfied. TA25 bit should NOT be checked with the following statement:

`while(ECanaRegs.CANTA.bit.TA25 == 0);`

Figure 16-4. eCAN-A Memory Map



eCAN−A Control and Status Registers
Mailbox Enable − CANME
Mailbox Direction − CANMD
Transmission Request Set − CANTRS
Transmission Request Reset − CANTRR
Transmission Acknowledge − CANTA
Abort Acknowledge − CANAA
Received Message Pending − CANRMP
Received Message Lost − CANRML
Remote Frame Pending − CANRFP
Global Acceptance Mask − CANGAM
Master Control − CANMC
Bit−Timing Configuration − CANBTC
Error and Status − CANES
Transmit Error Counter − CANTEC
Receive Error Counter − CANREC
Global Interrupt Flag 0 − CANGIF0
Global Interrupt Mask − CANGIM
Global Interrupt Flag 1 − CANGIF1
Mailbox Interrupt Mask − CANMIM
Mailbox Interrupt Level − CANMIL
Overwrite Protection Control − CANOPC
TX I/O Control − CANTIOC
RX I/O Control − CANRIOC
Time−Stamp Counter − CANTSC
Time−Out Control − CANTOC
Time−Out Status − CANTOS
Reserved
eCAN−A Registers (512 Bytes)
6000h
603Fh
Control and Status Registers
6040h
607Fh
Local Acceptance Masks (LAM) (32 × 32−Bit RAM)
6080h
60BFh
Message Object Time Stamps (MOTS) (32 × 32−Bit RAM)
60C0h
60FFh
Message Object Time−Out (MOTO) (32 × 32−Bit RAM)
eCAN−A Mailbox RAM (512 Bytes)
6100h−6107h Mailbox 0
6108h−610Fh Mailbox 1
6110h−6117h Mailbox 2
6118h−611Fh Mailbox 3
6120h−6127h Mailbox 4
61E0h−61E7h Mailbox 28
61E8h−61EFh Mailbox 29
61F0h−61F7h Mailbox 30
61F8h−61FFh Mailbox 31
Message Mailbox (16 Bytes)
61E8h−61E9h Message Identifier − MSGID (32 bits)
61EAh−61EBh Message Control − MSGCTRL (32 bits)
61ECh−61EDh Message Data Low − CANMDL (4 bytes)
61EEh−61EFh Message Data High − CANMDH (4 bytes)

16.3.3 eCAN Control and Status Registers

The eCAN registers listed below are used by the CPU to configure and control the CAN controller and the message objects.

Table 16-1. Register Map

REGISTER NAME	ECAN-A ADDRESS	SIZE (x32)	DESCRIPTION
CANME	0x6000	1	Mailbox enable
CANMD	0x6002	1	Mailbox direction
CANTRS	0x6004	1	Transmit request set
CANTRR	0x6006	1	Transmit request reset
CANTA	0x6008	1	Transmission acknowledge
CANAA	0x600A	1	Abort acknowledge
CANRMP	0x600C	1	Receive message pending
CANRML	0x600E	1	Receive message lost
CANRFP	0x6010	1	Remote frame pending
CANGAM	0x6012	1	Global acceptance mask
CANMC	0x6014	1	Master control
CANBTC	0x6016	1	Bit-timing configuration
CANES	0x6018	1	Error and status
CANTEC	0x601A	1	Transmit error counter
CANREC	0x601C	1	Receive error counter
CANGIF0	0x601E	1	Global interrupt flag 0
CANGIM	0x6020	1	Global interrupt mask
CANGIF1	0x6022	1	Global interrupt flag 1
CANMIM	0x6024	1	Mailbox interrupt mask
CANMIL	0x6026	1	Mailbox interrupt level
CANOPC	0x6028	1	Overwrite protection control
CANTIOC	0x602A	1	TX I/O control
CANRIOC	0x602C	1	RX I/O control
CANTSC	0x602E	1	Time stamp counter (Reserved in SCC mode)
CANTOC	0x6030	1	Time-out control (Reserved in SCC mode)
CANTOS	0x6032	1	Time-out status (Reserved in SCC mode)

NOTE: Only 32-bit accesses are allowed to the control and status registers. This restriction does not apply to the mailbox RAM area. See Section 16.3.2.1 for more information.

16.4 Message Objects

The eCAN module has 32 different message objects (mailboxes).

Each message object can be configured to either transmit or receive. Each message object has its individual acceptance mask.

A message object consists of a message mailbox with:

- The 29-bit message identifier
- The message control register
- 8 bytes of message data
- A 29-bit acceptance mask
- A 32-bit time stamp
- A 32-bit time-out value

Furthermore, corresponding control and status bits located in the registers allow control of the message objects.

16.5 Message Mailbox

The message mailboxes are the RAM area where the CAN messages are actually stored after they are received or before they are transmitted.

The CPU may use the RAM area of the message mailboxes that are not used for storing messages as normal memory.

Each mailbox contains:

- The message identifier
  - 29 bits for extended identifier
  - 11 bits for standard identifier
- The identifier extension bit, IDE (MSGID.31)
- The acceptance mask enable bit, AME (MSGID.30)
- The auto answer mode bit, AAM (MSGID.29)
- The transmit priority level, TPL (MSGCTRL.12-8)
- The remote transmission request bit, RTR (MSGCTRL.4)
- The data length code, DLC (MSGCTRL.3-0)
- Up to eight bytes for the data field

Each of the mailboxes can be configured as one of four message object types. Transmit and receive message objects are used for data exchange between one sender and multiple receivers (1 to n communication link), whereas request and reply message objects are used to set up a one-to-one communication link. Table 16-2 lists the mailbox RAM layout.

Table 16-2. eCAN-A Mailbox RAM Layout

Mailbox	MSGID MSGIDL-MSGIDH	MSGCTRL MSGCTRL-Rsvd	CANMDL CANMDL_L- CANMDL_H	CANMDH CANMDH_L- CANMDH_H
0	6100-6101h	6102-6103h	6104-6105h	6106-6107h
1	6108-6109h	610A-610Bh	610C-610Dh	610E-610Fh
2	6110 - 6111h	6112-6113h	6114-6115h	6116-6117h
3	6118-6119h	611A-611Bh	611C-611Dh	611E-611Fh
4	6120-6121h	6122-6123h	6124-6125h	6126-6127h
5	6128-6129h	612A-612Bh	612C-612Dh	612E-612Fh
6	6130-6131h	6132-6133h	6134-6135h	6136-6137h
7	6138-6139h	613A-613Bh	613C-613Dh	613E-613Fh
8	6140-6141h	6142-6143h	6144-6145h	6146-6147h
9	6148-6149h	614A-614Bh	614C-614Dh	614E-614Fh
10	6150-6151h	6152-6153h	6154-6155h	6156-6157h
11	6158-6159h	615A-615Bh	615C-615Dh	615E-615Fh
12	6160-6161h	6162-6163h	6164-6165h	6166-6167h
13	6168-6169h	616A-616Bh	616C-616Dh	616E-616Fh
14	6170-6171h	6172-6173h	6174-6175h	6176-6177h
15	6178-6179h	617A-617Bh	617C-617Dh	617E-617Fh
16	6180-6181h	6182-6183h	6184-6185h	6186-6187h
17	6188-6189h	618A-618Bh	618C-618Dh	618E-618Fh
18	6190-6191h	6192-6193h	6194-6195h	6196-6197h
19	6198-6199h	619A-619Bh	619C-619Dh	619E-619Fh
20	61A0-61A1h	61A2-61A3h	61A4-61A5h	61A6-61A7h
21	61A8-61A9h	61AA-61ABh	61AC-61ADh	61AE-61AFh
22	61B0-61B1h	61B2-61B3h	61B4-61B5h	61B6-61B7h
23	61B8-61B9h	61BA-61BBh	61BC-61BDh	61BE-61BFh
24	61C0-61C1h	61C2-61C3h	61C4-61C5h	61C6-61C7h
25	61C8-61C9h	61CA-61CBh	61CC-61CDh	61CE-61CFh
26	61D0-61D1h	61D2-61D3h	61D4-61D5h	61D6-61D7h
27	61D8-61D9h	61DA-61DBh	61DC-61DDh	61DE-61DFh
28	61E0-61E1h	61E2-61E3h	61E4-61E5h	61E6-61E7h
29	61E8-61E9h	61EA-61EBh	61EC-61EDh	61EE-61EFh
30	61F0-61F1h	61F2-61F3h	61F4-61F5h	61F6-61F7h
31	61F8-61F9h	61FA-61FBh	61FC-61FDh	61FE-61FFh

Table 16-3. Addresses of LAM, MOTS and MOTO registers for mailboxes (eCAN-A)

Mailbox	LAM	MOTS	MOT0
0	6040h-6041h	6080h-6081h	60C0h-60C1h
1	6042h-6043h	6082h-6083h	60C2h-60C3h
2	6044h-6045h	6084h-6085h	60C4h-60C5h
3	6046h-6047h	6086h-6087h	60C6h-60C7h
4	6048h-6049h	6088h-6089h	60C8h-60C9h
5	604Ah-604Bh	608Ah-608Bh	60CAh-60CBh
6	604Ch-604Dh	608Ch-608Dh	60CCh-60CDh
7	604Eh-604Fh	608Eh-608Fh	60CEh-60CFh
8	6050h-6051h	6090h-6091h	60D0h-60D1h
9	6052h-6053h	6092h-6093h	60D2h-60D3h
10	6054h-6055h	6094h-6095h	60D4h-60D5h
11	6056h-6057h	6096h-6097h	60D6h-60D7h
12	6058h-6059h	6098h-6099h	60D8h-60D9h
13	605Ah-605Bh	609Ah-609Bh	60DAh-60DBh
14	605Ch-605Dh	609Ch-609Dh	60DCh-60DDh
15	605Eh-605Fh	609Eh-609Fh	60DEh-60DFh
16	6060h-6061h	60A0h-60A1h	60E0h-60E1h
17	6062h-6063h	60A2h-60A3h	60E2h-60E3h
18	6064h-6065h	60A4h-60A5h	60E4h-60E5h
19	6066h-6067h	60A6h-60A7h	60E6h-60E7h
20	6068h-6069h	60A8h-60A9h	60E8h-60E9h
21	606Ah-606Bh	60AAh-60ABh	60EAh-60EBh
22	606Ch-606Dh	60ACh-60ADh	60ECh-60EDh
23	606Eh-606Fh	60AEh-60AFh	60EEh-60EFh
24	6070h-6071h	60B0h-60B1h	60F0h-60F1h
25	6072h-6073h	60B2h-60B3h	60F2h-60F3h
26	6074h-6075h	60B4h-60B5h	60F4h-60F5h
27	6076h-6077h	60B6h-60B7h	60F6h-60F7h
28	6078h-6079h	60B8h-60B9h	60F8h-60F9h
29	607Ah-607Bh	60BAh-60BBh	60FAh-60FBh
30	607Ch-607Dh	60BCh-60BDh	60FCh-60FDh
31	607Eh-607Fh	60BEh-60BFh	60FEh-60FFh

Table 16-4. Message Object Behavior Configuration

Message Object Behavior	Mailbox Direction Register (CANMD)	Auto-Answer Mode Bit (AAM)	Remote Transmission Request Bit (RTR)
Transmit message object	0	0	0
Receive message object	1	0	0
Request message object	1	0	1
Reply message object	0	1	0

16.5.1 Transmit Mailbox

The CPU stores the data to be transmitted in a mailbox configured as transmit mailbox. After writing the data and the identifier into the RAM, the message is sent if the corresponding TRS[n] bit has been set, provided the mailbox is enabled by setting the corresponding the CANME.n bit.

If more than one mailbox is configured as transmit mailbox and more than one corresponding TRS[n] is set, the messages are sent one after another in falling order beginning with the mailbox with the highest priority.

In the SCC-compatibility mode, the priority of the mailbox transmission depends on the mailbox number. The highest mailbox number (=15) comprises the highest transmit priority.

In the eCAN mode, the priority of the mailbox transmission depends on the setting of the TPL field in the message control field (MSGCTRL) register. The mailbox with the highest value in the TPL is transmitted first. Only when two mailboxes have the same value in the TPL is the higher numbered mailbox transmitted first.

If a transmission fails due to a loss of arbitration or an error, the message transmission will be reattempted. Before reattempting the transmission, the CAN module checks if other transmissions are requested and then transmits the mailbox with the highest priority.

16.5.2 Receive Mailbox

The identifier of each incoming message is compared to the identifiers held in the receive mailboxes using the appropriate mask. When equality is detected, the received identifier, the control bits, and the data bytes are written into the matching RAM location. At the same time, the corresponding receive-message-pending bit, RMP[n] (RMP.31-0), is set and a receive interrupt is generated if enabled. If no match is detected, the message is not stored.

When a message is received, the message controller starts looking for a matching identifier at the mailbox with the highest mailbox number. Mailbox 15 of the eCAN in SCC compatible mode has the highest receive priority; mailbox 31 has the highest receive priority of the eCAN in eCAN mode.

RMP[n] (RMP.31-0) has to be reset by the CPU after reading the data. If a second message has been received for this mailbox and the receive-message-pending bit is already set, the corresponding message-lost bit (RML[n] (RML.31-0)) is set. In this case, the stored message is overwritten with the new data if the overwrite-protection bit OPC[n] (OPC.31-0) is cleared; otherwise, the next mailboxes are checked.

If a mailbox is configured as a receive mailbox and the RTR bit is set for it, the mailbox can send a remote frame. Once the remote frame is sent, the TRS bit of the mailbox is cleared by the CAN module.

16.5.3 CAN Module Operation in Normal Configuration

If the CAN module is being used in normal configuration (i.e., not in self-test mode), there should be at least one more CAN module on the network, configured for the same bit rate. The other CAN module need NOT be configured to actually receive messages from the transmitting node. But, it should be configured for the same bit rate. This is because a transmitting CAN module expects at least one node in the CAN network to acknowledge the proper reception of a transmitted message. Per CAN protocol specification, any CAN node that received a message will acknowledge (unless the acknowledge mechanism has been explicitly turned off), irrespective of whether it has been configured to store the received message or not. It is not possible to turn off the acknowledge mechanism in C28x DSPs.

The requirement of another node does not exist for the self-test mode (STM). In this mode, a transmitting node generates its own acknowledge signal. The only requirement is that the node be configured for any valid bit-rate. That is, the bit timing registers should not contain a value that is not permitted by the CAN protocol.

It is not possible to achieve a direct digital loopback externally by connecting the CANTX and CANRX pins together (as is possible with SCI/SPI/McBSP modules). An internal loopback is possible in the self-test mode (STM).

16.6 eCAN Registers

16.6.1 Mailbox Enable Register (CANME)

This register is used to enable/disable individual mailboxes.

Figure 16-5. Mailbox-Enable Register (CANME)

31 ... 0
CANME[31:0]
R/W-0

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

Table 16-5. Mailbox-Enable Register (CANME) Field Descriptions

Bit	Field	Value	Description
31:0	CANME[31:0]		Mailbox enable bits. After power-up, all bits in CANME are cleared. Disabled mailboxes can be used as additional memory for the CPU.
		1	The corresponding mailbox is enabled for the CAN module. The mailbox must be disabled before writing to the contents of any identifier field. If the corresponding bit in CANME is set, the write access to the identifier of a mailbox is denied.
		0	The corresponding mailbox RAM area is disabled for the eCAN; however, it is accessible to the CPU as normal RAM.

16.6.2 Mailbox-Direction Register (CANMD)

This register is used to configure a mailbox for transmit or receive operation.

Figure 16-6. Mailbox-Direction Register (CANMD)

31 ... 0
CANMD[31:0]
R/W-0

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

Table 16-6. Mailbox-Direction Register (CANMD) Field Descriptions

Bit	Field	Value	Description
31:0	CANMD[31:0]		Mailbox direction bits. After power-up, all bits are cleared.
		1	The corresponding mailbox is configured as a receive mailbox.
		0	The corresponding mailbox is configured as a transmit mailbox.

16.6.3 Transmission-Request Set Register (CANTRS)

When mailbox *n* is ready to be transmitted, the CPU should set the TRS[*n*] bit to 1 to start the transmission.

These bits are normally set by the CPU and cleared by the CAN module logic. The CAN module can set these bits for a remote frame request. These bits are reset when a transmission is successful or aborted. If a mailbox is configured as a receive mailbox, the corresponding bit in CANTRS is ignored unless the receive mailbox is configured to handle remote frames. The TRS[*n*] bit of a receive mailbox is not ignored if the RTR bit is set. Therefore, a receive mailbox (whose RTR is set) can send a remote frame if its TRS bit is set. Once the remote frame is sent, the TRS[*n*] bit is cleared by the CAN module. Therefore, the same mailbox can be used to request a data frame from another mode. If the CPU tries to set a bit while the eCAN module tries to clear it, the bit is set.

Setting CANTRS[*n*] causes the particular message *n* to be transmitted. Several bits can be set simultaneously. Therefore, all messages with the TRS bit set are transmitted in turn, starting with the mailbox having the highest mailbox number (= highest priority), unless TPL bits dictate otherwise.

The bits in CANTRS are set by writing a 1 from the CPU. Writing a 0 has no effect. After power up, all bits are cleared.

Figure 16-7. Transmission-Request Set Register (CANTRS)

31 ... 0
TRS[31:0]
RS-0

LEGEND: RS = Read/Set; -*n* = value after reset

Table 16-7. Transmission-Request Set Register (CANTRS) Field Descriptions

Bit	Field	Value	Description
31:0	TRS[31:0]		Transmit-request-set bits
		1	Setting TRS*n* transmits the message in that mailbox. Several bits can be set simultaneously with all messages transmitted in turn.
		0	No operation

16.6.4 Transmission-Request-Reset Register (CANTRR)

These bits can only be set by the CPU and reset by the internal logic. These bits are reset when a transmission is successful or is aborted. If the CPU tries to set a bit while the CAN tries to clear it, the bit is set.

Setting the TRR[*n*] bit of the message object *n* cancels a transmission request if it was initiated by the corresponding bit (TRS[*n*]) and is not currently being processed. If the corresponding message is currently being processed, the bit is reset when a transmission is successful (normal operation) or when an aborted transmission due to a lost arbitration or an error condition is detected on the CAN bus line. When a transmission is aborted, the corresponding status bit (AA.31-0) is set. When a transmission is successful, the status bit (TA.31-0) is set. The status of the transmission request reset can be read from the TRS.31-0 bit.

The bits in CANTRR are set by writing a 1 from the CPU.

Figure 16-8. Transmission-Request-Reset Register (CANTRR)

31 … 0
TRR[31:0]
RS-0

LEGEND: RS = Read/Set; -*n* = value after reset

Table 16-8. Transmission-Request-Reset Register (CANTRR) Field Descriptions

Bit	Field	Value	Description
31:0	TRR[31:0]		Transmit-request-reset bits
		1	Setting TRR*n* cancels a transmission request
		0	No operation

16.6.5 Transmission-Acknowledge Register (CANTA)

If the message of mailbox *n* was sent successfully, the bit TA[*n*] is set. This also sets the GMIF0/GMIF1 (GIF0.15/GIF1.15) bit if the corresponding interrupt mask bit in the CANMIM register is set. The GMIF0/GMIF1 bit initiates an interrupt.

The CPU resets the bits in CANTA by writing a 1. This also clears the interrupt if an interrupt has been generated. Writing a 0 has no effect. If the CPU tries to reset the bit while the CAN tries to set it, the bit is set. After power-up, all bits are cleared.

Figure 16-9. Transmission-Acknowledge Register (CANTA)

31 … 0
TA[31:0]
RC-0

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 16-9. Transmission-Acknowledge Register (CANTA) Field Descriptions

Bit	Field	Value	Description
31:0	TA[31:0]		Transmit-acknowledge bits
		1	If the message of mailbox *n* is sent successfully, the bit *n* of this register is set.
		0	The message is not sent.

16.6.6 Abort-Acknowledge Register (CANAA)

If the transmission of the message in mailbox *n* was aborted, the bit AA[*n*] is set and the AAIF (GIF.14) bit is set, which may generate an interrupt if enabled.

The bits in CANAA are reset by writing a 1 from the CPU. Writing a 0 has no effect. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set. After power-up all bits are cleared.

Figure 16-10. Abort-Acknowledge Register (CANAA)

31 ... 0
AA[31:0]
RC-0

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

Table 16-10. Abort-Acknowledge Register (CANAA) Field Descriptions

Bit	Field	Value	Description
31-0	AA[31:0]		Abort-acknowledge bits
		0	If the transmission of the message in mailbox *n* is aborted, the bit *n* of this register is set.
		1	The transmission is not aborted.

16.6.7 Received-Message-Pending Register (CANRMP)

If mailbox *n* contains a received message, the bit RMP[*n*] of this register is set. These bits can be reset only by the CPU and set by the internal logic. A new incoming message overwrites the stored one if the OPC[*n*](OPC.31-0) bit is cleared, otherwise the next mailboxes are checked for a matching ID. If a mailbox is overwritten, the corresponding status bit RML[*n*] is set. The bits in the CANRMP and the CANRML registers are cleared by a write to register CANRMP, with a 1 at the corresponding bit location. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set.

The bits in the CANRMP register can set GMIF0/GMIF1 (GIF0.15/GIF1.15) if the corresponding interrupt mask bit in the CANMIM register is set. The GMIF0/GMIF1 bit initiates an interrupt.

Figure 16-11. Received-Message-Pending Register (CANRMP)

31 … 0
RMP[31:0]
RC-0

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 16-11. Received-Message-Pending Register (CANRMP) Field Descriptions

Bit	Field	Value	Description
31:0	RMP[31:0]		Received-message-pending bits
		1	If mailbox *n* contains a received message, bit RMP[*n*] of this register is set.
		0	The mailbox does not contain a message.

16.6.8 Received-Message-Lost Register (CANRML)

An RML[*n*] bit is set if an old message has been overwritten by a new one in mailbox *n*. These bits can only be reset by the CPU, and set by the internal logic. The bits can be cleared by a write access to the CANRMP register with a 1 at the corresponding bit location. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set. The CANRML register is not changed if the OPC[*n*] (OPC.31-0) bit is set.

If one or more of the bits in the CANRML register are set, the RMLIF (GIF0.11/ GIF1.11) bit is also set. This can initiate an interrupt if the RMLIM (GIM.11) bit is set.

Figure 16-12. Received-Message-Lost Register (CANRML)

31 — 0
RML[31:0]
R-0

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

Table 16-12. Received-Message-Lost Register (CANRML) Field Descriptions

Bit	Field	Value	Description
31:0	RML[31:0]		Received-message-lost bits
		1	An old unread message has been overwritten by a new one in that mailbox.
		0	No message was lost.
			Note: The RML*n* bit is cleared by clearing the set RMP*n* bit.

16.6.9 Remote-Frame-Pending Register (CANRFP)

Whenever a remote frame request is received by the CAN module, the corresponding bit RFP[*n*] in the remote frame pending register is set. If a remote frame is stored in a receive mailbox (AAM=0, CANMD=1), the RFP*n* bit will not be set.

To prevent an auto-answer mailbox from replying to a remote frame request, the CPU has to clear the RFP[*n*] flag and the TRS[*n*] bit by setting the corresponding transmission request reset bit TRR[*n*]. The AAM bit can also be cleared by the CPU to stop the module from sending the message.

If the CPU tries to reset a bit and the CAN module tries to set the bit at the same time, the bit is not set. The CPU cannot interrupt an ongoing transfer.

Figure 16-13. Remote-Frame-Pending Register (CANRFP)

31 ... 0
RFP.31:0
RC-0

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 16-13. Remote-Frame-Pending Register (CANRFP) Field Descriptions

Bit	Field	Value	Description
31:0	RFP.31:0		Remote-frame-pending register. For a receive mailbox, RFP*n* is set if a remote frame is received and TRS*n* is not affected. For a transmit mailbox, RFP*n* is set if a remote frame is received and TRS*n* is set if AAM of the mailbox is 1. The ID of the mailbox must match the remote frame ID.
		1	A remote-frame request was received by the module.
		0	No remote-frame request was received. The register is cleared by the CPU.

16.6.9.1 Handling of Remote Frames

If a remote frame is received (the incoming message has RTR (MSGCTRL.4) = 1), the CAN module compares the identifier to all identifiers of the mailboxes using the appropriate masks starting at the highest mailbox number in descending order.

In the case of a matching identifier (with the message object configured as send mailbox and AAM (MSGID.29) in this message object set) this message object is marked as to be sent (TRS[*n*] is set).

In case of a matching identifier with the mailbox configured as a send mailbox and bit AAM in this mailbox is not set, this message is not received in that mailbox.

After finding a matching identifier in a send mailbox no further compare is done.

With a matching identifier and the message object configured as receive mailbox, this message is handled like a data frame and the corresponding bit in the receive message pending (CANRMP) register is set. The CPU then has to decide how to handle this situation. For information about the CANRMP register, see Section 16.6.7.

For the CPU to change the data in a mailbox that is configured as a remote frame mailbox (AAM set) it has to set the mailbox number and the change data request bit (CDR [MC.8]) in the MCR first. The CPU can then do the access and clear the CDR bit to tell the eCAN that the access is finished. Until the CDR bit is cleared, the transmission of this mailbox is not permitted. Therefore, the newest data is sent.

To change the identifier in that mailbox, the mailbox must be disabled first (CANME*n* = 0).

For the CPU to request data from another node it configures the mailbox as a receive mailbox and sets the TRS bit. In this case the module sends a remote frame request and receives the data frame in the same mailbox that sent the request. Therefore, only one mailbox is necessary to do a remote request. Note that the CPU must set RTR (MSGCTRL.4) to enable a remote frame transmission. Once the remote frame is sent, the TRS bit of the mailbox is cleared by CAN. In this case, bit TA*n* will not be set for that mailbox.

The behavior of the message object *n* is configured with CANMD[*n*] (CANMD.31-0), the AAM (MSGID.29), and RTR (MSGCTRL.4). It shows how to configure a message object according to the desired behavior.

To summarize, a message object can be configured with four different behaviors:

1. A transmit message object is only able to transmit messages.
2. A receive message object is only able to receive messages.
3. A request message object is able to transmit a remote request frame and to wait for the corresponding data frame.
4. A reply message object is able to transmit a data frame whenever a remote request frame is received for the corresponding identifier.

NOTE: When a remote transmission request is successfully transmitted with a message object configured in request mode, the CANTA register is not set and no interrupt is generated. When the remote reply message is received, the behavior of the message object is the same as a message object configured in receive mode.

16.6.10 Global Acceptance Mask Register (CANGAM)

The global-acceptance mask is used by the eCAN in SCC mode. The global-acceptance mask is used for the mailboxes 6 to 15 if the AME bit (MSGID.30) of the corresponding mailbox is set. A received message is only stored in the first mailbox with a matching identifier.

The global-acceptance mask is used for the mailboxes 6 to 15 of the SCC.

Figure 16-14. Global Acceptance Mask Register (CANGAM)

31	30 29	28 16
AMI	Reserved	GAM[28:16]
RWI-0	R-0	RWI-0

15 0
GAM[28:16]
RWI-0

LEGEND: RWI = Read at any time, write during initialization mode only; -*n* = value after reset

Table 16-14. Global Acceptance Mask Register (CANGAM) Field Descriptions

Bit	Field	Value	Description
31	AMI		Acceptance-mask-identifier extension bit
		1	Standard and extended frames can be received. In case of an extended frame, all 29 bits of the identifier are stored in the mailbox and all 29 bits of global acceptance mask register are used for the filter. In case of a standard frame, only the first eleven bits (bit 28 to 18) of the identifier and the global acceptance mask are used.
			The IDE bit of the receive mailbox is a "don't care" and is overwritten by the IDE bit of the transmitted message. The filtering criterion must be satisfied in order to receive a message. The number of bits to be compared is a function of the value of the IDE bit of the transmitted message.
		0	The identifier extension bit stored in the mailbox determines which messages shall be received. The IDE bit of the receive mailbox determines the number of bits to be compared. Filtering is not applicable. The MSGIDs must match bit-for-bit in order to receive a message.
30:29	Reserved		Reads are undefined and writes have no effect.
28:0	GAM 28:0		Global-acceptance mask. These bits allow any identifier bits of an incoming message to be masked. Accept a 0 or a 1 (don't care) for the corresponding bit of the received identifier. Received identifier bit value must match the corresponding identifier bit of the MSGID register.

16.6.11 Master Control Register (CANMC)

This register is used to control the settings of the CAN module. Some bits of the CANMC register are EALLOW protected. For read/write operations, only 32-bit access is supported.

Figure 16-15. Master Control Register (CANMC)

31–17	16
Reserved	SUSP
R-0	R/W-0

15	14	13	12	11	10	9	8
MBCC	TCC	SCB	CCR	PDR	DBO	WUBA	CDR
R/WP-0	SP-x	R/WP-0	R/WP-1	R/WP-0	R/WP-0	R/WP-0	R/WP-0

7	6	5	4–0
ABO	STM	SRES	MBNR
R/WP-0	R/WP-0	R/S-0	R/W-0

LEGEND: R = Read, WP = Write in EALLOW mode only, S = Set in EALLOW mode only; -*n* = value after reset; x = Indeterminate
Note: eCAN only, reserved in the SCC

Table 16-15. Master Control Register (CANMC) Field Descriptions

Bit	Field	Value	Description
31:17	Reserved		Reads are undefined and writes have no effect.
16	SUSP		SUSPEND. This bit determines the action of the CAN module in SUSPEND (emulation stop such as breakpoint or single stepping).
		1	FREE mode. The peripheral continues to run in SUSPEND. The node would participate in CAN communication normally (sending acknowledge, generating error frames, transmitting/receiving data) while in SUSPEND.
		0	SOFT mode. The peripheral shuts down during SUSPEND after the current transmission is complete.
15	MBCC		Mailbox timestamp counter clear bit. This bit is reserved in SCC mode and it is EALLOW protected.
		1	The time stamp counter is reset to 0 after a successful transmission or reception of mailbox 16.
		0	The time stamp counter is not reset.
14	TCC		Time stamp counter MSB clear bit. This bit is reserved in SCC mode and it is EALLOW protected.
		1	The MSB of the time stamp counter is reset to 0. The TCC bit is reset after one clock cycle by the internal logic.
		0	The time stamp counter is not changed.
13	SCB		SCC compatibility bit. This bit is reserved in SCC mode and it is EALLOW protected.
		1	Select eCAN mode.
		0	The eCAN is in SCC mode. Only mailboxes 15 to 0 can be used.
12	CCR		Change-configuration request. This bit is EALLOW protected.
		1	The CPU requests write access to the configuration register CANBTC and the acceptance mask registers (CANGAM, LAM[0], and LAM[3]) of the SCC. After setting this bit, the CPU must wait until the CCE flag of CANES register is at 1 before proceeding to the CANBTC register. The CCR bit will also be set upon a bus-off condition, if the ABO bit is not set. The BO condition can be exited by clearing this bit (after 128 * 11 consecutive recessive bits on the bus).
		0	The CPU requests normal operation. This can be done only after the configuration register CANBTC was set to the allowed values. It also exits the bus-off state after the obligatory bus-off recovery sequence.
11	PDR		Power down mode request. This bit is automatically cleared by the eCAN module upon wakeup from low-power mode. This bit is EALLOW protected.
		1	The local power-down mode is requested.
		0	The local power-down mode is not requested (normal operation).
			Note: If an application sets the TRS*n* bit for a mailbox and then immediately sets the PDR bit, the CAN module goes into LPM without transmitting the data frame. This is because it takes about 80 CPU cycles for the data to be transferred from the mailbox RAM to the transmit buffer. Therefore, the application has to ensure that any pending transmission has been completed before writing to the PDR bit. The TA*n* bit could be polled to ensure completion of transmission.

Table 16-15. Master Control Register (CANMC) Field Descriptions (continued)

Bit	Field	Value	Description
10	DBO		Data byte order. This bit selects the byte order of the message data field. This bit is EALLOW protected.
		1	The data is received or transmitted least significant byte first.
		0	The data is received or transmitted most significant byte first.
9	WUBA		Wake up on bus activity. This bit is EALLOW protected.
		1	The module leaves the power-down mode after detecting any bus activity.
		0	The module leaves the power-down mode only after writing a 0 to the PDR bit.
8	CDR		Change data field request. This bit allows fast data message update.
		1	The CPU requests write access to the data field of the mailbox specified by the MBNR.4:0 field (MC.4-0). The CPU must clear the CDR bit after accessing the mailbox. The module does not transmit that mailbox content while the CDR is set. This is checked by the state machine before and after it reads the data from the mailbox to store it in the transmit buffer.
			Note: Once the TRS bit is set for a mailbox and then data is changed in the mailbox using the CDR bit, the CAN module fails to transmit the new data and transmits the old data instead. To avoid this, reset transmission in that mailbox using the TRR*n* bit and set the TRS*n* bit again. The new data is then transmitted.
		0	The CPU requests normal operation.
7	ABO		Auto bus on. This bit is EALLOW protected.
		1	After the bus-off state, the module goes back automatically into bus-on state after 128 * 11 recessive bits have been monitored.
		0	The bus-off state may only be exited after 128 * 11 consecutive recessive bits on the bus and after having cleared the CCR bit.
6	STM		Self test mode. This bit is EALLOW protected.
		1	The module is in self-test mode. In this mode, the CAN module generates its own acknowledge (ACK) signal, thus enabling operation without a bus connected to the module. The message is not sent, but read back and stored in the appropriate mailbox. The MSGID of the received frame is not stored in the MBR in STM.
			Note: In STM, if no MBX has been configured to receive a transmitted frame, then that frame will be stored in MBX0, even if MBX0 has not been configured for receive operations. If LAMs are configured such that some mailboxes can receive and store data frames, then a data frame that does not satisfy the acceptance mask filtering criterion for any receive mailbox will be lost.
		0	The module is in normal mode.
5	SRES		This bit can only be written and is always read as zero.
		1	A write access to this register causes a software reset of the module (all parameters, except the protected registers, are reset to their default values). The mailbox contents and the error counters are not modified. Pending and ongoing transmissions are canceled without perturbing the communication.
		0	0 No effect
4:0	MBNR 4:0		Mailbox number
		1	The bit MBNR.4 is for eCAN only, and is reserved in the SCC.
		0	Number of mailbox, for which the CPU requests a write access to the data field. This field is used in conjunction with the CDR bit.

16.6.11.1 CAN Module Action in SUSPEND

1. If there is no traffic on the CAN bus and SUSPEND mode is requested, the node goes into SUSPEND mode.
2. If there is traffic on the CAN bus and SUSPEND mode is requested, the node goes into SUSPEND mode when the ongoing frame is over.
3. If the node was transmitting, when SUSPEND is requested, it goes to SUSPEND state after it gets the acknowledgment. If it does not get an acknowledgment or if there are some other errors, it transmits an error frame and then goes to SUSPEND state. The TEC is modified accordingly. In the second case, i.e., it is suspended after transmitting an error frame, the node re-transmits the original frame after coming out of suspended state. The TEC is modified after transmission of the frame accordingly.
4. If the node was receiving, when SUSPEND is requested, it goes to SUSPEND state after transmitting

the acknowledgment bit. If there is any error, the node sends an error frame and go to SUSPEND state. The REC is modified accordingly before going to SUSPEND state.

5. If there is no traffic on the CAN bus and SUSPEND removal is requested, the node comes out of SUSPEND state.
6. If there is traffic on the CAN bus and SUSPEND removal is requested, the node comes out after the bus goes to idle. Therefore, a node does not receive any "partial" frame, which could lead to generation of error frames.
7. When the node is suspended, it does not participate in transmitting or receiving any data. Thus neither acknowledgment bit nor any error frame is sent. TEC and REC are not modified during SUSPEND state.

16.6.12 Bit-Timing Configuration Register (CANBTC)

The CANBTC register is used to configure the CAN node with the appropriate network-timing parameters. This register must be programmed before using the CAN module.

This register is write-protected in user mode and can only be written in initialization mode (see Section 3.6.1).

NOTE: To avoid unpredictable behavior of the CAN module, the CANBTC register should never be programmed with values not allowed by the CAN protocol specification and by the bit timing rules listed in Section 3.1.1.

Figure 16-16. Bit-Timing Configuration Register (CANBTC)

31–24	23–16
Reserved	BRP_{reg}
R-x	RWPI-0

15–10	9–8	7	6–3	2–0
Reserved	SJW_{reg}	SAM	$TSEG1_{reg}$	$TSEG2_{reg}$
R-0	RWPI-0	RWPI-0	RWPI-0	RWPI-0

LEGEND: RWPI = Read in all modes, write in EALLOW mode during initialization mode only; -*n* = value after reset

Table 16-16. Bit-Timing Configuration Register (CANBTC) Field Descriptions

Bit	Field	Value	Description
31:24	Reserved		Reads are undefined and writes have no effect.
23:16	BRP_{reg}.7:0		Baud rate prescaler. This register sets the prescaler for the baud rate settings. The length of one TQ is defined by: $$TQ = \frac{1}{SYSCLKOUT/2} \times (BRP_{reg} + 1)$$ where SYSCLKOUT /2 is the frequency of the CAN module clock. BRP_{reg} denotes the "register value" of the prescaler; i.e., value written into bits 23:16 of the CANBTC register. This value is automatically enhanced by 1 when the CAN module accesses it. The enhanced value is denoted by the symbol BRP (BRP = BRP_{reg} + 1). BRP is programmable from 1 to 256. **Note:** For the special case of BRP = 1, the Information Processing Time (IPT) is equal to 3 time quanta (TQ). This is not compliant to the ISO 11898 Standard, where the IPT is defined to be less than or equal to 2 TQ. Thus the usage of this mode (BRP_{reg} = 0) is not allowed.
15	Reserved		Reads are undefined and writes have no effect.
9:8	SJW_{reg} 1:0		Synchronization jump width. The parameter SJW indicates, by how many units of TQ a bit is allowed to be lengthened or shortened when resynchronizing. SJW_{reg} denotes the "register value" of the "resynchronization jump width;" i.e., the value written into bits 9:8 of the CANBTC register. This value is automatically enhanced by 1 when the CAN module accesses it. This enhanced value is denoted by the symbol SJW. $SJW = SJW_{reg} + 1$ SJW is programmable from 1 to 4 TQ. The maximum value of SJW is determined by the minimum value of TSEG2 and 4 TQ. $SJW_{(max)}$ = min [4 TQ, TSEG2]
7	SAM		This parameter sets the number of samples used by the CAN module to determine the actual level of the CAN bus. When the SAM bit is set, the level determined by the CAN bus corresponds to the result from the majority decision of the last three values. The sample points are at the sample point and twice before with a distance of ½ TQ. 1 The CAN module samples three times and make a majority decision. The triple sample mode shall be selected only for bit rate prescale values greater than 4 (BRP > 4). 0 The CAN module samples only once at the sampling point.

Table 16-16. Bit-Timing Configuration Register (CANBTC) Field Descriptions (continued)

Bit	Field	Value	Description
6:3	TSEG1 3:0		Time segment 1. The length of a bit on the CAN bus is determined by the parameters TSEG1, TSEG2, and BRP. All controllers on the CAN bus must have the same baud rate and bit length. For different clock frequencies of the individual controllers, the baud rate has to be adjusted by the said parameters.
			This parameter specifies the length of the TSEG1 segment in TQ units. TSEG1 combines PROP_SEG and PHASE_SEG1 segments:
			TSEG1 = PROP_SEG + PHASE_SEG1
			where PROP_SEG and PHASE_SEG1 are the length of these two segments in TQ units.
			TSEG1reg denotes the "register value" of "time segment 1;" i.e., the value written into bits 6:3 of the CANBTC register. This value is automatically enhanced by 1 when the CAN module accesses it. This enhanced value is denoted by the symbol TSEG1.
			$TSEG1 = TSEG1_{reg} + 1$
			TSEG1 value should be chosen such that TSEG1 is greater than or equal to TSEG2 and IPT. For more information on IPT, see Section 3.1.1.
2:0	$TSEG2_{reg}$		Time Segment 2. TSEG2 defines the length of PHASE_SEG2 segment in TQ units: TSEG2 is programmable in the range of 1 TQ to 8 TQ and has to fulfill the following timing rule: TSEG2 must be smaller than or equal to TSEG1 and must be greater than or equal to IPT. TSEG2reg denotes the "register value" of "time segment 2;" i.e., the value written into bits 2:0 of the CANBTC register. This value is automatically enhanced by 1 when the CAN module accesses it. This enhanced value is denoted by the symbol TSEG2. $TSEG2 = TSEG2_{reg} + 1$

16.6.13 Error and Status Register (CANES)

The status of the CAN module is shown by the Error and Status Register (CANES) and the error counter registers, which are described in this section.

The error and status register comprises information about the actual status of the CAN module and displays bus error flags as well as error status flags. If one of these error flags is set, then the current state of all other error flags is frozen. i.e. Only the first error is stored. In order to update the CANES register subsequently, the error flag which is set has to be acknowledged by writing a 1 to it. This action also clears the flag bit.

Figure 16-17. Error and Status Register (CANES)

31 25	24	23	22	21	20	19	18	17	16
Reserved	FE	BE	SA1	CRCE	SE	ACKE	BO	EP	EW
R-0	RC-0	RC-0	R-1	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0

15 6	5	4	3	2	1	0
Reserved	SMA	CCE	PDA	Rsvd	RM	TM
R-0	R-0	R-1	R-0	R-0	R-0	R-0

LEGEND: R = Read; C = Clear; -*n* = value after reset

Table 16-17. Error and Status Register (CANES) Field Descriptions

Bit	Field	Value	Description
31:25	Reserved		Reads are undefined and writes have no effect.
24	FE		Form error flag
		1	A form error occurred on the bus. This means that one or more of the fixed-form bit fields had the wrong level on the bus.
		0	No form error detected; the CAN module was able to send and receive correctly.
23	BE		Bit error flag
		1	The received bit does not match the transmitted bit outside of the arbitration field or during transmission of the arbitration field, a dominant bit was sent but a recessive bit was received.
		0	No bit error detected.
22	SA1		Stuck at dominant error. The SA1 bit is always at 1 after a hardware reset, a software reset, or a Bus-Off condition. This bit is cleared when a recessive bit is detected on the bus.
		1	The CAN module never detected a recessive bit.
		0	The CAN module detected a recessive bit.
21	CRCE		CRC error.
		1	The CAN module received a wrong CRC.
		0	The CAN module never received a wrong CRC.
20	SE		Stuff error.
		1	A stuff bit error occurred.
		0	No stuff bit error occurred.
19	ACKE		Acknowledge error.
		1	The CAN module received no acknowledge.
		0	All messages have been correctly acknowledged.
18	BO		Bus-off status. The CAN module is in bus-off state.
		1	There is an abnormal rate of errors on the CAN bus. This condition occurs when the transmit error counter (CANTEC) has reached the limit of 256. During Bus Off, no messages can be received or transmitted. The bus-off state can be exited by clearing the CCR bit in CANMC register or if the Auto Bus On (ABO) (CANMC.7) bit is set, after 128 * 11 receive bits have been received. After leaving Bus Off, the error counters are cleared.
		0	Normal operation
17	EP		Error-passive state
		1	The CAN module is in error-passive mode. CANTEC has reached 128.
		0	The CAN module is in error-active mode.

Table 16-17. Error and Status Register (CANES) Field Descriptions (continued)

Bit	Field	Value	Description
16	EW		Warning status
		1	One of the two error counters (CANREC or CANTEC) has reached the warning level of 96.
		0	Values of both error counters (CANREC and CANTEC) are less than 96.
15:6	Reserved		Reads are undefined and writes have no effect.
5	SMA		Suspend mode acknowledge. This bit is set after a latency of one clock cycle—up to the length of one frame—after the suspend mode was activated. The suspend mode is activated with the debugger tool when the circuit is not in run mode. During the suspend mode, the CAN module is frozen and cannot receive or transmit any frame. However, if the CAN module is transmitting or receiving a frame when the suspend mode is activated, the module enters suspend mode only at the end of the frame. Run mode is when SOFT mode is activated (CANMC.16 = 1).
		1	The module has entered suspend mode.
		0	The module is not in suspend mode.
4	CCE		Change configuration enable. This bit displays the configuration access right. This bit is set after a latency of one clock cycle.
		1	The CPU has write access to the configuration registers.
		0	The CPU is denied write access to the configuration registers.
			Note: The reset state of the CCE bit is 1. That is, upon reset, you can write to the bit timing registers. However, once the CCE bit is cleared (as part of the module initialization), the CANRX pin must be sensed high before you can set the CCE bit to 1 again.
3	PDA		Power-down mode acknowledge
		1	The CAN module has entered the power-down mode.
		0	Normal operation
2	Reserved		Reads are undefined and writes have no effect.
1	RM		Receive mode. The CAN module is in receive mode. This bit reflects what the CAN module is actually doing regardless of mailbox configuration.
		1	The CAN module is receiving a message.
		0	The CAN module is not receiving a message.
0	TM		Transmit mode. The CAN module is in transmit mode. This bit reflects what the CAN module is actually doing regardless of mailbox configuration.
		1	The CAN module is transmitting a message.
		0	The CAN module is not transmitting a message.

16.6.14 CAN Error Counter Registers (CANTEC/CANREC)

The CAN module contains two error counters: the receive error counter (CANREC) and the transmit error counter (CANTEC). The values of both counters can be read via the CPU interface. These counters are incremented or decremented according to the CAN protocol specification version 2.0.

Figure 16-18. Transmit-Error-Counter Register (CANTEC)

31 – 8	7 – 0
Reserved	TEC
R-x	R-0

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

Figure 16-19. Receive-Error-Counter Register (CANREC)

31 – 8	7 – 0
Reserved	REC
R-x	R-0

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

After reaching or exceeding the error passive limit (128), the receive error counter will not be increased anymore. When a message was received correctly, the counter is set again to a value between 119 and 127 (compare with CAN specification).

After reaching the bus-off state, the transmit error counter is undefined while the receive error counter changes its function. After reaching the bus-off state, the receive error counter is cleared. It is then incremented after every 11 consecutive recessive bits on the bus. These 11 bits correspond to the gap between two frames on the bus. If the counter reaches 128, the module automatically changes back to the bus-on status if this feature is enabled (Auto Bus On bit (ABO) (MC.7) set). All internal flags are reset and the error counters are cleared. After leaving initialization mode, the error counters are cleared.

16.6.15 Interrupt Registers

Interrupts are controlled by the interrupt flag registers, interrupt mask registers and mailbox interrupt level registers. These registers are described in the following subsections.

16.6.15.1 Global Interrupt Flag Registers (CANGIF0/CANGIF1)

These registers allow the CPU to identify the interrupt source.

The interrupt flag bits are set if the corresponding interrupt condition did occur. The global interrupt flags are set depending on the setting of the GIL bit in the CANGIM register. If that bit is set, the global interrupts set the bits in the CANGIF1 register; otherwise, in the CANGIF0 register. This also applies to the Interrupt Flags AAIF and RMLIF. These bits are set according to the setting of the appropriate GIL bit in the CANGIM register.

The following bits are set regardless of the corresponding interrupt mask bits in the CANGIM register: MTOF*n*, WDIF*n*, BOIF*n*, TCOF*n*, WUIF*n*, EPIF*n*, AAIF*n*, RMLIF*n*, and WLIF*n*.

For any mailbox, the GMIFn bit is set only when the corresponding mailbox interrupt mask bit (in the CANMIM register) is set.

If all interrupt flags are cleared and a new interrupt flag is set the interrupt output line is activated when the corresponding interrupt mask bit is set. The interrupt line stays active until the interrupt flag is cleared by the CPU by writing a 1 to the appropriate bit or by clearing the interrupt-causing condition.

The GMIFx flags must be cleared by writing a 1 to the appropriate bit in the CANTA register or the CANRMP register (depending on mailbox configuration) and cannot be cleared in the CANGIFx register. After clearing one or more interrupt flags and one or more interrupt flags still set, a new interrupt is generated. The interrupt flags are cleared by writing a 1 to the corresponding bit location. If the GMIFx is set the Mailbox Interrupt Vector MIVx indicates the mailbox number of the mailbox that caused the setting of the GMIFx. In case more than one mailbox interrupt is pending, it always displays the highest mailbox interrupt vector assigned to that interrupt line.

Figure 16-20. Global Interrupt Flag 0 Register (CANGIF0)

31–24
Reserved
R-x

23–18	17	16
Reserved	MTOF0	TCOF0
R-x	R-0	RC-0

15	14	13	12	11	10	9	8
GMIF0	AAIF0	WDIF0	WUIF0	RMLIF0	BOIF0	EPIF0	WLIF0
R/W-0	R-0	RC-0	RC-0	R-0	RC-0	RC-0	RC-0

7–5	4	3	2	1	0
Reserved	MIV0.4	MIV0.3	MIV0.2	MIV0.1	MIV0.0
R/W-0	R-0	R-0	R-0	R-0	R-0

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

Figure 16-21. Global Interrupt Flag 1 Register (CANGIF1)

31–24
Reserved
R-x

23–18	17	16
Reserved	MTOF1	TCOF1
R-x	R-0	RC-0

15	14	13	12	11	10	9	8
GMIF1	AAIF1	WDIF1	WUIF1	RMLIF1	BOIF1	EPIF1	WLIF1
R/W-0	R-0	RC-0	RC-0	R-0	RC-0	RC-0	RC-0

7–5	4	3	2	1	0
Reserved	MIV0.4	MIV0.3	MIV0.2	MIV0.1	MIV0.0
R/W-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R/W = Read/Write; R = Read; C = Clear; -*n* = value after reset
Note: eCAN only, reserved in the SCC

NOTE: The following bit descriptions are applicable to both the CANGIF0 and CANGIF1 registers. For the following interrupt flags, whether they are set in the CANGIF0 or the CANGIF1 register is determined by the value of the GIL bit in the CANGIM register: TCOF*n*, AAIF*n*, WDIF*n*, WUIF*n*, RMLIF*n*, BOIF*n*, EPIF*n*, and WLIF*n*.

If GIL = 0, these flags are set in the CANGIF0 register; if GIL = 1, they are set in the CANGIF1 register.

Similarly, the choice of the CANGIF0 and CANGIF1 register for the MTOFn and GMIFn bits is determined by the MILn bit in the CANMIL register.

Table 16-18. Global Interrupt Flag Registers (CANGIF0/CANGIF1) Field Descriptions

Bit	Field	Value	Description
31:18	Reserved		Reserved. Reads are undefined and writes have no effect.
17	MTOF0/1		Mailbox time-out flag. This bit is not available in the SCC mode.
		1	One of the mailboxes did not transmit or receive a message within the specified time frame.
		0	No time out for the mailboxes occurred.
			Note: Whether the MTOFn bit gets set in CANGIF0 or CANGIF1 depends on the value of MIL*n*. MTOFn gets cleared when TOSn is cleared. The TOSn bit will be cleared upon (eventual) successful transmission/reception.

Table 16-18. Global Interrupt Flag Registers (CANGIF0/CANGIF1) Field Descriptions (continued)

Bit	Field	Value	Description
16	TCOF0/1		Time stamp counter overflow flag.
		1	The MSB of the time stamp counter has changed from 0 to 1.
		0	The MSB of the time stamp counter is 0. That is, it has not changed from 0 to 1.
15	GMIF0/1		Global mailbox interrupt flag. This bit is set only when the corresponding mailbox interrupt mask bit in the CANMIM register is set.
		1	One of the mailboxes transmitted or received a message successfully.
		0	No message has been transmitted or received.
14	AAIF0/1		Abort-acknowledge interrupt flag
		1	A send transmission request has been aborted.
		0	No transmission has been aborted.
			Note: The AAIFn bit is cleared by clearing the set AAn bit.
13	WDIF0/WDIF1		Write-denied interrupt flag
		1	The CPU write access to a mailbox was not successful. The WDIF interrupt is asserted when the identifier field of a mailbox is written to, while it is enabled. Before writing to the MSGID field of a MBX, it should be disabled. If you try this operation when the MBX is still enabled, the WDIF bit will be set and a CAN interrupt asserted.
		0	The CPU write access to the mailbox was successful.
12	WUIF0/WUIF1		Wake-up interrupt flag
		1	During local power down, this flag indicates that the module has left sleep mode.
		0	The module is still in sleep mode or normal operation
11	RMLIF0/1		Receive-message-lost interrupt flag
		1	At least for one of the receive mailboxes, an overflow condition has occurred and the corresponding bit in the MILn register is cleared.
		0	No message has been lost.
			Note: The RMLIFn bit is cleared by clearing the set RMPn bit.
10	BOIF0/BOIF1		Bus off interrupt flag
		1	The CAN module has entered bus-off mode.
		0	The CAN module is still in bus-on mode.
9	EPIF0/EPIF1		Error passive interrupt flag
		1	The CAN module has entered error-passive mode.
		0	The CAN module is not in error-passive mode.
8	WLIF0/WLIF1		Warning level interrupt flag
		1	At least one of the error counters has reached the warning level.
		0	None of the error counters has reached the warning level.
7:5	Reserved		Reads are undefined and writes have no effect.
4:0	MIV0.4:0/MIV1.4:0		Mailbox interrupt vector. Only bits 3:0 are available in SCC mode.
			This vector indicates the number of the mailbox that set the global mailbox interrupt flag. It keeps that vector until the appropriate MIFn bit is cleared or when a higher priority mailbox interrupt occurred. Then the highest interrupt vector is displayed, with mailbox 31 having the highest priority. In the SCC mode, mailbox 15 has the highest priority. Mailboxes 16 to 31 are not recognized.
			If no flag is set in the TA/RMP register and GMIF1 or GMIF0 also cleared, this value is undefined.

16.6.15.2 Global Interrupt Mask Register (CANGIM)

The set up for the interrupt mask register is the same as for the interrupt flag register. If a bit is set, the corresponding interrupt is enabled. This register is EALLOW protected.

Figure 16-22. Global Interrupt Mask Register (CANGIM)

31–18	17	16
Reserved	MTOM	TCOM
R-0	R/WP-0	R/WP-0

15	14	13	12	11	10	9	8
Reserved	AAIM	WDIM	WUIM	RMLIM	BOIM	EPIM	WLIM
R-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0

7–3	2	1	0
Reserved	GIL	I1EN	I0EN
R-0	R/WP-0	R/WP-0	R/WP-0

LEGEND: R = Read; W = Write; WP = Write in EALLOW mode only; -*n* = value after reset

Table 16-19. Global Interrupt Mask Register (CANGIM) Field Descriptions

Bit	Field	Value	Description
31:18	Reserved		Reads are undefined and writes have no effect.
17	MTOM		Mailbox time-out interrupt mask
		1	Enabled
		0	Disabled
16	TCOM		Time stamp counter overflow mask
		1	Enabled
		0	Disabled
15	Reserved		Reads are undefined and writes have no effect.
14	AAIM		Abort Acknowledge Interrupt Mask.
		1	Enabled
		0	Disabled
13	WDIM		Write denied interrupt mask
		1	Enabled
		0	Disabled
12	WUIM		Wake-up interrupt mask
		1	Enabled
		0	Disabled
11	RMLIM		Received-message-lost interrupt mask
		1	Enabled
		0	Disabled
10	BOIM		Bus-off interrupt mask
		1	Enabled
		0	Disabled
9	EPIM		Error-passive interrupt mask
		1	Enabled
		0	Disabled
8	WLIM		Warning level interrupt mask
		1	Enabled
		0	Disabled
7:3	Reserved		Reads are undefined and writes have no effect.
2	GIL		Global interrupt level for the interrupts TCOF, WDIF, WUIF, BOIF, EPIF, RMLIF, AAIF and WLIF.
		1	All global interrupts are mapped to the ECAN1INT interrupt line.
		0	All global interrupts are mapped to the ECAN0INT interrupt line.

Table 16-19. Global Interrupt Mask Register (CANGIM) Field Descriptions (continued)

Bit	Field	Value	Description
1	I1EN		Interrupt 1 enable
		1	This bit globally enables all interrupts for the ECAN1INT line if the corresponding masks are set.
		0	The ECAN1INT interrupt line is disabled.
0	I0EN		Interrupt 0 enable
		1	This bit globally enables all interrupts for the ECAN0INT line if the corresponding masks are set.
		0	The ECAN0INT interrupt line is disabled.

The GMIF has no corresponding bit in the CANGIM because the mailboxes have individual mask bits in the CANMIM register.

16.6.15.3 Mailbox Interrupt Mask Register (CANMIM)

There is one interrupt flag available for each mailbox. This can be a receive or a transmit interrupt depending on the configuration of the mailbox. This register is EALLOW protected.

Figure 16-23. Mailbox Interrupt Mask Register (CANMIM)

31 … 0
MIM.31:0
R/W-0

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

Table 16-20. Mailbox Interrupt Mask Register (CANMIM) Field Descriptions

Bit	Field	Value	Description
31:0	MIM.31:0		Mailbox interrupt mask. After power up all interrupt mask bits are cleared and the interrupts are disabled. These bits allow any mailbox interrupt to be masked individually.
		1	Mailbox interrupt is enabled. An interrupt is generated if a message has been transmitted successfully (in case of a transmit mailbox) or if a message has been received without any error (in case of a receive mailbox).
		0	Mailbox interrupt is disabled.

16.6.15.4 Mailbox Interrupt Level Register (CANMIL)

Each of the 32 mailboxes may initiate an interrupt on one of the two interrupt lines. Depending on the setting in the mailbox interrupt level register (CANMIL), the interrupt is generated on ECAN0INT (MIL*n* = 0) or on line ECAN1INT (MIL[*n*] = 1).

Figure 16-24. Mailbox Interrupt Level Register (CANMIL)

31 … 0
MIL.31:0
R/W-0

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

Table 16-21. Mailbox Interrupt Level Register (CANMIL) Field Descriptions

Bit	Field	Value	Description
31:0	MIL.31:0		Mailbox interrupt level. These bits allow any mailbox interrupt level to be selected individually.
		1	The mailbox interrupt is generated on interrupt line 1.
		0	The mailbox interrupt is generated on interrupt line 0.

16.6.16 Overwrite Protection Control Register (CANOPC)

If there is an overflow condition for mailbox n (RMP[*n*] is set to 1 and a new receive message would fit for mailbox n), the new message is stored depending on the settings in the CANOPC register. If the corresponding bit OPC[*n*] is set to 1, the old message is protected against being overwritten by the new message; thus, the next mailboxes are checked for a matching ID. If no other mailbox is found, the message is lost without further notification. If the bit OPC[*n*] is cleared to 0, the old message is overwritten by the new one. This is notified by setting the receive message lost bit RML[*n*].

For read/write operations, only 32-bit access is supported.

Figure 16-25. Overwrite Protection Control Register (CANOPC)

31 ... 0
OPC.31:0
R/W-0

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

Table 16-22. Overwrite Protection Control Register (CANOPC) Field Descriptions

Bit	Field	Value	Description
31:0	OPC.31:0		Overwrite protection control bits
		1	1 If the bit OPC[*n*] is set to 1, an old message stored in that mailbox is protected against being overwritten by the new message.
		0	0 If the bit OPC[*n*] is not set, the old message can be overwritten by a new one.

16.6.17 eCAN I/O Control Registers (CANTIOC, CANRIOC)

The CANTX and CANRX pins should be configured for CAN use. This is done using the CANTIOC and CANRIOC registers.

Figure 16-26. TX I/O Control Register (CANTIOC)

31 16
Reserved
R-0

15 4	3	2 0
Reserved	TXFUNC	Reserved
R-0	RWP-0	

LEGEND: RWP = Read in all modes, write in EALLOW-mode only; R = Read only; -*n* = value after reset

Table 16-23. TX I/O Control Register (CANTIOC) Field Descriptions

Bit	Field	Value	Description
31:4	Reserved		Reads are undefined and writes have no effect.
3	TXFUNC		This bit must be set for CAN module function.
		1	The CANTX pin is used for the CAN transmit functions.
		0	Reserved
2:0	Reserved		Reserved

Figure 16-27. RX I/O Control Register (CANRIOC)

31 — 16
Reserved
R-x

15 — 4	3	2 — 0
Reserved	RXFUNC	Reserved
R-0	RWP-0	

LEGEND: RWP = Read in all modes, write in EALLOW-mode only; R = Read only; -*n* = value after reset; x = indeterminate

Table 16-24. RX I/O Control Register (CANRIOC) Field Descriptions

Bit	Field	Value	Description
31:4	Reserved		Reads are undefined and writes have no effect.
3	RXFUNC		This bit must be set for CAN module function.
		1	The CANRX pin is used for the CAN receive functions.
		0	Reserved
2:0	Reserved		Reserved

16.7 Timer Management Unit

Several functions are implemented in the eCAN to monitor the time when messages are transmitted/received. A separate state machine is included in the eCAN to handle the time-control functions. This state machine has lower priority when accessing the registers than the CAN state machine has. Therefore, the time-control functions may be delayed by other ongoing actions.

16.7.1 Time Stamp Functions

To get an indication of the time of reception or transmission of a message, a free-running 32-bit timer (TSC) is implemented in the module. Its content is written into the time stamp register of the corresponding mailbox (Message Object Time Stamp [MOTS]) when a received message is stored or a message has been transmitted.

The counter is driven from the bit clock of the CAN bus line. The timer is stopped during the initialization mode or if the module is in sleep or suspend mode. After power-up reset, the free-running counter is cleared.

The most significant bit of the TSC register is cleared by writing a 1 to TCC (CANMC.14). The TSC register can also be cleared when mailbox 16 transmitted or received (depending on the setting of CANMD.16 bit) a message successfully. This is enabled by setting the MBCC bit (CANMC.15). Therefore, it is possible to use mailbox 16 for global time synchronization of the network. The CPU can read and write the counter.

Overflow of the counter is detected by the TSC-counter-overflow-interrupt flag (TCOF*n*-CANGIF*n*.16). An overflow occurs when the highest bit of the TSC counter changes to 1. Therefore, the CPU has enough time to handle this situation.

16.7.1.1 Time-Stamp Counter Register (CANTSC)

This register holds the time-stamp counter value at any instant of time. This is a free-running 32-bit timer which is clocked by the bit clock of the CAN bus. For example, at a bit rate of 1 Mbps, CANTSC would increment every 1 µs.

Figure 16-28. Time-Stamp Counter Register (CANTSC)

31 … 0
TSC31:0
R/WP-0

LEGEND: R = Read; WP = Write in EALLOW enabled mode only; -*n* = value after reset
Note: eCAN mode only, reserved in the SCC

Table 16-25. Time-Stamp Counter Register (CANTSC) Field Descriptions

Bit	Field	Value	Description
31:0	TSC31:0		Time-stamp counter register. Value of the local network time counter used for the time-stamp and the time-out functions.

16.7.1.2 Message Object Time Stamp Registers (MOTS)

This register holds the value of the TSC when the corresponding mailbox data was successfully transmitted or received. Each mailbox has its own MOTS register.

Figure 16-29. Message Object Time Stamp Registers (MOTS)

31	0
MOTS31:0	
R/W-x	

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

Table 16-26. Message Object Time Stamp Registers (MOTS) Field Descriptions

Bit	Field	Value	Description
31:0	MOTS31:0		Message object time stamp register. Value of the time stamp counter (TSC) when the message has been actually received or transmitted.

16.7.2 Time-Out Functions

To ensure that all messages are sent or received within a predefined period, each mailbox has its own time-out register. If a message has not been sent or received by the time indicated in the time-out register and the corresponding bit TOC[*n*] is set in the TOC register, a flag is set in the time-out status register (TOS).

For transmit mailboxes the TOS[*n*] flag is cleared when the TOC[*n*] bit is cleared or when the corresponding TRS[*n*] bit is cleared, no matter whether due to successful transmission or abortion of the transmit request. For receive mailboxes, the TOS[*n*] flag is cleared when the corresponding TOC[*n*] bit is cleared.

The CPU can also clear the time-out status register flags by writing a 1 into the time-out status register.

The message object time-out registers (MOTO) are implemented as a RAM. The state machine scans all the MOTO registers and compares them to the TSC counter value. If the value in the TSC register is equal to or greater than the value in the time-out register, and the corresponding TRS bit (applies to transmit mailboxes only) is set, and the TOC[*n*] bit is set, the appropriate bit TOS[*n*] is set. Since all the time-out registers are scanned sequentially, there can be a delay before the TOS[*n*] bit is set.

16.7.2.1 Message-Object Time-Out Registers (MOTO)

This register holds the time-out value of the TSC by which the corresponding mailbox data should be successfully transmitted or received. Each mailbox has its own MOTO register.

Figure 16-30. Message-Object Time-Out Registers (MOTO)

31 … 0
MOTO31:0
R/W-x

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

Table 16-27. Message-Object Time-Out Registers (MOTO) Field Descriptions

Bit	Field	Value	Description
31:0	MOTO31:0		Message object time-out register. Limit-value of the time-stamp counter (TSC) to actually transmit or receive the message.

16.7.2.2 Time-Out Control Register (CANTOC)

This register controls whether or not time-out functionality is enabled for a given mailbox.

Figure 16-31. Time-Out Control Register (CANTOC)

31 … 0
TOC31:0
R/W-0

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

Table 16-28. Time-Out Control Register (CANTOC) Field Descriptions

Bit	Field	Value	Description
31:0	TOC31:0		Time-out control register
		1	The TOC[*n*] bit must be set by the CPU to enable the time-out function for mailbox *n*. Before setting the TOC[*n*] bit, the corresponding MOTO register should be loaded with the time-out value relative to TSC.
		0	The time-out function is disabled. The TOS[*n*] flag is never set.

16.7.2.3 Time-Out Status Register (CANTOS)

This register holds the status information of mailboxes that have timed out.

Figure 16-32. Time-Out Status Register (CANTOS)

31	0
TOS31:0	
R/C-0	

LEGEND: R/C = Read/Clear; -*n* = value after reset

Table 16-29. Time-Out Status Register (CANTOS) Field Descriptions

Bit	Field	Value	Description
31:0	TOS 31:0		Time-out status register
		1	Mailbox[*n*] has timed out. The value in the TSC register is larger or equal to the value in the time-out register that corresponds to mailbox *n* and the TOC[*n*] bit is set.
		0	No time-out occurred or it is disabled for that mailbox.

The TOS*n* bit is set when all three of the following conditions are met:

1. The TSC value is greater than or equal to the value in the time-out register (MOTOn).
2. The TOC*n* bit is set.
3. The TRS*n* bit is set.

The time-out registers are implemented as a RAM. The state machine scans all the time-out registers and compares them to the time stamp counter value. Since all the time out registers are scanned sequentially, it is possible that even though a transmit mailbox has timed out, the TOS*n* bit is not set. This can happen when the mailbox succeeded in transmitting and clearing the TRS*n* bit before the state machine scans the time-out register of that mailbox. This is true for the receive mailbox as well. In this case, the RMP*n* bit can be set to 1 by the time the state machine scans the time-out register of that mailbox. However, the receive mailbox probably did not receive the message before the time specified in the time-out register.

16.7.3 Behavior/Usage of MTOF0/1 Bit in User Applications

The MTOF0/1 bit is automatically cleared by the CPK (along with the TOS*n* bit) upon transmission/reception by the mailbox, which asserted this flag in the first place. It can also be cleared by the user (via the CPU). On a time-out condition, the MTOF0/1 bit (and the TOS.*n* bit) is set. On an (eventual) successful communication, these bits are automatically cleared by the CPK. Following are the possible behaviors/usage for the MTOF0/1 bit:

1. Time-out condition occurs. Both MTOF0/1 bit and TOS.*n* bits are set. Communication is never successful; i.e., the frame was never transmitted (or received). An interrupt is asserted. Application handles the issue and eventually clears both MTOF0/1 bit and TOS.*n* bit.
2. Time-out condition occurs. Both MTOF0/1 bit and TOS.*n* bits are set. However, communication is eventually successful; i.e., the frame gets transmitted (or received). Both MTOF0/1 bit and TOS.*n* bits are cleared automatically by the CPK. An interrupt is still asserted because, the interrupt occurrence was recorded in the PIE module. When the ISR scans the GIF register, it doesn't see the MTOF0/1 bit set. This is the phantom interrupt scenario. Application merely returns to the main code.
3. Time-out condition occurs. Both MTOF0/1 bit and TOS.*n* bits are set. While executing the ISR pertaining to time-out, communication is successful. This situation must be handled carefully. The application should not re-transmit a mailbox if the mailbox is sent between the time the interrupt is asserted and the time the ISR is attempting to take corrective action. One way of doing this is to poll the TM/RM bits in the GSR register. These bits indicate if the CPK is currently transmitting/receiving. If that is the case, the application should wait till the communication is over and then check the TOS.*n* bit again. If the communication is still not successful, then the application should take the corrective action.

16.8 Mailbox Layout

The following four 32-bit registers comprise each mailbox:

- MSGID − Stores the message ID
- MSGCTRL − Defines number of bytes, transmission priority and remote frames
- CANMDL − 4 bytes of data
- CANMDH − 4 bytes of data

16.8.1 Message Identifier Register (MSGID)

This register contains the message ID and other control bits for a given mailbox.

Figure 16-33. Message Identifier Register (MSGID) Register

31	30	29	28 – 0
IDE	AME	AAM	ID[28:0]
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 = indeterminate
Note: This register can be written only when mailbox n is disabled (CANME[n] (CANME.31-0) = 0).

Table 16-30. Message Identifier Register (MSGID) Field Descriptions

Bit	Field	Value	Description
31	IDE		Identifier extension bit. The characteristics of the IDE bit changes according to the value of the AMI bit. When AMI = 1: 1. The IDE bit of the receive mailbox is a "don't care." The IDE bit of the receive mailbox is overwritten by the IDE bit of the transmitted message. 2. The filtering criterion must be satisfied in order to receive a message. 3. The number of bits to be compared is a function of the value of the IDE bit of the transmitted message. When AMI = 0: 1. The IDE bit of the receive mailbox determines the number of bits to be compared. 2. Filtering is not applicable. The MSGIDs must match bit-for-bit in order to receive a message. When AMI = 1: IDE = 1: The RECEIVED message had an extended identifier IDE = 0: The RECEIVED message had a standard identifier When AMI = 0: IDE = 1: The message TO BE RECEIVED must have an extended identifier IDE = 0: The message TO BE RECEIVED must have a standard identifier.
30	AME		Acceptance mask enable bit. AME is only used for receive mailboxes. It must not be set for automatic reply (AAM[*n*]=1, CANMD[*n*]=0) mailboxes, otherwise the mailbox behavior is undefined. This bit is not modified by a message reception.
		1	The corresponding acceptance mask is used.
		0	No acceptance mask is used, all identifier bits must match to receive the message
29	AAM		Auto answer mode bit. This bit is only valid for message mailboxes configured as transmit. For receive mailboxes, this bit has no effect: the mailbox is always configured for normal receive operation. This bit is not modified by a message reception.
		1	Auto answer mode. If a matching remote request is received, the CAN module answers to the remote request by sending the contents of the mailbox.
		0	Normal transmit mode. The mailbox does not reply to remote requests. The reception of a remote request frame has no effect on the message mailbox.

Table 16-30. Message Identifier Register (MSGID) Field Descriptions (continued)

Bit	Field	Value	Description
28:0	ID[28:0]		Message identifier
		1	In standard identifier mode, if the IDE bit (MSGID.31) = 0, the message identifier is stored in bits ID.28:18. In this case, bits ID.17:0 have no meaning.
		0	In extended identifier mode, if the IDE bit (MSGID.31) = 1, the message identifier is stored in bits ID.28:0.

16.8.2 CPU Mailbox Access

Write accesses to the identifier can only be accomplished when the mailbox is disabled (CANME[*n*] (CANME.31-0) = 0). During access to the data field, it is critical that the data does not change while the CAN module is reading it. Hence, a write access to the data field is disabled for a receive mailbox.

For send mailboxes, an access is usually denied if the TRS (TRS.31-0) or the TRR (TRR.31-0) flag is set. In these cases, an interrupt can be asserted. A way to access those mailboxes is to set CDR (MC.8) before accessing the mailbox data.

After the CPU access is finished, the CPU must clear the CDR flag by writing a 0 to it. The CAN module checks for that flag before and after reading the mailbox. If the CDR flag is set during those checks, the CAN module does not transmit the message but continues to look for other transmit requests. The setting of the CDR flag also stops the write-denied interrupt (WDI) from being asserted.

16.8.3 Message-Control Register (MSGCTRL)

For a transmit mailbox, this register specifies the number of bytes to be transmitted and the transmission priority. It also specifies the remote-frame operation.

NOTE: As part of the CAN module initialization process, all the bits of the MSGCTRL*n* registers must first be initialized to zero before proceeding to initialize the various bit fields to the desired values.

Figure 16-34. Message-Control Register (MSGCTRL)

31–16
Reserved
R-0

15–13	12–8	7–5	4	3–0
Reserved	TPL	Reserved	RTR	DLC
R-0	RW-x	R-0	RW-x	RW-x

LEGEND: RW = Read any time, write when mailbox is disabled or configured for transmission; -*n* = value after reset; x = indeterminate
Note: The register MSGCTRL(*n*) can only be written if mailbox *n* is configured for transmission (CANMD[*n*] (CANMD.31-0)=0) or if the mailbox is disabled (CANME[*n*] (CANME.31-0) =0).

Table 16-31. Message-Control Register (MSGCTRL) Field Descriptions

Bit	Field	Value	Description
31:13	Reserved		Reserved
12:8	TPL.4:0		Transmit-priority level. This 5-bit field defines the priority of this mailbox as compared to the other 31 mailboxes. The highest number has the highest priority. When two mailboxes have the same priority, the one with the higher mailbox number is transmitted. TPL applies only for transmit mailboxes. TPL is not used in SCC-mode.
7:5	Reserved		Reserved
4	RTR		Remote-transmission-request bit
		1	For receive mailbox: If the TRS flag is set, a remote frame is transmitted and the corresponding data frame is received in the same mailbox. Once the remote frame is sent, the TRS bit of the mailbox is cleared by CAN. For transmit mailbox: If the TRS flag is set, a remote frame is transmitted, but the corresponding data frame has to be received in another mailbox.
		0	No remote frame is requested.
3:0	DLC 3:0		Data-length code. The number in these bits determines how many data bytes are sent or received. Valid value range is from 0 to 8. Values from 9 to 15 are not allowed.

16.8.4 Message Data Registers (CANMDL, CANMDH)

Eight bytes of the mailbox are used to store the data field of a CAN message. The setting of DBO (MC.10) determines the ordering of stored data. The data is transmitted or received from the CAN bus, starting with byte 0.

- When DBO (MC.10) = 1, the data is stored or read starting with the least significant byte of the CANMDL register and ending with the most significant byte of the CANMDH register.
- When DBO (MC.10) = 0, the data is stored or read starting with the most significant byte of the CANMDL register and ending with the least significant byte of the CANMDH register.

The registers CANMDL(*n*) and CANMDH(*n*) can be written only if mailbox *n* is configured for transmission (CANMD[*n*] (CANMD.31-0)=0) or the mailbox is disabled (CANME[*n*] (CANME.31-0)=0). If TRS[*n*] (TRS.31-0)=1, the registers CANMDL(*n*) and CANMDH(*n*) cannot be written, unless CDR (MC.8)=1, with MBNR (MC.4-0) set to *n*. These settings also apply for a message object configured in reply mode (AAM (MSGID.29)=1).

Figure 16-35. Message-Data-Low Register With DBO = 0 (CANMDL)

31 — 24	23 — 16	15 — 8	7 — 0
Byte 0	Byte 1	Byte 2	Byte 3

Figure 16-36. Message-Data-High Register With DBO = 0 (CANMDH)

31 — 24	23 — 16	15 — 8	7 — 0
Byte 4	Byte 5	Byte 6	Byte 7

Figure 16-37. Message-Data-Low Register With DBO = 1 (CANMDL)

31 — 24	23 — 16	15 — 8	7 — 0
Byte 3	Byte 2	Byte 1	Byte 0

Figure 16-38. Message-Data-High Register With DBO = 1 (CANMDH)

31 — 24	23 — 16	15 — 8	7 — 0
Byte 7	Byte 6	Byte 5	Byte 4

NOTE: The data field beyond the valid received data is modified by any message reception and is indeterminate.

16.9 Acceptance Filter

The identifier of the incoming message is first compared to the message identifier of the mailbox (which is stored in the mailbox). Then, the appropriate acceptance mask is used to mask out the bits of the identifier that should not be compared.

In the SCC-compatible mode, the global acceptance mask (GAM) is used for the mailboxes 6 to 15. An incoming message is stored in the highest numbered mailbox with a matching identifier. If there is no matching identifier in mailboxes 15 to 6, the incoming message is compared to the identifier stored in mailboxes 5 to 3 and then 2 to 0.

The mailboxes 5 to 3 use the local-acceptance mask LAM(3) of the SCC registers. The mailboxes 2 to 0 use the local-acceptance mask LAM(0) of the SCC registers. For specific uses, see Figure 16-39.

To modify the global acceptance mask register (CANGAM) and the two local-acceptance mask registers of the SCC, the CAN module must be set in the initialization mode (see Section 3.1).

Each of the 32 mailboxes of the eCAN has its own local-acceptance mask LAM(0) to LAM(31). There is no global-acceptance mask in the eCAN.

The selection of the mask to be used for the comparison depends on which mode (SCC or eCAN) is used.

16.9.1 Local-Acceptance Masks (CANLAM)

The local-acceptance filtering allows the user to locally mask (don't care) any identifier bits of the incoming message.

In the SCC, the local-acceptance-mask register LAM(0) is used for mailboxes 2 to 0. The local-acceptance-mask register LAM(3) is used for mailboxes 5 to 3. For the mailboxes 6 to 15, the global-acceptance-mask (CANGAM) register is used.

After a hardware or a software reset of the SCC module, CANGAM is reset to zero. After a reset of the eCAN, the LAM registers are not modified.

In the eCAN, each mailbox (0 to 31) has its own mask register, LAM(0) to LAM(31). An incoming message is stored in the highest numbered mailbox with a matching identifier.

Figure 16-39. Local-Acceptance-Mask Register (LAM*n*)

31	30 29	28 16
LAMI	Reserved	LAM*n*[28:16]
R/W-0	R/W-0	R/W-0

15 0
LAM*n*[15:0]
R/W-0

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

Table 16-32. Local-Acceptance-Mask Register (LAM*n*) Field Descriptions

Bit	Field	Value	Description
31	LAMI		Local-acceptance-mask identifier extension bit
		1	Standard and extended frames can be received. In case of an extended frame, all 29 bits of the identifier are stored in the mailbox and all 29 bits of the local-acceptance mask register are used for the filter. In case of a standard frame, only the first eleven bits (bits 28 to 18) of the identifier and the local-acceptance mask are used.
		0	The identifier extension bit stored in the mailbox determines which messages shall be received.
30:29	Reserved		Reads are undefined and writes have no effect.
28:0	LAM[28:0]		These bits enable the masking of any identifier bit of an incoming message.
		1	Accept a 0 or a 1 (don't care) for the corresponding bit of the received identifier.
		0	Received identifier bit value must match the corresponding identifier bit of the MSGID register.

You can locally mask any identifier bits of the incoming message. A 1 value means "don't care" or accept either a 0 or 1 for that bit position. A 0 value means that the incoming bit value must match identically to the corresponding bit in the message identifier.

If the local-acceptance mask identifier extension bit is set (LAMI = 1 => don't care) standard and extended frames can be received. An extended frame uses all 29 bits of the identifier stored in the mailbox and all 29 bits of local-acceptance mask register for the filter. For a standard frame only the first eleven bits (bit 28 to 18) of the identifier and the local-acceptance mask are used.

If the local-acceptance mask identifier extension bit is reset (LAMI = 0), the identifier extension bit stored in the mailbox determines the messages that are received.

16.10 CAN Module Initialization

The CAN module must be initialized before the utilization. Initialization is only possible if the module is in initialization mode. Figure 16-40 is a flow chart showing the process.

Programming CCR (CANMC.12) = 1 sets the initialization mode. The initialization can be performed only when CCE (CANES.4) = 1. Afterwards, the configuration registers can be written.

SCC mode only:

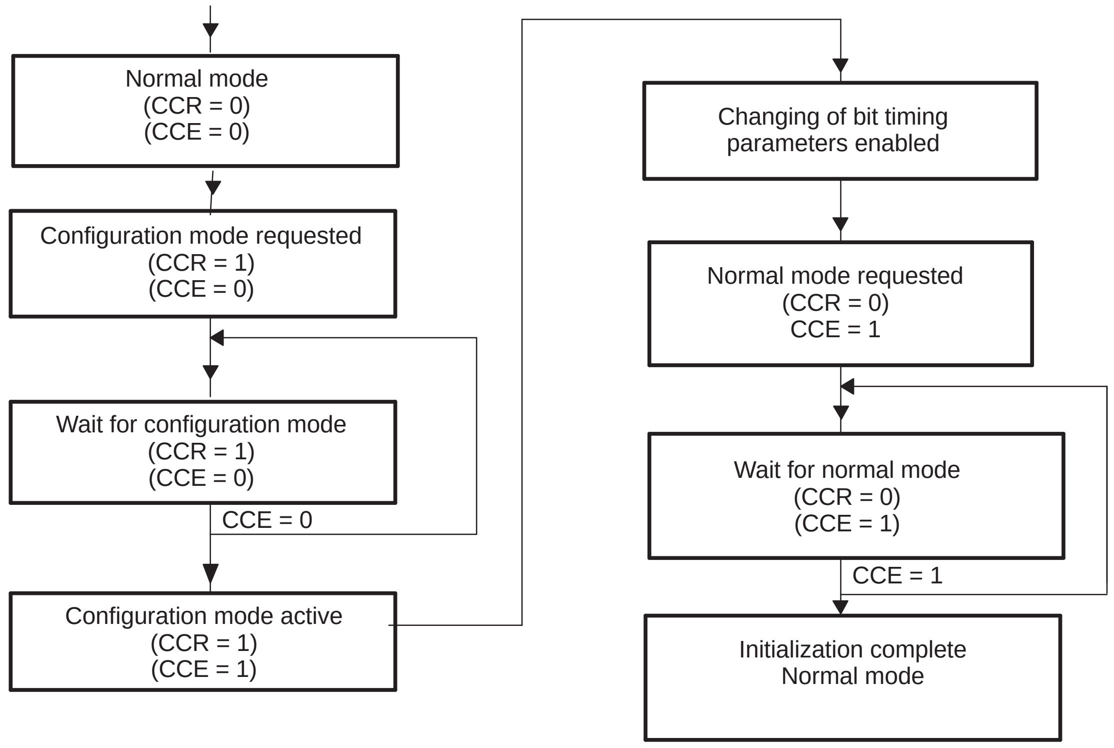
In order to modify the global acceptance mask register (CANGAM) and the two local acceptance mask registers [LAM(0) and LAM(3)], the CAN module also must be set in the initialization mode.

The module is activated again by programming CCR(CANMC.12) = 0.

After hardware reset, the initialization mode is active.

NOTE: If the CANBTC register is programmed with a zero value, or left with the initial value, the CAN module never leaves the initialization mode, i.e. CCE (CANES.4) bit remains at 1 when clearing the CCR bit.

Figure 16-40. Initialization Sequence



NOTE: The transition between initialization mode and normal mode and vice-versa is performed in synchronization with the CAN network. That is, the CAN controller waits until it detects a bus idle sequence (= 11 recessive bits) before it changes the mode. In the event of a stuck-to-dominant bus error, the CAN controller cannot detect a bus-idle condition and therefore is unable to perform a mode transition.

16.10.1 CAN Bit-Timing Configuration

The CAN protocol specification partitions the nominal bit time into four different time segments:

SYNC_SEG: This part of bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment. This segment is always 1 TIME QUANTUM (TQ).

PROP_SEG: This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation 'time on the bus line, the input comparator delay, and the output driver delay. This segment is programmable from 1 to 8 TIME QUANTA (TQ).

PHASE_SEG1: This phase is used to compensate for positive edge phase error. This segment is programmable from 1 to 8 TIME QUANTA (TQ) and can be lengthened by resynchronization.

PHASE_SEG2: This phase is used to compensate for negative edge phase error. This segment is programmable from 2 to 8 TIME QUANTA (TQ) and can be shortened by resynchronization.

In the eCAN module, the length of a bit on the CAN bus is determined by the parameters TSEG1 (BTC.6-3), TSEG2 (BTC.2-0), and BRP (BTC.23.16).

TSEG1 combines the two time segments PROP_SEG and PHASE_SEG1 as defined by the CAN protocol. TSEG2 defines the length of the time segment PHASE_SEG2.

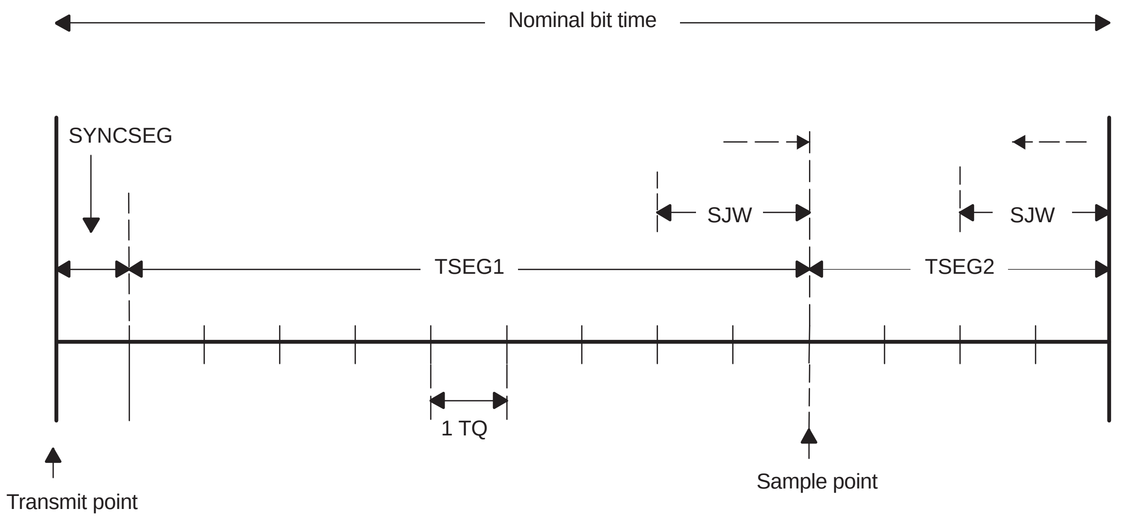
IPT (information processing time) corresponds to the time necessary for the processing of the bit read. IPT corresponds to two units of TQ.

The following bit timing rules must be fulfilled when determining the bit segment values:

- TSEG1(min) ≥ TSEG2

- IPT ≤ TSEG1 ≤ 16 TQ
- IPT ≤ TSEG2 ≤ 8 TQ
- IPT = 3/BRP (the resulting IPT has to be rounded up to the next integer value)
- 1 TQ ≤ SJW min[4 TQ, TSEG2] (SJW = Synchronization jump width)
- To utilize three-time sampling mode, BRP ≥ 5 has to be selected

Figure 16-41. CAN Bit Timing



A TSEG1 can be lengthened or TSEG2 shortened by the SJW

16.10.2 CAN Bit Rate Calculation

Bit-rate is calculated in bits per second as follows:

$$\text{Bit rate} = \frac{SYSCLKOUT/2}{BRP \times \text{Bit Time}}$$

Where bit-time is the number of time quanta (TQ) per bit. SYSCLKOUT is the CAN module system clock frequency, which is the same as the CPU clock frequency. BRP is the value of BRP$_{reg}$ + 1 (CANBTC.23-16).

Bit-time is defined as follows:

Bit-time = (TSEG1$_{reg}$ + 1) + (TSEG2$_{reg}$ + 1) + 1

In the above equation TESG1$_{reg}$ and TSEG2$_{reg}$ represent the actual values written in the corresponding fields in the CANBTC register. The parameters TSEG1$_{reg}$, TSEG2r$_{eg}$, SJW$_{reg}$, and BRP$_{reg}$ are automatically enhanced by 1 when the CAN module accesses these parameters. TSEG1, TSEG2 and SJW, represent the values as applicable per Figure 16-41.

Bit-time = TSEG1 + TSEG2 + 1

16.10.3 Bit Configuration Parameters for 30 -MHz CAN Clock

This section provides example values for the CANBTC bit fields for various CAN module clocks, bit rates and sampling points. Note that these values are for illustrative purposes only. In a real-world application, the propagation delay introduced by various entities such as the network cable, transceivers/ isolators must be taken into account before choosing the timing parameters.

Table 16-33 shows how the BRP$_{reg}$ field may be changed to achieve different bit rates with a BT of 15 for an 80% SP.

Table 16-33. BRP Field for Bit Rates (BT = 15, $TSEG1_{reg}$ = 10, $TSEG2_{reg}$ = 2, Sampling Point = 80%)

CAN Bus Speed	BRP	CAN Module Clock
1 Mbps	BRP_{reg} +1 = 2	15 MHz
500 kbps	BRP_{reg} +1 = 4	7.5 MHz
250 kbps	BRP_{reg} +1 = 8	3.75 MHz
125 kbps	BRP_{reg} +1 = 16	1.875 MHz
100 kbps	BRP_{reg} +1 = 20	1.5 MHz
50 kbps	BRP_{reg} +1 = 40	0.75 MHz

Table 16-34 shows how to achieve different sampling points with a BT of 15.

Table 16-34. Achieving Different Sampling Points With a BT of 15

$TSEG1_{reg}$	$TSEG2_{reg}$	SP
10	2	80%
9	3	73%
8	4	66%
7	5	60%

Table 16-35 shows how BRP_{reg} field may be changed to achieve different bit rates with a BT of 10 for an 80% sampling point.

Table 16-35. BRP Field for Bit Rates (BT = 10, $TSEG1_{reg}$ = 6, $TSEG2_{reg}$ = 1, Sampling Point = 80%)

CAN Bus Speed	BRP	CAN Module Clock
1 Mbps	BRP_{reg} +1 = 3	10 MHz
500 kbps	BRP_{reg} +1 = 6	5 MHz
250 kbps	BRP_{reg} +1 = 12	2.5 MHz
125 kbps	BRP_{reg} +1 = 24	1.25 MHz
100 kbps	BRP_{reg} +1 = 30	1 MHz
50 kbps	BRP_{reg} +1 = 60	0.5 MHz

16.10.4 Bit Configuration Parameters for 100 -MHz CAN Clock

Table 16-36 shows how the BRP_{reg} field may be changed to achieve different bit rates with a BT of 10 for an 80% SP.

Table 16-36. BRP Field for Bit Rates (BT = 10, $TSEG1_{reg}$ = 6, $TSEG2_{reg}$ = 1, Sampling Point = 80%)

CAN Bus Speed	BRP	CAN Module Clock
1 Mbps	BRP_{reg} +1 = 10	10 MHz
500 kbps	BRP_{reg} +1 = 20	5 MHz
250 kbps	BRP_{reg} +1 = 40	2.5 MHz
125 kbps	BRP_{reg} +1 = 80	1.25 MHz
100 kbps	BRP_{reg} +1 = 100	1 MHz
50 kbps	BRP_{reg} +1 = 200	0.5 MHz

Table 16-37 shows how to achieve different sampling points with a BT of 20.

Table 16-37. Achieving Different Sampling Points With a BT of 20

$TSEG1_{reg}$	$TSEG2_{reg}$	SP
15	2	85%
14	3	80%
13	4	75%
12	5	70%
11	6	65%
10	7	60%

NOTE: For a SYSCLKOUT of 60 MHz, the lowest bit-rate that can be achieved is 4.687 kbps.

16.10.5 EALLOW Protection

To protect against inadvertent modification, some critical registers/bits of the eCAN module are EALLOW protected. These registers/bits can be changed only if the EALLOW protection has been disabled. Following are the registers/ bits that are EALLOW protected in the eCAN module:

- CANMC[15..9] & MCR[7..6]
- CANBTC
- CANGIM
- MIM[31..0]
- TSC[31..0]
- IOCONT1[3]
- IOCONT2[3]

16.11 Steps to Configure eCAN

NOTE: This sequence must be done with EALLOW enabled.

The following steps must be performed to configure the eCAN for operation:

Step 1. Enable clock to the CAN module.

Step 2. Set the CANTX and the CANRX pins to CAN functions:

(a) Write CANTIOC.3:0 = 0x08

(b) Write CANRIOC.3:0 = 0x08

Step 3. After a reset, bit CCR (CANMC.12) and bit CCE (CANES.4) are set to 1. This allows the user to configure the bit-timing configuration register (CANBTC).
If the CCE bit is set (CANES.4 = 1), proceed to next step; otherwise, set the CCR bit (CANMC.12 = 1) and wait until CCE bit is set (CANES.4 = 1).

Step 4. Program the CANBTC register with the appropriate timing values. Make sure that the values TSEG1 and TSEG2 are not 0. If they are 0, the module does not leave the initialization mode.

Step 5. For the SCC, program the acceptance masks now. For example:
Write LAM(3) = 0x3C0000

Step 6. Program the master control register (CANMC) as follows:

(a) Clear CCR (CANMC.12) = 0

(b) Clear PDR (CANMC.11) = 0

(c) Clear DBO (CANMC.10) = 0

(d) Clear WUBA (CANMC.9)= 0

(e) Clear CDR (CANMC.8) = 0

(f) Clear ABO (CANMC.7) = 0

(g) Clear STM (CANMC.6) = 0

(h) Clear SRES (CANMC.5) = 0

(i) Clear MBNR (CANMC.4-0) = 0

Step 7. Initialize all bits of MSGCTRLn registers to zero.

Step 8. Verify the CCE bit is cleared (CANES.4 = 0), indicating that the CAN module has been configured.

This completes the setup for the basic functionality.

16.11.1 Configuring a Mailbox for Transmit

To transmit a message, the following steps need to be performed (in this example, for mailbox 1):

1. Clear the appropriate bit in the CANTRS register to 0:

   Clear CANTRS.1 = 0 (Writing a 0 to TRS has no effect; instead, set TRR.1 and wait until TRS.1 clears.) If the RTR bit is set, the TRS bit can send a remote frame. Once the remote frame is sent, the TRS bit of the mailbox is cleared by the CAN module. The same node can be used to request a data frame from another node.
2. Disable the mailbox by clearing the corresponding bit in the mailbox enable (CANME) register.

   Clear CANME.filter1 = 0
3. Load the message identifier (MSGID) register of the mailbox. Clear the AME (MSGID.30) and AAM (MSGID.29) bits for a normal send mailbox (MSGID.30 = 0 and MSGID.29 = 0). This register is usually not modified during operation. It can only be modified when the mailbox is disabled. For example:
   (a) Write MSGID(1) = 0x15AC0000
   (b) Write the data length into the DLC field of the message control field register (MSGCTRL.3:0). The RTR flag is usually cleared (MSGCTRL.4 = 0).
   (c) Set the mailbox direction by clearing the corresponding bit in the CANMD register.
   (d) Clear CANMD.1 = 0
4. Set the mailbox enable by setting the corresponding bit in the CANME register

   Set CANME.1 = 1

This configures mailbox 1 for transmit mode.

16.11.2 Transmitting a Message

To start a transmission (in this example, for mailbox:

1. Write the message data into the mailbox data field.
   (a) Since DBO (MC.10) is set to zero in the configuration section and MSGCTRL(1) is set to 2, the data are stored in the 2 MSBytes of CANMDL(1).
   (b) Write CANMDL(1) = xxxx0000h
2. Set the corresponding flag in the transmit request register (CANTRS.1 = 1) to start the transmission of the message. The CAN module now handles the complete transmission of the CAN message.
3. Wait until the transmit-acknowledge flag of the corresponding mailbox is set (TA.1 = 1). After a successful transmission, this flag is set by the CAN module.
4. The TRS flag is reset to 0 by the module after a successful or aborted transmission (TRS.1 = 0).
5. The transmit acknowledge must be cleared for the next transmission (from the same mailbox).
   (a) Set TA.1 = 1
   (b) Wait until read TA.1 is 0
6. To transmit another message in the same mailbox, the mailbox RAM data must be updated. Setting the TRS.1 flag starts the next transmission. Writing to the mailbox RAM can be half-word (16 bits) or full word (32 bits) but the module always returns 32-bit from even boundary. The CPU must accept all the 32 bits or part of it.

16.11.3 Configuring Mailboxes for Receive

To configure a mailbox to receive messages, the following steps must be performed (in this example, mailbox 3):

1. Disable the mailbox by clearing the corresponding bit in the mailbox enable (CANME) register.
   Clear CANME.3 = 0
2. Write the selected identifier into the corresponding MSGID register. The identifier extension bit must be configured to fit the expected identifier. If the acceptance mask is used, the acceptance mask enable (AME) bit must be set (MSGID.30 = 1). For example:
   Write MSGID(3) = 0x4f780000

3. If the AME bit is set to 1, the corresponding acceptance mask must be programmed.
   Write LAM(3) = 0x03c0000.
4. Configure the mailbox as a receive mailbox by setting the corresponding flag in the mailbox direction register (CANMD.3 = 1). Make sure no other bits in this register are affected by this operation.
5. If data in the mailbox is to be protected, the overwrite protection control register (CANOPC) should be programmed now. This protection is useful if no message must be lost. If OPC is set, the software has to make sure that an additional mailbox (buffer mailbox) is configured to store ’overflow’ messages. Otherwise messages can be lost without notification.
   Write OPC.3 = 1
6. Enable the mailbox by setting the appropriate flag in the mailbox enable register (CANME). This should be done by reading CANME, and writing back (CANME |= 0x0008) to make sure no other flag has changed accidentally.

The object is now configured for the receive mode. Any incoming message for that object is handled automatically.

16.11.4 Receiving a Message

This example uses mailbox 3. When a message is received, the corresponding flag in the receive message pending register (CANRMP) is set to 1 and an interrupt can be initiated. The CPU can then read the message from the mailbox RAM. Before the CPU reads the message from the mailbox, it should first clear the RMP bit (RMP.3 = 1). The CPU should also check the receive message lost flag RML.3 = 1. Depending on the application, the CPU has to decide how to handle this situation.

After reading the data, the CPU needs to check that the RMP bit has not been set again by the module. If the RMP bit has been set to 1, the data may have been corrupted. The CPU needs to read the data again because a new message was received while the CPU was reading the old one.

16.11.5 Handling of Overload Situations

If the CPU is not able to handle important messages fast enough, it may be advisable to configure more than one mailbox for that identifier. Here is an example where the objects 3, 4, and 5 have the same identifier and share the same mask. For the SCC, the mask is LAM(3). For the eCAN, each object has its own LAM: LAM(3), LAM(4), and LAM(5), all of which need to be programmed with the same value.

To make sure that no message is lost, set the OPC flag for objects 4 and 5, which prevents unread messages from being overwritten. If the CAN module must store a received message, it first checks mailbox 5. If the mailbox is empty, the message is stored there. If the RMP flag of object 5 is set (mailbox occupied), the CAN module checks the condition of mailbox 4. If that mailbox is also busy, the module checks in mailbox 3 and stores the message there since the OPC flag is not set for mailbox 3. If mailbox 3 contents have not been previously read, it sets the RML flag of object 3, which can initiate an interrupt.

It is also advisable to have object 4 generate an interrupt telling the CPU to read mailboxes 4 and 5 at once. This technique is also useful for messages that require more than 8 bytes of data (i.e., more than one message). In this case, all data needed for the message can be collected in the mailboxes and be read at once.

16.12 Handling of Remote Frame Mailboxes

There are two functions for remote frame handling. One is a request by the module for data from another node, the other is a request by another node for data that the module needs to answer.

16.12.1 Requesting Data From Another Node

In order to request data from another node, the object is configured as receive mailbox. Using object 3 for this example, the CPU needs to do the following:

1. Set the RTR bit in the message control field register (CANMSGCTRL) to 1.
   Write MSGCTRL(3) = 0x12
2. Write the correct identifier into the message identifier register (MSGID).
   Write MSGID(3) = 0x4F780000

3. Set the CANTRS flag for that mailbox. Since the mailbox is configured as receive, it only sends a remote request message to the other node.
   Set CANTRS.3 = 1
4. The module stores the answer in that mailbox and sets the RMP bit when it is received. This action can initiate an interrupt. Also, make sure no other mailbox has the same ID.
   Wait for RMP.3 = 1
5. Read the received message.

16.12.2 Answering a Remote Request

1. Configure the object as as a transmit mailbox.
2. Set the auto answer mode (AAM) (MSGID.29) bit in the MSGID register before the mailbox is enabled.
   MSGID(1) = 0x35AC0000
3. Update the data field.
   MDL, MDH(1) = xxxxxxxxh
4. Enable the mailbox by setting the CANME flag to 1.
   CANME.1 = 1
   When a remote request is received from another node, the TRS flag is set automatically and the data is transmitted to that node. The identifier of the received message and the transmitted message are the same.
   After transmission of the data, the TA flag is set. The CPU can then update the data.
   Wait for TA.1 = 1

16.12.3 Updating the Data Field

To update the data of an object that is configured in auto answer mode, the following steps need to be performed. This sequence can also be used to update the data of an object configured in normal transmission with TRS flag set.

1. Set the change data request (CDR) (MC.8) bit and the mailbox number (MBNR) of that object in the master control register (CANMC). This tells the CAN module that the CPU wants to change the data field. For example, for object 1:
   Write MC = 0x0000101
2. Write the message data into the mailbox data register. For example:
   Write CANMDL(1) = xxxx0000h
3. Clear the CDR bit (MC.8) to enable the object.
   Write MC = 0x00000000

16.13 Interrupts

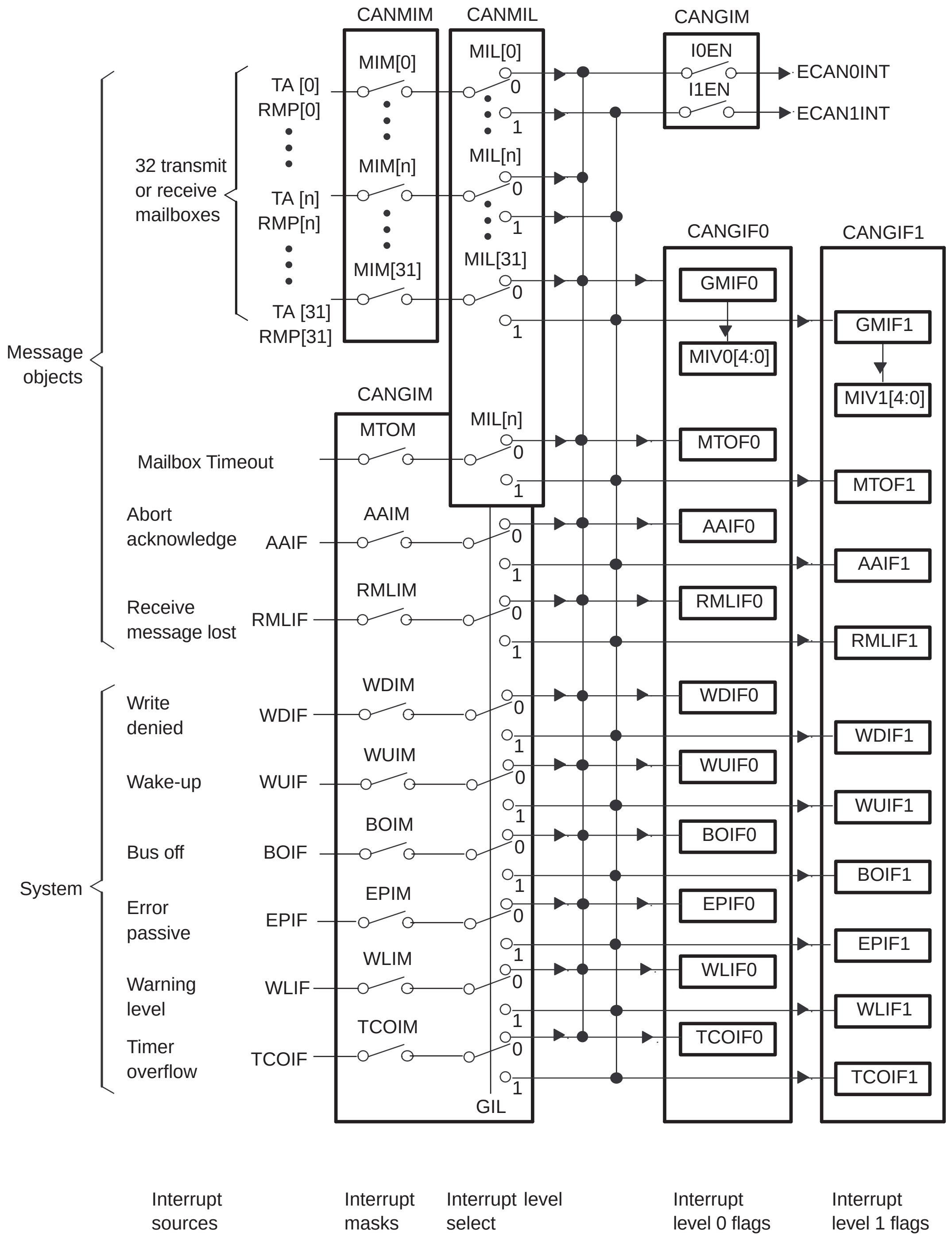
There are two different types of interrupts. One type of interrupt is a mailbox related interrupt, for example, the receive-message-pending interrupt or the abort-acknowledge interrupt. The other type of interrupt is a system interrupt that handles errors or system-related interrupt sources, for example, the error-passive interrupt or the wake-up interrupt. See Figure 16-42.

The following events can initiate one of the two interrupts:

- Mailbox interrupts
  - Message reception interrupt: a message was received
  - Message transmission interrupt: a message was transmitted successfully
  - Abort-acknowledge interrupt: a pending transmission was aborted
  - Received-message-lost interrupt: an old message was overwritten by a new one (before the old message was read)
  - Mailbox timeout interrupt (eCAN mode only): one of the messages was not transmitted or received within a predefined time frame
- System interrupts
  - Write-denied interrupt: the CPU tried to write to a mailbox but was not allowed to

– Wake-up interrupt: this interrupt is generated after a wake up
– Bus-off interrupt: the CAN module enters the bus-off state
– Error-passive interrupt: the CAN module enters the error-passive mode
– Warning level interrupt: one or both error counters are greater than or equal to 96
– Time-stamp counter overflow interrupt (eCAN only): the time-stamp counter had an overflow

Figure 16-42. Interrupts Scheme



16.13.1 Interrupts Scheme

The interrupt flags are set if the corresponding interrupt condition occurred. The system interrupt flags are set depending on the setting of GIL (CANGIM.2). If set, the global interrupts set the bits in the CANGIF1 register, otherwise they set in the CANGIF0 register.

The GMIF0/GMIF1(CANGIF0.15/CANGIF1.15) bit is set depending on the setting of the MIL[n] bit that corresponds to the mailbox originating that interrupt. If the MIL[n] bit is set, the corresponding mailbox interrupt flag MIF[n] sets the GMIF1 flag in the CANGIF1 register, otherwise, it sets the GMIF0 flag.

If all interrupt flags are cleared and a new interrupt flag is set, the CAN module interrupt output line (ECAN0INT or ECAN1INT) is activated if the corresponding interrupt mask bit is set. The interrupt line stays active until the interrupt flag is cleared by the CPU by writing a 1 to the appropriate bit.

The GMIF0 (CANGIF0.15) or GMIF1 (CANGIF0.15) bit must be cleared by writing a 1 to the appropriate bit in the CANTA register or the CANRMP register (depending on mailbox configuration) and cannot be cleared in the CANGIF0/CANGIF1 register.

After clearing one or more interrupt flags, and one or more interrupt flags are still pending, a new interrupt is generated. The interrupt flags are cleared by writing a 1 to the corresponding bit location. If the GMIF0 or GMIF1 bit is set, the mailbox interrupt vector MIV0 (CANGIF0.4-0) or MIV1 (CANGIF1.4-0) indicates the mailbox number of the mailbox that caused the setting of the GMIF0/1. It always displays the highest mailbox interrupt vector assigned to that interrupt line.

16.13.2 Mailbox Interrupt

Each of the 32 mailboxes in the eCAN or the 16 mailboxes in the SCC can initiate an interrupt on one of the two interrupt output lines 1 or 0. These interrupts can be receive or transmit interrupts depending on the mailbox configuration.

There is one interrupt mask bit (MIM[n]) and one interrupt level bit (MIL[n]) dedicated to each mailbox. To generate a mailbox interrupt upon a receive/transmit event, the MIM bit has to be set. If a CAN message is received (RMP[n]=1) in a receive mailbox or transmitted (TA[n]=1) from a transmit mailbox, an interrupt is asserted. If a mailbox is configured as remote request mailbox (CANMD[n]=1, MSGCTRL.RTR=1), an interrupt occurs upon reception of the reply frame. A remote reply mailbox generates an interrupt upon successful transmission of the reply frame (CANMD[n]=0, MSGID.AAM=1).

The setting of the RMP[n] bit or the TA[n] bit also sets the GMIF0/GMIF1 (GIF0.15/GIF1.15) flag in the GIF0/GIF1 register if the corresponding interrupt mask bit is set. The GMIF0/GMIF1 flag then generates an interrupt and the corresponding mailbox vector (= mailbox number) can be read from the bit field MIV0/MIV1 in the GIF0/GIF1 register. If more than one mailbox interrupts are pending, the actual value of MIV0/MIV1 reflects the highest priority interrupt vector. The interrupt generated depends on the setting in the mailbox interrupt level (MIL) register.

The abort acknowledge flag (AA[n]) and the abort acknowledge interrupt flag (AAIF) in the GIF0/GIF1 register are set when a transmit message is aborted by setting the TRR[n] bit. An interrupt is asserted upon transmission abortion if the mask bit AAIM in the GIM register is set. Clearing the AA[n] flag(s) clears the AAIF0/AAIF1 flag.

A lost receive message is notified by setting the receive message lost flag RML[n] and the receive message lost interrupt flag RMLIF0/RMLIF1in the GIF0/GIF1 register. If an interrupt shall be generated upon the lost receive message event, the receive message lost interrupt mask bit (RMLIM) in the GIM register has to be set. Clearing the RML[n] flag does not reset the RMLIF0/RMLIF1 flag. The interrupt flag has to be cleared separately.

Each mailbox of the eCAN (in eCAN mode only) is linked to a message- object, time-out register (MOTO). If a time-out event occurs (TOS[n] = 1), a mailbox timeout interrupt is asserted to one of the two interrupt lines if the mailbox timeout interrupt mask bit (MTOM) in the CANGIM register is set. The interrupt line for mailbox timeout interrupt is selected in accordance with the mailbox interrupt level (MIL[n]) of the concerned mailbox.

16.13.3 Interrupt Handling

The CPU is interrupted by asserting one of the two interrupt lines. After handling the interrupt, which should generally also clear the interrupt source, the interrupt flag must be cleared by the CPU. To do this, the interrupt flag must be cleared in the CANGIF0 or CANGIF1 register. This is generally done by writing a 1 to the interrupt flag. There are some exceptions to this as stated in Table 16-38. This also releases the interrupt line if no other interrupt is pending.

Table 16-38. eCAN Interrupt Assertion/Clearing[1]

Interrupt Flag	Interrupt Condition	GIF0/GIF1 Determination	Clearing Mechanism
WLIFn	One or both error counters are >= 96	GIL bit	Cleared by writing a 1 to it
EPIFn	CAN module has entered “error passive” mode	GIL bit	Cleared by writing a 1 to it
BOIFn	CAN module has entered “bus-off” mode	GIL bit	Cleared by writing a 1 to it
RMLIFn	An overflow condition has occurred in one of the receive mailboxes.	GIL bit	Cleared by clearing the set RMPn bit.
WUIFn	CAN module has left the local power-down mode	GIL bit	Cleared by writing a 1 to it
WDIFn	A write access to a mailbox was denied	GIL bit	Cleared by writing a 1 to it
AAIFn	A transmission request was aborted	GIL bit	Cleared by clearing the set AAn bit.
GMIFn	One of the mailboxes successfully transmitted/received a message	MILn bit	Cleared by appropriate handling of the interrupt causing condition. Cleared by writing a 1 to the ap-propriate bit in CANTA or CANRMP registers
TCOFn	The MSB of the the TSC has changed from 0 to 1	GIL bit	Cleared by writing a 1 to it
MTOFn	One of the mailboxes did not transmit/receive within the specified time frame.	MILn bit	Cleared by clearing the set TOSn bit.

[1] Key to interpreting the table above:
1) Interrupt flag: This is the name of the interrupt flag bit as applicable to CANGIF0/CANGIF1 registers.
2) Interrupt condition: This column illustrates the conditions that cause the interrupt to be asserted.
3) GIF0/GIF1 determination: Interrupt flag bits can be set in either CANGIF0 or CANGIF1 registers. This is determined by either the GIL bit in CANGIM register or MILn bit in the CANMIL register, depending on the interrupt under consideration. This column illustrates whether a particular interrupt is dependant on GIL bit or MILn bit.
4) Clearing mechanism: This column explains how a flag bit can be cleared. Some bits are cleared by writing a 1 to it. Other bits are cleared by manipulating some other bit in the CAN control register.

16.13.3.1 Configuring for Interrupt Handling

To configure for interrupt handling, the mailbox interrupt level register (CANMIL), the mailbox interrupt mask register (CANMIM), and the global interrupt mask register (CANGIM) need to be configured. The steps to do this are described below:

1. Write the CANMIL register. This defines whether a successful transmission asserts interrupt line 0 or 1. For example, CANMIL = 0xFFFFFFFF sets all mailbox interrupts to level 1.
2. Configure the mailbox interrupt mask register (CANMIM) to mask out the mailboxes that should not cause an interrupt. This register could be set to 0xFFFFFFFF, which enables all mailbox interrupts. Mailboxes that are not used do not cause any interrupts anyhow.
3. Now configure the CANGIM register. The flags AAIM, WDIM, WUIM, BOIM, EPIM, and WLIM (GIM.14-9) should always be set (enabling these interrupts). In addition, the GIL (GIM.2) bit can be set to have the global interrupts on another level than the mailbox interrupts. Both the I1EN (GIM.1) and I0EN (GIM.0) flags should be set to enable both interrupt lines. The flag RMLIM (GIM.11) can also be set depending on the load of the CPU.

This configuration puts all mailbox interrupts on line 1 and all system interrupts on line 0. Thus, the CPU can handle all system interrupts (which are always serious) with high priority, and the mailbox interrupts (on the other line) with a lower priority. All messages with a high priority can also be directed to the interrupt line 0.

16.13.3.2 Handling Mailbox Interrupts

There are three interrupt flags for mailbox interrupts. These are listed below:

GMIF0/GMIF1: One of the objects has received or transmitted a message. The number of the mailbox is in MIV0/MIV1(GIF0.4-0/GIF1.4-0). The normal handling routine is as follows:

1. Do a half-word read on the GIF register that caused the interrupt. If the value is negative, a mailbox caused the interrupt. Otherwise, check the AAIF0/AAIF1 (GIF0.14/GIF1.14) bit (abort-acknowledge interrupt flag) or the RMLIF0/RMLIF1 (GIF0.11/GIF1.11) bit (receive-message-lost interrupt flag). Otherwise, a system interrupt has occurred. In this case, each of the system-interrupt flags must be checked.
2. If the RMLIF (GIF0.11) flag caused the interrupt, the message in one of the mailboxes has been overwritten by a new one. This should not happen in normal operation. The CPU needs to clear that flag by writing a 1 to it. The CPU must check the receive-message-lost register (RML) to find out which mailbox caused that interrupt. Depending on the application, the CPU has to decide what to do next. This interrupt comes together with an GMIF0/GMIF1 interrupt.
3. If the AAIF (GIF.14) flag caused the interrupt, a send transmission operation was aborted by the CPU. The CPU should check the abort acknowledge register (AA.31-0) to find out which mailbox caused the interrupt and send that message again if requested. The flag must be cleared by writing a 1 to it.
4. If the GMIF0/GMIF1 (GIF0.15/GIF1.15) flag caused the interrupt, the mailbox number that caused the interrupt can be read from the MIV0/MIV1 (GIF0.4-0/GIF1.4-0) field. This vector can be used to jump to a location where that mailbox is handled. If it is a receive mailbox, the CPU should read the data as described above and clear the RMP.31-0 flag by writing a 1 to it. If it is a send mailbox, no further action is required, unless the CPU needs to send more data. In this case, the normal send procedure as described above is necessary. The CPU needs to clear the transmit acknowledge bit (TA.31-0) by writing a 1 to it.

16.13.3.3 Interrupt Handling Sequence

In order for the CPU core to recognize and service CAN interrupts, the following must be done in any CAN ISR:

1. The flag bit in the CANGIF0/CANGIF1 register which caused the interrupt in the first place must be cleared. There are two kinds of bits in these registers:
   (a) the very same bit that needs to be cleared. The following bits fall under this category: TCOFn, WDIFn, WUIFn, BOIFn, EPIFn, WLIFn
   (b) The second group of bits are cleared by writing to the corresponding bits in the associated registers. The following bits fall under this category: MTOFn, GMIFn, AAIFn, RMLIFn
      (i) The MTOFn bit is cleared by clearing the corresponding bit in the TOS register. For example, if mailbox 27 caused a time-out condition due to which the MTOFn bit was set, the ISR (after taking appropriate actions for the timeout condition) needs to clear the TOS27 bit in order to clear the MTOFn bit.
      (ii) The GMIFn bit is cleared by clearing the appropriate bit in TA or RMP register. For example, if mailbox 19 has been configured as a transmit mailbox and has completed a transmission, TA19 is set, which in turn sets GMIFn. The ISR (after taking appropriate actions) needs to clear the TA19 bit in order to clear the GMIFn bit. If mailbox 8 has been configured as a receive mailbox and has completed a reception, RMP8 is set, which in turn sets GMIFn. The ISR (after taking appropriate actions) needs to clear the RMP8 bit in order to clear the GMIFn bit.
      (iii) The AAIFn bit is cleared by clearing the corresponding bit in the AA register. For example, if mailbox 13's transmission was aborted due to which the AAIFn bit was set, the ISR needs to clear the AA13 bit in order to clear the AAIFn bit.
      (iv) The RMLIFn bit is cleared by clearing the corresponding bit in the RMP register. For example, if mailbox 13's message was overwritten due to which the RMLIFn bit was set, the ISR needs to

clear the RMP13 bit in order to clear the RMLIFn bit.

2. The PIEACK bit corresponding to the CAN module must be written with a 1, which can be accomplished with the following C language statement:

   ```
   PieCtrlRegs.PIEACK.bit.ACK9 = 1; // Enables PIE to drive a pulse into the CPU
   ```

3. The interrupt line into the CPU corresponding to the CAN module must be enabled, which can be accomplished with the following C language statement:

   ```
   IER |= 0x0100; // Enable INT9
   ```

4. The CPU interrupts must be enabled globally by clearing the INTM bit.

16.14 CAN Power-Down Mode

A local power-down mode has been implemented where the CAN module internal clock is de-activated by the CAN module itself.

16.14.1 Entering and Exiting Local Power-Down Mode

During local power-down mode, the clock of the CAN module is turned off (by the CAN module itself) and only the wake-up logic is still active. The other peripherals continue to operate normally.

The local power-down mode is requested by writing a 1 to the PDR (CANMC.11) bit, allowing transmission of any packet in progress to complete. After the transmission is completed, the status bit PDA (CANES.3) is set. This confirms that the CAN module has entered the power-down mode.

The value read on the CANES register is 0x08 (PDA bit is set). All other register read accesses deliver the value 0x00.

The module leaves the local power-down mode when the PDR bit is cleared or if any bus activity is detected on the CAN bus line (if the wake-up-on bus activity is enabled).

The automatic wake-up-on bus activity can be enabled or disabled with the configuration bit WUBA of CANMC register. If there is any activity on the CAN bus line, the module begins its power-up sequence. The module waits until it detects 11 consecutive recessive bits on the CANRX pin and then it goes bus-active.

NOTE: The first CAN message, which initiates the bus activity, cannot be received. This means that the first message received in power-down and automatic wake-up mode is lost.

After leaving the sleep mode, the PDR and PDA bits are cleared. The CAN error counters remain unchanged.

If the module is transmitting a message when the PDR bit is set, the transmission is continued until a successful transmission, a lost arbitration, or an error condition on the CAN bus line occurs. Then, the PDA bit is activated so the module causes no error condition on the CAN bus line.

To implement the local power-down mode, two separate clocks are used within the CAN module. One clock stays active all the time to ensure power-down operation (i.e., the wake-up logic and the write and read access to the PDA (CANES.3) bit). The other clock is enabled depending on the setting of the PDR bit.

16.14.2 Precautions for Entering and Exiting Device Low-Power Modes (LPM)

The 28x DSP features two low-power modes, STANDBY and HALT, in which the peripheral clocks are turned off. Since the CAN module is connected to multiple nodes across a network, you must take care before entering and exiting device low-power modes such as STANDBY and HALT. A CAN packet must be received in full by all the nodes; therefore, if transmission is aborted half-way through the process, the aborted packet would violate the CAN protocol resulting in all the nodes generating error frames. The node exiting LPM should do so unobtrusively. For example, if a node exits LPM when there is traffic on the CAN bus it could “see” a truncated packet and disturb the bus with error frames.

The following points must be considered before entering a device low-power mode:

1. The CAN module has completed the transmission of the last packet requested.

2. The CAN module has signaled to the CPU that it is ready to enter LPM.

In other words, device low-power modes should be entered into only after putting the CAN module in local power-down mode.

16.14.3 Enabling/Disabling Clock to the CAN Module

The CAN module cannot be used unless the clock to the module is enabled. It is enabled or disabled by using bit 14 of the PCLKCR register. This bit is useful in applications that do not use the CAN module at all. In such applications, the CAN module clock can be permanently turned off, resulting in some power saving. This bit is not intended to put the CAN module in low-power mode and should not be used for that purpose. Like all other peripherals, clock to the CAN module is disabled upon reset.

16.14.4 Possible Failure Modes External to the CAN Controller Module

This section lists some potential failure modes in a CAN based system. The failure modes listed are external to the CAN controller and hence, need to be evaluated at the system level.

- CAN_H and CAN_ L shorted together
- CAN_H and/or CAN_ L shorted to ground
- CAN_H and/or CAN_ L shorted to supply
- Failed CAN transceiver
- Electrical disturbance on CAN bus

Universal Serial Bus (USB) Controller

This chapter discusses the features and functions of the universal serial bus (USB) controller.

Topic **Page**

17.1 Introduction

The USB controller operates as a full-speed function controller during point-to-point communications with the USB host device. The controller complies with the USB 2.0 standard, which includes SUSPEND and RESUME signaling, eight endpoints comprised of two hard-wired for control transfers (one endpoint for IN and one endpoint for OUT) plus six endpoints defined by firmware, along with a dynamic sizable FIFO support multiple packet queueing. DMA access to the FIFO allows minimal interference from system software. Software-controlled connect and disconnect allows flexibility during USB device startup.

17.2 Features

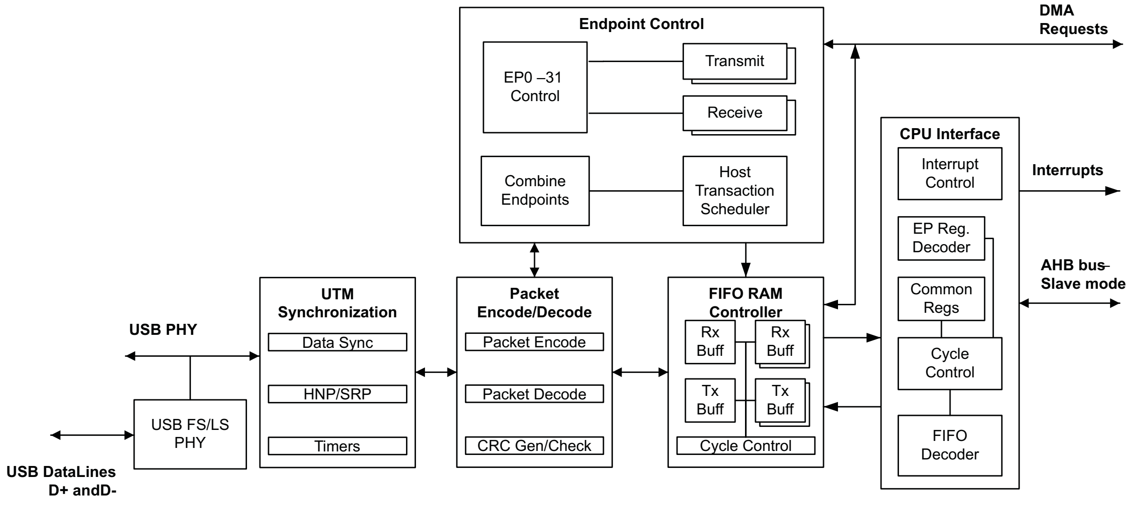
The USB module has the following features:

- Complies with USB-IF certification standards
- USB 2.0 full-speed (12 Mbps) operation in host and device modes as well as low-speed (1.5 Mbps) operation in host mode
- Integrated PHY
- Four transfer types: Control, Interrupt, Bulk, and Isochronous
- Eight endpoints
  - One dedicated control IN endpoint and one dedicated control OUT endpoint
  - Three configurable IN endpoints and three configurable OUT endpoints
- Four KB dedicated endpoint memory: one endpoint may be defined for double-buffered 1023-byte isochronous packet size
- Efficient transfers using direct memory access controller (DMA)
  - All six endpoints can trigger separate DMA events
  - Channel requests asserted when FIFO contains required amount of data

17.2.1 Block Diagram

The USB block diagram is shown in the figure below.

Figure 17-1. USB Block Diagram



17.2.2 Signal Description

The USB controller requires a total of three signals (D+, D-, and VBus) to operate in device mode and two signals (D+, D-) to operate in embedded host mode. Because of the high speed needed for USB, the pins D+ and D- have special buffers to support USB. As such, their position on the chip is not user-selectable. These pins at reset are, by default, GPIOs. They must be configured before being used as USB function pins. The USBIOEN bit in the GPIO A Control 2 (GPACTRL2) register should be set to choose the USB function. The signals USB bus voltage (VBUS), External power enable (EPEN), and Power fault (PFLT) are not hardwired to any pin and some applications will require they be implemented in software via a GPIO. Software that implements these signals is available in the USB software library.

17.2.3 Signal Pinout Tables

The signal pinouts are shown in the table below.

Table 17-1. Signal Pinouts

Package PIn		**GPIO Number**	**Function**
100 LQFP (PZ)	**80 LQFP (PN)**		
78	62	26	USB D+ Positive Differential Half of USB signal
77	61	27	USB D+ Negative Differential Half of USB signal

17.2.4 VBus Recommendations

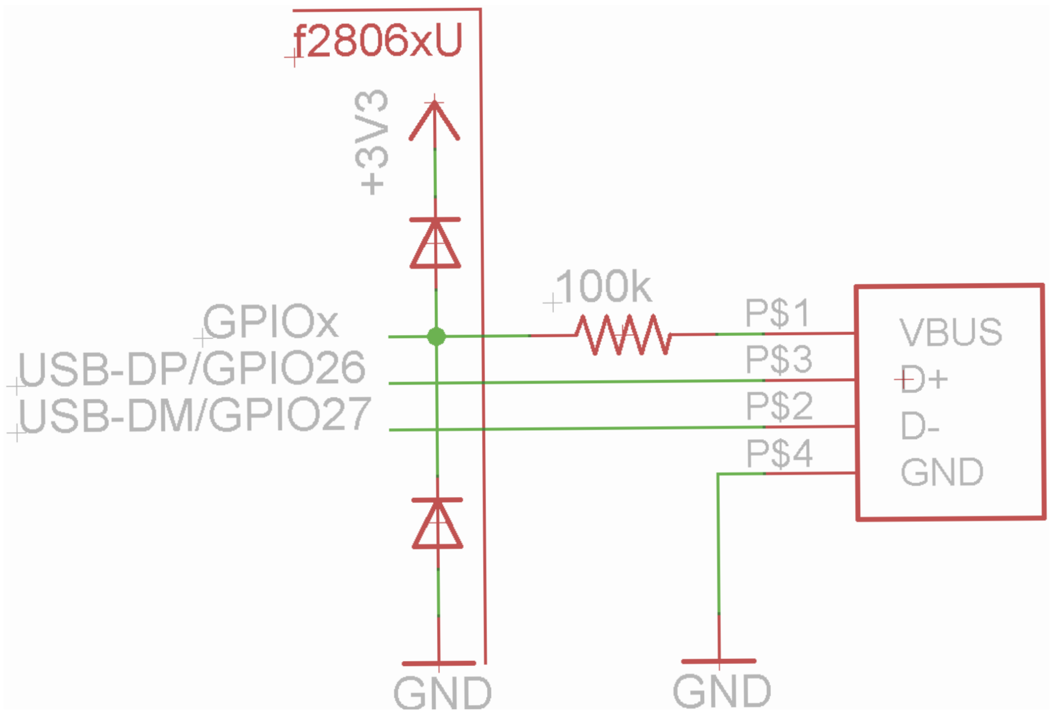
Most applications do not have a need to monitor VBus, and because of this, a dedicated VBus monitoring pin was not included on this processor. If you are designing a bus-powered device application or an embedded host application, you do NOT need to monitor VBus. If you are designing a self-powered device, you will however, need to actively monitor the state of the VBus pin in order to ensure compliance with the USB specification. In Section 7.1.5 and Section 7.2.1 of the USB Specification Revision 2.0™ it is stated respectively that:

- "The voltage source on the [speed identification] pull-up resistor must be derived from or controlled by the power supplied on the USB cable such that when VBUS is removed, the pull-up resistor does not supply current on the data line to which it is attached.
- When VBUS is removed, the device must remove power from the D+/D- pull-up resistor within 10 seconds.
- Later in the timing tables (Section 7.3.2) of the USB Specification 2.0 it is also stated that the D+/D- pull-up resistor should be applied within 100ms of VBus reaching a valid level."

Meeting the above specification is very easy because of the slow timing requirements. In this chapter we will discuss the hardware part of the VBus monitoring solution. The corresponding software will be discussed breifly, but for examples and an explanation the user should consult the USB software guide.

The pins of this microprocessor are NOT 5V tolerant, and because of this the VBus signal cannot be directly connected to a pin. Directly connecting 5V to a pin of the microcontroller WILL destroy the pin and possible more of the chip. The most cost effective way of making any pin capable of reading a 5V input is to use a series resistance in conjunction with the ESD diode clamps already present inside the device on every pin. We recommend the use of a 100k series resistance between the VBus signal and the pin choosen to monitor it. A diagram of this setup is shown below.

Figure 17-2. USB Scheme



In the above diagram, if VBus is above or below 3.3V and 0V respectively, one of the ESD clamp diodes will be forward-biased, allowing current to flow through the 100K resistor. The purpose of the diode clamps is to protect the pins of the microcontroller from very short overvoltage spikes of a high magnitude. They do this by clamping the voltage excursion to one of the supply rails. We are effectively asking the ESD clamps to do the same thing they were designed to do, but instead of a short high magnitude pulse, we are giving them a long low magnitude static value via the 100k resistor.

Any pin that has digital input/output functionality could potentially be used to monitor VBus, but the use of an interrupt capable GPIO is recommended. A pin that does not have external interrupt capability may also be used, but the input state of the pin must be polled periodically by the application software to ensure appropriate action is taken whenever VBus is applied or removed. If an interrupt capable GPIO is chosen, it should be configured to generate an interrupt on both the rising and falling edge. More information on external interrupts can be found in the *System Control and Interrupts* chapter. Example code that implements VBus monitoring using external interrupts and takes the appropriate actions is documented in the USB Software Guide and can be found in the associated USB software package.

17.3 Functional Description

The USB controller can be configured to act as either a dedicated host or device. However, when the USB controller is acting as a self-powered device, a GPIO input or analog comparator input must be connected to VBUS and configured to generate an interrupt when the VBUS level drops. This interrupt is used to disable the pullup resistor on the USB0DP signal.

Note: When USB is used in the system, the minimum system frequency is 20 MHz.

17.3.1 Operation as a Device

This section describes how the USB controller performs when it is being used as a USB device. IN endpoints, OUT endpoints, entry into and exit from SUSPEND mode, and recognition of start of frame (SOF) are all described.

When in device mode, IN transactions are controlled by the endpoint transmit interface and uses the transmit endpoint registers for the given endpoint. OUT transactions are handled with the endpoins receive interface and uses the receive endpoint registers for the given endpoint. When configuring the size of the FIFOs for endpoints, take into account the maximum packet size for an endpoint.

- Bulk. Bulk endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used (described further in the following section).
- Interrupt. Interrupt endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used.
- Isochronous. Isochronous endpoints are more flexible and can be up to 1023 bytes.
- Control. It is also possible to specify a separate control endpoint for a USB device. However, in most cases the USB device should use the dedicated control endpoint on the USB controller's endpoint 0.

17.3.1.1 Control and Configurable Endpoints

When operating as a device, the USB controller provides two dedicated control endpoints (IN and OUT) and six configurable endpoints (three IN and three OUT) that can be used for communications with a host controller. The endpoint number and direction associated with an endpoint is directly related to its register designation. For example, when the Host is transmitting to endpoint 1, all configuration and data is in the endpoint 1 transmit register interface. Endpoint 0 is a dedicated control endpoint used for all control transactions to endpoint 0 during enumeration or when any other control requests are made to endpoint 0. Endpoint 0 uses the first 64 bytes of the USB controller's FIFO RAM as a shared memory for both IN and OUT transactions. The remaining six endpoints can be configured as control, bulk, interrupt, or isochronous endpoints. They should be treated as three configurable IN and three configurable OUT endpoints. The endpoint pairs are not required to have the same type for their IN and OUT endpoint configuration. For example, the OUT portion of an endpoint pair could be a bulk endpoint, while the IN portion of that endpoint pair could be an interrupt endpoint. The address and size of the FIFOs attached to each endpoint can be modified to fit the application's needs.

17.3.1.1.1 IN Transactions as a Device

When operating as a USB device, data for IN transactions is handled through the FIFOs attached to the transmit endpoints. The sizes of the FIFOs for the three configurable IN endpoints are determined by the USB Transmit FIFO Start Address (USBTXFIFOADD) register. The maximum size of a data packet that may be placed in a transmit endpoint's FIFO for transmission is programmable and is determined by the value written to the USB Maximum Transmit Data Endpoint n (USBTXMAXPn) register for that endpoint. The endpoint's FIFO can also be configured to use double-packet or single-packet buffering. When double-packet buffering is enabled, two data packets can be buffered in the FIFO, which also requires that the FIFO is at least two packets in size. When double-packet buffering is disabled, only one packet can be buffered, even if the packet size is less than half the FIFO size.

Note: The maximum packet size set for any endpoint must not exceed the FIFO size. The USBTXMAXPn register should not be written to while data is in the FIFO as unexpected results may occur.

Single-Packet Buffering

If the size of the transmit endpoint's FIFO is less than twice the maximum packet size for this endpoint (as set in the USB Transmit Dynamic FIFO Sizing (USBTXFIFOSZ) register), only one packet can be buffered in the FIFO and single-packet buffering is required. When each packet is completely loaded into the transmit FIFO, the TXRDY bit in the USB Transmit Control and Status Endpoint n Low (USBTXCSRLn) register must be set. If the AUTOSET bit in the USB Transmit Control and Status Endpoint n High (USBTXCSRHn) register is set, the TXRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, the TXRDY bit must be set manually. When the TXRDY bit is set, either manually or automatically, the packet is ready to be sent. When the packet has been successfully sent, both TXRDY and FIFONE are cleared, and the appropriate transmit endpoint interrupt signaled. At this point, the next packet can be loaded into the FIFO.

Double-Packet Buffering

If the size of the transmit endpoint's FIFO is at least twice the maximum packet size for this endpoint, two packets can be buffered in the FIFO and double-packet buffering is allowed. As each packet is loaded into the transmit FIFO, the TXRDY bit in the USBTXCSRLn register must be set. If the AUTOSET bit in the USBTXCSRHn register is set, the TXRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, TXRDY must be set manually. When the TXRDY bit is set, either manually or automatically, the packet is ready to be sent. After the first packet is loaded, TXRDY is immediately cleared and an interrupt is generated. A second packet can now be loaded into the transmit FIFO and TXRDY set again (either manually or automatically if the packet is the maximum size). At this point, both packets are ready to be sent. After each packet has been successfully sent, TXRDY is automatically cleared and the appropriate transmit endpoint interrupt signaled to indicate that another packet can now be loaded into the transmit FIFO. The state of the FIFONE bit in the USBTXCSRLn register at this point indicates how many packets may be loaded. If the FIFONE bit is set, then another packet is in the FIFO and only one more packet can be loaded. If the FIFONE bit is clear, then no packets are in the FIFO and two more packets can be loaded.

Note: Double-packet buffering is disabled if an endpoint's corresponding EPn bit is set in the USB Transmit Double Packet Buffer Disable (USBTXDPKTBUFDIS) register. This bit is set by default, so it must be cleared to enable double-packet buffering.

17.3.1.1.2 Out Transactions as a Device

When in device mode, OUT transactions are handled through the USB controller receive FIFOs. The sizes of the receive FIFOs for the 3 configurable OUT endpoints are determined by the USB Receive FIFO Start Address (USBRXFIFOADD) register. The maximum amount of data received by an endpoint in any packet is determined by the value written to the USB Maximum Receive Data Endpoint n (USBRXMAXPn) register for that endpoint. When double-packet buffering is enabled, two data packets can be buffered in the FIFO. When double-packet buffering is disabled, only one packet can be buffered even if the packet is less than half the FIFO size.

Note: In all cases, the maximum packet size must not exceed the FIFO size.

Single-Packet Buffering

If the size of the receive endpoint FIFO is less than twice the maximum packet size for an endpoint, only one data packet can be buffered in the FIFO and single-packet buffering is required. When a packet is received and placed in the receive FIFO, the RXRDY and FULL bits in the USB Receive Control and Status Endpoint n Low (USBRXCSRL[*n*]) register are set and the appropriate receive endpoint is signaled, indicating that a packet can now be unloaded from the FIFO. After the packet has been unloaded, the RXRDY bit must be cleared in order to allow further packets to be received. This action also generates the acknowledge signaling to the Host controller. If the AUTOCL bit in the USB Receive Control and Status Endpoint n High (USBRXCSRH[*n*]) register is set and a maximum-sized packet is unloaded from the FIFO, the RXRDY and FULL bits are cleared automatically. For packet sizes less than the maximum, RXRDY must be cleared manually.

Double-Packet Buffering

If the size of the receive endpoint FIFO is at least twice the maximum packet size for the endpoint, two data packets can be buffered and double-packet buffering can be used. When the first packet is received and loaded into the receive FIFO, the RXRDY bit in the USBRXCSRL[*n*] register is set and the appropriate receive endpoint interrupt is signaled to indicate that a packet can now be unloaded from the FIFO.

Note: The FULL bit in USBRXCSRL[*n*] is not set when the first packet is received. It is only set if a second packet is received and loaded into the receive FIFO.

After each packet has been unloaded, the RXRDY bit must be cleared to allow further packets to be received. If the AUTOCL bit in the USBRXCSRH[*n*] register is set and a maximum-sized packet is unloaded from the FIFO, the RXRDY bit is cleared automatically. For packet sizes less than the maximum, RXRDY must be cleared manually. If the FULL bit is set when RXRDY is cleared, the USB controller first clears the FULL bit, then sets RXRDY again to indicate that there is another packet waiting in the FIFO to be unloaded.

Note: Double-packet buffering is disabled if an endpoint's corresponding EPn bit is set in the USB Receive Double Packet Buffer Disable (USBRXDPKTBUFDIS) register. This bit is set by default, so it must be cleared to enable double-packet buffering.

17.3.1.1.3 Scheduling

The device has no control over the scheduling of transactions as scheduling is determined by the Host controller. The USB controller can set up a transaction at any time. The USB controller waits for the request from the Host controller and generates an interrupt when the transaction is complete or if it was terminated due to some error. If the Host controller makes a request and the device controller is not ready, the USB controller sends a busy response (NAK) to all requests until it is ready.

17.3.1.1.4 Additional Actions

The USB controller responds automatically to certain conditions on the USB bus or actions by the Host controller such as when the USB controller automatically stalls a control transfer or unexpected zero length OUT data packets.

Stalled Control Transfer

The USB controller automatically issues a STALL handshake to a control transfer under the following conditions:

1. The Host sends more data during an OUT data phase of a control transfer than was specified in the device request during the SETUP phase. This condition is detected by the USB controller when the Host sends an OUT token (instead of an IN token) after the last OUT packet has been unloaded and the DATAEND bit in the USB Control and Status Endpoint 0 Low (USBCSRL0) register has been set.
2. The Host requests more data during an IN data phase of a control transfer than was specified in the device request during the SETUP phase. This condition is detected by the USB controller when the Host sends an IN token (instead of an OUT token) after the CPU has cleared TXRDY and set DATAEND in response to the ACK issued by the Host to what should have been the last packet.
3. The Host sends more than USBRXMAXPn bytes of data with an OUT data token.
4. The Host sends more than a zero length data packet for the OUT STATUS phase.

Zero Length OUT Data Packets

A zero-length OUT data packet is used to indicate the end of a control transfer. In normal operation, such packets should only be received after the entire length of the device request has been transferred. However, if the Host sends a zero-length OUT data packet before the entire length of device request has been transferred, it is signaling the premature end of the transfer. In this case, the USB controller automatically flushes any IN token ready for the data phase from the FIFO and sets the DATAEND bit in the USBCSRL0 register.

Setting the Device Address

When a Host is attempting to enumerate the USB device, it requests that the device change its address from zero to some other value. The address is changed by writing the value that the Host requested to the USB Device Functional Address (USBFADDR) register. However, care should be taken when writing to USBFADDR to avoid changing the address before the transaction is complete. This register should only be set after the SET_ADDRESS command is complete. Like all control transactions, the transaction is only complete after the device has left the STATUS phase. In the case of a SET_ADDRESS command, the transaction is completed by responding to the IN request from the Host with a zero-byte packet. Once the device has responded to the IN request, the USBFADDR register should be programmed to the new value as soon as possible to avoid missing any new commands sent to the new address.

Note: If the USBFADDR register is set to the new value as soon as the device receives the OUT transaction with the SET_ADDRESS command in the packet, it changes the address during the control transfer. In this case, the device does not receive the IN request that allows the USB transaction to exit the STATUS phase of the control transfer because it is sent to the old address. As a result, the Host does not get a response to the IN request, and the Host fails to enumerate the device.

17.3.1.1.5 Device Mode Suspend

When no activity has occurred on the USB bus for 3 ms, the USB controller automatically enters SUSPEND mode. If the SUSPEND interrupt has been enabled in the USB Interrupt Enable (USBIE) register, an interrupt is generated at this time. When in SUSPEND mode, the PHY also goes into SUSPEND mode. When RESUME signaling is detected, the USB controller exits SUSPEND mode and takes the PHY out of SUSPEND. If the RESUME interrupt is enabled, an interrupt is generated. The USB controller can also be forced to exit SUSPEND mode by setting the RESUME bit in the USB Power (USBPOWER) register. When this bit is set, the USB controller exits SUSPEND mode and drives RESUME signaling onto the bus. The RESUME bit must be cleared after 10 ms (a maximum of 15 ms) to end RESUME signaling. To meet USB power requirements, the controller can be put into Deep Sleep mode which keeps the controller in a static state.

17.3.1.1.6 Start of Frame

When the USB controller is operating in device mode, it receives a Start-Of-Frame (SOF) packet from the Host once every millisecond. When the SOF packet is received, the 11-bit frame number contained in the packet is written into the USB Frame Value (USBFRAME) register, and an SOF interrupt is also signaled and can be handled by the application. Once the USB controller has started to receive SOF packets, it expects one every millisecond. If no SOF packet is received after 1.00358 ms, the packet is assumed to have been lost, and the USBFRAME register is not updated. The USB controller continues and resynchronizes these pulses to the received SOF packets when these packets are successfully received again.

17.3.1.1.7 USB Reset

When the USB controller is in device mode and a RESET condition is detected on the USB bus, the USB controller automatically performs the following actions:

- Clears the USBFADDR register
- Clears the USB Endpoint Index (USBEPIDX) register
- Flushes all endpoint FIFOs
- Clears all control/status registers
- Enables all endpoint interrupts
- Generates a RESET interrupt

17.3.1.1.8 Connect/Disconnect

The USB controller connection to the USB bus is handled by software. The USB PHY can be switched between normal mode and non-driving mode by setting or clearing the SOFTCONN bit of the USBPOWER register. When the SOFTCONN bit is set, the PHY is placed in its normal mode, and the USB0DP/USB0DM lines of the USB bus are enabled. At the same time, the USB controller is placed into a state, in which it does not respond to any USB signaling except a USB RESET. When the SOFTCONN bit is cleared, the PHY is put into non-driving mode, USB0DP and USB0DM are tristated, and the USB controller appears to other devices on the USB bus as if it has been disconnected. The non-driving mode is the default so the USB controller appears disconnected until the SOFTCONN bit has been set. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is complete, and the system is ready to perform enumeration before connecting to the USB bus. Once the SOFTCONN bit has been set, the USB controller can be disconnected by clearing this bit.

Note: The USB controller does not generate an interrupt when the device is connected to the Host. However, an interrupt is generated when the Host terminates a session.

17.3.2 Operation as a Host

When the USB controller is operating in Host mode, it can either be used for point-to-point communications with another USB device or, when attached to a hub, for communication with multiple devices. Full-speed and low-speed USB devices are supported, both for point-to-point communication and for operation through a hub. The USB controller automatically carries out the necessary transaction translation needed to allow a low-speed or full-speed device to be used with a USB 2.0 hub. Control, bulk, isochronous, and interrupt transactions are supported. This section describes the USB controller's actions when it is being used as a USB Host. Configuration of IN endpoints, OUT endpoints, entry into and exit from SUSPEND mode, and RESET are all described.

When in Host mode, IN transactions are controlled by an endpoint's receive interface. All IN transactions use the receive endpoint registers and all OUT endpoints use the transmit endpoint registers for a given endpoint. As in device mode, the FIFOs for endpoints should take into account the maximum packet size for an endpoint.

- Bulk. Bulk endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used (described further in the following section).
- Interrupt. Interrupt endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used.
- Isochronous. Isochronous endpoints are more flexible and can be up to 1023 bytes.
- Control. It is also possible to specify a separate control endpoint to communicate with a device. However, in most cases the USB controller should use the dedicated control endpoint to communicate with a device's endpoint 0.

17.3.2.1 Endpoint Registers

The endpoint registers are used to control the USB endpoint interfaces which communicate with device(s) that are connected. The endpoints consist of a dedicated control IN endpoint, a dedicated control OUT endpoint, 3 configurable OUT endpoints, and 3 configurable IN endpoints.

The dedicated control interface can only be used for control transactions to endpoint 0 of devices. These control transactions are used during enumeration or other control functions that communicate using endpoint 0 of devices. This control endpoint shares the first 64 bytes of the USB controller's FIFO RAM for IN and OUT transactions. The remaining IN and OUT interfaces can be configured to communicate with control, bulk, interrupt, or isochronous device endpoints.

These USB interfaces can be used to simultaneously schedule as many as 3 independent OUT and 3 independent IN transactions to any endpoints on any device. The IN and OUT controls are paired in three sets of registers. However, they can be configured to communicate with different types of endpoints and different endpoints on devices. For example, the first pair of endpoint controls can be split so that the OUT portion is communicating with a device's bulk OUT endpoint 1, while the IN portion is communicating with a device's interrupt IN endpoint 2.

Before accessing any device, whether for point-to-point communications or for communications via a hub, the relevant USB Receive Functional Address Endpoint n (USBRXFUNCADDRn) or USB Transmit Functional Address Endpoint n (USBTXFUNCADDRn) registers must be set for each receive or transmit endpoint to record the address of the device being accessed.

The USB controller also supports connections to devices through a USB hub by providing a register that specifies the hub address and port of each USB transfer. The FIFO address and size are customizable and can be specified for each USB IN and OUT transfer. Customization includes allowing one FIFO per transaction, sharing a FIFO across transactions, and allowing for double-buffered FIFOs.

17.3.2.2 IN Transactions as a Host

IN transactions are handled in a similar manner to the way in which OUT transactions are handled when the USB controller is in device mode except that the transaction first must be initiated by setting the REQPKT bit in the USBCSRL0 register, indicating to the transaction scheduler that there is an active transaction on this endpoint. The transaction scheduler then sends an IN token to the target device. When the packet is received and placed in the receive FIFO, the RXRDY bit in the USBCSRL0 register is set, and the appropriate receive endpoint interrupt is signaled to indicate that a packet can now be unloaded from the FIFO.

When the packet has been unloaded, RXRDY must be cleared. The AUTOCL bit in the USBRXCSRHn register can be used to have RXRDY automatically cleared when a maximum-sized packet has been unloaded from the FIFO. The AUTORQ bit in USBRXCSRHn causes the REQPKT bit to be automatically set when the RXRDY bit is cleared. The AUTOCL and AUTORQ bits can be used with DMA accesses to perform complete bulk transfers without main processor intervention. When the RXRDY bit is cleared, the controller sends an acknowledge to the device. When there is a known number of packets to be transferred, the USB Request Packet Count in Block Transfer Endpoint n (USBRQPKTCOUNTn) register associated with the endpoint should be configured to the number of packets to be transferred. The USB controller decrements the value in the USBRQPKTCOUNTn register following each request. When the USBRQPKTCOUNTn value decrements to 0, the AUTORQ bit is cleared to prevent any further transactions being attempted. For cases where the size of the transfer is unknown, USBRQPKTCOUNTn should be cleared. AUTORQ then remains set until cleared by the reception of a short packet (that is, less than the MAXLOAD value in the USBRXMAXPn register) such as may occur at the end of a bulk transfer.

If the device responds to a bulk or interrupt IN token with a NAK, the USB Host controller keeps retrying the transaction until any NAK Limit that has been set has been reached. If the target device responds with a STALL, however, the USB Host controller does not retry the transaction but sets the STALLED bit in the USBCSRL0 register. If the target device does not respond to the IN token within the required time, or the packet contained a CRC or bit-stuff error, the USB Host controller retries the transaction. If after three attempts the target device has still not responded, the USB Host controller clears the REQPKT bit and sets the ERROR bit in the USBCSRL0 register.

17.3.2.3 OUT Transactions as a Host

OUT transactions are handled in a similar manner to the way in which IN transactions are handled when the USB controller is in device mode. The TXRDY bit in the USBTXCSRLn register must be set as each packet is loaded into the transmit FIFO. Again, setting the AUTOSET bit in the USBTXCSRHn register automatically sets TXRDY when a maximum-sized packet has been loaded into the FIFO. Furthermore, AUTOSET can be used with the DMA controller to perform complete bulk transfers without software intervention.

If the target device responds to the OUT token with a NAK, the USB Host controller keeps retrying the transaction until the NAK Limit that has been set has been reached. However, if the target device responds with a STALL, the USB controller does not retry the transaction but interrupts the main processor by setting the STALLED bit in the USBTXCSRLn register. If the target device does not respond to the OUT token within the required time, or the packet contained a CRC or bit-stuff error, the USB Host controller retries the transaction. If after three attempts the target device has still not responded, the USB controller flushes the FIFO and sets the ERROR bit in the USBTXCSRLn register.

17.3.2.4 Transaction Scheduling

Scheduling of transactions is handled automatically by the USB Host controller. The Host controller allows configuration of the endpoint communication scheduling based on the type of endpoint transaction. Interrupt transactions can be scheduled to occur in the range of every frame to every 255 frames in 1 frame increments. Bulk endpoints do not allow scheduling parameters, but do allow for a NAK timeout in the event an endpoint on a device is not responding. Isochronous endpoints can be scheduled from every frame to every 216 frames, in powers of 2.

The USB controller maintains a frame counter. If the target device is a full-speed device, the USB controller automatically sends an SOF packet at the start of each frame and increments the frame counter. If the target device is a low-speed device, a K state is transmitted on the bus to act as a keep-alive to stop the low-speed device from going into SUSPEND mode.

After the SOF packet has been transmitted, the USB Host controller cycles through all the configured endpoints looking for active transactions. An active transaction is defined as a receive endpoint for which the REQPKT bit is set or a transmit endpoint for which the TXRDY bit and/or the FIFONE bit is set.

An isochronous or interrupt transaction is started if the transaction is found on the first scheduler cycle of a frame and if the interval counter for that endpoint has counted down to zero. As a result, only one interrupt or isochronous transaction occurs per endpoint every n frames, where n is the interval set via the USB Host Transmit Interval Endpoint n (USBTXINTERVAL[*n*]) or USB Host Receive Interval Endpoint n (USBRXINTERVAL[*n*]) register for that endpoint.

An active bulk transaction starts immediately, provided sufficient time is left in the frame to complete the transaction before the next SOF packet is due. If the transaction must be retried (for example, because a NAK was received or the target device did not respond), then the transaction is not retried until the transaction scheduler has first checked all the other endpoints for active transactions. This process ensures that an endpoint that is sending a lot of NAKs does not block other transactions on the bus. The controller also allows the user to specify a limit to the length of time for NAKs to be received from a target device before the endpoint times out.

17.3.2.5 USB Hubs

The following setup requirements apply to the USB Host controller only if it is used with a USB hub. When a full- or low-speed device is connected to the USB controller via a USB 2.0 hub, details of the hub address and the hub port also must be recorded in the corresponding USB Receive Hub Address Endpoint n (USBRXHUBADDRn) and USB Receive Hub Port Endpoint n (USBRXHUBPORTn) or the USB Transmit Hub Address Endpoint n (USBTXHUBADDRn) and USB Transmit Hub Port Endpoint n (USBTXHUBPORTn) registers. In addition, the speed at which the device operates (full or low) must be recorded in the USB Type Endpoint 0 (USBTYPE0) (endpoint 0), USB Host Configure Transmit Type Endpoint n (USBTXTYPEn), or USB Host Configure Receive Type Endpoint n (USBRXTYPEn) registers for each endpoint that is accessed by the device.

For hub communications, the settings in these registers record the current allocation of the endpoints to the attached USB devices. To maximize the number of devices supported, the USB Host controller allows this allocation to be changed dynamically by simply updating the address and speed information recorded in these registers. Any changes in the allocation of endpoints to device functions must be made following the completion of any on-going transactions on the endpoints affected.

17.3.2.6 Babble

The USB Host controller does not start a transaction until the bus has been inactive for at least the minimum inter-packet delay. The controller also does not start a transaction unless it can be finished before the end of the frame. If the bus is still active at the end of a frame, then the USB Host controller assumes that the target device to which it is connected has malfunctioned, and the USB controller suspends all transactions and generates a babble interrupt.

17.3.2.7 Host SUSPEND

If the SUSPEND bit in the USBPOWER register is set, the USB Host controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions are started and no SOF packets are generated.

To exit SUSPEND mode, set the RESUME bit and clear the SUSPEND bit. While the RESUME bit is set, the USB Host controller generates RESUME signaling on the bus. After 20 ms, the RESUME bit must be cleared, at which point the frame counter and transaction scheduler start. The Host supports the detection of a remote wake-up.

17.3.2.8 USB RESET

If the RESET bit in the USBPOWER register is set, the USB Host controller generates USB RESET signaling on the bus. The RESET bit must be set for at least 20 ms to ensure correct resetting of the target device. After the CPU has cleared the bit, the USB Host controller starts its frame counter and transaction scheduler.

17.3.2.9 Connect/Disconnect

A session is started by setting the SESSION bit in the USB device Control (USBDEVCTL) register, enabling the USB controller to wait for a device to be connected. When a device is detected, a connect interrupt is generated. The speed of the device that has been connected can be determined by reading the USBDEVCTL register where the FSDEV bit is set for a full-speed device, and the LSDEV bit is set for a low-speed device. The USB controller must generate a RESET to the device, and then the USB Host controller can begin device enumeration. If the device is disconnected while a session is in progress, a disconnect interrupt is generated.

17.3.3 DMA Operation

The USB peripheral provides an interface connected to the DMA controller with separate channels for 3 transmit endpoints and 3 receive endpoints. The DMA operation of the USB is enabled through the USBTXCSRHn and USBRXCSRHn registers, for the TX and RX channels respectively. When DMA operation is enabled, the USB asserts a DMA request on the enabled receive or transmit channel when the associated FIFO can transfer data. When either FIFO can transfer data, the request for that channel is asserted. The size of the DMA transfer must be restricted to whole multiples of the size of the USB FIFO. Both read and write transfers of the USB FIFOs using DMA must be configured in this manner. For example, if the USB endpoint is configured with a FIFO size of 64 bytes, the DMA channel can be used to transfer 64 bytes to or from the endpoint FIFO. If the number of bytes to transfer is less than 64, then a programmed I/O method must be used to copy the data to or from the FIFO.

If the DMAMOD bit in the USBTXCSRHn/USBRXCSRHn register is clear, an interrupt is generated after every packet is transferred, but the DMA continues transferring data. If the DMAMOD bit is set, an interrupt is generated only when the entire DMA transfer is complete. The interrupt occurs on the USB interrupt vector. Therefore, if interrupts are used for USB operation and the DMA is enabled, the USB interrupt handler must be designed to handle the DMA completion interrupt.

Care must be taken when using the DMA to unload the receive FIFO as data is read from the receive FIFO in 4 byte chunks regardless of value of the MAXLOAD field in the USBRXCSRHn register. The RXRDY bit is cleared as follows.

To enable DMA operation for the endpoint receive channel, the DMAEN bit of the USBRXCSRH[*n*] register should be set. To enable DMA operation for the endpoint transmit channel, the DMAEN bit of the USBTXCSRH[*n*] register must be set.

See the Direct Memory Access (DMA)" Users Guide for more details about programming the DMA controller.

17.3.4 Address/Data Bus Bridge

The USB controller on this device is the same controller that is on the Stellaris devices. This controller was originally designed to connect to an ARM AHB bus, but has been modified in order to function with the C28x device's bus architecture. The modifications made are largely invisible to the user application, but there are some things to note.

- The USB memory space is 8 bits wide, while the C28x memory space is 16 bits wide.
- 32 and 16 bit accesses (r/w) are completely transparent to the user application code, no changes need be made.
- The C28x core only supports 8 bit accesses through a byte intrinsic type. This can be used to perform 8 bit reads or writes to the USB controller.
  - int &__byte(int *array, unsigned int byte_index);
  - *array = ptr to address to access, byte_index = always 0 (for USB)
    See Table 17-2 for example.
  - See the TMS320C28x Optimizing C/C++ Compiler User's Guide (SPRU514) and the TMS320C28x Assembly Language Tools User's Guide (SPRU513)
- Because of the bridge, the memory view of the USB controller memory space in CCS isn't a 1:1 representation of what is in the controller
  - When the view mode is
    - 32 bit or 16 bit, even address are effectively duplicated, ignore odd addresses.
    - 8 bit, Even addresses from within the controller are duplicated into odd address in the view window; old addresses from within the controller are not displayed.
      See Table 17-3 for example.

Table 17-2. USB Memory Access From Software

USB Controller Memory			C28x 8 Bit	
Address	**Reg. Name**	**Data**	**Access**	**Data**
0x00	FADDR	0x00	__byte((int *)0x00,0)	0x0000
0x01	POWER	0x11	__byte((int *)0x01,0)	0x0011
0x02	TXIS (LSB)	0x22	__byte((int *)0x02,0)	0x0022
0x03	TXIS (MSB)	0x33	__byte((int *)0x03,0)	0x0033
0x04	RXIS (LSB)	0x44	__byte((int *)0x04,0)	0x0044
0x05	RXIS (MSB)	0x55	__byte((int *)0x05,0)	0x0055

Table 17-2. USB Memory Access From Software (continued)

USB Controller Memory			C28x 8 Bit	
0x06	TXIE (LSB)	0x66	__byte((int *)0x06,0)	0x0066
0x07	TXIE (MSB)	0x77	__byte((int *)0x07,0)	0x0077
0x08	RXIE (LSB)	0x88	__byte((int *)0x08,0)	0x0088
0x09	RXIE (MSB)	0x99	__byte((int *)0x09,0)	0x0099
0x0A	USBIS	0xAA	__byte((int *)0x0A,0)	0x00AA
0x0B	USBIE	0xBB	__byte((int *)0x0B,0)	0x00BB
0x0C	FRAME (LSB)	0xCC	__byte((int *)0x0C,0)	0x00CC
0x0D	FRAME (MSB)	0xDD	__byte((int *)0x0D,0)	0x00DD
0x0E	EPIDX	0xEE	__byte((int *)0x0E,0)	0x00EE
0x0F	TEST	0xFF	__byte((int *)0x0F,0)	0x00FF
C28x 16 Bit			**C28x 32 Bit**	
Access	**Data**		**Access**	**Data**
(*((short *)(0x00)))	0x1100		(*((long *)(0x00)))	0x33221100
(*((short *)(0x01)))	0x1100		(*((long *)(0x01)))	0x33221100
(*((short *)(0x02)))	0x3322		(*((long *)(0x02)))	0x33221100
(*((short *)(0x03)))	0x3322		(*((long *)(0x03)))	0x33221100
(*((short *)(0x04)))	0x5544		(*((long *)(0x04)))	0x77665544
(*((short *)(0x05)))	0x5544		(*((long *)(0x05)))	0x77665544
(*((short *)(0x06)))	0x7766		(*((long *)(0x06)))	0x77665544
(*((short *)(0x07)))	0x7766		(*((long *)(0x07)))	0x77665544
(*((short *)(0x08)))	0x9988		(*((long *)(0x08)))	0xBBAA9988
(*((short *)(0x09)))	0x9988		(*((long *)(0x09)))	0xBBAA9988
(*((short *)(0x0A)))	0xBBAA		(*((long *)(0x0A)))	0xBBAA9988
(*((short *)(0x0B)))	0xBBAA		(*((long *)(0x0B)))	0xBBAA9988
(*((short *)(0x0C)))	0xDDCC		(*((long *)(0x0C)))	0xFFEEDDCC
(*((short *)(0x0D)))	0xDDCC		(*((long *)(0x0D)))	0xFFEEDDCC
(*((short *)(0x0E)))	0xFFEE		(*((long *)(0x0E)))	0xFFEEDDCC
(*((short *)(0x0F)))	0xFFEE		(*((long *)(0x0F)))	0xFFEEDDCC

Table 17-3. USB Memory Access From CCS

CCS 8 Bit		CCS 16 Bit		CCS 32 Bit	
Address	**Displayed Data**	**Address**	**Displayed Data**	**Address**	**Displayed Data**
0x00	0x00	0x00	0x1100	0x00	0x11001100
0x01	0x00	0x01	0x1100	0x02	0x33223322
0x02	0x22	0x02	0x3322	0x04	0x55445544
0x03	0x22	0x03	0x3322	0x06	0x77667766
0x04	0x44	0x04	0x5544	0x08	0x99889988
0x05	0x44	0x05	0x5544	0x0A	0xBBAABBAA
0x06	0x66	0x06	0x7766	0x0C	0xDDCCDDCC
0x07	0x66	0x07	0x7766	0x0E	0xFFEEFFEE
0x08	0x88	0x08	0x9988		
0x09	0x88	0x09	0x9988		
0x0A	0xAA	0x0A	0xBBAA		
0x0B	0xAA	0x0B	0xBBAA		
0x0C	0xCC	0x0C	0xDDCC		
0x0D	0xCC	0x0D	0xDDCC		

Table 17-3. USB Memory Access From CCS (continued)

CCS 8 Bit		CCS 16 Bit		CCS 32 Bit	
0x0E	0xEE	0x0E	0xFFEE		
0x0F	0xEE	0x0F	0xFFEE		

17.4 Initialization and Configuration

To use the USB Controller, the clock for the USB controller must first be configured. USBCLK is driven via the second PLL within the chip. The PLL should be configured to operate at 60MHz via the PLL2CTL, PLL2MULT, PLL2STS registers. After the PLL is enabled and locked,the controller can be clocked by enabling the peripheral in the PCLKCR3 register. In addition to the clock,the USB PHY must be enabled. Configure the USBIOEN field in the GPACTRL2 register to enable USB functionality on the designated pins. After the clock has been enabled and the USB PHY turned on the USB peripheral is ready for operation and the associated software initialization routines may be called.

17.4.1 Pin Configuration

In order to give the user more flexabliity, the signals External Power Enable (EPEN) and Power Fault (PFLT) were not implemented in hardware. Instead, it is left up to the user to implement these signals in software. Examples of how to implement these signals in software can be found in the f2806x USB Software Guide.

When using the device controller portion of the USB controller in a system that also provides Host functionality, the power to VBUS must be disabled to allow the external Host controller to supply power. Usually, the EPEN signal is used to control the external regulator and should be negated to avoid having two devices driving the VBUS power pin on the USB connector.

When the USB controller is acting as a Host, it is in control of two signals that are attached to an external voltage supply that provides power to VBUS. The Host controller uses the EPEN signal to enable or disable power to the VBUS pin on the USB connector. An input pin, PFLT, provides feedback when there has been a power fault on VBUS. The PFLT signal can be configured to either automatically negate the EPEN signal to disable power, and/or it can generate an interrupt to the interrupt controller to allow software to handle the power fault condition. The polarity and actions related to both EPEN and PFLT are fully configurable in the USB controller. The controller also provides interrupts on device insertion and removal to allow the Host controller code to respond to these external events.

17.4.2 Endpoint Configuration

To start communication in Host or device mode, the endpoint registers must first be configured. In Host mode, this configuration establishes a connection between an endpoint register and an endpoint on a device. In device mode, an endpoint must be configured before enumerating to the Host controller.

In both cases, the endpoint 0 configuration is limited because it is a fixed-function, fixed-FIFO-size endpoint. In device and Host modes, the endpoint requires little setup but does require a software-based state machine to progress through the setup, data, and status phases of a standard control transaction. In device mode, the configuration of the remaining endpoints is done once before enumerating and then only changed if an alternate configuration is selected by the Host controller. In Host mode, the endpoints must be configured to operate as control, bulk, interrupt or isochronous mode. Once the type of endpoint is configured, a FIFO area must be assigned to each endpoint. In the case of bulk, control and interrupt endpoints, each has a maximum of 64 bytes per transaction. Isochronous endpoints can have packets with up to 1023 bytes per packet. In either mode, the maximum packet size for the given endpoint must be set prior to sending or receiving data.

Configuring each endpoint’s FIFO involves reserving a portion of the overall USB FIFO RAM to each endpoint. The total FIFO RAM available is 4 Kbytes with the first 64 bytes reserved for endpoint 0. The endpoint’s FIFO must be at least as large as the maximum packet size. The FIFO can also be configured as a double-buffered FIFO so that interrupts occur at the end of each packet and allow filling the other half of the FIFO.

If operating as a device, the USB device controller's soft connect must be enabled when the device is ready to start communications, indicating to the host controller that the device is ready to start the enumeration process. If operating as a Host controller, the device soft connect must be disabled and power must be provided to VBUS via the USB0EPEN signal.

17.5 Register Map

Table 17-4 lists the registers. All addresses given are relative to the USB base address of 0x4000. Note that the USB controller clock must be enabled before the registers can be programmed (see System Control module).

Table 17-4. Universal Serial Bus (USB) Controller Register Map

Offset	Name	Type	Reset	Description	Location
0x000	USBFADDR(1)	R/W	0x00	USB Device Functional Address	Section 17.6.1
0x001	USBPOWER(1)(2)	R/W	0x20	USB Power	Section 17.6.2
0x002	USBTXIS(1)(2)	RO	0x0000	USB Transmit Interrupt Status	Section 17.6.3
0x004	USBRXIS(1)(2)	RO	0x0000	USB Receive Interrupt Status	Section 17.6.4
0x006	USBTXIE(1)(2)	R/W	0xFFFF	USB Transmit Interrupt Enable	Section 17.6.5
0x008	USBRXIE(1)(2)	R/W	0xFFFE	USB Receive Interrupt Enable	Section 17.6.6
0x00A	USBIS(1)(2)	RO	0x00	USB General Interrupt Status	Section 17.6.7
0x00B	USBIE(1)(2)	R/W	0x06	USB Interrupt Enable	Section 17.6.8
0x00C	USBFRAME(1)(2)	RO	0x0000	USB Frame Value	Section 17.6.9
0x00E	USBEPIDX(1)(2)	R/W	0x00	USB Endpoint Index	Section 17.6.10
0x00F	USBTEST(1)(2)	R/W	0x00	USB Test Mode	Section 17.6.11
0x020	USBFIFO0(1)(2)	R/W	0x0000.0000	USB FIFO Endpoint 0	Section 17.6.12
0x024	USBFIFO1(1)(2)	R/W	0x0000.0000	USB FIFO Endpoint 1	Section 17.6.12
0x028	USBFIFO2(1)(2)	R/W	0x0000.0000	USB FIFO Endpoint 2	Section 17.6.12
0x02C	USBFIFO3(1)(2)	R/W	0x0000.0000	USB FIFO Endpoint 3	Section 17.6.12
0x060	USBDEVCTL(2)	R/W	0x80	USB Device Control	Section 17.6.13
0x062	USBTXFIFOSZ(1)(2)	R/W	0x00	USB Transmit Dynamic FIFO Sizing	Section 17.6.14
0x063	USBRXFIFOSZ(1)(2)	R/W	0x00	USB Receive Dynamic FIFO Sizing	Section 17.6.15
0x064	USBTXFIFOADD(1)(2)	R/W	0x0000	USB Transmit FIFO Start Address	Section 17.6.16
0x066	USBRXFIFOADD(1)(2)	R/W	0x0000	USB Receive FIFO Start Address	Section 17.6.17
0x07A	USBCONTIM(1)(2)	R/W	0x5C	USB Connect Timing	Section 17.6.18
0x07D	USBFSEOF(1)(2)	R/W	0x77	USB Full-Speed Last Transaction to End of Frame Timing	Section 17.6.19
0x07E	USBLSEOF(1)(2)	R/W	0x72	USB Low-Speed Last Transaction to End of Frame Timing	Section 17.6.20
0x080	USBTXFUNCADDR0(2)	R/W	0x00	USB Transmit Functional Address Endpoint 0	Section 17.6.21
0x082	USBTXHUBADDR0(2)	R/W	0x00	USB Transmit Hub Address Endpoint 0	Section 17.6.22
0x083	USBTXHUBPORT0(2)	R/W	0x00	USB Transmit Hub Port Endpoint 0	Section 17.6.23

Table 17-4. Universal Serial Bus (USB) Controller Register Map (continued)

Offset	Name	Type	Reset	Description	Location
0x088	USBTXFUNCADDR1(2)	R/W	0x00	USB Transmit Functional Address Endpoint 1	Section 17.6.21
0x08A	USBTXHUBADDR1(2)	R/W	0x00	USB Transmit Hub Address Endpoint 1	Section 17.6.22
0x08B	USBTXHUBPORT1(2)	R/W	0x00	USB Transmit Hub Port Endpoint 1	Section 17.6.23
0x08C	USBRXFUNCADDR1(2)	R/W	0x00	USB Receive Functional Address Endpoint 1	Section 17.6.24
0x08E	USBRXHUBADDR1(2)	R/W	0x00	USB Receive Hub Address Endpoint 1	Section 17.6.25
0x08F	USBRXHUBPORT1(2)	R/W	0x00	USB Receive Hub Port Endpoint 1	Section 17.6.26
0x090	USBTXFUNCADDR2(2)	R/W	0x00	USB Transmit Functional Address Endpoint 2	Section 17.6.21
0x092	USBTXHUBADDR2(2)	R/W	0x00	USB Transmit Hub Address Endpoint 2	Section 17.6.22
0x093	USBTXHUBPORT2(2)	R/W	0x00	USB Transmit Hub Port Endpoint 2	Section 17.6.23
0x094	USBRXFUNCADDR2(2)	R/W	0x00	USB Receive Functional Address Endpoint 2	Section 17.6.24
0x096	USBRXHUBADDR2(2)	R/W	0x00	USB Receive Hub Address Endpoint 2	Section 17.6.25
0x097	USBRXHUBPORT2(2)	R/W	0x00	USB Receive Hub Port Endpoint 2	Section 17.6.26
0x098	USBTXFUNCADDR3(2)	R/W	0x00	USB Transmit Functional Address Endpoint 3	Section 17.6.21
0x09A	USBTXHUBADDR3(2)	R/W	0x00	USB Transmit Hub Address Endpoint 3	Section 17.6.22
0x09B	USBTXHUBPORT3(2)	R/W	0x00	USB Transmit Hub Port Endpoint 3	Section 17.6.23
0x09C	USBRXFUNCADDR3(2)	R/W	0x00	USB Receive Functional Address Endpoint 3	Section 17.6.24
0x09E	USBRXHUBADDR3(2)	R/W	0x00	USB Receive Hub Address Endpoint 3	Section 17.6.25
0x09F	USBRXHUBPORT3(2)	R/W	0x00	USB Receive Hub Port Endpoint 3	Section 17.6.26
0x102	USBCSRL0(1)(2)	W1C	0x00	USB Control and Status Endpoint 0 Low	Section 17.6.28
0x103	USBCSRH0(1)(2)	W1C	0x00	USB Control and Status Endpoint 0 High	Section 17.6.29
0x108	USBCOUNT0(1)(2)	R/o	0x00	USB Receive Byte Count Endpoint 0	Section 17.6.30
0x10A	USBTYPE0(2)	R/W	0x00	USB Type Endpoint 0	Section 17.6.31
0x10B	USBNAKLMT(2)	R/W	0x00	USB NAK Limit	Section 17.6.32
0x110	USBTXMAXP1(1)(2)	R/W	0x0000	USB Maximum Transmit Data Endpoint 1	Section 17.6.27
0x112	USBTXCSRL1(1)(2)	R/W	0x00	USB Transmit Control and Status Endpoint 1 Low	Section 17.6.33

Table 17-4. Universal Serial Bus (USB) Controller Register Map (continued)

Offset	Name	Type	Reset	Description	Location
0x113	USBTXCSRH1(1)(2)	R/W	0x00	USB Transmit Control and Status Endpoint 1 High	Section 17.6.34
0x114	USBRXMAXP1(1)(2)	R/W	0x0000	USB Maximum Receive Data Endpoint 1	Section 17.6.35
0x116	USBRXCSRL1(1)(2)	R/W	0x00	USB Receive Control and Status Endpoint 1 Low	Section 17.6.36
0x117	USBRXCSRH1(1)(2)	R/W	0x00	USB Receive Control and Status Endpoint 1 High	Section 17.6.37
0x118	USBRXCOUNT1(1)(2)	RO	0x0000	USB Receive Byte Count Endpoint 1	Section 17.6.38
0x11A	USBTXTYPE1(2)	R/W	0x00	USB Host Transmit Configure Type Endpoint 1	Section 17.6.39
0x11B	USBTXINTERVAL1(2)	R/W	0x00	USB Host Transmit Interval Endpoint 1	Section 17.6.40
0x11C	USBRXTYPE1(2)	R/W	0x00	USB Host Configure Receive Type Endpoint 1	Section 17.6.41
0x11D	USBRXINTERVAL1(2)	R/W	0x00	USB Host Receive Polling Interval Endpoint 1	Section 17.6.42
0x120	USBTXMAXP2(1)(2)	R/W	0x0000	USB Maximum Transmit Data Endpoint 2	Section 17.6.27
0x122	USBTXCSRL2(1)(2)	R/W	0x00	USB Transmit Control and Status Endpoint 2 Low	Section 17.6.33
0x123	USBTXCSRH2(1)(2)	R/W	0x00	USB Transmit Control and Status Endpoint 2 High	Section 17.6.34
0x124	USBRXMAXP2(1)(2)	R/W	0x0000	USB Maximum Receive Data Endpoint 2	Section 17.6.35
0x126	USBRXCSRL2(1)(2)	R/W	0x00	USB Receive Control and Status Endpoint 2 Low	Section 17.6.36
0x127	USBRXCSRH2(1)(2)	R/W	0x00	USB Receive Control and Status Endpoint 2 High	Section 17.6.37
0x128	USBRXCOUNT2(1)(2)	RO	0x0000	USB Receive Byte Count Endpoint 2	Section 17.6.38
0x12A	USBTXTYPE2(2)	R/W	0x00	USB Host Transmit Configure Type Endpoint 2	Section 17.6.39
0x12B	USBTXINTERVAL2(2)	R/W	0x00	USB Host Transmit Interval Endpoint 2	Section 17.6.40
0x12C	USBRXTYPE2(2)	R/W	0x00	USB Host Configure Receive Type Endpoint 2	Section 17.6.41
0x12D	USBRXINTERVAL2(2)	R/W	0x00	USB Host Receive Polling Interval Endpoint 2	Section 17.6.42
0x130	USBTXMAXP3(1)(2)	R/W	0x0000	USB Maximum Transmit Data Endpoint 3	Section 17.6.27

Table 17-4. Universal Serial Bus (USB) Controller Register Map (continued)

Offset	Name	Type	Reset	Description	Location
0x132	USBTXCSRL3[1][2]	R/W	0x00	USB Transmit Control and Status Endpoint 3 Low	Section 17.6.33
0x133	USBTXCSRH3[1][2]	R/W	0x00	USB Transmit Control and Status Endpoint 3 High Section 17.6.34	
0x134	USBRXMAXP3[1][2]	R/W	0x0000	USB Maximum Receive Data Endpoint 3	Section 17.6.35
0x136	USBRXCSRL3[1][2]	R/W	0x00	USB Receive Control and Status Endpoint 3 Low	Section 17.6.33
0x137	USBRXCSRH3[1][2]	R/W	0x00	USB Receive Control and Status Endpoint 3 High	Section 17.6.36
0x138	USBRXCOUNT3[1][2]	RO	0x0000	USB Receive Byte Count Endpoint 3	Section 17.6.38
0x13A	USBTXTYPE3[2]	R/W	0x00	USB Host Transmit Configure Type Endpoint 3	Section 17.6.39
0x13B	USBTXINTERVAL3[2]	R/W	0x00	USB Host Transmit Interval Endpoint 3	Section 17.6.40
0x13C	USBRXTYPE3[2]	R/W	0x00	USB Host Configure Receive Type Endpoint 3	Section 17.6.41
0x13D	USBRXINTERVAL3[2]	R/W	0x00	USB Host Receive Polling Interval Endpoint 3	Section 17.6.42
0x304	USBRQPKTCOUNT1[2]	R/W	0x0000 1	USB Request Packet Count in Block Transfer Endpoint 1	Section 17.6.43
0x308	USBRQPKTCOUNT2[2]	R/W	0x0000	USB Request Packet Count in Block Transfer Endpoint 2	Section 17.6.43
0x30C	USBRQPKTCOUNT3[2]	R/W	0x0000	USB Request Packet Count in Block Transfer Endpoint 3	Section 17.6.43
0x340	USBRXDPKTBUFDIS[1][2]	R/W	0x0000	USB Receive Double Packet Buffer Disable	Section 17.6.44
0x342	USBTXDPKTBUFDIS[1][2]	R/W	0x0000	USB Transmit Double Packet Buffer Disable	Section 17.6.45
0x400	USBEPC[1][2]	R/W	0x0000.0000	USB External Power Control	Section 17.6.46
0x404	USBEPCRIS[1][2]	RO	0x0000.0000	USB External Power Control Raw Interrupt Status	Section 17.6.47
0x408	USBEPCIM[2][1]	R/W	0x0000.0000	USB External Power Control Interrupt Mask	Section 17.6.48
0x40C	USBEPCISC[1][2]	R/W	0x0000.0000	USB External Power Control Interrupt Status and Clear	Section 17.6.49
0x410	USBDRRIS[1][2]	RO	0x0000.0000	USB Device RESUME Raw Interrupt Status	Section 17.6.50
0x414	USBDRIM[1][2]	R/W	0x0000.0000	USB Device RESUME Interrupt Mask	Section 17.6.51

Table 17-4. Universal Serial Bus (USB) Controller Register Map (continued)

Offset	Name	Type	Reset	Description	Location
0x418	USBDRISC(1)(2)	W1C	0x0000.0000	USB Device RESUME Interrupt Status and Clear	Section 17.6.52
0x41C	USBGPCS(1)(2)	R/W	0x0000.0000	USB General-Purpose Control and Status	Section 17.6.53
0x450	USBDMASEL(1)(2)	R/W	0x0033.2211	USB DMA Select	Section 17.6.54

(1) This register is used in Device mode. Some registers are used for both Host and Device mode and may have different bit definitions depending on the mode.

(2) This register is used in Host mode. Some registers are used for both Host and Device mode and may have different bit definitions depending on the mode. The USB controller is in Device mode upon reset, so the reset values shown for these registers apply to the Device mode definition.

17.6 Register Descriptions

17.6.1 USB Device Functional Address Register (USBFADDR), offset 0x000

The USB function address 8-bit register (USBFADDR) contains the 7-bit address of the device part of the transaction.

When the USB controller is being used in device mode (the HOST bit in the USBDEVCTL register is clear), this register must be written with the address received through a SET_ADDRESS command, which is then used for decoding the function address in subsequent token packets.

Mode(s): Device

For special considerations when writing this register, see the *Setting the Device Address* in Section 17.3.1.1.4.

USBFADDR is shown in Figure 17-3 and described in Table 17-5.

Figure 17-3. Function Address Register (USBFADDR)

7	6 ... 0
Reserved	FUNCADDR
R-0	R/W-0

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

Table 17-5. Function Address Register (USBFADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	FUNCADDR	0-7Fh	Function Address of Device as received through SET_ADDRESS.

17.6.2 USB Power Management Register (USBPOWER), offset 0x001

The power management 8-bit register (USBPOWER) is used for controlling SUSPEND and RESUME signaling, and some basic operational aspects of the USB controller.

Mode(s): Host Device

USBPOWER in Host Mode is shown in Figure 17-4 and described in Table 17-6.

Figure 17-4. Power Management Register (USBPOWER) in Host Mode

7 - 4	3	2	1	0
Reserved	RESET	RESUME	SUSPEND	PWRDNPHY
R-0	R/W-0	R/W-0	R/W-1S	R/W-0

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

Table 17-6. Power Management Register (USBPOWER) in Host Mode Field Descriptions

Bit	Field	Value	Description
7-4	Reserved	0	Reserved
3	RESET		RESET signaling.
		0	Ends RESET signaling on the bus.
		1	Enables RESET signaling on the bus.
2	RESUME		RESUME signaling. The bit should be cleared by software 20 ms after being set.
		0	Ends RESUME signaling on the bus.
		1	Enables RESUME signaling when the Device is in SUSPEND mode.
1	SUSPEND		SUSPEND mode
		0	No effect
		1	Enables SUSPEND mode.
0	PWRDNPHY		Power Down PHY
		0	No effect
		1	Powers down the internal USB PHY.

USBPOWER in Device Mode is shown in Figure 17-5 and described in Table 17-7.

Figure 17-5. Power Management Register (USBPOWER) in Device Mode

7	6	5 - 4	3	2	1	0
ISOUPDATE	SOFTCONN	Reserved	RESET	RESUME	SUSPEND	PWRDNPHY
R/W-0	R/W-0	R-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 17-7. Power Management Register (USBPOWER) in Device Mode Field Descriptions

Bit	Field	Value	Description
7	ISOUPDATE		Isochronous Update
		0	No effect
		1	The USB controller waits for an SOF token from the time the TXRDY bit is set in the USBTXCSRLn register before sending the packet. If an IN token is received before an SOF token, then a zero-length data packet is sent.
6	SOFTCONN		Soft Connect/Disconnect
		0	The USB D+/D- lines are tri-stated.
		1	The USB D+/D- lines are enabled.
5-4	Reserved	0	Reserved

Table 17-7. Power Management Register (USBPOWER) in Device Mode Field Descriptions (continued)

Bit	Field	Value	Description
3	RESET		RESET signaling
		0	Ends RESET signaling on the bus.
		1	Enables RESET signaling on the bus.
2	RESUME		RESUME signaling. The bit should be cleared by software 10 ms (a maximum of 15 ms) after being set.
		0	Ends RESUME signaling on the bus.
		1	Enables RESUME signaling when the Device is in SUSPEND mode.
1	SUSPEND		SUSPEND mode.
		0	This bit is cleared when software reads the interrupt register or sets the RESUME bit above.
		1	The USB controller is in SUSPEND mode.
0	PWRDNPHY		Power Down PHY
		0	No effect
		1	Powers down the internal USB PHY.

17.6.3 USB Transmit Interrupt Status Register (USBTXIS), offset 0x002

NOTE: Use caution when reading this register. Performing a read may change bit status.

The USB transmit interrupt status 16-bit read-only register (USBTXIS) indicates which interrupts are currently active for endpoint 0 and the transmit endpoints 1–3. The meaning of the EPn bits in this register is based on the mode of the device. The EP1 through EP3 bits always indicate that the USB controller is sending data; however, in Host mode, the bits refer to OUT endpoints; while in Device mode, the bits refer to IN endpoints. The EP0 bit is special in Host and Device modes and indicates that either a control IN or control OUT endpoint has generated an interrupt.

Mode(s): Host Device

USBTXIS is shown in Figure 17-6 and described in Table 17-8.

Figure 17-6. USB Transmit Interrupt Status Register (USBTXIS)

15–4	3	2	1	0
Reserved	EP3	EP2	EP1	EP0
R-0	R-0	R-0	R-0	R-0

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

Table 17-8. USB Transmit Interrupt Status Register (USBTXIS) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Reserved
3	EP3		TX Endpoint 3 Interrupt
		0	No interrupt
		1	The Endpoint 3 transmit interrupt is asserted.
2	EP2		TX Endpoint 2 Interrupt
		0	No interrupt
		1	The Endpoint 2 transmit interrupt is asserted.
1	EP1		TX Endpoint 1 Interrupt
		0	No interrupt
		1	The Endpoint 1 transmit interrupt is asserted.
0	EP0		TX and RX Endpoint 0 Interrupt
		0	No interrupt
		1	The Endpoint 0 transmit and receive interrupt is asserted.

17.6.4 USB Receive Interrupt Status Register (USBRXIS), offset 0x004

NOTE: Use caution when reading this register. Performing a read may change bit status.

The USB receive interrupt status 16-bit read-only register (USBRXIS) indicates which interrupts are currently active for receive endpoints 1–3.

Note: Bits relating to endpoints that have not been configured always return 0. All active interrupts are cleared when this register is read.

Mode(s): Host Device

USBRXIS is shown in Figure 17-7 and described in Table 17-9.

Figure 17-7. USB Receive Interrupt Status Register (USBRXIS)

15 – 4	3	2	1	0
Reserved	EP3	EP2	EP1	Rsvd
R-0	R-0	R-0	R-0	R-0

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

Table 17-9. USB Receive Interrupt Status Register (USBRXIS) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Reserved
3	EP3		RX Endpoint 3 Interrupt
		0	No interrupt
		1	The Endpoint 3 receive interrupt is asserted.
2	EP2		RX Endpoint 2 Interrupt
		0	No interrupt
		1	The Endpoint 2 receive interrupt is asserted.
1	EP1		RX Endpoint 1 Interrupt
		0	No interrupt
		1	The Endpoint 1 receive interrupt is asserted.
0	Reserved	0	Reserved

17.6.5 USB Transmit Interrupt Enable Register (USBTXIE), offset 0x006

The USB transmit interrupt enable 16-bit register (USBTXIE) provides interrupt enable bits for the interrupts in the USBTXIS register. When a bit is set, the USB interrupt is asserted to the interrupt controller when the corresponding interrupt bit in the USBTXIS register is set. When a bit is cleared, the interrupt in the USBTXIS register is still set but the USB interrupt to the interrupt controller is not asserted. On reset, all interrupts are enabled.

Mode(s): Host Device

USBTXIS is shown in Figure 17-8 and described in Table 17-10.

Figure 17-8. USB Transmit Interrupt Status Enable Register (USBTXIE)

15 - 4	3	2	1	0
Reserved	EP3	EP2	EP1	EP0
R-0	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 17-10. USB Transmit Interrupt Status Register (USBTXIE) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Reserved
3	EP3	0	TX Endpoint 3 Interrupt Enable
			The EP3 transmit interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the EP3 bit in the USBTXIS register is set.
2	EP2		TX Endpoint 2 Interrupt Enable
		0	The EP2 transmit interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the EP2 bit in the USBTXIS register is set.
1	EP1		TX Endpoint 1 Interrupt Enable
		0	The EP1 transmit interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the EP1 bit in the USBTXIS register is set.
0	EP0		TX and RX Endpoint 0 Interrupt Enable
		0	The EP0 transmit and receive interrupt is suppressed and not sent to the interupt controller.
		1	An interrupt is sent to the interrupt controller when the EP0 bit in the USBTXIS register is set.

17.6.6 USB Receive Interrupt Enable Register (USBRXIE), offset 0x008

The USB receive interrupt enable 16-bit register (USBTXIE) provides interrupt enable bits for the interrupts in the USBRXIS register. When a bit is set, the USB interrupt is asserted to the interrupt controller when the corresponding interrupt bit in the USBRXIS register is set. When a bit is cleared, the interrupt in the USBRXIS register is still set but the USB interrupt to the interrupt controller is not asserted. On reset, all interrupts are enabled.

Mode(s): Host Device

USBRXIE is shown in Figure 17-8 and described in Table 17-10.

Figure 17-9. USB Receive Interrupt Enable Register (USBRXIE)

15-4	3	2	1	0
Reserved	EP3	EP2	EP1	Rsvd
R-0	R/W-1	R/W-1	R/W-1	R-0

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

Table 17-11. USB Receive Interrupt Register (USBRXIE) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Reserved
3	EP3		RX Endpoint 3 Interrupt Enable
		0	The EP3 receive interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the EP3 bit in the USBRXIS register is set.
2	EP2		RX Endpoint 2 Interrupt Enable
		0	The EP2 receive interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the EP2 bit in the USBRXIS register is set.
1	EP1		RX Endpoint 1 Interrupt Enable
		0	The EP1 receive interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the EP1 bit in the USBRXIS register is set.
0	Reserved	0	Reserved

17.6.7 USB General Interrupt Status Register (USBIS), offset 0x00A

NOTE: Use caution when reading this register. Performing a read may change bit status.

The USB general interrupt status 8-bit read-only register (USBIS) indicates which USB interrupts are currently active. All active interrupts are cleared when this register is read.

Mode(s): Host Device

USBIS in Host Mode is shown in Figure 17-10 and described in Table 17-12.

Figure 17-10. USB General Interrupt Status Register (USBIS) in Host Mode

7	6	5	4	3	2	1	0
VBUSERR	SESREQ	DISCON	CONN	SOF	BABBLE	RESUME	Reserved
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 17-12. USB General Interrupt Status Register (USBIS) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	VBUSERR		VBUS Error
		0	No interrupt
		1	VBUS has dropped below the VBUS Valid threshold during a session.
6	SESREQ		Session Request
		0	No interrupt
		1	SESSION REQUEST signaling has been detected.
5	DISCON		Session Disconnect
		0	No interrupt
		1	A Device disconnect has been detected.
4	CONN		Session Connect
		0	No interrupt
		1	A Device connection has been detected.
3	SOF		Start of Frame
		0	No interrupt
		1	A new frame has started.
2	BABBLE		Babble Detected
		0	No interrupt
		1	Babble has been detected. This interrupt is active only after the first SOF has been sent.
1	RESUME		RESUME Signaling Detected. This interrupt can only be used if the USB controller's system clock is enabled. If the user disables the clock programming, the USBDRRIS, USBDRIM, and USBDRISC registers should be used.
		0	No effect
		1	RESUME signaling has been detected on the bus while the USB controller is in SUSPEND mode.
0	Reserved	0	Reserved

USBIS in Device Mode is shown in Figure 17-11 and described in Table 17-13.

Figure 17-11. USB General Interrupt Status Register (USBIS) in Device Mode

7	6	5	4	3	2	1	0
Reserved		DISCON	Reserved	SOF	RESET	RESUME	SUSPEND
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 17-13. USB General Interrupt Status Register (USBIS) in Device Mode Field Descriptions

Bit	Field	Value	Description
7-6	Reserved	0	Reserved
5	DISCON		Session Disconnect
		0	No interrupt
		1	The device has been disconnected from the host.
4	Reserved	0	Reserved
3	SOF		Start of frame
		0	No interrupt
		1	A new frame has started.
2	RESET		RESET Signaling Detected
		0	No interrupt
		1	RESET signaling has been detected on the bus.
1	RESUME		RESUME Signaling Detected. This interrupt can only be used if the USB controller's system clock is enabled. If the user disables the clock programming, the USBDRRIS, USBDRIM, and USBDRISC registers should be used.
		0	No interrupt
		1	RESUME signaling has been detected on the bus while the USB controller is in SUSPEND mode.
0	SUSPEND		SUSPEND Signaling Detected
		0	No interrupt
		1	SUSPEND signaling has been detected on the bus.

17.6.8 USB Interrupt Enable Register (USBIE), offset 0x00B

NOTE: Use caution when reading this register. Performing a read may change bit status.

The USB interrupt enable 8-bit register (USBIE) provides interrupt enable bits for each of the interrupts in USBIS. At reset interrupts 1 and 2 are enabled in device mode.

Mode(s): Host Device

USBIE in Host Mode is shown in Figure 17-12 and described in Table 17-14.

Figure 17-12. USB Interrupt Enable Register (USBIE) in Host Mode

7	6	5	4	3	2	1	0
VBUSERR	SESREQ	DISCON	CONN	SOF	BABBLE	RESUME	Reserved
R-W	R-W	R-W	R-W	R-W	R-W	R-W	R-0

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

Table 17-14. USB Interrupt Enable Register (USBIE) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	VBUSERR		Enable VBUS Error Interrupt
		0	The VBUSERR interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the VBUSERR bit in the USBIS register is set.
6	SESREQ		Enable Session Request
		0	The SESREQ interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the SESREEQ bit in the USBIS register is set.
5	DISCON		Enable Disconnect Interrupt
		0	The DISCON interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the DISCON bit in the USBIS register is set.
4	CONN		Enable Connect Interrupt
		0	The CONN interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the CONN bit in the USBIS register is set.
3	SOF		Start of Frame
		0	The SOF interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the SOF bit in the USBIS register is set.
2	BABBLE		Babble Detected
		0	The BABBLE interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the BABBLE bit in the USBIS register is set.
1	RESUME		RESUME Signaling Detected. This interrupt can only be used if the USB controller's system clock is enabled. If the user disables the clock programming, the USBDRRIS, USBDRIM, and USBDRISC registers should be used.
		0	The RESUME interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the RESUME bit in the USBIS register is set.
0	Reserved	0	Reserved

USBIE in Device Mode is shown in Figure 17-11 and described in Table 17-13.

Figure 17-13. USB Interrupt Enable Register (USBIE) in Device Mode

7	6	5	4	3	2	1	0
Reserved		DISCON	Reserved	SOF	RESET	RESUME	SUSPEND
R-0		R/W-0	R-0	R/W-0	R/W-1	RW-1	R/W-0

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

Table 17-15. USB Interrupt Enable Register (USBIE) in Device Mode Field Descriptions

Bit	Field	Value	Description
7-6	Reserved	0	Reserved
5	DISCON		Enable Disconnect Interrupt
		0	The DISCON interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the DISCON bit in the USBIS register is set.
4	Reserved	0	Reserved
3	SOF		Start of frame
		0	The SOF interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the SOF bit in the USBIS register is set.
2	RESET		RESET Signaling Detected
		0	The RESET interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the RESET bit in the USBIS register is set.
1	RESUME		RESUME Signaling Detected. This interrupt can only be used if the USB controller's system clock is enabled. If the user disables the clock programming, the USBDRRIS, USBDRIM, and USBDRISC registers should be used.
		0	The RESUME interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the RESUME bit in the USBIS register is set.
0	SUSPEND		SUSPEND Signaling Detected
		0	The SUSPEND interrupt is suppressed and not sent to the interrupt controller.
		1	An interrupt is sent to the interrupt controller when the DISCON bit in the USBIS register is set.

17.6.9 USB Frame Value Register (USBFRAME), offset 0x00C

The frame number 16-bit read-only register (USBFRAME) holds the last received frame number.

Mode(s): Host Device

USBFRAME is shown in Figure 17-14 and described in Table 17-16.

Figure 17-14. Frame Number Register (FRAME)

15 — 11	10 — 0
Reserved	FRAME
R-0	R-0

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

Table 17-16. Frame Number Register (FRAME) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	FRAME	0-7FFh	Last received frame number

17.6.10 USB Endpoint Index Register (USBEPIDX), offset 0x00E

Each endpoint buffer can be accessed by configuring a FIFO size and starting address. The endpoint index 16-bit register (USBEPIDX) is used with the USBTXFIFOSZ, USBRXFIFOSZ, USBTXFIFOADD, and USBRXFIFOADD registers.

Mode(s): Host Device

USBEPIDX is shown in Figure 17-15 and described in Table 17-17.

Figure 17-15. USB Endpoint Index Register (USBEPIDX)

7 — 4	3 — 0
Reserved	EPIDX
R-0	R/W-0

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

Table 17-17. USB Endpoint Index Register (USBEPIDX) Field Descriptions

Bit	Field	Value	Description
7-4	Reserved	0	Reserved
3-0	EPIDX	0-4h	Endpoint Index. This bit field configures which endpoint is accessed when reading or writing to one of the USB controller's indexed registers. A value of 0x0 corresponds to Endpoint 0 and a value of 0xF corresponds to Endpoint 15.

17.6.11 USB Test Mode Register (USBTEST), offset 0x00F

The USB test mode 8-bit register (USBTEST) is primarily used to put the USB controller into one of the four test modes for operation described in the USB Specification 2.0 , in response to a SET FEATURE: USBTESTMODE command. This register is not used in normal operation.

Note: Only one of these bits should be set at any time.

Mode(s): Host Device

USBTEST in Host Mode is shown in Figure 17-16 and described in Table 17-18.

Figure 17-16. USB Test Mode Register (USBTEST) in Host Mode

7	6	5	4-0
FORCEH	FIFOACC	FORCEFS	Reserved
R/W-0	R/W1S-0	R/W-0	R-0

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

Table 17-18. USB Test Mode Register (USBTEST) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	FORCEH		Force Host Mode. While in this mode, status of the bus connection may be read using the DEV bit of the USBDEVCTL register. The operating speed is determined from the FORCEFS bit.
		0	No effect
		1	Forces the USB controller to enter Host mode when the SESSION bit is set, regardless of whether the USB controller is connected to any peripheral. The state of the USB0DP and USB0DM signals is ignored. The USB controller then remains in Host mode until the SESSION bit is cleared, even if a Device is disconnected. If the FORCEH bit remains set, the USB controller re-enters Host mode the next time the SESSION bit is set.
6	FIFOACC		FIFO Access
		0	No effect
		1	Transfers the packet in the endpoint 0 transmit FIFO to the endpoint 0 receive FIFO.
5	FORCEFS		Force Full-Speed Mode
		0	The USB controller operates at Low Speed.
		1	Forces the USB controller into Full-Speed mode upon receiving a USB RESET.
4-0	Reserved	0	Reserved

USBTEST in Device Mode is shown in Figure 17-17 and described in Table 17-19.

Figure 17-17. USB Test Mode Register (USBTEST) in Device Mode

7	6	5	4-0
Reserved	FIFOACC	FORCEFS	Reserved
R-0	R/W1S-0	R/W-0	R-0

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

Table 17-19. USB Test Mode Register (USBTEST) in Device Mode Field Descriptions

Bit	Field	Value	Description
7	Reserved		Force Host Mode. While in this mode, status of the bus connection may be read using the DEV bit of the USBDEVCTL register. The operating speed is determined from the FORCEFS bit.
6	FIFOACC		FIFO Access
		0	No effect
		1	Transfers the packet in the endpoint 0 transmit FIFO to the endpoint 0 receive FIFO.

Table 17-19. USB Test Mode Register (USBTEST) in Device Mode Field Descriptions (continued)

Bit	Field	Value	Description
5	FORCEFS		Force Full-Speed Mode
		0	The USB controller operates at Low Speed.
		1	Forces the USB controller into Full-Speed mode upon receiving a USB RESET.
4-0	Reserved	0	Reserved

17.6.12 USB FIFO Endpoint n Register (USBFIFO[0]-USBFIFO[3])

NOTE: Use caution when reading these registers. Performing a read may change bit status.

The USB FIFO endpoint *n* 32-bit registers (USBFIFO[*n*]) provide an address for CPU access to the FIFOs for each endpoint. Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

Transfers to and from FIFOs can be 8-bit, 16-bit or 32-bit as required, and any combination of accesses is allowed provided the data accessed is contiguous. All transfers associated with one packet must be of the same width so that the data is consistently byte-, halfword- or word-aligned. However, the last transfer may contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.

Depending on the size of the FIFO and the expected maximum packet size, the FIFOs support either single-packet or double-packet buffering (see *Single-Packet Buffering* in Section 17.3.1.1.2). Burst writing of multiple packets is not supported as flags must be set after each packet is written.

Following a STALL response or a transmit error on endpoint 1–3, the associated FIFO is completely flushed.

For the specific offset for each FIFO register see Table 17-4.

Mode(s): Host Device

USBFIFO0-3 are shown in Figure 17-18 and described in Table 17-20.

Figure 17-18. USB FIFO Endpoint *n* Register (USBFIFO[*n*])

31 … 0
EPDATA
R/W-0

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

Table 17-20. USB FIFO Endpoint *n* Register (USBFIFO[*n*]) Field Descriptions

Bit	Field	Reset	Description
31-0	EPDATA	0x0000.0000	Endpoint Data. Writing to this register loads the data into the Transmit FIFO and reading unloads data from the Receive FIFO.

17.6.13 USB Device Control Register (USBDEVCTL), offset 0x060

The USB device control 8-bit register (USBDEVCTL) is used for controlling and monitoring the USB VBUS line. If the PHY is suspended, no PHY clock is received and the VBUS is not sampled. In addition, in Host mode, USBDEVCTL provides the status information for the current operating mode (Host or Device) of the USB controller. If the USB controller is in Host mode, this register also indicates if a full- or low-speed Device has been connected.

Mode(s): Host Device

USBDEVCTL is shown in Figure 17-19 and described in Table 17-21.

Figure 17-19. USB Device Control Register (USBDEVCTL)

7	6	5	4-3	2	1	0
DEV	FSDEV	LSDEV	VBUS	HOSTMODE	HOSTREQ	SESSION
R-1	R-0	R-0	R-0	R-0	R/W-0	R/W-0

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

Table 17-21. USB Device Control Register (USBDEVCTL) Field Descriptions

Bit	Field	Value	Description
7	DEV		Device mode
		0	The USB controller is operating on the OTG A side of the cable.
		1	The USB controller is operating on the OTG B side of the cable.
			Only valid while a session is in progress.
6	FSDEV		Full-Speed Device Detected
		0	A full-speed Device has not been detected on the port.
		1	A full-speed Device has been detected on the port.
5	LSDEV		Low-Speed Device Detected
		0	A low-speed Device has not been detected on the port.
		1	A low-speed Device has been detected on the port.
4-3	VBUS	0-3h	These read-only bits encode the current VBus level as follows:
		0	Below Session End. VBUS is detected as under 0.5 V.
		1h	Above Session End, below AValid. VBUS is detected as above 0.5 V and under 1.5 V.
		2h	Above AValid, below VBusValid. VBUS is detected as above 1.5 V and below 4.75 V.
		3h	Above VBusValid. VBUS is detected as above 4.75 V.
2	HOSTMODE		This read-only bit is set when the USB controller is acting as a Host.
		0	The USB controller is acting as a Device.
		1	The USB controller is acting as a Host.
			Only valid while a session is in progress.
1	HOSTREQ		When set, the USB controller will initiate the Host Negotiation when Suspend mode is entered. It is cleared when Host Negotiation is completed.
		0	No effect
		1	Initiates the Host Negotiation when SUSPENDmode is entered.

Table 17-21. USB Device Control Register (USBDEVCTL) Field Descriptions (continued)

Bit	Field	Value	Description
0	SESSION		Session Start/End When operating as a Host:
		0	When cleared by software, this bit ends a session.
		1	When set by software, this bit starts a session.
			When operating as a Device:
		0	The USB controller has ended a session. When the USB controller is in SUSPEND mode, this bit may be cleared by software to perform a software disconnect.
		1	The USB controller has started a session. When set by software, the Session Request Protocol is initiated.
			Clearing this bit when the USB controller is not suspended results in undefined behavior.

17.6.14 USB Transmit Dynamic FIFO Sizing Register (USBTXFIFOSZ), offset 0x062

The USB transmit dynamic FIFO sizing 8-bit register (USBTXFIFOSZ) allows the selected TX endpoint FIFOs to be dynamically sized. USBEPIDX is used to configure each transmit endpoint's FIFO size.

Mode(s): Host Device

USBTXFIFOSZ is shown in Figure 17-20 and described in Table 17-22.

Figure 17-20. USB Transmit Dynamic FIFO Sizing Register (USBTXFIFOSZ)

7 5	4	3 0
Reserved	DPB	SZ
R-0	R/W-0	R-0

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

Table 17-22. USB Transmit Dynamic FIFO Sizing Register (USBTXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved
4	DPB		Double Packet Buffering Support
		0	Single packet buffering is supported.
		1	Double packet buffering is enabled.
3-0	SZ		Maximum packet size to be allowed. If DPB = 0, the FIFO also is this size; if DPB = 1, the FIFO is twice this size. Packet size in bytes:
		0h	8
		1h	16
		2h	32
		3h	64
		4h	128
		5h	256
		6h	512
		7h	1024
		8h	2048
		9-Fh	Reserved

17.6.15 USB Receive Dynamic FIFO Sizing Register (USBRXFIFOSZ), offset 0x063

The USB receive dynamic FIFO sizing 8-bit register (USBRXFIFOSZ) allows the selected RX endpoint FIFOs to be dynamically sized.

Mode(s): Host Device

USBRXFIFOSZ is shown in Figure 17-21 and described in Table 17-23.

Figure 17-21. USB Receive Dynamic FIFO Sizing Register (USBRXFIFOSZ)

7 - 5	4	3 - 0
Reserved	DPB	SZ
R-0	R/W-0	R-0

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

Table 17-23. USB Receive Dynamic FIFO Sizing Register (USBRXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved
4	DPB		Double Packet Buffering Support
		0	Single packet buffering is supported.
		1	Double packet buffering is enabled.
3-0	SZ		Maximum packet size to be allowed. If DPB = 0, the FIFO also is this size; if DPB = 1, the FIFO is twice this size. Packet size in bytes:
		0h	8
		1h	16
		2h	32
		3h	64
		4h	128
		5h	256
		6h	512
		7h	1024
		8h	2048
		9-Fh	Reserved

17.6.16 USB Transmit FIFO Start Address Register (USBTXFIFOADD), offset 0x064

The USB transmit FIFO start address 16-bit register (USBTXFIFOADD) controls the start address of the selected transmit endpoint FIFOs.

Mode(s): Host Device

USBTXFIFOADDR is shown in Figure 17-22 and described in Table 17-24.

Figure 17-22. USB Transmit FIFO Start Address Register (USBTXFIFOADDR])

15 - 9	8 - 0
Reserved	ADDR
R-0	R/W-0

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

Table 17-24. USB Transmit FIFO Start Address Register (USBTXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved
8-0	ADDR		Start Address of the endpoint FIFO in units of 8 bytes.
		0h	0
		1h	8
		2h	16
		3h	24
		4h	32
		5h	40
		6h	48
		7h	56
		8h	64
		..	..
		1FFh	4095

17.6.17 USB Receive FIFO Start Address Register (USBRXFIFOADD), offset 0x066

The USB receive FIFO start address 16-bit register (USBRXFIFOADD) controls the start address of the selected receive endpoint FIFOs.

Mode(s): Host Device

USBRXFIFOADDR is shown in Figure 17-23 and described in Table 17-25.

Figure 17-23. USB Receive FIFO Start Address Register (USBRXFIFOADDR)

15 - 9	8 - 0
Reserved	ADDR
R-0	R/W-0

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

Table 17-25. USB Receive FIFO Start Address Register (USBRXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved
8-0	ADDR		Start Address of the endpoint FIFO in units of 8 bytes.
		0h	0
		1h	8
		2h	16
		3h	24
		4h	32
		5h	40
		6h	48
		7h	56
		8h	64
		..	..
		1FFh	4095

17.6.18 USB Connect Timing Register (USBCONTIM), offset 0x07A

The USB connect timing 8-bit configuration register (USBCONTIM) specifies connection and negotiation delays.

Mode(s): Host Device

USBCONTIM is shown in Figure 17-24 and described in Table 17-26.

Figure 17-24. USB Connect Timing Register (USBCONTIM)

7 4	3 0
WTCON	WTID
R/W-1	R/W-1

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

Table 17-26. USB Connect Timing Register (USBCONTIM) Field Descriptions

Bit	Field	Value	Description
7-4	WTCON	5h	The connect wait field configures the wait required to allow for the user’s connect/disconnect filter, in units of 533.3 ns. The default corresponds to 2.667 µs.
3-0	WTID	Ch	The wait ID field configures the delay required from the enable of the ID detection to when the ID value is valid, in units of 4.369 ms. The default corresponds to 52.43 ms.

17.6.19 USB Full-Speed Last Transaction to End of Frame Timing Register (USBFSEOF), offset 0x07D

USB full-speed last transaction to end of frame timing 8-bit configuration register (USBFSEOF) specifies the minimum time gap allowed between the start of the last transaction and the EOF for full-speed transactions.

Mode(s): Host Device

USBFSEOF is shown in Figure 17-25 and described in Table 17-27.

Figure 17-25. USB Full-Speed Last Transaction to End of Frame Timing Register (USBFSEOF)

7 … 0
FSEOFG
R/W-0x77

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

Table 17-27. USB Full-Speed Last Transaction to End of Frame Timing Register (USBFSEOF) Field Descriptions

Bit	Field	Reset	Description
7-0	FSEOFG	77h	The full-speed end-of-frame gap field is used during full-speed transactions to configure the gap between the last transaction and the End-of-Frame (EOF), in units of 533.3 ns. The default corresponds to 63.46 μs.

17.6.20 USB Low-Speed Last Transaction to End of Frame Timing Register (USBLSEOF), offset 0x07E

The USB low-speed last transaction to end of frame timing 8-bit configuration register (USBLSEOF) specifies the minimum time gap that is to be allowed between the start of the last transaction and the EOF for low-speed transactions.

Mode(s): Host Device

USBLSEOF is shown in Figure 17-26 and described in Table 17-28.

Figure 17-26. USB Low-Speed Last Transaction to End of Frame Timing Register (USBLSEOF)

7 … 0
LSEOFG
R/W-0x72

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

Table 17-28. USB Low-Speed Last Transaction to End of Frame Timing Register (USBLSEOF) Field Descriptions

Bit	Field	Reset	Description
7-0	LSEOFG	72h	The low-speed end-of-frame gap field is used during low-speed transactions to set the gap between the last transaction and the End-of-Frame (EOF), in units of 1.067 μs. The default corresponds to 121.6 μs.

17.6.21 USB Transmit Functional Address Endpoint n Registers (USBTXFUNCADDR[0]-USBTXFUNCADDR[3])

The transmit functional address endpoint *n* 8-bit registers (USBTXFUNCADDR[*n*]) record the address of the target function to be accessed through the associated endpoint (EP*n*). USBTXFUNCADDRx must be defined for each transmit endpoint that is used.

Note: USBTXFUNCADDR0 is used for both receive and transmit for endpoint 0.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBTXFUNCADDR[*n*] registers are shown in Figure 17-27 and described in Table 17-29.

Figure 17-27. USB Transmit Functional Address Endpoint *n* Registers (USBTXFUNCADDR[*n*])

7	6 - 0
Reserved	ADDR
R-0	R/W-0

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

Table 17-29. USB Transmit Functional Address Endpoint *n* Registers (USBTXFUNCADDR[*n*]) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	ADDR	0	Device Address specifies the USB bus address for the target Device.

17.6.22 USB Transmit Hub Address Endpoint n Registers (USBTXHUBADDR[0]-USBTXHUBADDR[3])

The transmit hub address endpoint *n* 8-bit read/write registers (USBTXHUBADDR[*n*]), like USBTXHUBPORT[*n*], must be written only when a USB device is connected to transmit endpoint EP*n* via a USB 2.0 hub. This register records the address of the USB 2.0 hub through which the target associated with the endpoint is accessed.

Note: USBTXHUBADDR0 is used for both receive and transmit for endpoint 0.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBTXHUBADDR[*n*] registers are shown in Figure 17-27 and described in Table 17-29.

Figure 17-28. USB Transmit Hub Address Endpoint *n* Registers (USBTXHUBADDR[*n*])

7	6 ... 0
Reserved	ADDR
R-0	R/W-0

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

Table 17-30. USB Transmit Hub Address Endpoint *n* Registers(USBTXHUBADDR[*n*]) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	ADDR	0	Device Address specifies the USB bus address for the target Device.

17.6.23 USB Transmit Hub Port Endpoint n Registers (USBTXHUBPORT[0]-USBTXHUBPORT[3])

The transmit hub port endpoint *n* 8-bit read/write registers (USBTXHUBPORT[*n*]), like USBTXHUBADDR[*n*], must be written only when a full- or low-speed Device is connected to transmit endpoint EP*n* via a USB 2.0 hub. This register records the port of the USB 2.0 hub through which the target associated with the endpoint is accessed.

Note: USBTXHUBPORT0 is used for both receive and transmit for endpoint 0.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBTXHUBPORT*n* registers are shown in Figure 17-29 and described in Table 17-31.

Figure 17-29. USB Transmit Hub Port Endpoint *n* Registers (USBTXHUBPORT[*n*])

7	6–0
Reserved	PORT
R-0	R/W-0

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

Table 17-31. USB Transmit Hub Port Endpoint *n* Registers(USBTXHUBPORT[*n*]) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	PORT	0	Hub Port specifies the USB hub port number.

17.6.24 USB Receive Functional Address Endpoint n Registers (USBRXFUNCADDR[1]-USBRXFUNCADDR[3])

The recieve functional address endpoint *n* 8-bit read/write registers (USBRXFUNCADDR[*n*]) record the address of the target function to be accessed through the associated endpoint (EP*n*). USBRXFUNCADDRx must be defined for each receive endpoint that is used.

Note: USBTXFUNCADDR0 is used for both receive and transmit for endpoint 0.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBRXFUNCADDR[*n*] registers are shown in Figure 17-30 and described in Table 17-32.

Figure 17-30. USB Receive Functional Address Endpoint *n* Registers (USBFIFO[*n*])

7	6 — 0
Reserved	ADDR
R-0	R/W-0

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

Table 17-32. USB Recieve Functional Address Endpoint *n* Registers(USBFIFO[*n*]) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	ADDR	0	Device Address specifies the USB bus address for the target Device.

17.6.25 USB Receive Hub Address Endpoint n Registers (USBRXHUBADDR[1]-USBRXHUBADDR[3])

The receive hub address endpoint *n* 8-bit read/write registers (USBRXHUBADDR[*n*]), like [*n*], must be written only when a full- or low-speed Device is connected to receive endpoint EP*n* via a USB 2.0 hub. Each register records the address of the USB 2.0 hub through which the target associated with the endpoint is accessed.

Note: USBTXHUBADDR0 is used for both receive and transmit for endpoint 0.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBRXHUBADDR[*n*] registers are shown in Figure 17-31 and described in Table 17-33.

Figure 17-31. USB Receive Hub Address Endpoint *n* Registers (USBRXHUBADDR[*n*])

7	6 ... 0
MULTTRAN	ADDR
R/w-0	R/W-0

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

Table 17-33. USB Receive Hub Address Endpoint *n* Registers(USBRXHUBADDR[*n*]) Field Descriptions

Bit	Field	Value	Description
7	MULTTRAN		Multiple Translators
		0	Clear to indicate that the hub has a single transaction translator.
		1	Set to indicate that the hub has multiple transaction translators.
6-0	ADDR	0	Device Address specifies the USB bus address for the target Device.

17.6.26 USB Receive Hub Port Endpoint n Registers (USBRXHUBPORT[1]-USBRXHUBPORT[3])

The receive hub port endpoint *n* 8-bit read/write registers (USBRXHUBPORT[*n*]), like USBRXHUBADDR[*n*], must be written only when a full- or low-speed device is connected to receive endpoint EP*n* via a USB 2.0 hub. Each register records the port of the USB 2.0 hub through which the target associated with the endpoint is accessed.

Note: USBTXHUBPORT0 is used for both receive and transmit for endpoint 0.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBRXHUBPORT*n* registers are shown in Figure 17-32 and described in Table 17-34.

Figure 17-32. USB Transmit Hub Port Endpoint *n* Registers (USBRXHUBPORT[*n*])

7	6 - 0
Reserved	PORT
R-0	R/W-0

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

Table 17-34. USB Transmit Hub Port Endpoint *n* Registers(USBRXHUBPORT[*n*]) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	PORT	0	Hub Port specifies the USB hub port number.

17.6.27 USB Maximum Transmit Data Endpoint n Registers (USBTXMAXP[1]-USBTXMAXP[3])

The USB maximum transmit data endpoint *n* 16-bit registers (USBTXMAXP[*n*]) define the maximum amount of data that can be transferred through the selected transmit endpoint in a single operation.

Bits 10:0 define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the *USB Specification* on packet sizes for bulk, interrupt and isochronous transfers in full-speed operation.

The total amount of data represented by the value written to this register must not exceed the FIFO size for the transmit endpoint, and must not exceed half the FIFO size if double-buffering is required.

If this register is changed after packets have been sent from the endpoint, the transmit endpoint FIFO must be completely flushed (using the FLUSH bit in USBTXCSRLn) after writing the new value to this register.

Note: USBTXMAXP[*n*] must be set to an even number of bytes for proper interrupt generation in DMA Basic Mode.

For the specific offset for each register see Table 17-4.

Mode(s): Host Device

The USBTXMAXP[*n*] registers are shown in Figure 17-33 and described in Table 17-35.

Figure 17-33. USB Maximum Transmit Data Endpoint *n* Registers (USBTXMAXP[*n*])

15 - 11	10 - 0
Reserved	MAXLOAD
R-0	R/W-000

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

Table 17-35. USB Maximum Transmit Data Endpoint *n* Registers(USBTXMAXP[*n*]) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	MAXLOAD		Maximum Payload specifies the maximum payload in bytes per transaction.

17.6.28 USB Control and Status Endpoint 0 Low Register (USBCSRL0), offset 0x102

The USB control and status endpoint 0 low 8-bit register (USBCSRL0) provides control and status bits for endpoint 0.

Mode(s): Host Device

USBCSRL0 in Host mode is shown in Figure 17-34 and described in Table 17-36.

Figure 17-34. USB Control and Status Endpoint 0 Low Register (USBCSRL0) in Host Mode

7	6	5	4	3	2	1	0
NAKTO	STATUS	REQPKT	ERROR	SETUP	STALLED	TXRDY	RXRDY
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 17-36. USB Control and Status Endpoint 0 Low Register(USBCSRL0) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	NAKTO		NAK Timeout. Software must clear this bit to allow the endpoint to continue.
		0	No timeout
		1	Indicates that endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the USBNAKLMT register.
6	STATUS		Status Packet. Setting this bit ensures that the DT bit is set in the USBCSRH0 register so that a DATA1 packet is used for the STATUS stage transaction.
		0	No transaction
		1	Initiates a STATUS stage transaction. This bit must be set at the same time as the TXRDY or REQPKT bit is set.
			This bit is automatically cleared when the STATUS stage is over.
5	REQPKT		Request Packet. This bit is cleared when the RXRDY bit is set.
		0	No request
		1	Requests an IN transaction.
4	ERROR		Error. Software must clear this bit.
		0	No error
		1	Three attempts have been made to perform a transaction with no response from the peripheral. The EP0 bit in the USBTXIS register is also set in this situation.
3	SETUP		Setup Packet. Setting this bit always clears the DT bit in the USBCSRH0 register to send a DATA0 packet.
		0	Sends an OUT token.
		1	Sends a SETUP token instead of an OUT token for the transaction. This bit should be set at the same time as the TXRDY bit is set.
2	STALLED		Endpoint Stalled. Software must clear this bit.
		0	No handshake has been received.
		1	A STALL handshake has been received.
1	TXRDY		Transmit Packet Ready. If both the TXRDY and SETUP bits are set, a setup packet is sent. If just TXRDY is set, an OUT packet is sent.
		0	No transmit packet is ready.
		1	Software sets this bit after loading a data packet into the TX FIFO. The EP0 bit in the USBTXIS register is also set in this situation.
0	RXRDY		Receive Packet Ready. Software must clear this bit after the packet has been read from the FIFO to acknowledge that the data has been read from the FIFO.
		0	No receive packet has been received.
		1	Indicates that a data packet has been received in the RX FIFO. The EP0 bit in the USBTXIS register is also set in this situation.

USBCSRL0 in Device mode is shown in Figure 17-35 and described in Table 17-37.

Figure 17-35. USB Control and Status Endpoint 0 Low Register (USBCSRL0) in Device Mode

7	6	5	4	3	2	1	0
SETENDC	RXRDYC	STALL	SETEND	DATAEND	STALLED	TXRDY	RXRDY
W1C-0	W1C-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 17-37. USB Control and Status Endpoint 0 Low Register (USBCSRL0) in Device Mode Field Descriptions

Bit	Field	Value	Description
7	SETENDC		Setup End Clear
		0	No effect
		1	Writing a 1 to this bit clears the SETEND bit.
6	RXRDYC		RXRDY Clear
		0	No effect
		1	Writing a 1 to this bit clears the RXRDY bit.
5	STALL		Send Stall.
		0	No effect
		1	Terminates the current transaction and transmits the STALL handshake.
			This bit is cleared automatically after the STALL handshake is transmitted.
4	SETEND		Setup end.
		0	A control transaction has not ended or ended after the DATAEND bit was set.
		1	A control transaction has ended before the DATAEND bit has been set. The EP0 bit in the USBTXIS register is also set in this situation.
			This bit is cleared by writing a 1 to the SETENDC bit.
3	DATAEND		Data end.
		0	No effect
		1	Set this bit in the following situations: • When setting TXRDY for the last data packet • When clearing RXRDY after unloading the last data packet • When setting TXRDY for a zero-length data packet
			This bit is cleared automatically.
2	STALLED		Endpoint Stalled. Software must clear this bit.
		0	A STALL handshake has not been transmitted.
		1	A STALL handshake has been transmitted.
1	TXRDY		Transmit Packet Ready. If both the TXRDY and SETUP bits are set, a setup packet is sent. If just TXRDY is set, an OUT packet is sent.
		0	No transmit packet is ready.
		1	Software sets this bit after loading an IN data packet into the TX FIFO. The EP0 bit in the USBTXIS register is also set in this situation.
0	RXRDY		Receive Packet Ready.
		0	No receive packet has been received.
		1	A data packet has been received. The EP0 bit in the USBTXIS register is also set in this situation.
			This bit is cleared by writing a 1 to the RXRDYC bit.

17.6.29 USB Control and Status Endpoint 0 High Register (USBCSRH0), offset 0x103

The USB control and status endpoint 0 high 8-bit register (USBCSRH0) provides control and status bits for endpoint 0.

Mode(s): Host Device

USBCSRH0 in Host mode is shown in Figure 17-36 and described in Table 17-38.

Figure 17-36. USB Control and Status Endpoint 0 High Register (USBCSRH0) in Host Mode

7 3	2	1	0
Reserved	DTWE	DT	FLUSH
R-0	R/W-0	R/W-0	R/W-0

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

Table 17-38. USB Control and Status Endpoint 0 High Register (USBCSRH0) in Host Mode Field Descriptions

Bit	Field	Value	Description
7-3	Reserved	0	Reserved
2	DTWE		Data Toggle Write Enable. This bit is automatically cleared once the new value is written.
		0	The DT bit cannot be written.
		1	Enables the current state of the endpoint 0 data toggle to be written (see DT bit).
1	DT		Data Toggle. When read, this bit indicates the current state of the endpoint 0 data toggle. If DTWE is set, this bit may be written with the required setting of the data toggle. If DTWE is Low, this bit cannot be written. Care should be taken when writing to this bit as it should only be changed to RESET USB endpoint 0.
0	FLUSH		Flush FIFO. This bit is automatically cleared after the flush is performed.
		0	No effect
		1	Flushes the next packet to be transmitted/read from the endpoint 0 FIFO. The FIFO pointer is reset and the TXRDY/RXRDY bit is cleared. **Note:** This bit should only be set when TXRDY/RXRDY is set. At other times, it may cause data to be corrupted.

USBCSRH0 in Device mode is shown in Figure 17-37 and described in Table 17-39.

Figure 17-37. USB Control and Status Endpoint 0 High Register (USBCSRH0) in Device Mode

7 1	0
Reserved	FLUSH
R-0	R/W-0

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

Table 17-39. USB Control and Status Endpoint 0 High Register (USBCSRH0) in Device Mode Field Descriptions

Bit	Field	Value	Description
7-1	Reserved	0	Reserved
0	FLUSH		Flush FIFO. This bit is automatically cleared after the flush is performed.
		0	No effect
		1	Flushes the next packet to be transmitted/read from the endpoint 0 FIFO. The FIFO pointer is reset and the TXRDY/RXRDY bit is cleared. **Note:** This bit should only be set when TXRDY/RXRDY is set. At other times, it may cause data to be corrupted.

17.6.30 USB Receive Byte Count Endpoint 0 Register (USBCOUNT0), offset 0x108

The USB receive byte count endpoint 0 8-bit read-only register (USBCOUNT0) indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while the RXRDY bit is set.

Mode(s): Host Device

USBCOUNT0 is shown in Figure 17-38 and described in Table 17-29.

Figure 17-38. USB Receive Byte Count Endpoint 0 Register (USBCOUNT0)

7	6 — 0
Reserved	COUNT
R-0	R/W-0

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

Table 17-40. USB Receive Byte Count Endpoint 0 Register (USBCOUNT0) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	COUNT	0	FIFO Count. COUNT is a read-only value that indicates the number of received data bytes in the endpoint 0 FIFO.

17.6.31 USB Type Endpoint 0 Register (USBTYPE0), offset 0x10A

The USB type endpoint 0 8-bit register (USBTYPE0) must be written with the operating speed of the targeted Device being communicated with using endpoint 0.

Mode(s): Host

USBTYPE0 is shown in Figure 17-39 and described in Table 17-41.

Figure 17-39. USB Type Endpoint 0 Register (USBTYPE0)

7 — 6	5 — 0
SPEED	Reserved
R/W-0	R-0

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

Table 17-41. USB Type Endpoint 0 Register (USBTYPE0) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED	0	Operating Speed specifies the operating speed of the target Device. If selected, the target is assumed to have the same connection speed as the USB controller.
		0-1h	Reserved
		2h	Full
		3h	Low
5-0	Reserved	0	Reserved

17.6.32 USB NAK Limit Register (USBNAKLMT), offset 0x10B

The USB NAK limit 8-bit read-only register (USBNAKLMT) sets the number of frames after which endpoint 0 should time out on receiving a stream of NAK responses. (Equivalent settings for other endpoints can be made through their USBTXINTERVAL[*n*] and USBRXINTERVAL[*n*] registers.)

The number of frames selected is 2(m-1) (where m is the value set in the register, with valid values of 2–16). If the Host receives NAK responses from the target for more frames than the number represented by the limit set in this register, the endpoint is halted.

Note: A value of 0 or 1 disables the NAK timeout function.

Mode(s): Host

USBNAKLMT is shown in Figure 17-40 and described in Table 17-42.

Figure 17-40. USB NAK Limit Register (USBNAKLMT)

7 5	4 0
Reserved	NAKLMT
R-0	R/W-0

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

Table 17-42. USB NAK Limit Register (USBNAKLMT) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved
4-0	NAKLMT	0	EP0 NAK Limit specifies the number of frames after receiving a stream of NAK responses.

17.6.33 USB Transmit Control and Status Endpoint n Low Register (USBTXCSRL[1]-USBTXCSRL[3])

The USB transmit control and status endpoint *n* low 8-bit registers (USBTXCSRL[*n*]) provide control and status bits for transfers through the currently selected transmit endpoint.

For the specific offset for each register see Table 17-4.

Mode(s): Host Device

The USBTXCSRL[*n*] registers in Host Mode are shown in Figure 17-41 and described in Table 17-43.

Figure 17-41. USB Transmit Control and Status Endpoint n Low Register (USBTXCSRL[*n*]) in Host Mode

7	6	5	4	3	2	1	0
NAKTO	CLRDT	STALLED	SETUP	FLUSH	ERROR	FIFONE	TXRDY
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 17-43. USB Transmit Control and Status Endpoint n Low Register (USBTXCSRL[*n*]) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	NAKTO		NAK Timeout. Software must clear this bit to allow the endpoint to continue.
		0	No timeout
		1	Bulk endpoints only: Indicates that the transmit endpoint is halted following the receipt of NAK responses for longer than the time set by the NAKLMT field in the USBTXINTERVAL[*n*] register.
6	CLRDT		Clear DataToggle
		0	No effect
		1	Writing a 1 to this bit clears the DT bit in the USBTXCSRH[*n*] register.
5	STALLED		Endpoint Stalled. Software must clear this bit.
		0	A STALL handshake has not been received
		1	Indicates that a STALL handshake has been received. When this bit is set, any DMA request that is in progress is stopped, the FIFO is completely flushed, and the TXRDY bit is cleared.
4	SETUP		Setup Packet.
		0	No SETUP token is sent.
		1	Sends a SETUP token instead of an OUT token for the transaction. This bit should be set at the same time as the TXRDY bit is set.
			Note: Setting this bit also clears the DT bit in the USBTXCSRH[*n*] register.
3	FLUSH		Flush FIFO. This bit can be set simultaneously with the TXRDY bit to abort the packet that is currently being loaded into the FIFO. Note that if the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO.
		0	No effect
		1	Flushes the latest packet from the endpoint transmit FIFO. The FIFO pointer is reset and the TXRDY bit is cleared. The EPn bit in the USBTXIS register is also set in this situation.
			Note: This bit should only be set when the TXRDY bit is set. At other times, it may cause data to be corrupted.
2	ERROR		Error. Software must clear this bit.
		0	No error
		1	Three attempts have been made to send a packet and no handshake packet has been received. The TXRDY bit is cleared, the EPn bit in the USBTXIS register is set, and the FIFO is completely flushed in this situation.
			Note: This bit is valid only when the endpoint is operating in Bulk or Interrupt mode.
1	FIFONE		FIFO Not Empty
		0	The FIFO is empty
		1	At least one packet is in the transmit FIFO.

Table 17-43. USB Transmit Control and Status Endpoint n Low Register (USBTXCSRL[*n*]) in Host Mode Field Descriptions (continued)

Bit	Field	Value	Description
0	TXRDY		Transmit Packet Ready. This bit is cleared automatically when a data packet has been transmitted. The EP*n* bit in the USBTXIS register is also set at this point. TXRDY is also automatically cleared prior to loading a second packet into a double-buffered FIFO.
		0	No transmit packet is ready.
		1	Software sets this bit after loading a data packet into the TX FIFO.

The USBTXCSRL[*n*] registers in Device Mode are shown in Table 17-43 and described in Figure 17-42.

Figure 17-42. USB Transmit Control and Status Endpoint n Low Register (USBTXCSRL[*n*]) in Device Mode

7	6	5	4	3	2	1	0
Reserved	CLRDT	STALLED	STALL	FLUSH	UNDRN	FIFONE	TXRDY
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; -*n* = value after reset

Table 17-44. USB Transmit Control and Status Endpoint n Low Register (USBTXCSRL[*n*]) in Device Mode Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6	CLRDT		Clear Data Toggle
		0	No effect
		1	Writing a 1 to this bit clears the DT bit in the USBTXCSRH[*n*] register.
5	STALLED		Endpoint Stalled. Software must clear this bit.
		0	A STALL handshake has not been transmitted.
		1	A STALL handshake has been transmitted. The FIFO is flushed and the TXRDY bit is cleared.
4	STALL		Send Stall. Software clears this bit to terminate the STALL condition. **Note:** This bit has no effect in isochronous transfers.
		0	No effect
		1	Issues a STALL handshake to an IN token.
3	FLUSH		Flush FIFO. This bit may be set simultaneously with the TXRDY bit to abort the packet that is currently being loaded into the FIFO. Note that if the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO. **Note:** This bit should only be set when the TXRDY bit is set. At other times, it may cause data to be corrupted.
		0	No effect
		1	Flushes the latest packet from the endpoint transmit FIFO. The FIFO pointer is reset and the TXRDY bit is cleared. The EPn bit in the USBTXIS register is also set in this situation.
			This bit is cleared automatically.
2	UNDRN		Underrun. Software must clear this bit.
		0	No underrun
		1	An IN token has been received when TXRDY is not set.
1	FIFONE		FIFO Not Empty
		0	The FIFO is empty.
		1	At least one packet is in the transmit FIFO.

Table 17-44. USB Transmit Control and Status Endpoint n Low Register (USBTXCSRL[*n*]) in Device Mode Field Descriptions (continued)

Bit	Field	Value	Description
0	TXRDY		Transmit Packet Ready. This bit is cleared automatically when a data packet has been transmitted. The EPn bit in the USBTXIS register is also set at this point. TXRDY is also automatically cleared prior to loading a second packet into a double-buffered FIFO.
		0	No transmit packet is ready.
		1	Software sets this bit after loading a data packet into the TX FIFO. This bit is cleared by writing a 1 to the RXRDYC bit.

17.6.34 USB Transmit Control and Status Endpoint n High Register (USBTXCSRH[1]-USBTXCSRH[3])

The USB transmit control and status endpoint *n* high 8-bit registers (USBTXCSRH[*n*]) provide additional control for transfers through the currently selected transmit endpoint.

For the specific offset for each register see Table 17-4.

Mode(s): Host Device

The USBTXCSRH[*n*] registers in Host Mode are shown in Figure 17-43 and described in Table 17-45.

Figure 17-43. USB Transmit Control and Status Endpoint n High Register (USBTXCSRH[*n*]) in Host Mode

7	6	5	4	3	2	1	0
AUTOSET	Reserved	MODE	DMAEN	FDT	DMAMOD	DTWE	DT
R/W-0	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; -*n* = value after reset

Table 17-45. USB Transmit Control and Status Endpoint n High Register (USBTXCSRH[*n*]) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	AUTOSET		Auto Set
		0	The TXRDY bit must be set manually.
		1	Enables the TXRDY bit to be automatically set when data of the maximum packet size (value in USBTXMAXP[*n*]) is loaded into the transmit FIFO. If a packet of less than the maximum packet size is loaded, then the TXRDY bit must be set manually.
6	Reserved	0	Reserved
5	MODE		Mode **Note:** This bit only has an effect when the same endpoint FIFO is used for both transmit and receive transactions.
		0	Enables the endpoint direction as RX.
		1	Enables the endpoint direction as TX.
4	DMAEN		DMA Request Enable **Note:** Three TX and three /RX endpoints can be connected to the DMA module. If this bit is set for a particular endpoint, the DMAATX, DMABTX, or DMACTX field in the USB DMA Select (USBDMASEL) register must be programmed correspondingly.
		0	Disables the DMA request for the transmit endpoint.
		1	Enables the DMA request for the transmit endpoint.
3	FDT		Force Data Toggle
		0	No effect
		1	Forces the endpoint DT bit to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This bit can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints. **Note:** This bit should only be set when the TXRDY bit is set. At other times, it may cause data to be corrupted.
2	DMAMOD		DMA Request Mode **Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared.
		0	An interrupt is generated after every DMA packet transfer.
		1	An interrupt is generated only after the entire DMA transfer is complete. **Note:** This bit is valid only when the endpoint is operating in Bulk or Interrupt mode.
1	DTWE		Data Toggle Write Enable. This bit is automatically cleared once the new value is written.
		0	The DT bit cannot be written.
		1	Enables the current state of the transmit endpoint data to be written (see DT bit).

Table 17-45. USB Transmit Control and Status Endpoint n High Register (USBTXCSRH[*n*]) in Host Mode Field Descriptions (continued)

Bit	Field	Value	Description
0	DT		Data Toggle. When read, this bit indicates the current state of the transmit endpoint data toggle. If DTWE is High, this bit can be written with the required setting of the data toggle. If DTWE is Low, any value written to this bit is ignored. Care should be taken when writing to this bit as it should only be changed to RESET the transmit endpoint.

The USBTXCSRH[*n*] registers in Device Mode are shown in Figure 17-44 and described in Table 17-46.

Figure 17-44. USB Transmit Control and Status Endpoint n High Register (USBTXCSRH[*n*]) in Device Mode

7	6	5	4	3	2	1	0
AUTOSET	ISO	MODE	DMAEN	FDT	DMAMOD	Reserved	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	

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

Table 17-46. USB Transmit Control and Status Endpoint n High Register (USBTXCSRH[*n*]) in Device Mode Field Descriptions

Bit	Field	Value	Description
7	AUTOSET		Auto Set
		0	The TXRDY bit must be set manually.
		1	Enables the TXRDY bit to be automatically set when data of the maximum packet size (value in USBTXMAXP[*n*]) is loaded into the transmit FIFO. If a packet of less than the maximum packet size is loaded, then the TXRDY bit must be set manually.
6	ISO		Isochronous Transfers
		0	Enables the transmit endpoint for bulk or interrupt transfers.
		1	Enables the transmit endpoint for isochronous transfers.
5	MODE		Mode **Note:** This bit only has an effect when the same endpoint FIFO is used for both transmit and receive transactions.
		0	Enables the endpoint direction as RX.
		1	Enables the endpoint direction as TX.
4	DMAEN		DMA Request Enable **Note:** Three TX and three /RX endpoints can be connected to the DMA module. If this bit is set for a particular endpoint, the DMAATX, DMABTX, or DMACTX field in the USB DMA Select (USBDMASEL) register must be programmed correspondingly.
		0	Disables the DMA request for the transmit endpoint.
		1	Enables the DMA request for the transmit endpoint.
3	FDT		Force Data Toggle
		0	No effect
		1	Forces the endpoint DT bit to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This bit can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.
2	DMAMOD		DMA Request Mode **Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared.
		0	An interrupt is generated after every DMA packet transfer.
		1	An interrupt is generated only after the entire DMA transfer is complete.
0	Reserved	0	Reserved

17.6.35 USB Maximum Receive Data Endpoint n Registers (USBRXMAXP[1]-USBRXMAXP[3])

The USB maximum receive data endpoint *n* 16-bit registers (USBRXMAXP[*n*]) define the maximum amount of data that can be transferred through the selected receive endpoint in a single operation.

Bits 10:0 define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the *USB Specification* on packet sizes for bulk, interrupt and isochronous transfers in full-speed operation.

The total amount of data represented by the value written to this register must not exceed the FIFO size for the transmit endpoint, and must not exceed half the FIFO size if double-buffering is required.

Note: USBRXMAXP[*n*] must be set to an even number of bytes for proper interrupt generation in DMA Basic Mode.

For the specific offset for each register see Table 17-4.

Mode(s): Host Device

The USBRXMAXP[*n*] registers are shown in Figure 17-45 and described in Table 17-47.

Figure 17-45. USB Maximum Receive Data Endpoint *n* Registers (USBRXMAXP[*n*])

15 — 11	10 — 0
Reserved	MAXLOAD
R-0	R/W-000

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

Table 17-47. USB Maximum Receive Data Endpoint *n* Registers (USBTXMAXP[*n*]) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	MAXLOAD		Maximum Payload specifies the maximum payload in bytes per transaction.

17.6.36 USB Receive Control and Status Endpoint n Low Register (USBRXCSRL[1]-USBRXCSRL[3])

The USB receive control and status endpoint *n* low 8-bit register (USBCSRL[*n*]) provides control and status bits for transfers through the currently selected receive endpoint.

For the specific offset for each register see Table 17-4.

Mode(s): Host Device

The USBCSRL[*n*] registers in Host mode are shown in Figure 17-46 and described in Table 17-48.

Figure 17-46. USB Receive Control and Status Endpoint *n* Low Register (USBCSRL[*n*]) in Host Mode

7	6	5	4	3	2	1	0
CLRDT	STALLED	REQPKT	FLUSH	DATAERR / NAKTO	ERROR	FULL	RXRDY
W1C-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 17-48. USB Control and Status Endpoint n Low Register(USBCSRL[*n*]) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	NAKTO		Clear Data Toggle.
		0	No effect
		1	Writing a 1 to this bit clears the DT bit in the USBRXCSRH[*n*] register.
6	STALLED		Endpoint Stalled. Software must clear this bit.
		0	No handshake has been received.
		1	A STALL handshake has been received. The EP*n* bit in the USBRXIS register is also set.
5	REQPKT		Request Packet. This bit is cleared when the RXRDY bit is set.
		0	No request
		1	Requests an IN transaction.
4	FLUSH		Flush FIFO. If the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO. **Note:**This bit should only be set when the RXRDY bit is set. At other times, it may cause data to be corrupted.
		0	No effect
		1	Flushes the next packet to be read from the endpoint receive FIFO. The FIFO pointer is reset and the RXRDY bit is cleared.
3	DATAERR / NAKTO		Data Error / NAK Timeout
		0	Normal operation
		1	Isochronous endpoints only: Indicates that RXRDY is set and the data packet has a CRC or bit-stuff error. This bit is cleared when RXRDY is cleared. Bulk endpoints only: Indicates that the receive endpoint is halted following the receipt of NAK responses for longer than the time set by the NAKLMT field in the USBRXINTERVAL[*n*] register. Software must clear this bit to allow the endpoint to continue.
2	ERROR		Error. Software must clear this bit. **Note:** This bit is only valid when the receive endpoint is operating in Bulk or Interrupt mode. In Isochronous mode, it always returns zero.
		0	No error
		1	Three attempts have been made to receive a packet and no data packet has been received. The EP*n* bit in the USBRXIS register is set in this situation.
1	FULL		FIFO Full
		0	The receive FIFO is not full.
		1	No more packets can be loaded into the receive FIFO.

Table 17-48. USB Control and Status Endpoint n Low Register(USBCSRL[*n*]) in Host Mode Field Descriptions (continued)

Bit	Field	Value	Description
0	RXRDY		Receive Packet Ready. If the AUTOCLR bit in the USBRXCSRH[*n*] register is set, then the this bit is automatically cleared when a packet of USBRXMAXP[*n*] bytes has been unloaded from the receive FIFO. If the AUTOCLR bit is clear, or if packets of less than the maximum packet size are unloaded, then software must clear this bit manually when the packet has been unloaded from the receive FIFO.
		0	No data packet has been received.
		1	Indicates that a data packet has been received. The EP*n* bit in the USBTXIS register is also set in this situation.

USBCSRL0 in Device mode is shown in Figure 17-47 and described in Table 17-49.

Figure 17-47. USB Control and Status Endpoint *n* Low Register (USBCSRL[*n*]) in Device Mode

7	6	5	4	3	2	1	0
CLRDT	STALLED	STALL	FLUSH	DATAERR	OVER	FULL	RXRDY
W1C-0	W1C-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 17-49. USB Control and Status Endpoint 0 Low Register(USBCSRL[*n*]) in Device Mode Field Descriptions

Bit	Field	Value	Description
7	CLRDT		Clear Data Toggle
		0	No effect
		1	Writing a 1 to this bit clears the DT bit in the USBRXCSRH[*n*] register.
6	STALLED		Endpoint Stalled. Software must clear this bit.
		0	A STALL handshake has been transmitted.
		1	A STALL handshake has been transmitted.
5	STALL		Send Stall. Software must clear this bit to terminate the STALL condition. **Note:** This bit has no effect where the endpoint is being used for isochronous transfers.
		0	No effect
		1	Issues a STALL handshake.
4	FLUSH		Flush FIFO. The CPU writes a 1 to this bit to flush the next packet to be read from the endpoint receive FIFO. The FIFO pointer is reset and the RXRDY bit is cleared. Note that if the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO.
		0	No effect
		1	Flushes the next packet from the endpoint receive FIFO. The FIFO pointer is reset and the RXRDY bit is cleared. **Note:** This bit should only be set when the RXRDY bit is set. At other times, it may cause data to be corrupted.
3	DATAEND		Data error. This bit is cleared when RXRDY is cleared. **Note:** This bit is only valid when the endpoint is operating in Isochronous mode. In Bulk mode, it always returns zero.
		0	Normal operation
		1	Indicates that RXRDY is set and the data packet has a CRC or bit-stuff error. This bit is cleared automatically.
2	OVER		Overrun. Software must clear this bit. **Note:** This bit is only valid when the endpoint is operating in Isochronous mode. In Bulk mode, it always returns zero.
		0	No overrun error
		1	Indicates an OUT packet cannot be loaded into the receive FIFO.

Table 17-49. USB Control and Status Endpoint 0 Low Register(USBCSRL[*n*]) in Device Mode Field Descriptions (continued)

Bit	Field	Value	Description
1	FULL		FIFO Full
		0	The receive FIFO is not full.
		1	No more packets can be loaded into the receive FIFO.
0	RXRDY		Receive Packet Ready. If the AUTOCLR bit in the USBRXCSRH[*n*] register is set, then the this bit is automatically cleared when a packet of USBRXMAXP[*n*] bytes has been unloaded from the receive FIFO. If the AUTOCLR bit is clear, or if packets of less than the maximum packet size are unloaded, then software must clear this bit manually when the packet has been unloaded from the receive FIFO.
		0	No data packet has been received.
		1	A data packet has been received. The EP*n* bit in the USBTXIS register is also set in this situation.
			This bit is cleared by writing a 1 to the RXRDYC bit.

17.6.37 USB Receive Control and Status Endpoint n High Register (USBRXCSRH[1]-USBRXCSRH[3])

The USB receive control and status endpoint *n* high 8-bit register (USBCSRL[*n*]) provides additional control and status bits for transfers through the currently selected receive endpoint.

For the specific offset for each register see Table 17-4.

Mode(s): Host Device

The USBCSRH[*n*] registers in OTG A/Host mode are shown in Figure 17-48 and described in Table 17-50.

Figure 17-48. USB Receive Control and Status Endpoint *n* High Register (USBCSRH[*n*]) in Host Mode

7	6	5	4	3	2	1	0
AUTOCL	AUTORQ	DMAEN	PIDERR	DMAMOD	DTWE	DT	Reserved
W1C-0	R/W-0	R/W-0	R-0	R/W-0	R-0	R-0	R-0

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

Table 17-50. USB Control and Status Endpoint n High Register (USBCSRH[*n*]) in Host Mode Field Descriptions

Bit	Field	Value	Description
7	AUTOCL		Auto Clear
		0	No effect
		1	Enables the RXRDY bit to be automatically cleared when a packet of USBRXMAXP[*n*] bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, RXRDY must be cleared manually. Care must be taken when using DMA to unload the receive FIFO as data is read from the receive FIFO in 4-byte chunks regardless of the value of the MAXLOAD field in the USBRXMAXP[*n*] register, see Section 17.3.3.
6	AUTORQ		Auto Request **Note:** This bit is automatically cleared when a short packet is received.
		0	No effect
		1	Enables the REQPKT bit to be automatically set when the RXRDY bit is cleared.
5	DMAEN		DMA Request Enable **Note:** Three TX and three RX endpoints can be connected to the DMA module. If this bit is set for a particular endpoint, the DMAARX, DMABRX, or DMACRX field in the USB DMA Select (USBDMASEL) register must be programmed correspondingly.
		0	Disables the DMA request for the receive endpoint.
		1	Enables the DMA request for the receive endpoint.
4	PIDERR		PID Error. This bit is ignored in bulk or interrupt transactions.
		0	No error
		1	Indicates a PID error in the received packet of an isochronous transaction.
3	DMAMOD		DMAMOD **Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared.
		0	An interrupt is generated after every DMA packet transfer.
		1	An interrupt is generated only after the entire DMA transfer is complete.
2	DTWE		Data Toggle Write Enable. This bit is automatically cleared once the new value is written.
		0	The DT bit cannot be written.
		1	Enables the current state of the receive endpoint data to be written (see DT bit).
1	DT		Data Toggle. When read, this bit indicates the current state of the receive data toggle. If DTWE is High, this bit may be written with the required setting of the data toggle. If DTWE is Low, any value written to this bit is ignored. Care should be taken when writing to this bit as it should only be changed to RESET the receive endpoint.
0	Reserved	0	Reserved

The USBCSRH[*n*] registers in Device mode are shown in Figure 17-49 and described in Table 17-51.

Figure 17-49. USB Control and Status Endpoint *n* High Register (USBCSRH[*n*]) in Device Mode

7	6	5	4	3	2		0
AUTOCL	ISO	DMAEN	DISNYET / PIDERR	DMAMOD		Reserved	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R-0	

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

Table 17-51. USB Control and Status Endpoint 0 High Register(USBCSRH[*n*]) in Device Mode Field Descriptions

Bit	Field	Value	Description
7	AUTOCL		Auto Clear
		0	No effect
		1	Enables the RXRDY bit to be automatically cleared when a packet of USBRXMAXP[*n*] bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, RXRDY must be cleared manually. Care must be taken when using DMA to unload the receive FIFO as data is read from the receive FIFO in 4-byte chunks regardless of the value of the MAXLOAD field in the USBRXMAXP[*n*] register, see Section 17.3.3.
6	ISO		Isochronous Transfers
		0	Enables the receive endpoint for isochronous transfers.
		1	Enables the receive endpoint for bulk/interrupt transfers.
5	DMAEN		DMA Request Enable **Note:** Three TX and three RX endpoints can be connected to the DMA module. If this bit is set for a particular endpoint, the DMAARX, DMABRX, or DMACRX field in the USB DMA Select (USBDMASEL) register must be programmed correspondingly.
		0	Disables the DMA request for the receive endpoint.
		1	Enables the DMA request for the receive endpoint.
4	DISNYET/PIDERR		Disable NYET / PID Error
		0	No effect
		1	For bulk or interrupt transactions: Disables the sending of NYET handshakes. When this bit is set, all successfully received packets are acknowledged, including at the point at which the FIFO becomes full.
			For isochronous transactions: Indicates a PID error in the received packet.
3	DMAMOD		DMA Request Mode **Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared.
		0	An interrupt is generated after every DMA packet transfer.
		1	An interrupt is generated only after the entire DMA transfer is complete.
0	Reserved	0	Reserved

17.6.38 USB Receive Byte Count Endpoint n Registers (USBRXCOUNT[1]-USBRXCOUNT[3])

The USB receive byte count endpoint *n* 16-bit read-only registers hold the number of data bytes in the packet currently in line to be read from the receive FIFO. If the packet is transmitted as multiple bulk packets, the number given is for the combined packet.

Note: The value returned changes as the FIFO is unloaded and is only valid while the RXRDY bit in the USBRXCSRLn register is set.

For the specific offset for each register see Table 17-4.

Mode(s): Host Device

The USBRXCOUNT[*n*] registers are shown in Figure 17-50 and described in Table 17-52.

Figure 17-50. USB Maximum Receive Data Endpoint *n* Registers (USBRXCOUNT[*n*])

15 - 13	12 - 0
Reserved	COUNT
R-0	R-0

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

Table 17-52. USB Maximum Receive Data Endpoint *n* Registers (USBRXCOUNT[*n*]) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved
12-0	COUNT		Receive Packet Count indicates the number of bytes in the receive packet.

17.6.39 USB Host Transmit Configure Type Endpoint n Register (USBTXTYPE[1]-USBTXTYPE[3])

The USB host transmit configure type endpoint *n* 8-bit registers (USBTXTYPE[*n*]) must be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently selected transmit endpoint, and its operating speed.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBTXTYPE[*n*] registers are shown in Figure 17-51 and described in Table 17-53.

Figure 17-51. USB Host Transmit Configure Type Endpoint n Register (USBTXTYPE[*n*])

7-6	5-4	3-0
SPEED	PROTO	TEP
R/W-0	R/W-0	R/W-0

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

Table 17-53. USB Host Transmit Configure Type Endpoint n Register(USBTXTYPE[*n*]) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED		Operating Speed. This bit field specifies the operating speed of the target Device:
		0h	Default. The target is assumed to be using the same connection speed as the USB controller.
		1h	Reserved
		2h	Full
		3h	Low
5-4	PROTO		Protocol. Software must configure this bit field to select the required protocol for the transmit endpoint:
		0h	Control
		1h	Isochronous
		2h	Bulk
		3h	Interrupt
3-0	TEP	0	Target Endpoint Number. Software must configure this value to the endpoint number contained in the transmit endpoint descriptor returned to the USB controller during Device enumeration.

17.6.40 USB Host Transmit Interval Endpoint n Register (USBTXINTERVAL[1]USBTXINTERVAL[3])

The USB host transmit interval endpoint *n* 8-bit registers (USBTXINTERVAL[*n*]), for interrupt and isochronous transfers, define the polling interval for the currently selected transmit endpoint. For bulk endpoints, this register defines the number of frames after which the endpoint should time out on receiving a stream of NAK responses.

The USBTXINTERVAL[*n*] registers values define a number of frames, as follows:

Table 17-54. USBTXINTERVAL[*n*] Frame Numbers

Transfer Type	Speed	Valid Values (m)	Interpretation
Interrupt	Low-speed or Full-speed	0x01-0xFF	The polling interval is *m* frames.
Isochronous	Full-speed	0x01-0x10	The polling interval is $2^{(m-1)}$ frames.
Bulk	Full-speed	0x02-0x10	The NAK Limit is $2^{(m-1)}$ frames. A value of 0 or 1 disables the NAK timeout function.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBTXINTERVAL[*n*] registers are shown in Figure 17-51 and described in Table 17-53.

Figure 17-52. USB Host Transmit Interval Endpoint n Register (USBTXINTERVAL[*n*])

7 … 0
TXPOLL / NAKLMT
R/W-0

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

Table 17-55. USB Host Transmit Interval Endpoint n Register(USBTXINTERVAL[*n*]) Field Descriptions

Bit	Field	Value	Description
7-0	TXPOLL / NAKLMT	0	TX Polling / NAK Limit The polling interval for interrupt/isochronous transfers; the NAK limit for bulk transfers. See Table 17-54 for valid entries; other values are reserved.

17.6.41 USB Host Configure Receive Type Endpoint n Register (USBRXTYPE[1]-USBRXTYPE[3])

The USB host configure receive type endpoint *n* 8-bit registers (USBRXTYPE[*n*]) must be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently selected receive endpoint, and its operating speed.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBRXTYPE[*n*] registers are shown in Figure 17-53 and described in Table 17-56.

Figure 17-53. USB Host Configure Receive Type Endpoint n Register (USBRXTYPE[*n*])

7-6	5-4	3-0
SPEED	PROTO	TEP
R/W-0	R/W-0	R/W-0

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

Table 17-56. USB Host Configure Receive Type Endpoint n Register(USBRXTYPE[*n*]) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED		Operating Speed. This bit field specifies the operating speed of the target Device:
		0h	Default. The target is assumed to be using the same connection speed as the USB controller.
		1h	Reserved
		2h	Full
		3h	Low
5-4	PROTO		Protocol. Software must configure this bit field to select the required protocol for the receive endpoint:
		0h	Control
		1h	Isochronous
		2h	Bulk
		3h	Interrupt
3-0	TEP	0	Target Endpoint Number. Software must configure this value to the endpoint number contained in the transmit endpoint descriptor returned to the USB controller during Device enumeration.

17.6.42 USB Host Receive Polling Interval Endpoint n Register (USBRXINTERVAL[1]-USBRXINTERVAL[3])

The USB host receive polling interval endpoint *n* 8-bit registers (USBRXINTERVAL[*n*]), for interrupt and isochronous transfers, define the polling interval for the currently selected transmit endpoint. For bulk endpoints, this register defines the number of frames after which the endpoint should time out on receiving a stream of NAK responses.

The USBRXINTERVAL[*n*] registers values define a number of frames, as follows:

Table 17-57. USBRXINTERVAL[*n*] Frame Numbers

Transfer Type	Speed	Valid Values (m)	Interpretation
Interrupt	Low-speed or Full-speed	0x01-0xFF	The polling interval is *m* frames.
Isochronous	Full-speed	0x01-0x10	The polling interval is $2^{(m-1)}$ frames.
Bulk	Full-speed	0x02-0x10	The NAK Limit is $2^{(m-1)}$ frames. A value of 0 or 1 disables the NAK timeout function.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBRXINTERVAL[*n*] registers are shown in Figure 17-51 and described in Table 17-53.

Figure 17-54. USB Host Receive Polling Interval Endpoint n Register (USBRXINTERVAL[*n*])

7 — 0
TXPOLL / NAKLMT
R/W-0

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

Table 17-58. USB Host Receive Polling Interval Endpoint n Register(USBRXINTERVAL[*n*]) Field Descriptions

Bit	Field	Value	Description
7-0	TXPOLL / NAKLMT	0	TX Polling / NAK Limit The polling interval for interrupt/isochronous transfers; the NAK limit for bulk transfers. See Table 17-57 for valid entries; other values are reserved.

17.6.43 USB Request Packet Count in Block Transfer Endpoint n Registers (USBRQPKTCOUNT[1]-USBRQPKTCOUNT[3])

The USB receive packet count in block transfer endpoint *n* 16-bit read/writer registers are used in Host mode to specify the number of packets that are to be transferred in a block transfer of one or more bulk packets to receive endpoint *n*. The USB controller uses the value recorded in this register to determine the number of requests to issue where the AUTORQ bit in the USBRXCSRH[*n*] register has been set. For more information about IN transactions as a host, see Section 17.3.2.2.

Note: Multiple packets combined into a single bulk packet within the FIFO count as one packet.

For the specific offset for each register see Table 17-4.

Mode(s): Host

The USBRQPKTCOUNT[*n*] registers are shown in Figure 17-55 and described in Table 17-59.

Figure 17-55. USB Request Packet Count in Block Transfer Endpoint *n* Registers (USBRQPKTCOUNT[*n*])

15 13	12 0
Reserved	COUNT
R-0	R-0

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

Table 17-59. USB Request Packet Count in Block Transfer Endpoint *n* Registers (USBRQPKTCOUNT[*n*]) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved
12-0	COUNT		Block Transfer Packet Count sets the number of packets of the size defined by the MAXLOAD bit field that are to be transferred in a block transfer. **Note:** This is only used in Host mode when AUTORQ is set. The bit has no effect in Device mode or when AUTORQ is not set.

17.6.44 USB Receive Double Packet Buffer Disable Register (USBRXDPKTBUFDIS), offset 0x340

The USB receive double packet buffer disable 16-bit register (USBTXIE) indicates which of the receive endpoints have disabled the double-packet buffer functionality (see *Double-Packet Buffering* in Section 17.3.1.1.1).

Mode(s): Host Device

USBRXDPKTBUFDIS is shown in Figure 17-56 and described in Table 17-60.

Figure 17-56. USB Receive Double Packet Buffer Disable Register (USBRXDPKTBUFDIS)

15–4	3	2	1	0
Reserved	EP3	EP2	EP1	Rsvd
R-0	R/W-1	R/W-1	R/W-1	R-0

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

Table 17-60. USB Receive Double Packet Buffer Disable Register (USBRXDPKTBUFDIS) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Reserved
3	EP3		EP3 RX Double-Packet Buffer Disable
		0	Disables double-packet buffering.
		1	Enables double-packet buffering.
2	EP2		EP2 RX Double-Packet Buffer Disable
		0	Disables double-packet buffering.
		1	Enables double-packet buffering.
1	EP1		EP1 RX Double-Packet Buffer Disable
		0	Disables double-packet buffering.
		1	Enables double-packet buffering.
0	Reserved	0	Reserved

17.6.45 USB Transmit Double Packet Buffer Disable Register (USBTXDPKTBUFDIS), offset 0x342

The USB transmit double packet buffer disable 16-bit register (USBTXIE) indicates which of the transmit endpoints have disabled the double-packet buffer functionality (see *Double-Packet Buffering* in Section 17.3.1.1.1).

Mode(s): Host Device

USBTXDPKTBUFDIS is shown in Figure 17-57 and described in Table 17-61.

Figure 17-57. USB Transmit Double Packet Buffer Disable Register (USBTXDPKTBUFDIS)

15–4	3	2	1	0
Reserved	EP3	EP2	EP1	Rsvd
R-0	R/W-1	R/W-1	R/W-1	R-0

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

Table 17-61. USB Transmit Double Packet Buffer Disable Register (USBTXDPKTBUFDIS) Field Descriptions

Bit	Field	Value	Description
15-4	Reserved		Reserved
3	EP3		EP3 RX Double-Packet Buffer Disable
		0	Disables double-packet buffering.
		1	Enables double-packet buffering.
2	EP2		EP2 RX Double-Packet Buffer Disable
		0	Disables double-packet buffering.
		1	Enables double-packet buffering.
1	EP1		EP1 RX Double-Packet Buffer Disable
		0	Disables double-packet buffering.
		1	Enables double-packet buffering.
0	Reserved	0	Reserved

17.6.46 USB External Power Control Register (USBEPC), offset 0x400

The USB external power control 32-bit register (USBEPC) specifies the function of the two-pin external power interface (USB0EPEN and USB0PFLT). The assertion of the power fault input may generate an automatic action, as controlled by the hardware configuration registers. The automatic action is necessary because the fault condition may require a response faster than one provided by firmware.

Mode(s): Host Device

USBEPC is shown in Figure 17-58 and described in Table 17-62.

Figure 17-58. USB External Power Control Register (USBEPC)

31–16
Reserved
R-0

15–10	9–8
Reserved	PFLTACT
R-0	R/W-0

7	6	5	4	3	2	1–0
Reserved	PFLTAEN	PFLTSEN	PFLTEN	Reserved	EPENDE	EPEN
R-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

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

Table 17-62. USB External Power Control Register (USBEPC) Field Descriptions

Bit	Field	Value	Description
31-10	Reserved	0	Reserved
9-8	PFLTACT		Power Fault Action. This bit field specifies how the USB0EPEN signal is changed when detecting a USB power fault.
		0h	Unchanged. USB0EPEN is controlled by the combination of the EPEN and EPENDE bits.
		1h	Tristate. USB0EPEN is undriven (tristate).
		2h	Low. USB0EPEN is driven Low.
		3h	High. USB0EPEN is driven High.
7	Reserved	0	Reserved
6	PFLTAEN		Power Fault Action Enable. This bit specifies whether a USB power fault triggers any automatic corrective action regarding the driven state of the USB0EPEN signal.
		0	Disabled. USB0EPEN is controlled by the combination of the EPEN and EPENDE bits.
		1	Enabled. The USB0EPEN output is automatically changed to the state specified by the PFLTACT field.
5	PFLTSEN		Power Fault Sense. This bit specifies the logical sense of the USB0PFLT input signal that indicates an error condition. The complementary state is the inactive state.
		0	Low Fault. If USB0PFLT is driven Low, the power fault is signaled internally (if enabled by the PFLTEN bit).
		1	High Fault. If USB0PFLT is driven High, the power fault is signaled internally (if enabled by the PFLTEN bit).
4	PFLTEN		Power Fault Input Enable. This bit specifies whether the USB0PFLT input signal is used in internal logic.
		0	Not Used. The USB0PFLT signal is ignored.
		1	Used. The USB0PFLT signal is used internally
3	Reserved	0	Reserved

Table 17-62. USB External Power Control Register (USBEPC) Field Descriptions (continued)

Bit	Field	Value	Description
2	EPENDE		EPEN Drive Enable. This bit specifies whether the USB0EPEN signal is driven or undriven (tristate). When driven, the signal value is specified by the EPEN field. When not driven, the EPEN field is ignored and the USB0EPEN signal is placed in a high-impedance state. The USB0EPEN signal is undriven at reset because the sense of the external power supply enable is unknown. By adding the high-impedance state, system designers can bias the power supply enable to the disabled state using a large resistor (100 kΩ) and later configure and drive the output signal to enable the power supply.
		0	Not Driven. The USB0EPEN signal is high impedance.
		1	Driven. The USB0EPEN signal is driven to the logical value specified by the value of the EPEN field.
1-0	EPEN		External Power Supply Enable Configuration. This bit field specifies and controls the logical value driven on the USB0EPEN signal.
		0h	Power Enable Active Low. The USB0EPEN signal is driven Low if the EPENDE bit is set.
		1h	Power Enable Active High. The USB0EPEN signal is driven High if the EPENDE bit is set.
		2h	Power Enable High if VBUS Low. The USB0EPEN signal is driven High when the A device is not recognized.
		3h	Power Enable High if VBUS High. The USB0EPEN signal is driven High when the A device is recognized.

17.6.47 USB External Power Control Raw Interrupt Status Register (USBEPCRIS), offset 0x404

The USB external power control raw interrupt status 32-bit register (USBEPCRIS) specifies the unmasked interrupt status of the two-pin external power interface.

Mode(s): Host Device

USBEPCRIS is shown in Figure 17-59 and described in Table 17-63.

Figure 17-59. USB External Power Control Raw Interrupt Status Register (USBEPCRIS)

31–1	0
Reserved	PF
R-0	R-0

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

Table 17-63. USB External Power Control Raw Interrupt Status Register (USBEPCRIS) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	PF		USB Power Fault Interrupt Status. This bit is cleared by writing a 1 to the PF bit in the USBEPCISC register.
		0	A Power Fault status has been detected.
		1	An interrupt has not occurred.

17.6.48 USB External Power Control Interrupt Mask Register (USBEPCIM), offset 0x408

The USB external power control interrupt mask 32-bit register (USBEPCIM) specifies the interrupt mask of the two-pin external power interface.

Mode(s): Host Device

USBEPCIM is shown in Figure 17-59 and described in Table 17-63.

Figure 17-60. USB External Power Control Interrupt Mask Register (USBEPCIM)

31 – 1	0
Reserved	PF
R-0	R-0

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

Table 17-64. USB External Power Control Interrupt Mask Register (USBEPCIM) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	PF		USB Power Fault Interrupt Mask.
		0	The raw interrupt signal from a detected power fault is sent to the interrupt controller.
		1	A detected power fault does not affect the interrupt status.

17.6.49 USB External Power Control Interrupt Status and Clear Register (USBEPCISC), offset 0x40C

The USB external power control interrupt status and clear 32-bit register (USBEPCISC) specifies the unmasked interrupt status of the two-pin external power interface.

Mode(s): Host Device

USBEPCISC is shown in Figure 17-61 and described in Table 17-65.

Figure 17-61. USB External Power Control Interrupt Status and Clear Register (USBEPCISC)

31 … 1	0
Reserved	PF
R-0	R-0

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

Table 17-65. USB External Power Control Interrupt Status and Clear Register (USBEPCISC) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved. Reset is 0x0000.000.
0	PF		USB Power Fault Interrupt Status and Clear. This bit is cleared by writing a 1. Clearing this bit also clears the PF bit in the USBEPCISC register.
		0	The PF bits in the USBEPCRIS and USBEPCIM registers are set, providing an interrupt to the interrupt controller.
		1	No interrupt has occurred or the interrupt is masked.

17.6.50 USB Device RESUME Raw Interrupt Status Register (USBDRRIS), offset 0x410

The USB device RESUME raw interrupt status register (USBDRRIS) is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt prior to masking. A write has no effect.

Mode(s): Host Device

USBDRRIS is shown in Figure 17-62 and described in Table 17-66.

Figure 17-62. USB Device RESUME Raw Interrupt Status Register (USBDRRIS)

31 ... 1	0
Reserved	RESUME
R-0	R-0

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

Table 17-66. USB Device RESUME Raw Interrupt Status Register (USBDRRIS) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved. Reset is 0x0000.000.
0	PF		RESUME Interrupt Status This bit is cleared by writing a 1 to the RESUME bit in the USBDRISC register.
		0	A RESUME status has been detected.
		1	An interrupt has not occurred.

17.6.51 USB Device RESUME Raw Interrupt Mask Register (USBDRIM), offset 0x414

The USB device RESUME raw interrupt status register (USBDRIM) is the masked interrupt status register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.

Mode(s): Host Device

USBDRIM is shown in Figure 17-63 and described in Table 17-67.

Figure 17-63. USB Device RESUME Raw Interrupt Status Register (USBDRRIS)

31 — 1	0
Reserved	RESUME
R-0	R-0

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

Table 17-67. USB Device RESUME Raw Interrupt Status Register (USBDRRIS) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved. Reset is 0x0000.000.
0	PF		RESUME Interrupt Mask
		0	The raw interrupt signal from a detected RESUME is sent to the interrupt controller. This bit should only be set when a SUSPEND has been detected (the SUSPEND bit in the USBIS register is set).
		1	A detected RESUME does not affect the interrupt status.

17.6.52 USB Device RESUME Interrupt Status and Clear Register (USBDRISC), offset 0x418

The USB device RESUME interrupt status and clear register (USBDRRIS) is the raw interrupt clear register. On a write of 1, the corresponding interrupt is cleared. A write of 0 has no effect.

Mode(s): Host Device

USBDRISC is shown in Figure 17-64 and described in Table 17-68.

Figure 17-64. USB Device RESUME Interrupt Status and Clear Register (USBDRISC)

31 – 1	0
Reserved	RESUME
R-0	R/W1C

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

Table 17-68. USB Device RESUME Interrupt Status and Clear Register (USBDRISC) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved. Reset is 0x0000.000.
0	RESUME		RESUME Interrupt Status and Clear. This bit is cleared by writing a 1. Clearing this bit also clears the RESUME bit in the USBDRCRIS register.
		0	The RESUME bits in the USBDRRIS and USBDRCIM registers are set, providing an interrupt to the interrupt controller.
		1	No interrupt has occurred or the interrupt is masked.

17.6.53 USB General-Purpose Control and Status Register (USBGPCS), offset 0x41C

The USB general-purpose control and status register (USBGPCS) provides the state of the internal ID signal.

When the USB controller is used as either a dedicated Host or Device, the DEVMODOTG and DEVMOD bits in the USB General-Purpose Control and Status (USBGPCS) register should be used to connect the ID inputs to fixed levels internally. For proper self-powered Device operation, the VBUS value must be monitored to assure that if the Host removes VBUS, the self-powered Device disables the D+/D- pull-up resistors. This function can be accomplished by connecting a standard GPIO to VBUS.

Mode(s): Host Device

USBGPCS is shown in Figure 17-65 and described in Table 17-69.

Figure 17-65. USB General-Purpose Control and Status Register (USBGPCS)

31 – 2	1	0
Reserved	DEVMODOTG	DEVMOD
R-0	R/W-0	R/W-0

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

Table 17-69. USB General-Purpose Control and Status Register (USBGPCS) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved. Reset is 0x0000.000.
1	DEVMODOTG		Enable Device Mode. This bit enables the DEVMOD bit to control the state of the internal ID signal in OTG mode.
		0	The mode is specified by the state of the internal ID signal.
		1	This bit enables the DEVMOD bit to control the internal ID signal.
0	DEVMOD		Device Mode This bit specifies the state of the internal ID signal in Host mode and in OTG mode when the DEVMODOTG bit is set. In Device mode this bit is ignored (assumed set).
		0	Host mode
		1	Device mode

17.6.54 USB DMA Select Register (USBDMASEL), offset 0x450

The USB DMA select 32-bit register (USBDMASEL) specifies whether the unmasked interrupt status of the ID value is valid.

Mode(s): Host Device

USBDMASEL is shown in Figure 17-66 and described in Table 17-70.

Figure 17-66. USB DMA Select Register (USBDMASEL)

31–24	23–20	19–16
Reserved	DMACTX	DMACRX
R/0	R/W-0	R/W-0

15–12	11–8	7–4	3–0
DMABTX	DMABRX	DMAATX	DMAARX
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 17-70. USB DMA Select Register (USBDMASEL) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved. Reset is 0x0000.000.
23-20	DMACTX		DMA C TX Select specifies the TX mapping of the third USB endpoint on DMA channel 5 (primary assignment).
		0h	Reserved
		1h	Endpoint 1 TX
		2h	Endpoint 2 TX
		3h	Endpoint 3 TX
19-16	DMACRX		DMA C RX Select specifies the RX and TX mapping of the third USB endpoint on DMA channel 4 (primary assignment).
		0h	Reserved
		1h	Endpoint 1 RX
		2h	Endpoint 2 RX
		3h	Endpoint 3 RX
15-12	DMABTX		DMA B TX Select specifies the TX mapping of the second USB endpoint on DMA channel 3 (primary assignment).
		0h	Reserved
		1h	Endpoint 1 TX
		2h	Endpoint 2 TX
		3h	Endpoint 3 TX
11-8	DMABRX		DMA B RX Select Specifies the RX mapping of the second USB endpoint on DMA channel 2 (primary assignment).
		0h	Reserved
		1h	Endpoint 1 RX
		2h	Endpoint 2 RX
		3h	Endpoint 3 RX
7-4	DMAATX		DMA A TX Select specifies the TX mapping of the first USB endpoint on DMA channel 1 (primary assignment).
		0h	Reserved
		1h	Endpoint 1 TX
		2h	Endpoint 2 TX
		3h	Endpoint 3 TX

Table 17-70. USB DMA Select Register (USBDMASEL) Field Descriptions (continued)

Bit	Field	Value	Description
3-0	DMAARX		DMA A RX Select specifies the RX mapping of the first USB endpoint on DMA channel 0 (primary assignment).
		0h	Reserved
		1h	Endpoint 1 RX
		2h	Endpoint 2 RX
		3h	Endpoint 3 RX

Revision History

Table A-1 lists the changes made since the previous version of this document.

Table A-1. Revisions

Chapter	Location	Additions/Modifications/Deletions
ADC	Figure 8-37	Changed 80ns min to 156ns min in two places. Removed the second line of the note following the graphic.
BootROM	Table 2-6	Deleted the note following the table and the corresponding reference to it in the table.
	Table 2-6	Underneath the table, inserted new list item and description - Jump to OTP Memory
CLA	MCCNDD and MRCNDD	Removed comma from code instances, MRCNDD, UNC and MRCNDD, NEQ
eCAP	Table 6-8	For bit 3, RE-ARM, reworded the description
SPI	Section 12.2.1	Removed the bottom row of text (Ch1, 200 V, Ch2, etc.) from each waveform graphic in this section.
	General	Removed references to 2802x
SysCtrl	Table 1-108	Changed INT13 - removed "External Interrupt 13".. and "for RTOS use" Changed INT14; removed "fir RTOS use"
	Figure 1-37	Revised this figure to correspond with the data manual CPU-watchdog Module figure
	Table 1-63	Changed the note from "The SCI-B and eQEP2 peripherals are not available on the 80-pin PN/PFP package" to "eQEP2 is not available on the 80-pin PN/PFP package."
	Section 1.2.3	Revised the paragraph and added instructions for use of the non-security feature
VCU	VMOVIX VRa, #16I	Description for VMOVIX says "Leave the upper 16-bits of the register unchanged." Changed to "Leave the lower 16-bits …".

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		