TMS320x2833x, TMS320x2823x

Technical Reference Manual

Literature Number: SPRUI07
March 2020

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.

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

Notational Conventions

This document uses the following conventions.

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

Glossary

TI Glossary —This glossary lists and explains terms, acronyms, and definitions.

Related Documentation From Texas Instruments

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

Documentation Feedback

Use the link at the bottom of the page to submit documentation feedback.

Support Resources

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

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

Export Control Notice

Recipient agrees to not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S., EU, and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from disclosing party under nondisclosure obligations (if any), or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S. or other applicable laws, without obtaining prior authorization from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws.

Trademarks

E2E, Code Composer Studio, 27x, C2xLP, TMS470, LF240xA are trademarks of Texas Instruments.

System Control and Interrupts

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

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

For more information on the Viterbi, Complex Math, CRC Unit (VCU), please refer to TMS320C28x Extended Instruction Sets Technical Reference Manual.

Topic **Page**

1.1 Flash and OTP Memory Blocks

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

1.1.1 Flash Memory

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

- **Multiple sectors**

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

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

  To save power when the flash is not in use, two levels of low power modes are available. See Section 1.1.3 for more information on the available flash power modes.
- **Configurable wait states**

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

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

1.1.2 OTP Memory

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

1.1.3 Flash and OTP Power Modes

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

- **Reset or Sleep State**

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

  In this state, the bank and pump are in standby power mode state. This state uses more power 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.

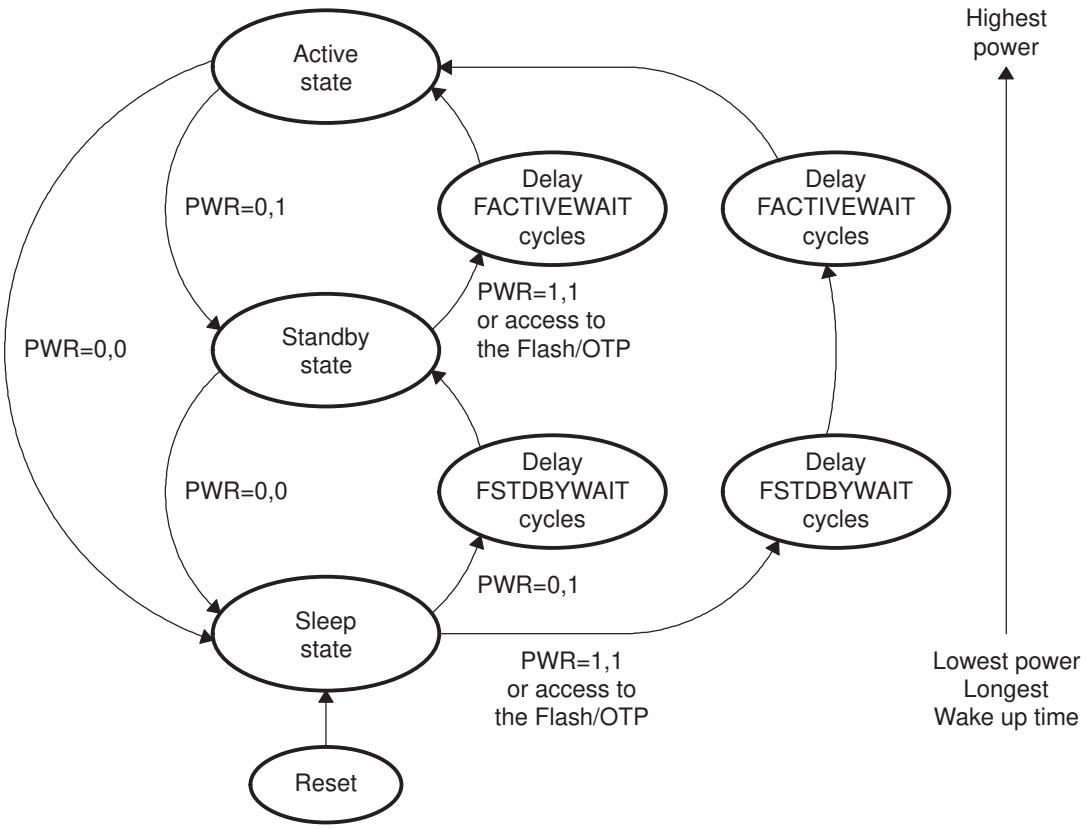
- **To move to a higher power state**

  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. F2833x/F2823x devices require a minimum of 1 wait-state for PAGE/RANDOM flash access. 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.

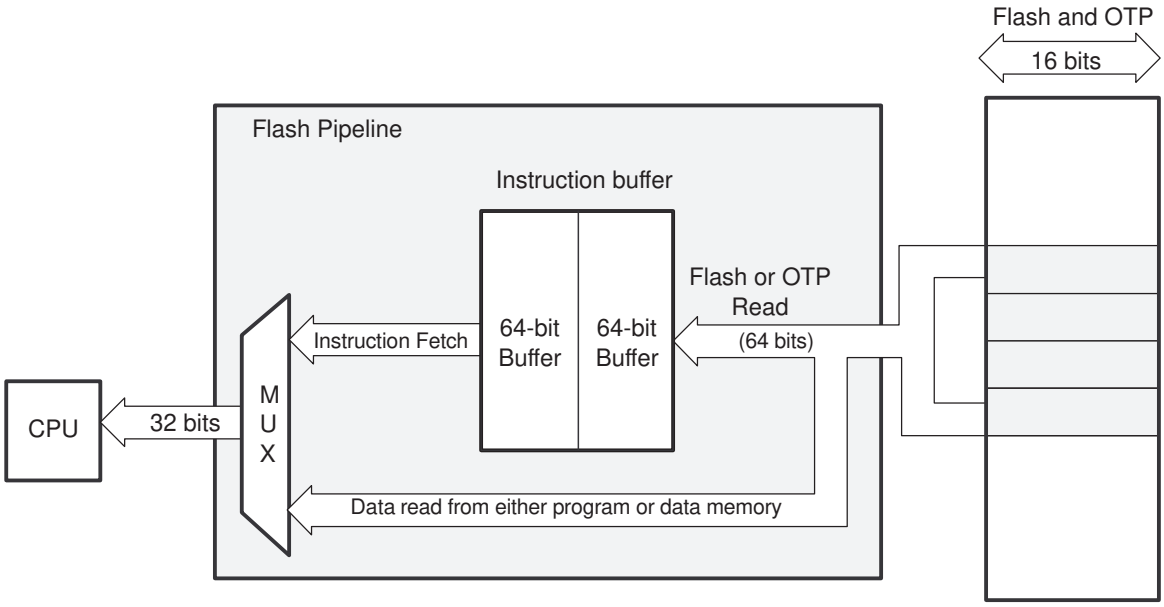
1.1.3.2 Flash Pipeline Mode

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 0x33 FFF6 and 0x33 FFF7 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 0x33 FFF6. If you program a branch instruction here that will then re-direct code execution to the entry point of the application.
2. For code security operation, all addresses between 0x33 FF80 and 0x33 FFF5 cannot be used for

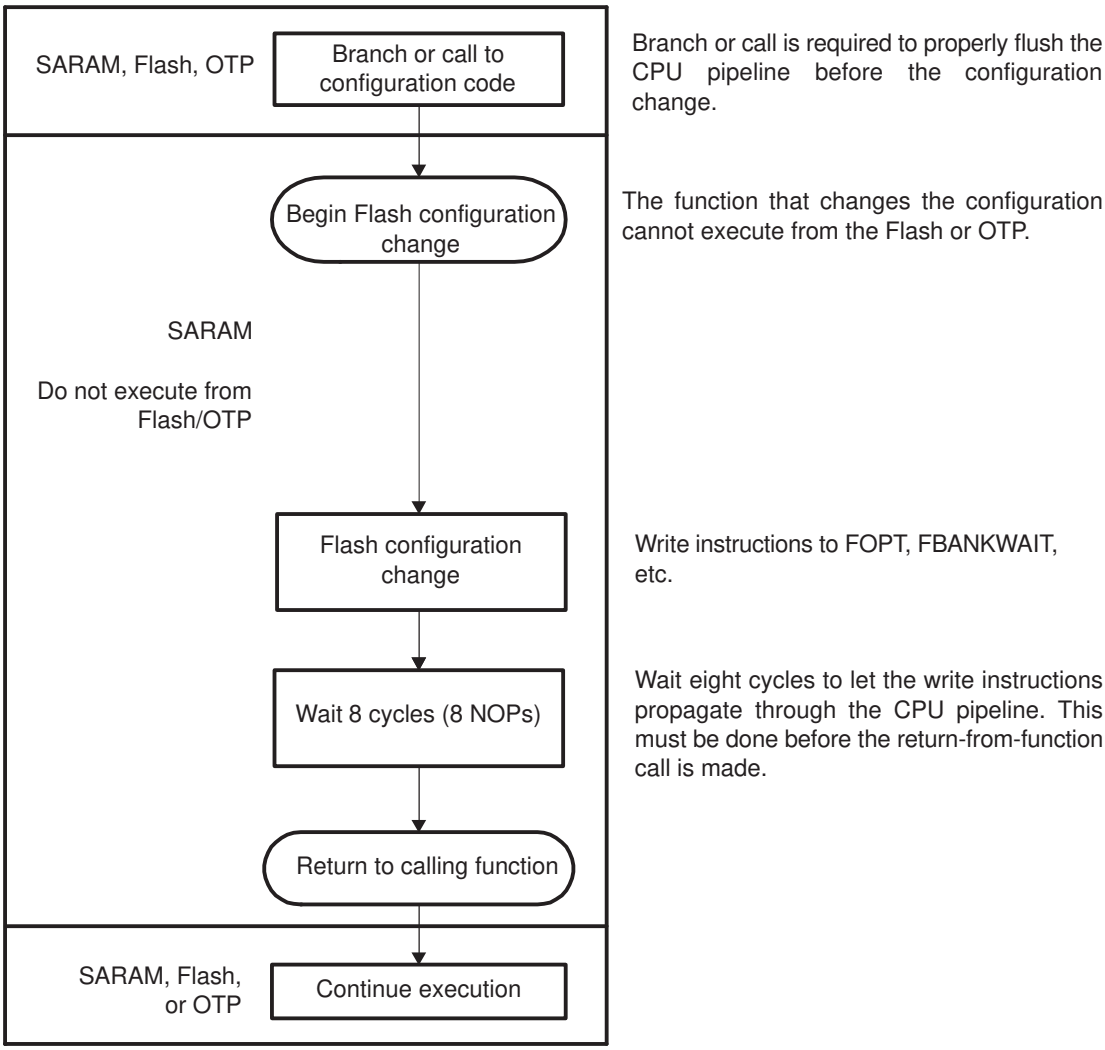
program code or data, but must be programmed to 0x0000 when the Code Security Password is programmed. If security is not a concern, addresses 0x33 FF80 through 0x33 FFF5 may be used for code or data. See Section 1.2 for information in using the Code Security Module.

3. Addresses from 0x33 FFF0 to 0x33 FFF5 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



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		
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		
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		Reserved
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		Reserved
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 Modes Status Bits. These bits indicate which power mode the flash/OTP is currently in.
			The PWRS bits are set to the new power mode only after the appropriate timing delays have expired.
		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	0	Reserved
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	0	Reserved
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		Reserved
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	Illegal value. PAGEWAIT must be set greater then 0.
		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		Reserved
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	0	Reserved
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 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 (0x33 FFF8 - 0x33 FFFF). 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, L1, L2 or L3 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: Reserved Flash Locations When Using Code Security

For code security operation, **all addresses between 0x33 FF80 and 0x33 FFF5 cannot be used as program code or data, but must be programmed to 0x0000** when the Code Security Password is programmed. If security is not a concern, addresses 0x33 FF80 through 0x33 FFF5 may be used for code or data. The 128-bit password (at 0x33 FFF8 - 0x33 FFFF) must not be programmed to zeros. Doing so would permanently lock the device.

Addresses 0x33 FFF0 through 0x33 FFF5 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 8FFF	L0 SARAM (4K X 16)
0x00 9000 - 0x00 9FFF	L1 SARAM (4K X 16)
0x00 A000 - 0x00 AFFF	L2 SARAM (4K X 16)
0x00 B000 - 0x00 BFFF	L3 SARAM (4K X 16)
0x30 0000 - 0x33 FFFF	Flash (64K X 16, 32 X 16, or 16 X 16)
0x38 0000 - 0x38 03FF	TI One-Time Programmable (OTP)(1) (1K X 16)
0x38 0400 - 0x38 07FF	User One-Time Programmable (OTP) (1K X 16)
0x3F 8000 - 0x3F 8FFF	L0 SARAM (4K X 16), mirror
0x3F 9000 - 0x3F 9FFF	L1 SARAM (4K X 16), mirror
0x3F A000 - 0x3F AFFF	L2 SARAM (4K X 16), mirror
0x3F B000 - 0x3F BFFF	L3 SARAM (4K X 16), mirror

(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 - 0x00 CFFF	L4 SARAM (4K x 16)
0x00 D000 - 0x00 DFFF	L5 SARAM (4K x 16)
0x00 E000 - 0x00 EFFF	L6 SARAM (4K x 16)
0x00 F000 - 0x00 FFFF	L7 SARAM (4K x 16)
0x3F F000 - 0x3F FFFF	Boot ROM (4K x 16)

To summarize, it is possible to load code onto the unprotected on-chip program SARAM shown in Table 1-11 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 required in the development phase of a project; however, security is needed once a robust code is developed. 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 (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 memor, 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 a typical 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)]	0x005F	CSM status and control register
Password Locations (PWL) in Flash Memory - Reserved for the CSM password only			
0x33 - FFF8	PWL0	User defined	Low word of the 128-bit password
0x33 - FFF9	PWL1	User defined	Second word of the 128-bit password
0x33 - FFFA	PWL2	User defined	Third word of the 128-bit password
0x33 - FFFB	PWL3	User defined	Fourth word of the 128-bit password
0x33 - FFFC	PWL4	User defined	Fifth word of the 128-bit password
0x33 - FFFD	PWL5	User defined	Sixth word of the 128-bit password
0x33 - FFFE	PWL6	User defined	Seventh word of the 128-bit password
0x33 - FFFF	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 7	6 1	0
FORCESEC	Reserved	Reserved	SECURE
R/W-1	R-0	R-10111	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.

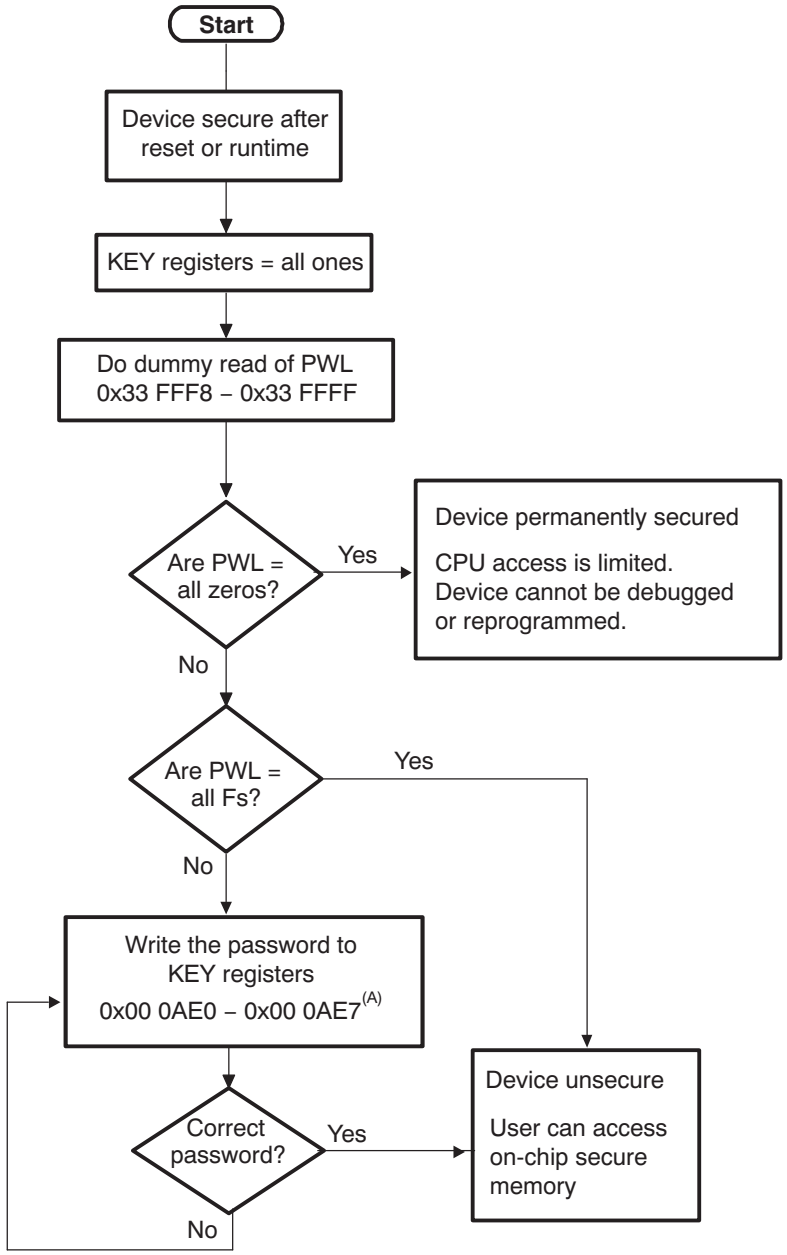
The unsecuring sequence is identical in all the above situations. This sequence is referred to as the *password match flow (PMF)* for simplicity. Figure 1-12 explains the sequence of operation that is required every time the user attempts to unsecure a device. A code example is listed for clarity.

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)



A The KEY registers are EALLOW protected.

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 (0x33 FFF8 - 0x33 FFFF in memory). In addition, locations 0x33 FF80 - 0x33 FFF5 should be programmed with all 0x0000 and not used for program and/or data storage. The following are steps to unsecure this device:

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

Case 2: Device Without Code Security

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

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

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

1.2.3.3.1 C Code Example to Unsecure

```
volatile int *CSM = (volatile int *)0x000AE0; //CSM register file
volatile int *PWL = (volatile int *)0x0033FFF8; //Password location
volatile int tmp;

int I;

   // Read the 128-bits of the password locations (PWL)
   // in flash at address 0x33 FFF8 - 0x33 FFFF
   // 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 0x33 FF80 - 0x33 FFF5 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 0x33 FF80 - 0x33 FFF5 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 (iCode 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 and System Control

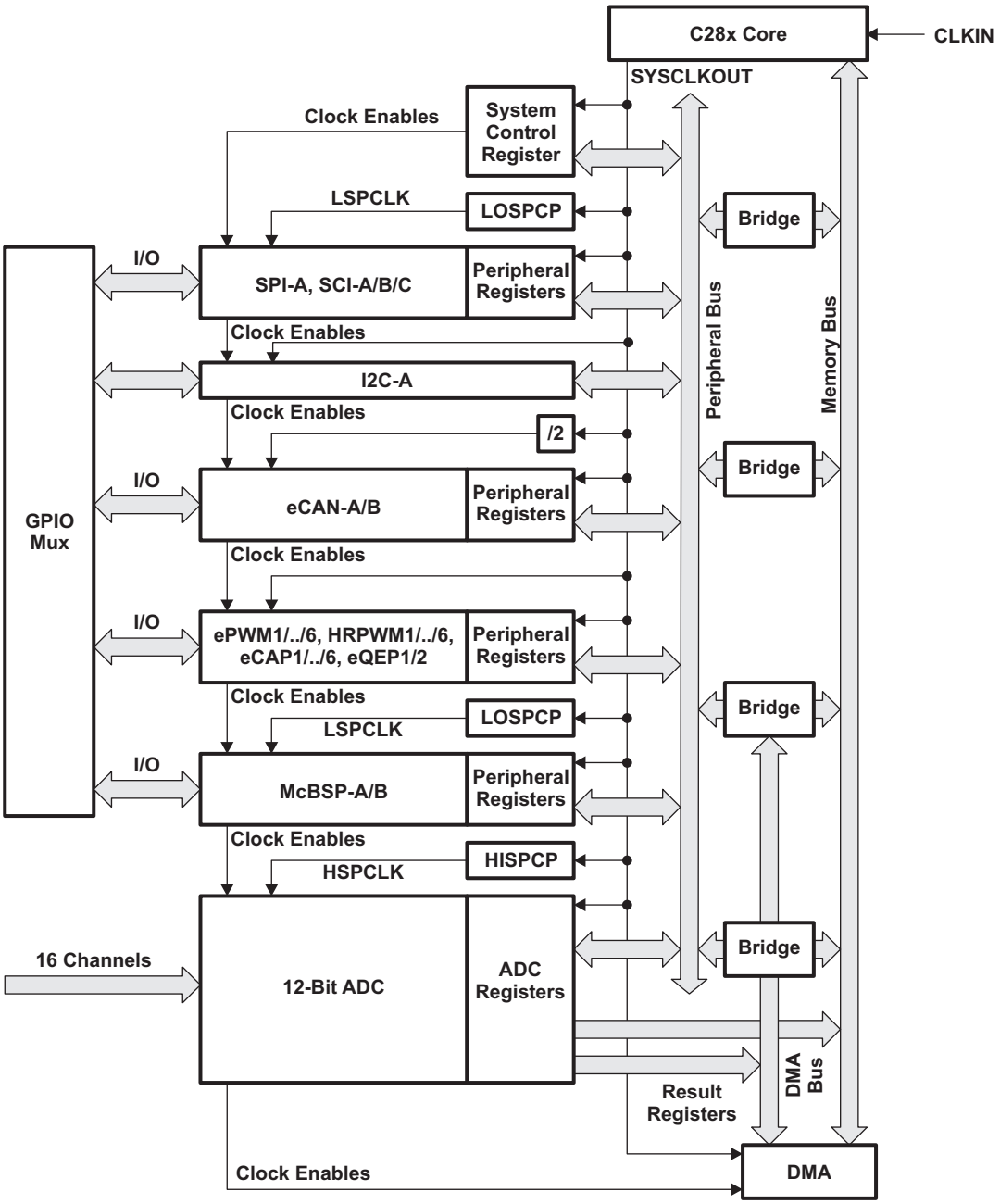
This section describes the oscillator, PLL and clocking mechanisms, the watchdog function, and the low-power modes.

1.3.1 Clocking

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

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

Figure 1-13. Clock and Reset Domains



A CLKIN is the clock into the CPU. It is passed out of the CPU as SYSCLKOUT (that is, CLKIN is the same frequency as SYSCLKOUT).

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

Name	Address	Size (x16)	Description[1]	Bit Description
PLLSTS [2]	0x7011	1	PLL Status Register	Figure 1-24
HISPCP	0x701A	1	High-Speed Peripheral Clock (HSPCLK) Prescaler Register	Figure 1-17
LOSPCP	0x701B	1	Low-Speed Peripheral Clock (LSPCLK) Prescaler Register	Figure 1-18
PCLKCR0	0x701C	1	Peripheral Clock Control Register 0	Figure 1-14
PCLKCR1	0x701D	1	Peripheral Clock Control Register 1	Figure 1-15
LPMCR0	0x701E	1	Low Power Mode Control Register 0	Figure 1-18
PCLKCR3	0x7020	1	Peripheral Clock Control Register 3	Figure 1-16
PLLCR [2]	0x7021	1	PLL Control Register	Figure 1-23
SCSR	0x7022	1	System Control & Status Register	Figure 1-27
WDCNTR	0x7023	1	Watchdog Counter Register.	Figure 1-28
WDKEY	0x7025	1	Watchdog Reset Key Register	Figure 1-29
WDCR	0x7029	1	Watchdog Control Register	Figure 1-30

(1) All of the registers in this table are EALLOW protected. See Section 1.5.2 for more information.
(2) The PLL control register (PLLCR) and PLL Status Register (PLLSTS) are reset to a known 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.

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 MUXing, 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 is not 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
ECANBENCLK	ECANAENCLK	MCBSPBENCLK	MCBSPAENCLK	SCIBENCLK	SCIAENCLK	Reserved	SPIAENCLK
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0
7	6	5	4	3	2	1	0
Reserved		SCICENCLK	I2CAENCLK	ADCENCLK	TBCLKSYNC	Reserved	
R-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 1-15. Peripheral Clock Control 0 Register (PCLKCR0) Field Descriptions

Bit	Field	Value	Description [1]
15	ECANBENCLK		ECAN-B Clock enable
		0	The eCAN-B module is not clocked. (default) [2]
		1	The eCAN-B module is clocked (SYSCLKOUT/2).
14	ECANAENCLK		ECAN-A clock enable
		0	The eCAN-A module is not clocked. (default) [2]
		1	The eCAN-A module is clocked (SYSCLKOUT/2).
13	MCBSPBENCLK		McBSP-B Clock Enable. This bit is reserved on devices without the McBSP-B module.[3]
		0	The McBSP-B module is not clocked. (default)
		1	The McBSP-B module is clocked by the low-speed clock (LSPCLK).

(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) On devices without a particular peripheral, the clock selection bit is reserved. On these devices, the bit should not be written to with a 1.

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

Bit	Field	Value	Description [1]
12	MCBSPAENCLK		McBSP-A Clock Enable
		0	The McBSP-A module is not clocked. (default)
		1	The McBSP-A module is clocked by the low-speed clock (LSPCLK).
11	SCIBENCLK		SCI-B clock enable
		0	SCI-B module is not clocked. (default) [2]
		1	The SCI-B module is clocked by the low-speed clock (LSPCLK).
10	SCIAENCLK		SCI-A clock enable
		0	The SCI-A module is not clocked. (default) [2]
		1	The SCI-A module is clocked by the low-speed clock (LSPCLK).
9	Reserved	0	Reserved
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).
7:6	Reserved	0	Reserved
5	SCICENCLK		SCI-C clock enable. This bit is reserved on devices without the SCI-C module.[3]
		0	The SCI-C module is not clocked. (default)
		1	The SCI-C module is clocked by the low-speed clock (LSPCLK).
4	I2CAENCLK		I2C clock enable
		0	The I2C module is not clocked. (default) [2]
		1	The I2C module is clocked by SYSCLKOUT.
3	ADCENCLK		ADC clock enable
		0	The ADC is not clocked. (default) [2]
		1	The ADC module is clocked by the high-speed clock (HSPCLK)
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.
1-0	Reserved		Reserved

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

15	14	13	12	11	10	9	8
EQEP2ENCLK	EQEP1ENCLK	ECAP6ENCLK	ECAP5ENCLK	ECAP4ENCLK	ECAP3ENCLK	ECAP2ENCLK	ECAP1ENCLK
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
Reserved		EPWM6ENCLK	EPWM5ENCLK	EPWM4ENCLK	EPWM3ENCLK	EPWM2ENCLK	EPWM1ENCLK
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 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	ECAP6ENCLK		eCAP6 clock enable. This bit is reserved on devices without the eCAP6 module.
		0	The eCAP6 module is not clocked. (default)
		1	The eCAP6 module is clocked by the system clock (SYSCLKOUT).
12	ECAP5ENCLK		eCAP5 clock enable. This bit is reserved on devices without the eCAP5 module.
		0	The eCAP5 module is not clocked. (default)
		1	The eCAP5 module is clocked by the system clock (SYSCLKOUT).
11	ECAP4ENCLK		eCAP4 clock enable
		0	The eCAP4 module is not clocked. (default) [2]
		1	The eCAP4 module is clocked by the system clock (SYSCLKOUT).
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).
7:6	Reserved	0	Reserved
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).

(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]
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	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 3 Register (PCLKCR3)

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

7	0
Reserved	
R-0	

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

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

Bit	Field	Value	Description
15:14	Reserved		Reserved
13	GPIOINENCLK		GPIO Input Clock Enable
		0	The GPIO module is not clocked.
		1	The GPIO module is clocked.
12	XINTFENCLK		External Interface (XINTF) Clock Enable
		0	The external memory interface is not clocked.
		1	The external memory interface is clocked.
11	DMAENCLK		DMA Clock Enable
		0	The DMA module is not clocked.
		1	The DMA module 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.
7:0	Reserved		Reserved

The high-speed peripheral and low-speed peripheral clock prescale (HISPCP and LOSPCP) registers are used to configure the high-speed and low-speed peripheral clocks, respectively. See Figure 1-17 for the HISPCP bit layout and Figure 1-18 for the LOSPCP layout.

Figure 1-17. High-Speed Peripheral Clock Prescaler (HISPCP) Register

15	3	2	0
Reserved		HSPCLK	
R-0		R/W-001	

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

Table 1-18. High-Speed Peripheral Clock Prescaler (HISPCP) Field Descriptions

Bits	Field	Value	Description [1]
15-3	Reserved		Reserved
2-0	HSPCLK		These bits configure the high-speed peripheral clock (HSPCLK) rate relative to SYSCLKOUT: If HISPCP[2] ≠ 0, HSPCLK = SYSCLKOUT/(HISPCP X 2) If HISPCP = 0, HSPCLK = SYSCLKOUT
		000	High speed clock = SYSCLKOUT/1
		001	High speed clock = SYSCLKOUT/2 (reset default)
		010	High speed clock = SYSCLKOUT/4
		011	High speed clock = SYSCLKOUT/6
		100	High speed clock = SYSCLKOUT/8
		101	High speed clock = SYSCLKOUT/10
		110	High speed clock = SYSCLKOUT/12
		111	High speed clock = SYSCLKOUT/14

(1) This register is EALLOW protected. See Section 1.5.2 for more information.
(2) HISPCP in this equation denotes the value of bits 2:0 in the HISPCP register.

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[2]		These bits configure the low-speed peripheral clock (LSPCLK) rate relative to SYSCLKOUT: If LOSPCP[3] ≠ 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) Check the device datasheet for the valid range of values of this register.
(3) LOSPCP in this equation denotes the value of bits 2:0 in the LOSPCP register.

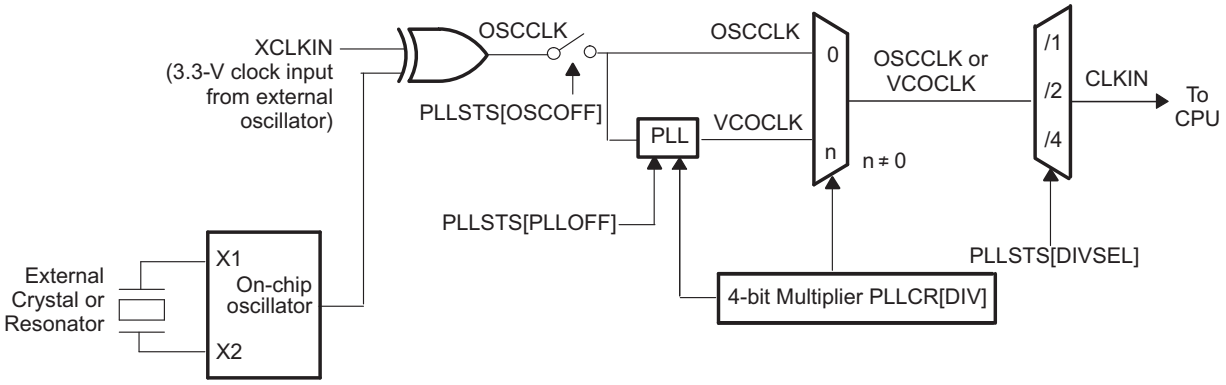
1.3.2 OSC and PLL Block

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

1.3.2.1 PLL-Based Clock Module

The 2833x, 2823x devices have an on-chip, PLL-based clock module. The PLL has a 4-bit ratio control to select different CPU clock rates. Figure 1-19 shows the OSC and PLL block.

Figure 1-19. OSC and PLL Block



The PLL-based clock module provides two modes of operation:

- **Crystal/Resonator Operation:**

  The on-chip oscillator enables the use of an external crystal/resonator to be attached to the device to provide the time base to the device. The crystal/resonator is connected to the X1/X2 pins and XCLKIN is tied low.

- **External clock source operation:**

  If the on-chip oscillator is not used, this mode allows the internal oscillator to be bypassed. The device clocks are generated from an external clock source input on either the X1 or the XCLKIN pin.

  Option 1: External clock on the XCLKIN pin:

  When using XCLKIN as the external clock source, you must tie X1 low and leave X2 disconnected. In this case, an external oscillator clock is connected to the XCLKIN pin, which allows for a 3.3-V clock source to be used.

  Option 2: External clock on the X1 pin:

  When using X1 as the clock source, you must tie XCLKIN low and leave X2 disconnected. In this case, an external oscillator clock is connected to the X1 pin, which allows for a 1.8-V clock source to be used.

The OSC circuit enables attachment of a crystal using the X1 and X2 pins. If a crystal is not used, then an external oscillator can be directly connected to the XCLKIN pin, the X2 pin is left unconnected, and the X1 pin is tied low. See the *TMS320F28335, TMS320F28334, TMS28332, TMS320F28235, TMS320F28234, TMS28232 Digital Signal Controllers (DSCs) Data Manual* (literature number SPRS439). The input clock and PLLCR[DIV] bits should be chosen in such a way that the output frequency of the PLL (VCOCLK) does not exceed 300 MHz .

Table 1-20. Possible PLL Configuration Modes

PLL Mode	Remarks	PLLSTS[DIVSEL](1)	SYSCLKOUT
PLL Off	Invoked by the user setting the PLLOFF bit in the PLLSTS register. The PLL block is disabled in this mode. 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 ($\overline{XRS}$). 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	OSCCLK*n/4 OSCCLK*n/2

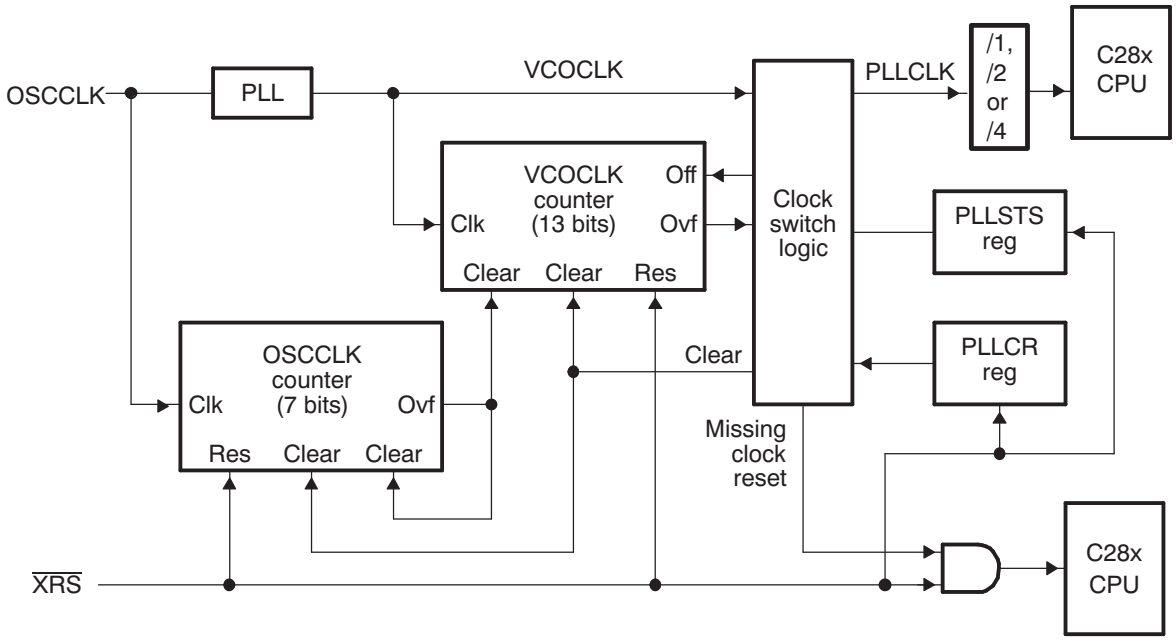
(1) PLLSTS[DIVSEL] must be 0 before writing to the PLLCR and should be changed only after PLLSTS[PLLLOCKS] = 1. See Figure 1-22.

1.3.2.2 Main Oscillator Fail Detection

Due to vibrations, it is possible for the external clock source to the DSP to become detached and fail to clock the device. 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 1-20. The first counter is incremented by the OSCCLK signal itself either from the X1/X2 or XCLKIN input. 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 1-20. Oscillator Fail-Detection 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 ($\overline{\text{MCLKRES}}$). The $\overline{\text{MCLKRES}}$ is an internal reset only. The external $\overline{\text{XRS}}$ pin of the device is not pulled low by $\overline{\text{MCLKRES}}$ and the PLLCR and PLLSTS registers are not 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 reset the part and that the CPU is now running either at or one-half of the limp mode frequency.

Software should check the PLLSTS[MCLKSTS] bit after a reset to determine if the device was reset by $\overline{\text{MCLKRES}}$ 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 and the V_{DD3VFL} rail.

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 1-22 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 1-22.
- **The watchdog is not functional without an external clock.**

  The watchdog is not functional and cannot generate a reset when OSCCLK is not present. No special hardware has been added to switch the watchdog to the limp mode clock should OSCCLK become missing.
- **Limp mode may not work from power up.**

  The PLL may not generate a limp mode clock if OSCCLK is missing from power-up. Only if OSCCLK is initially present will a limp mode clock be generated by the PLL.
- **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.

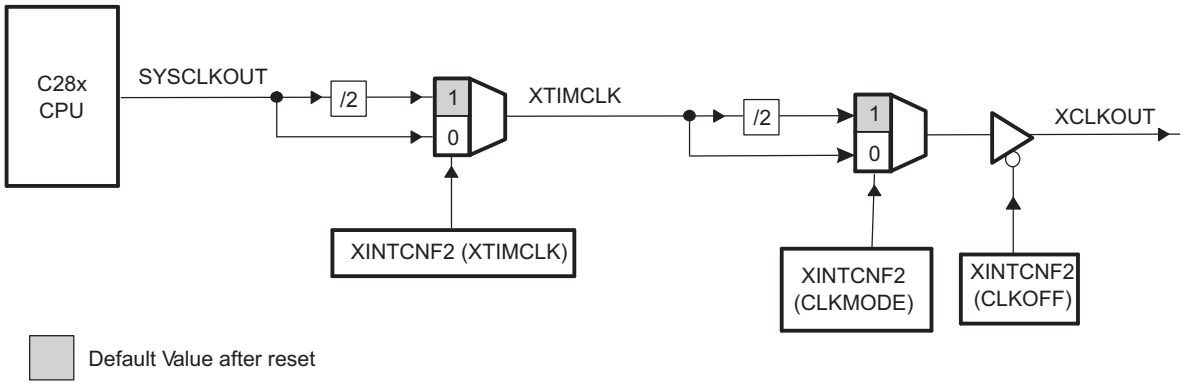
- **HALT low power mode**

  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.

1.3.2.3 XCLKOUT Generation

The XCLKOUT signal is directly derived from the system clock SYSCLKOUT as shown in Figure 1-21. 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 . Note that the boot ROM changes SYSCLKOUT from OSCCLK/4 to OSCCLK/2. Therefore, if the boot ROM has been executed after a reset, XCLKOUT will be OSCCLK/8.

Figure 1-21. XCLKOUT Generation



The XCLKOUT signal is active when reset is active. Since XCLKOUT should reflect SYSCLKOUT/4 when reset is low, you can monitor this signal to detect if the device is being properly clocked during debug. There is no internal pullup or pulldown on the XCLKOUT pin.

If XCLKOUT is not being used, it can be turned off by setting the CLKOFF bit to 1 in the XINTCNF2 register.

1.3.2.4 PLL Control (PLLCR) Register

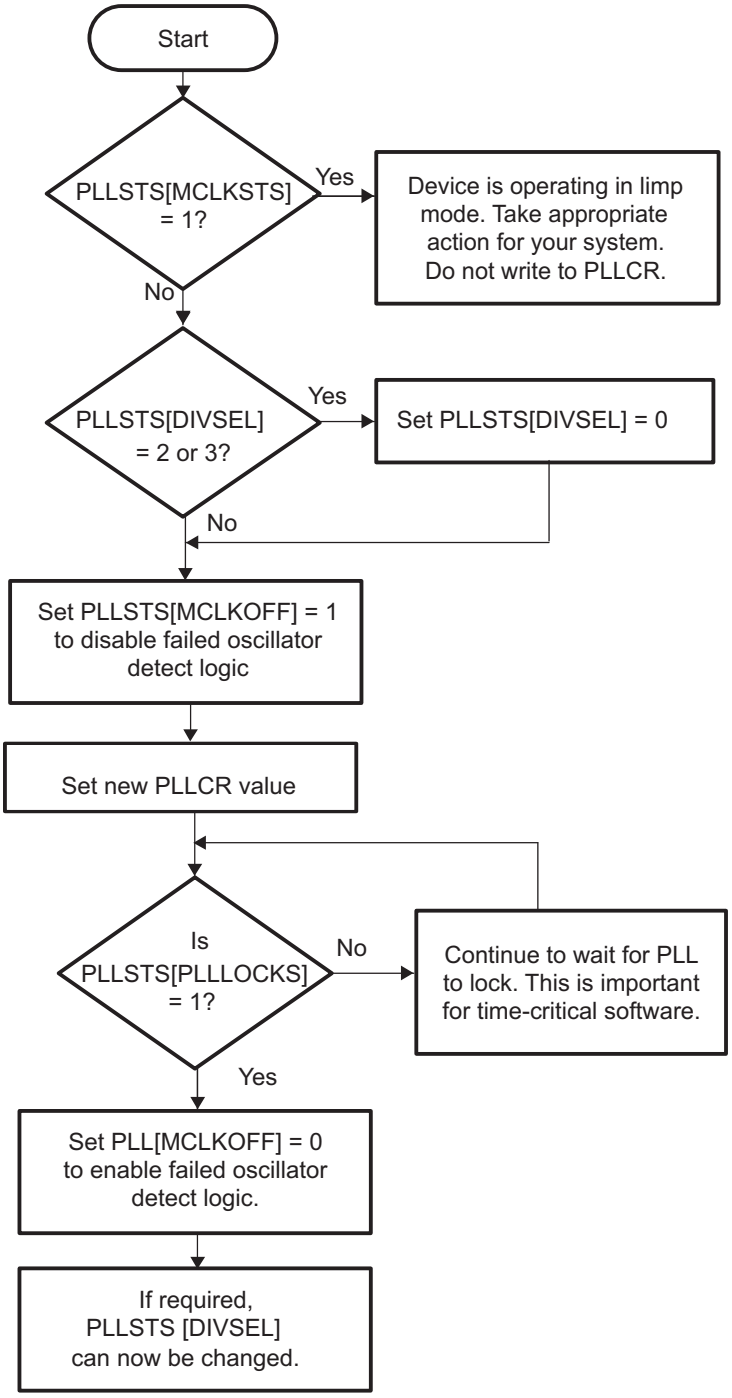
The PLLCR register is used to change the PLL multiplier of the device. Before writing to the PLLCR register, the following requirements must be met:

- 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.
- The device must not be operating in "limp mode". That is, the PLLSTS[MCLKSTS] bit must be 0.

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 1-21. 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 1-22 any time you are writing to the PLLCR register.

Figure 1-22. PLLCR Change Procedure Flow Chart



1.3.2.5 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 01, 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 1-22.

Figure 1-23. PLLCR Register Layout

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

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

Table 1-21. PLLCR Bit Descriptions[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
0001	(OSCCLK * 1)/4	(OSCCLK*1)/2	–
0010	(OSCCLK * 2)/4	(OSCCLK*2)/2	–
0011	(OSCCLK * 3)/4	(OSCCLK*3)/2	–
0100	(OSCCLK * 4)/4	(OSCCLK*4)/2	–
0101	(OSCCLK * 5)/4	(OSCCLK*5)/2	–
0110	(OSCCLK * 6)/4	(OSCCLK*6)/2	–
0111	(OSCCLK * 7)/4	(OSCCLK*7)/2	–
1000	(OSCCLK * 8)/4	(OSCCLK*8)/2	–
1001	(OSCCLK * 9)/4	(OSCCLK*9)/2	–
1010	(OSCCLK * 10)/4	(OSCCLK*10)/2	–
1011 - 1111	Reserved	Reserved	Reserved

(1) This register is EALLOW protected. See Section 1.5.2 for more information.
(2) PLLSTS[DIVSEL] must be 0 before writing to the PLLCR and should be changed only after PLLSTS[PLLLOCKS] = 1. See Figure 1-22.
(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 1-24. PLL Status Register (PLLSTS)

15						9	8
Reserved							DIVSEL
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 1-22. PLL Status Register (PLLSTS) Field Descriptions

Bits	Field	Value	Description (1) (2)
15-9	Reserved		Reserved
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. (This mode can be used only when PLL is off or bypassed.)
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, X1/X2 or XCLKIN is fed to the PLL block. (default)
		1	The OSCCLK signal from X1, 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 or 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 powered 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.
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		Reserved
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.

(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.

1.3.2.6 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 1-23 summarizes the various modes.

The various low-power modes operate as shown in Table 1-24.

See the *TMS320F28335, TMS320F28334, TMS320F28332, TMS320F28235, TMS320F28234, TMS320F28232 Digital Signal Controllers (DSCs) Data Manual* (literature number SPRS439) for exact timing for entering and exiting the low power modes.

Table 1-23. Low-Power Mode Summary

Mode	LPMCR0[1:0]	OSCCLK	CLKIN	SYSCLKOUT	Exit[1]
IDLE	00	On	On	On[2]	$\overline{\text{XRS}}$, Watchdog interrupt, Any enabled interrupt
STANDBY	01	On (watchdog still running)	Off	Off	$\overline{\text{XRS}}$, Watchdog interrupt, GPIO Port A signal, Debugger[3]
HALT	1X	Off (oscillator and PLL turned off, watchdog not functional)	Off	Off	$\overline{\text{XRS}}$, GPIO Port A Signal, Debugger[3]

(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) The IDLE mode on the 28x behaves differently than on the 24x/240x. On the 28x, the clock output from the CPU (SYSCLKOUT) is still functional while on the 24x/240x the clock is turned off.

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

Table 1-24. Low Power Modes

Mode	Description
IDLE Mode:	This mode is exited by any enabled interrupt or an NMI. 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.
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 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. • 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. • If the above conditions are met, – WAKE_INT ISR will be executed first, followed by the instruction(s) after IDLE, if INTM = 0. – WAKE_INT ISR will not be executed and instruction(s) after IDLE will be executed, if INTM = 1.

Table 1-24. Low Power Modes (continued)

Mode	Description
	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. Once the oscillator has stabilized, the PLL lock sequence is initiated. 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 1-25).

Figure 1-25. 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-1	R/W-0

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

Table 1-25. 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		Reserved
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 (default)
		000001	3 OSCCLKs
		. . .	. . .
		111111	65 OSCCLKs
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 [3]
		11	Set the low power mode to HALT [3]

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

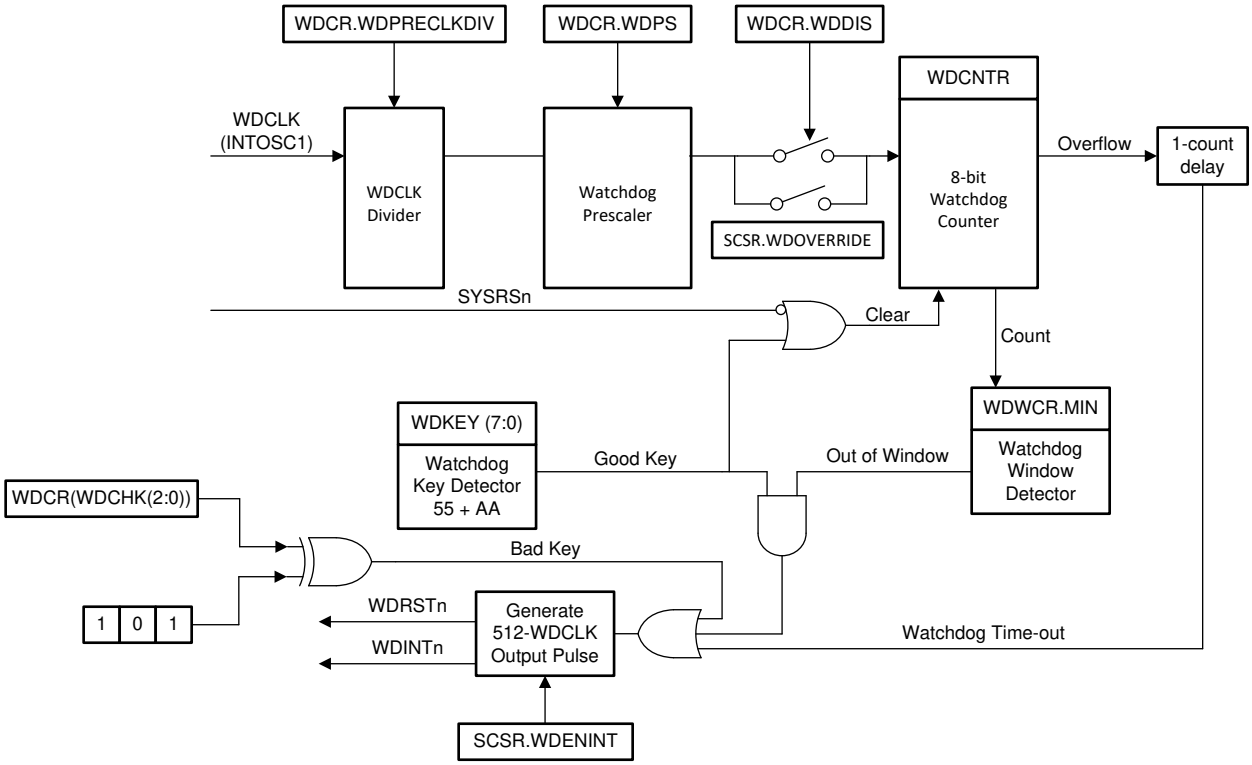
(2) The low power mode bits (LPM) only take effect when the IDLE instruction is executed. Therefore, you must set the LPM bits to the appropriate mode before executing the IDLE instruction.

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

1.3.4 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 1-26 shows the various functional blocks within the watchdog module.

Figure 1-26. 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. Watchdog is not functional and cannot generate a reset when OSCCLK is not present.

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 1-26. 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 1-26 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 disabled before $\overline{\text{WDINT}}$ goes inactive, the 512-cycle count will halt and $\overline{\text{WDINT}}$ will remain active. The count will resume when the watchdog is enabled again.

  If the watchdog is 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 1-27 describes the bit functions of the SCSR register.

Figure 1-27. 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 1-27. System Control and Status Register (SCSR) Field Descriptions

Bit	Field	Value	Description (1)
15-3	Reserved		
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 1-28. 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 1-28. Watchdog Counter Register (WDCNTR) Field Descriptions

Bits	Field	Description
15-8	Reserved	Reserved
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 1-29. 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 1-29. Watchdog Reset Key Register (WDKEY) Field Descriptions

Bits	Field	Value	Description [1]
15-8	Reserved		Reserved
7-0	WDKEY		Refer to Table 1-26 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 1-30. Watchdog Control Register (WDCR)

15	8
Reserved	

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 1-30. Watchdog Control Register (WDCR) Field Descriptions

Bits	Field	Value	Description [1]
15-8	Reserved		Reserved
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 1-30. 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 will happen even if the watchdog module is disabled. These three bits always read back as zero (0, 0, 0). This feature can be used to generate a software reset of the DSP.
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 (Figure 1-31) (TIMER0/1/2).

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 1-32.

Figure 1-31. CPU Timers

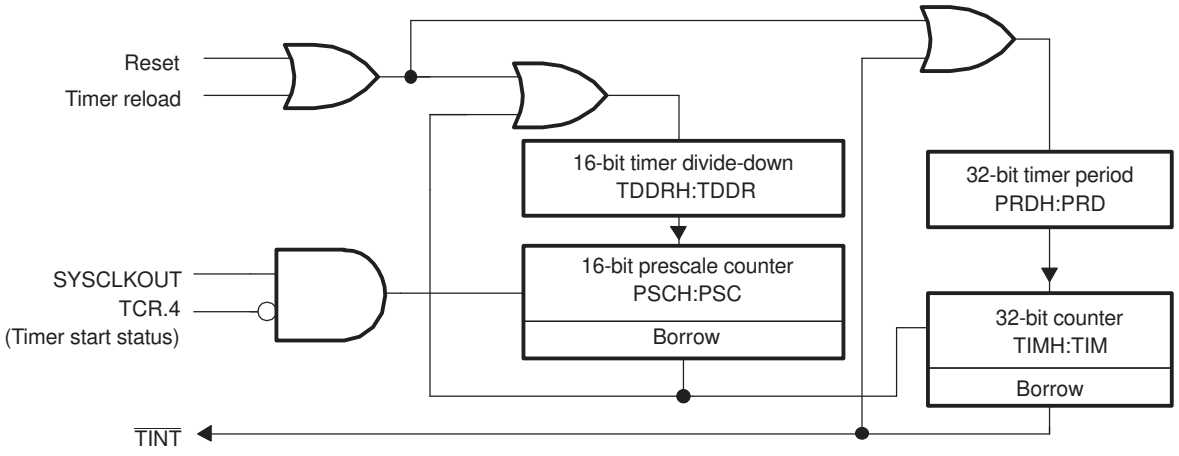
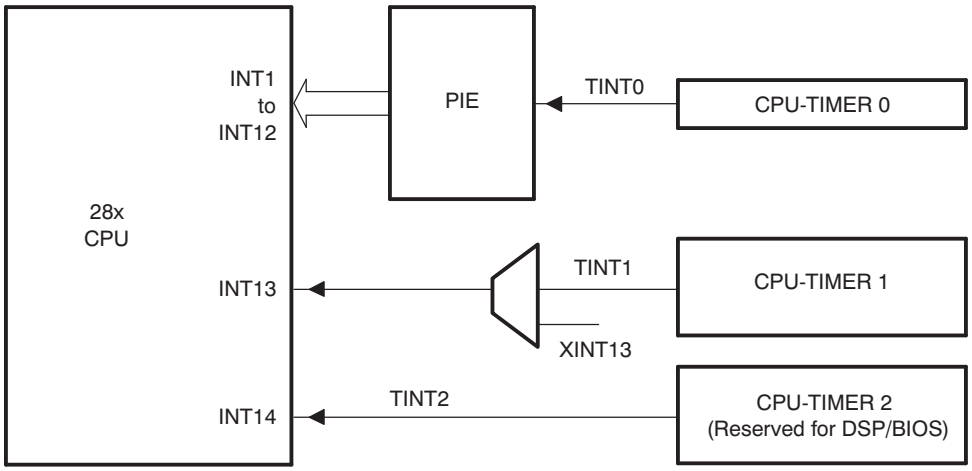


Figure 1-32. CPU-Timer Interrupt Signals and Output Signal



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 1-31 are used to configure the timers.

Table 1-31. 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 1-33
TIMER0TIMH	0x0C01	1	CPU-Timer 0, Counter Register High	Figure 1-34
TIMER0PRD	0x0C02	1	CPU-Timer 0, Period Register	Figure 1-35
TIMER0PRDH	0x0C03	1	CPU-Timer 0, Period Register High	Figure 1-36
TIMER0TCR	0x0C04	1	CPU-Timer 0, Control Register	Figure 1-37
Reserved	0x0C05	1		
TIMER0TPR	0x0C06	1	CPU-Timer 0, Prescale Register	Figure 1-38
TIMER0TPRH	0x0C07	1	CPU-Timer 0, Prescale Register High	Figure 1-39
TIMER1TIM	0x0C08	1	CPU-Timer 1, Counter Register	Figure 1-33
TIMER1TIMH	0x0C09	1	CPU-Timer 1, Counter Register High	Figure 1-34
TIMER1PRD	0x0C0A	1	CPU-Timer 1, Period Register	Figure 1-35
TIMER1PRDH	0x0C0B	1	CPU-Timer 1, Period Register High	Figure 1-36
TIMER1TCR	0x0C0C	1	CPU-Timer 1, Control Register	Figure 1-37
Reserved	0x0C0D	1		
TIMER1TPR	0x0C0E	1	CPU-Timer 1, Prescale Register	Figure 1-38
TIMER1TPRH	0x0C0F	1	CPU-Timer 1, Prescale Register High	Figure 1-39
TIMER2TIM	0x0C10	1	CPU-Timer 2, Counter Register	Figure 1-33
TIMER2TIMH	0x0C11	1	CPU-Timer 2, Counter Register High	Figure 1-34
TIMER2PRD	0x0C12	1	CPU-Timer 2, Period Register	Figure 1-35
TIMER2PRDH	0x0C13	1	CPU-Timer 2, Period Register High	Figure 1-36
TIMER2TCR	0x0C14	1	CPU-Timer 2, Control Register	Figure 1-37
Reserved	0x0C15	1		
TIMER2TPR	0x0C16	1	CPU-Timer 2, Prescale Register	Figure 1-38
TIMER2TPRH	0x0C17	1	CPU-Timer 2, Prescale Register High	Figure 1-39

Figure 1-33. 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 1-32. 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 1-34. 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 1-33. TIMERxTIMH Register Field Descriptions

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

Figure 1-35. TIMERxPRD Register (x = 0, 1, 2)

15	0
PRD	
R/W-0xFFFF	

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

Table 1-34. 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 1-36. 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 1-35. TIMERxPRDH Register Field Descriptions

Bits	Field	Description
15-0	PRDH	See description for TIMERxPRD

Figure 1-37. 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 1-36. TIMERxTCR Register Field Descriptions

Bits	Field	Value	Description
15	TIF		CPU-Timer Overflow Flag.
		0	The CPU-Timer has not decremented to zero. TIF indicates whether a timer overflow has happened since TIF was last cleared. TIF is not cleared automatically and does not need to be cleared to enable the next timer interrupt.
			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		Reserved
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		Reserved
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		Reserved

Figure 1-38. 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 1-37. 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 1-39. 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 1-38. TIMERxTPRH Register Field Descriptions

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

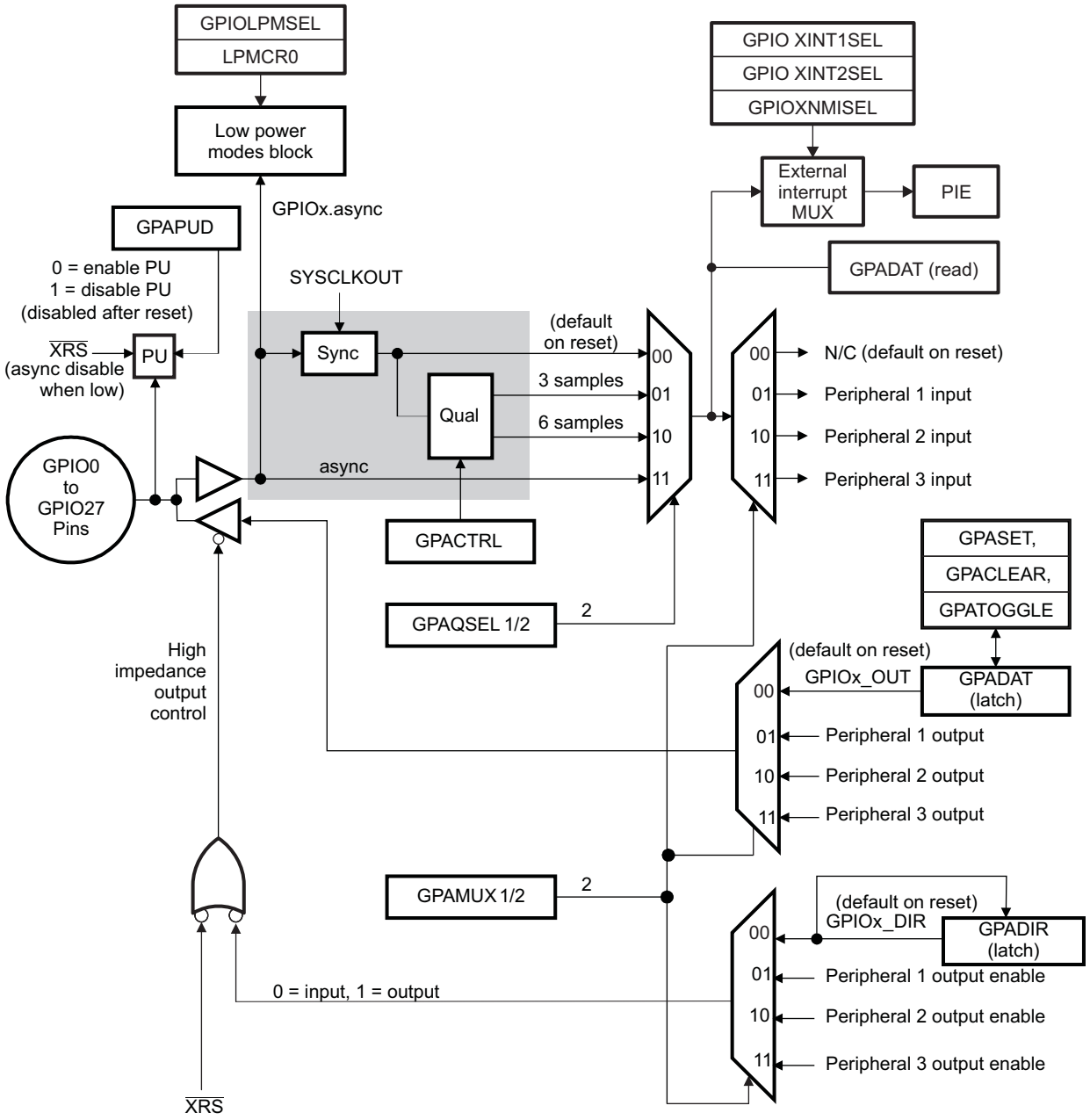
1.4 General-Purpose Input/Output (GPIO)

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 - GPIO87). 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 32-bit I/O ports. Port A consists of GPIO0-GPIO31, port B consists of GPIO32-GPIO63, and port C consists of GPIO64-87. Figure 1-40 shows the basic modes of operation for the GPIO module.

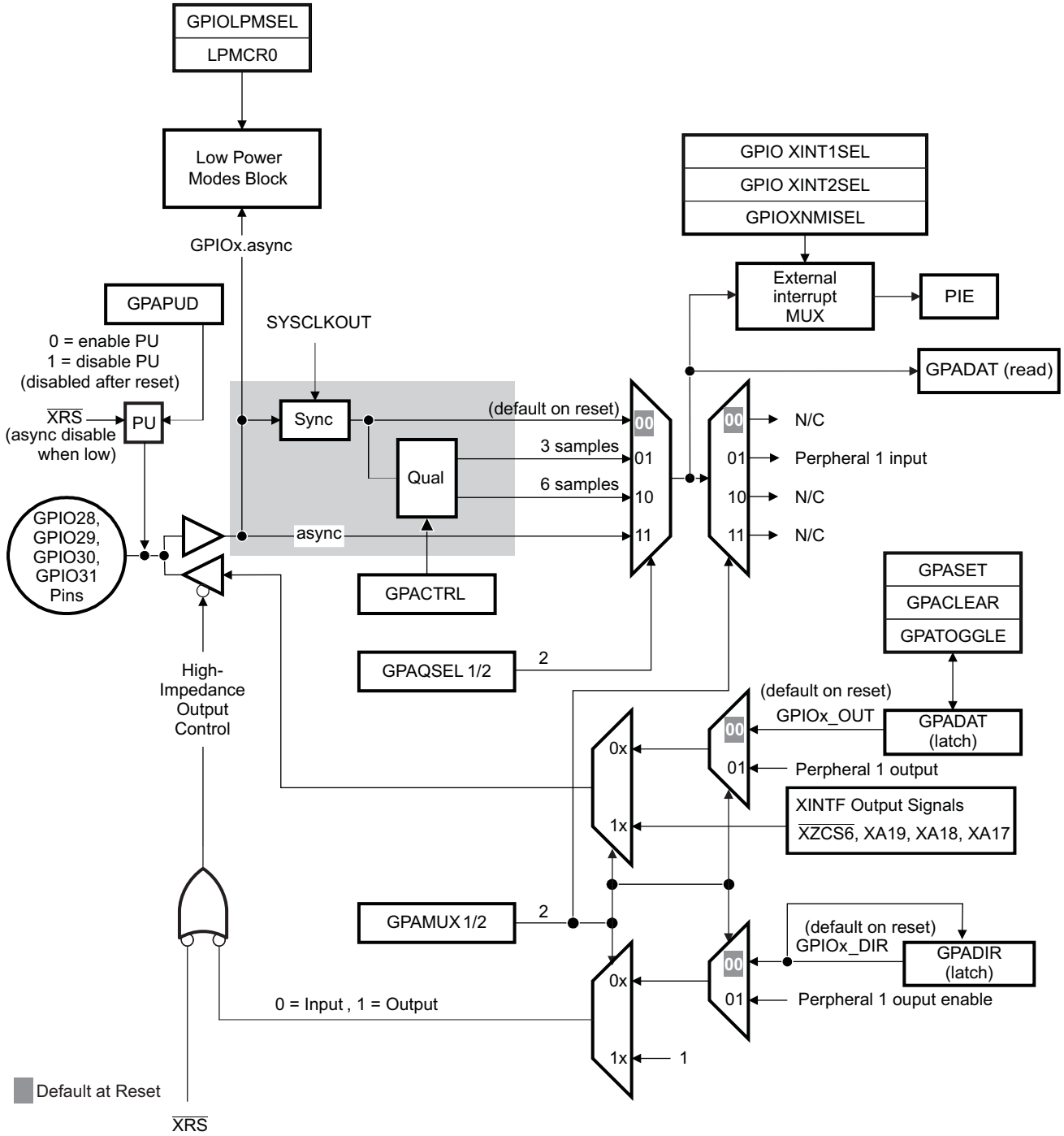
Figure 1-40. GPIO0 to GPIO27 Multiplexing Diagram



A The shaded area is disabled in the above GPIOs when the GPIOINENCLK bit is cleared to 0 in the PCLKCR3 register and the respective pin is configured as an output. This is to reduce power consumption when a pin is configured as an output. Clearing the GPIOINCLK bit will reset the sync and qualification logic so no residual value is left.

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

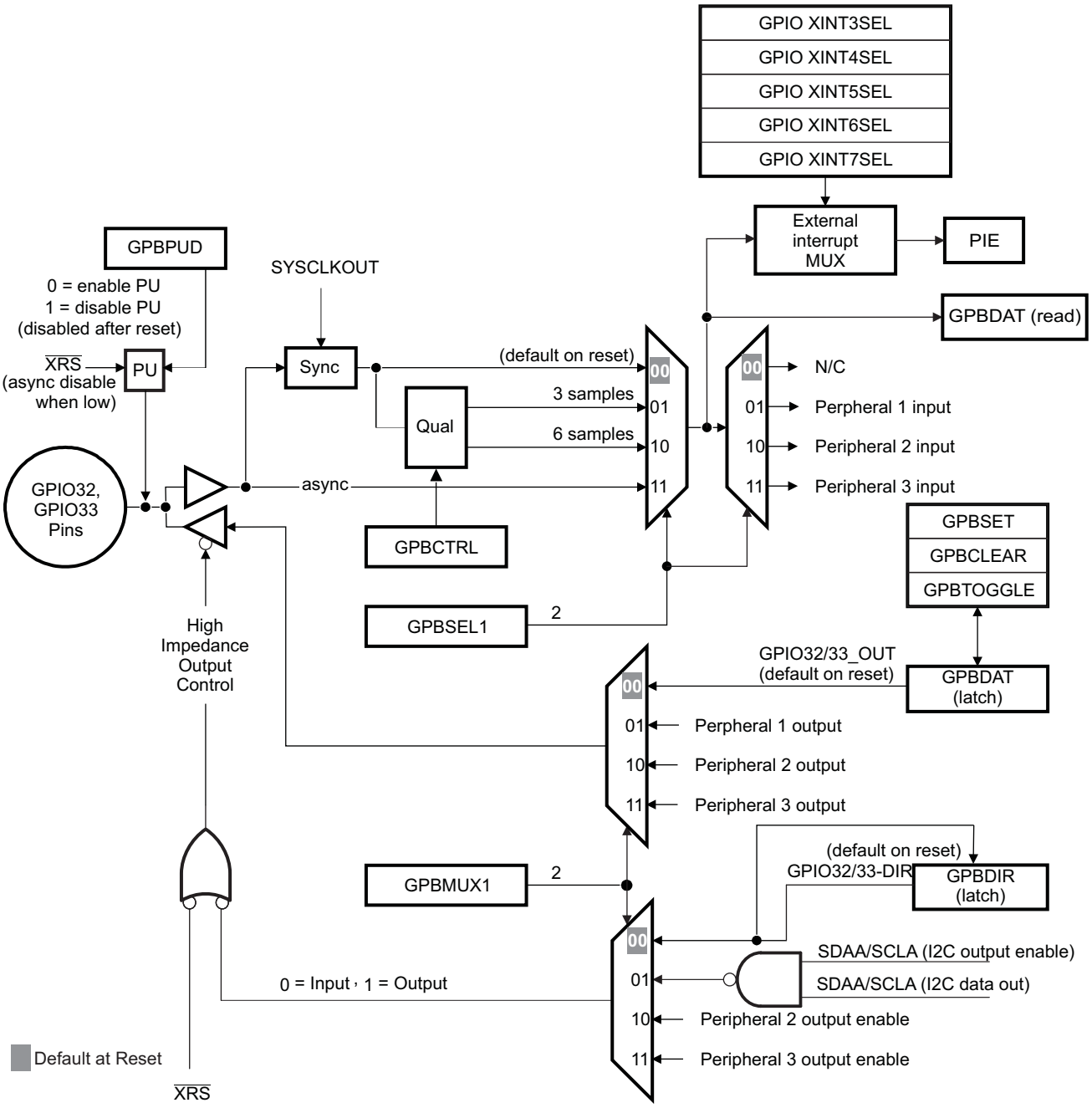
Figure 1-41. GPIO28 to GPIO31 Multiplexing Diagram (Peripheral 2 and Peripheral 3 Outputs Merged)



A The shaded area is disabled in the above GPIOs when the GPIOINENCLK bit is cleared to 0 in the PCLKCR3 register and the respective pin is configured as an output. This is to reduce power consumption when a pin is configured as an output. Clearing the GPIOINCLK bit will reset the sync and qualification logic so no residual value is left.

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

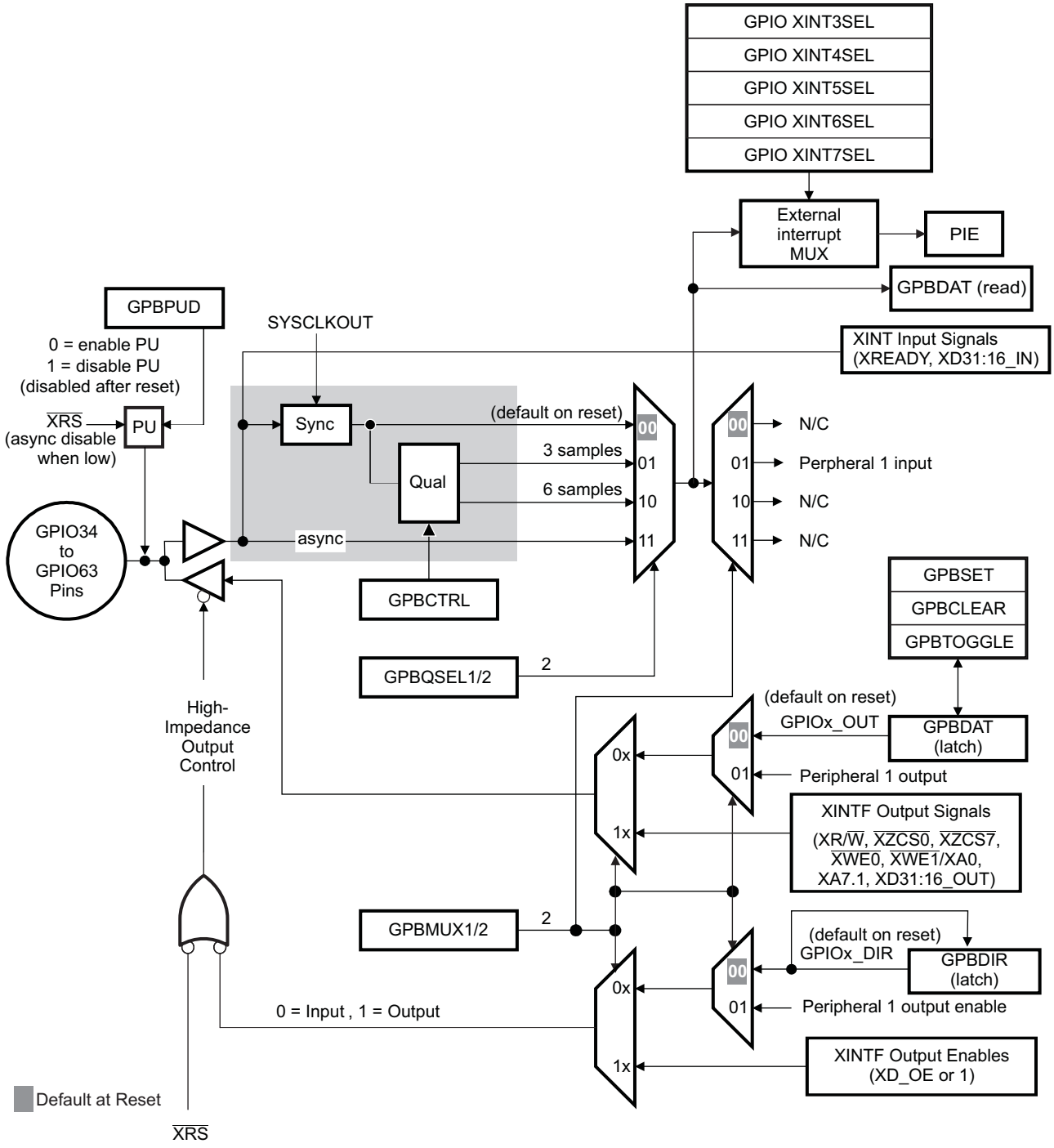
Figure 1-42. GPIO32, GPIO33 Multiplexing Diagram



A The GPIOINENCLK bit in the PCLKCR3 register does not affect the above GPIOs (I2C pins) since the pins are bi-directional.

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

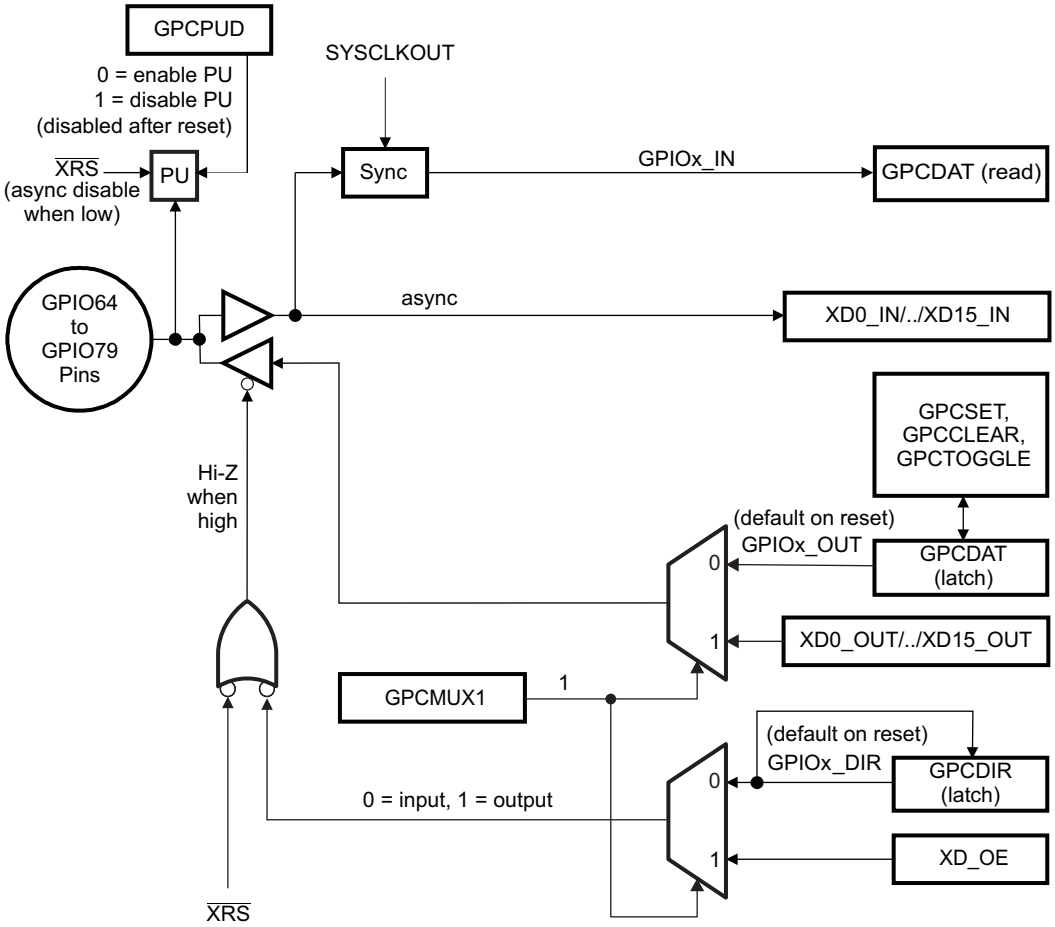
Figure 1-43. GPIO34 to GPIO63 Multiplexing Diagram (Peripheral 2 and Peripheral 3 Outputs Merged)



A The shaded area is disabled in the above GPIOs when the GPIOINENCLK bit is cleared to "0" in the PCLKCR3 register and the respective pin is configured as an output. This is to reduce power consumption when a pin is configured as an output. Clearing the GPIOINCLK bit will reset the sync and qualification logic so no residual value is left.

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

Figure 1-44. GPIO64 to GPIO79 Multiplexing Diagram (Minimal GPIOs Without Qualification)



1.4.2 Configuration Overview

The pin function assignments, input qualification, and the external interrupt (XINT1 – XINT7, XNMI) 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 1-39 and Table 1-40 list the registers that are used to configure the GPIO pins to match the system requirements.

Table 1-39. GPIO Control Registers

Name [1]	Address	Size (x16)	Register Description	Bit Description
GPACTRL	0x6F80	2	GPIO A Control Register (GPIO0-GPIO31)	Figure 1-53
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register (GPIO0-GPIO15)	Figure 1-55
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register (GPIO16-GPIO31)	Figure 1-56
GPAMUX1	0x6F86	2	GPIO A MUX 1 Register (GPIO0-GPIO15)	Figure 1-47
GPAMUX2	0x6F88	2	GPIO A MUX 2 Register (GPIO16-GPIO31)	Figure 1-48
GPADIR	0x6F8A	2	GPIO A Direction Register (GPIO0-GPIO31)	Figure 1-59
GPAPUD	0x6F8C	2	GPIO A Pull Up Disable Register (GPIO0-GPIO31)	Figure 1-62
GPBCTRL	0x6F90	2	GPIO B Control Register (GPIO32-GPIO63)	Figure 1-54
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register (GPIO32-GPIO47)	Figure 1-57
GPBQSEL2	0x6F94	2	GPIO B Qualifier Select 2 Register (GPIO48 - GPIO63)	Figure 1-58
GPBMUX1	0x6F96	2	GPIO B MUX 1 Register (GPIO32-GPIO47)	Figure 1-49
GPBMUX2	0x6F98	2	GPIO B MUX 2 Register (GPIO48-GPIO63)	Figure 1-50
GPBDIR	0x6F9A	2	GPIO B Direction Register (GPIO32-GPIO63)	Figure 1-60
GPBPUD	0x6F9C	2	GPIO B Pull Up Disable Register (GPIO32-GPIO63)	Figure 1-63
GPCMUX1	0x6FA6	2	GPIO C MUX 1 Register (GPIO64-GPIO79)	Figure 1-51
GPCMUX2	0x6FA8	2	GPIO C MUX 2 Register (GPIO80-GPIO87)	Figure 1-52
GPCDIR	0x6FAA	2	GPIO C Direction Register (GPIO64-GPIO87)	Figure 1-61
GPCPUD	0x6FAC	2	GPIO C Pull Up Disable Register (GPIO64-GPIO87)	Figure 1-64

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

Table 1-40. 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 1-71
GPIOXINT2SEL	0x6FE1	1	XINT2 Source Select Register (GPIO0-GPIO31)	Figure 1-71
GPIOXNMISEL	0x6FE2	1	XNMI Source Select Register (GPIO0-GPIO31)	Figure 1-71
GPIOXINT3SEL	0x6FE3	1	XINT3 Source Select Register (GPIO32 - GPIO63)	Table 1-82
GPIOXINT4SEL	0x6FE4	1	XINT4 Source Select Register (GPIO32 - GPIO63)	Table 1-82
GPIOXINT5SEL	0x6FE5	1	XINT5 Source Select Register (GPIO32 - GPIO63)	Table 1-82
GPIOXINT6SEL	0x6FE6	1	XINT6 Source Select Register (GPIO32 - GPIO63)	Table 1-82
GPIOXINT7SEL	0x6FE7	1	XINT7 Source Select Register (GPIO32 - GPIO63)	Table 1-82
GPIOLPMSEL	0x6FE8	1	LPM wakeup Source Select Register (GPIO0-GPIO31)	Figure 1-72

(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 pullup resistors:**

To enable or disable the internal pullup resistors, write to the respective bits in the GPIO pullup disable (GPAPUD, GPBPUD, and GPCPUD) registers. For pins that can function as ePWM output pins (GPIO0-GPIO11), the internal pullup resistors are disabled by default. All other GPIO-capable pins have the pullup enabled by default.

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 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, and GPCDIR 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 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 - XINT7, and XNMI interrupts. For each interrupt you can specify one of the port A signals (for XINT1/2/3) or port B signals (XINT4/5/6/7) as the source. This is done by specifying the source in the GPIOXINTnSEL, and GPIOXNMISEL registers. The polarity of the interrupts can be configured in the XINTnCR, and the XNMICR registers as described in Section 1.6.5.

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

1.4.3 Digital General Purpose I/O Control

For pins that are configured as GPIO you can change the values on the pins by using the registers in Table 1-41.

Table 1-41. GPIO Data Registers

Name	Address	Size (x16)	Register Description	Bit Description
GPADAT	0x6FC0	2	GPIO A Data Register (GPIO0-GPIO31)	Figure 1-65
GPASET	0x6FC2	2	GPIO A Set Register (GPIO0-GPIO31)	Figure 1-68
GPACLEAR	0x6FC4	2	GPIO A Clear Register (GPIO0-GPIO31)	Figure 1-68
GPATOGGLE	0x6FC6	2	GPIO A Toggle Register (GPIO0-GPIO31)	Figure 1-68
GPBDAT	0x6FC8	2	GPIO B Data Register (GPIO32-GPIO63)	Figure 1-66
GPBSET	0x6FCA	2	GPIO B Set Register (GPIO32-GPIO63)	Figure 1-69
GPBCLEAR	0x6FCC	2	GPIO B Clear Register (GPIO32-GPIO63)	Figure 1-69

Table 1-41. GPIO Data Registers (continued)

Name	Address	Size (x16)	Register Description	Bit Description
GPBTOGGLE	0x6FCE	2	GPIO B Toggle Register (GPIO32-GPIO63)	Figure 1-69
GPCDAT	0x6FD0	2	GPIO C Data Register (GPIO64 - GPIO87)	Figure 1-67
GPCSET	0x6FD2	2	GPIO C Set Register (GPIO64 - GPIO87)	Figure 1-70
GPCCLEAR	0x6FD4	2	GPIO C Clear Register (GPIO64 - GPIO87)	Figure 1-70
GPCTOGGLE	0x6FD6	2	GPIO C Toggle Register (GPIO64 - GPIO87)	Figure 1-70

- **GPxDAT 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. Writing to the GPxDAT 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 GPIOA0 by writing to the GPADAT register bit 0, using a read-modify-write instruction. The problem can occur if another I/O port A signal changes level between the read and the write stage of the instruction. You can also change the state of that output latch. You can avoid this scenario by using the GPxSET, GPxCLEAR, and GPxTOGGLE registers to load the output latch instead.

- **GPxSET 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 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 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, eCAN, and I2C. In addition, it may be desirable to have the ePWM trip zone ($\overline{TZ1}$-$\overline{TZ6}$) 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.

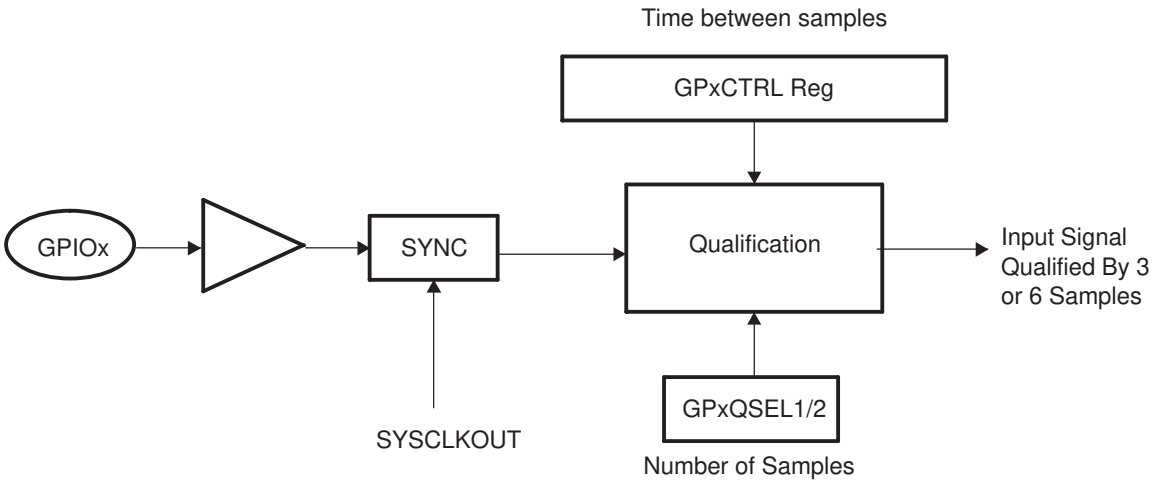
1.4.4.2 Synchronization to SYSCLKOUT Only

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 1-45 and Figure 1-46 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 1-45. 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 1-42 and Table 1-43 show the relationship between the sampling period or sampling frequency and the GPxCTRL[QUALPRDn] setting.

Table 1-42. Sampling Period

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

Table 1-43. Sampling Frequency

	Sampling Frequency
If GPxCTRL[QUALPRDn] = 0	$f_{SYSCLKOUT}$
If GPxCTRL[QUALPRDn] ≠ 0	$f_{SYSCLKOUT} \times 1 \div (2 \times$ 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, $f_{SYSCLKOUT}$ = 150 MHz

then the signal will be sampled at 150 MHz or one sample every 6.67 ns.

Example: Minimum Sampling Frequency:

If GPxCTRL[QUALPRDn] = 0xFF (255)

then the sampling frequency is $f_{SYSCLKOUT} \times 1 \div (2 \times$ GPxCTRL[QUALPRDn])

If, for example, $f_{SYSCLKOUT}$ = 150 MHz

then the signal will be sampled at 150 MHz × 1 ÷ (2 × 255) or one sample every 3.4 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 1-46. 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 1-42. Likewise, for a six-sample window, the sampling window width is 5 sampling periods wide. Table 1-44 and Table 1-45 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 1-44. Case 1: Three-Sample Sampling Window Width

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

Table 1-45. 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:

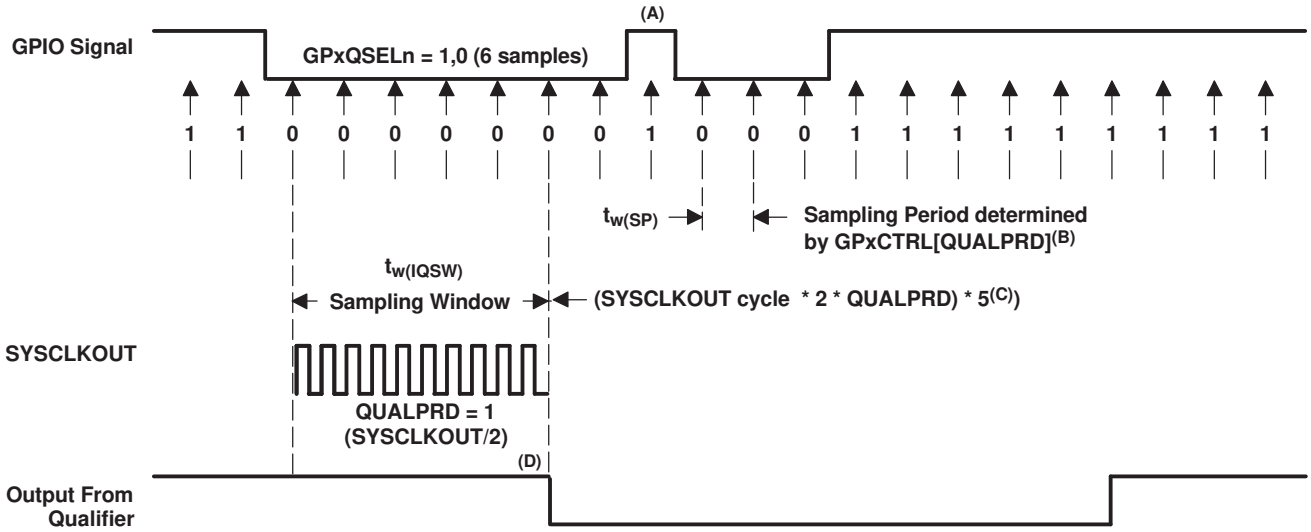
For the example shown in Figure 1-46, the input qualification has been configured as follows:

- 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, $T_{SYSCLKOUT}$ = 6.67 ns, then the duration of the sampling window is:
  $t_w(IQSW) = 5 \times 2 \times 6.67$ ns = 67 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:
  $t_w(SP) + T_{SYSCLKOUT}$ = 13.34 ns + 6.67 ns = 20 ns
- In Figure 1-46, the glitch (A) is shorter then the qualification window and will be ignored by the input qualifier.

Figure 1-46. Input Qualifier Clock Cycles



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 1-47, Table 1-48, and Table 1-49 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 GPIO7 pin is controlled by writing to GPAMUX[15:14]. By writing to these bits, the pin is configured as either GPIO7, or one of up to three peripheral functions. The GPIO7 pin can be configured as follows:

GPAMUX1[15:14] Bit Setting	Pin Functionality Selected
If GPAMUX1[15:14] = 0,0	Pin configured as GPIO7
If GPAMUX1[15:14] = 0,1	Pin configured as EPWM4B (O)
If GPAMUX1[15:14] = 1,0	Pin configured as MCLKRA (I/O)
If GPAMUX1[15:14] = 1,1	Pin configured as ECAP2 (I/O)

All devices in the 2833x, 2823x family have the same multiplexing scheme. The only difference is that 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 1-47, Table 1-48, and Table 1-49) these options are indicated as "Reserved".

Some peripherals can be assigned to more than one pin via the MUX registers. For example, the CAP1 function can be assigned to either the GPIO5 or GPIO24 pin, depending on individual system requirements as shown below:

Pin Assigned to CAP1		MUX Configuration
Choice 1	GPIO5	GPAMUX1[11:10] = 1,1
or Choice 2	GPIO24	GPAMUX2[17:16] = 0,1

If no pin is configured as an input to a peripheral, or if more than one pin is configured as an input for the same peripheral, then the input to the peripheral will either default to a 0 or a 1 as shown in Table 1-46. For example, if ECAP1 were assigned to both GPIO5 and GPIO24, the input to the eCAP1 peripheral would default to a high state as shown in Table 1-46 and the input would not be connected to GPIO5 or GPIO24.

Table 1-46. Default State of Peripheral Input

Peripheral Input	Description	Default Input [1]
$\overline{\text{TZ1}}$-$\overline{\text{TZ6}}$	Trip zone 1-6	1
EPWMSYNCI	ePWM Synch Input	0
ECAPn	eCAP input	1
EQEPnA	eQEP input	1
EQEPnI	eQEP index	1
EQEPnS	eQEP strobe	1
SPICLKx	SPI clock	1
$\overline{\text{SPISTE}}$x	SPI transmit enable	0
SPISIMOx	SPI Slave-in, master-out	1
SPISOMIx	SPI Slave-out, master-in	1
SCIRXDx	SCI receive	1
CANRXx	CAN receive	1
SDAA	I2C data	1
SCLA1	I2C 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 1-47. GPIOA MUX

	Default at Reset Primary I/O Function	Peripheral Selection	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[1]	Reserved[1]
3-2	GPIO1	EPWM1B (O)	ECAP6 (I/O)	MFSRB (I/O)[1]
5-4	GPIO2	EPWM2A (O)	Reserved[1]	Reserved[1]
7-6	GPIO3	EPWM2B (O)	ECAP5 (I/O)	MCLKRB (I/O)[1]
9-8	GPIO4	EPWM3A (O)	Reserved[1]	Reserved[1]
11-10	GPIO5	EPWM3B (O)	MFSRA (I/O)	ECAP1 (I/O)
13-12	GPIO6	EPWM4A (O)	EPWMSYNCI (I)	EPWMSYNCO (O)
15-14	GPIO7	EPWM4B (O)	MCLKRA (I/O)	ECAP2 (I/O)
17-16	GPIO8	EPWM5A (O)	CANTXB (O)	$\overline{\text{ADCSOCAO}}$ (O)
19-18	GPIO9	EPWM5B (O)	SCITXDB (O)	ECAP3 (I/O)
21-20	GPIO10	EPWM6A (O)	CANRXB (I)	$\overline{\text{ADCSOCBO}}$ (O)
23-22	GPIO11	EPWM6B (O)	SCIRXDB (I)	ECAP4 (I/O)
25-24	GPIO12	$\overline{\text{TZ1}}$ (I)	CANTXB (O)	MDXB (O)
27-26	GPIO13	$\overline{\text{TZ2}}$ (I)	CANRXB (I)	MDRB (I)
29-28	GPIO14	$\overline{\text{TZ3}}$/$\overline{\text{XHOLD}}$ (I)	SCITXDB (O)	MCLKXB (I/O)
31-30	GPIO15	$\overline{\text{TZ4}}$/$\overline{\text{XHOLDA}}$ (O)	SCIRXDB (I)	MFSXB (I/O)
GPAMUX2 Register Bits	**(GPAMUX2 bits = 00)**	**(GPAMUX2 bits = 01)**	**(GPAMUX2 bits = 10)**	**(GPAMUX2 bits = 11)**
1-0	GPIO16	SPISIMOA (I/O)	CANTXB (O)	$\overline{\text{TZ5}}$ (I)
3-2	GPIO17	SPISOMIA (I/O)	CANRXB (I)	$\overline{\text{TZ6}}$ (I)
5-4	GPIO18	SPICLKA (I/O)	SCITXDB (O)	CANRXA (I)
7-6	GPIO19	$\overline{\text{SPISTEA}}$ (I/O)	SCIRXDB (I)	CANTXA (O)
9-8	GPIO20	EQEP1A (I)	MDXA (O)	CANTXB (O)
11-10	GPIO21	EQEP1B (I)	MDRA (I)	CANRXB (I)
13-12	GPIO22	EQEP1S (I/O)	MCLKXA (I/O)	SCITXDB (O)
15-14	GPIO23	EQEP1I (I/O)	MFSXA (I/O)	SCIRXDB (I)
17-16	GPIO24	ECAP1 (I/O)	EQEP2A (I)	MDXB (O)
19-18	GPIO25	ECAP2 (I/O)	EQEP2B (I)	MDRB (I)
21-20	GPIO26	ECAP3 (I/O)	EQEP2I (I/O)	MCLKXB (I/O)
23-22	GPIO27	ECAP4 (I/O)	EQEP2S (I/O)	MFSXB (I/O)
25-24	GPIO28	SCIRXDA (I)	$\overline{\text{XZCS6}}$ (O)	$\overline{\text{XZCS6}}$ (O)
27-26	GPIO29	SCITXDA (O)	XA19 (O)	XA19 (O)
29-28	GPIO30	CANRXA (I)	XA18 (O)	XA18 (O)
31-30	GPIO31	CANTXA (O)	XA17 (O)	XA17 (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.

Table 1-48. GPIOB MUX

	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 (I/O)	SDAA (I/OC)	EPWMSYNCI (I)	ADCSOCAO (O)
3,2	GPIO33 (I/O)	SCLA (I/OC)	EPWMSYNCO (O)	ADCSOCBO (O)
5,4	GPIO34 (I/O)	ECAP1 (I/O)	XREADY (I)	XREADY (I)
7,6	GPIO35 (I/O)	SCITXDA (O)	XR/W (O)	XR/W (O)
9,8	GPIO36 (I/O)	SCIRXDA (I)	XZCS0 (O)	XZCS0 (O)
11,10	GPIO37 (I/O)	ECAP2 (I/O)	XZCS7 (O)	XZCS7 (O)
13,12	GPIO38 (I/O)	Reserved	XWE0 (O)	XWE0 (O)
15,14	GPIO39 (I/O)	Reserved	XA16 (O)	XA16 (O)
17,16	GPIO40 (I/O)	Reserved	XA0/XWE1 (O)	XA0/XWE1 (O)
19,18	GPIO41 (I/O)	Reserved	XA1 (O)	XA1 (O)
21,20	GPIO42 (I/O)	Reserved	XA2 (O)	XA2 (O)
23,22	GPIO43 (I/O)	Reserved	XA3 (O)	XA3 (O)
25,24	GPIO44 (I/O)	Reserved	XA4 (O)	XA4 (O)
27,26	GPIO45 (I/O)	Reserved	XA5 (O)	XA5 (O)
29,28	GPIO46 (I/O)	Reserved	XA6 (O)	XA6 (O)
31,30	GPIO47 (I/O)	Reserved	XA7 (O)	XA7 (O)
GPBMUX2 Register Bits	**(GPBMUX2 bits = 00)**	**(GPBMUX2 bits = 01)**	**(GPBMUX2 bits = 10 or 11)**	
1,0	GPIO48 (I/O)	ECAP5 (I/O)	XD31 (I/O)	
3,2	GPIO49 (I/O)	ECAP6 (I/O)	XD30 (I/O)	
5,4	GPIO50 (I/O)	EQEP1A (I)	XD29 (I/O)	
7,6	GPIO51 (I/O)	EQEP1B (I)	XD28 (I/O)	
9,8	GPIO52 (I/O)	EQEP1S (I/O)	XD27 (I/O)	
11,10	GPIO53 (I/O)	EQEP1I (I/O)	XD26 (I/O)	
13,12	GPIO54 (I/O)	SPISIMOA (I/O)	XD25 (I/O)	
15,14	GPIO55 (I/O)	SPISOMIA (I/O)	XD24 (I/O)	
17,16	GPIO56 (I/O)	SPICLKA (I/O)	XD23 (I/O)	
19,18	GPIO57 (I/O)	SPISTEA (I/O)	XD22 (I/O)	
21,20	GPIO58 (I/O)	MCLKRA (I/O)	XD21 (I/O)	
23,22	GPIO59 (I/O)	MFSRA (I/O)	XD20 (I/O)	
25,24	GPIO60 (I/O)	MCLKRB (I/O)	XD19 (I/O)	
27,26	GPIO61 (I/O)	MFSRB (I/O)	XD18 (I/O)	
29,28	GPIO62 (I/O)	SCIRXDC (I)	XD17 (I/O)	
31,30	GPIO63 (I/O)	SCITXDC (O)	XD16 (I/O)	

Table 1-49. GPIOC MUX

	Default at Reset Primary I/O Function	Peripheral Selection 2 or 3
GPCMUX1 Register Bits	**(GPCMUX1 bits = 00 or 01)**	**(GPCMUX1 bits = 10 or 11)**
1,0	GPIO64 (I/O)	XD15 (I/O)
3,2	GPIO65 (I/O)	XD14 (I/O)
5,4	GPIO66 (I/O)	XD13 (I/O)
7,6	GPIO67 (I/O)	XD12 (I/O)
9,8	GPIO68 (I/O)	XD11 (I/O)
11,10	GPIO69 (I/O)	XD10 (I/O)
13,12	GPIO70 (I/O)	XD9 (I/O)
15,14	GPIO71 (I/O)	XD8 (I/O)
17,16	GPIO72 (I/O)	XD7 (I/O)
19,18	GPIO73 (I/O)	XD6 (I/O)
21,20	GPIO74 (I/O)	XD5 (I/O)
23,22	GPIO75 (I/O)	XD4 (I/O)
25,24	GPIO76 (I/O)	XD3 (I/O)
27,26	GPIO77 (I/O)	XD2 (I/O)
29,28	GPIO78 (I/O)	XD1 (I/O)
31,30	GPIO79 (I/O)	XD0 (I/O)
GPCMUX2 Register Bits	**GPCMUX2 bits = 00 or 01**	**GPCMUX2 bits = 10 or 11**
1,0	GPIO80 (I/O)	XA8 (O)
3,2	GPIO81 (I/O)	XA9 (O)
5,4	GPIO82 (I/O)	XA10 (O)
7,6	GPIO83 (I/O)	XA11 (O)
9,8	GPIO84 (I/O)	XA12 (O)
11,10	GPIO85 (I/O)	XA13 (O)
13,12	GPIO86 (I/O)	XA14 (O)
15,14	GPIO87 (I/O)	XA15 (O)
16 – 31	Reserved	Reserved

1.4.6 Register Bit Definitions

Figure 1-47. 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 1-50. 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	$\overline{TZ4}$ - Trip Zone 4 (I) or $\overline{XHOLDA}$ (O). The pin function for this option is based on the direction chosen in the GPADIR register. If the pin is configured as an input, then $\overline{TZ4}$ function is chosen. If the pin is configured as an output, then $\overline{XHOLDA}$ function is chosen. $\overline{XHOLDA}$ is driven active (low) when the XINTF has granted an $\overline{XHOLD}$ request. All XINTF buses and strobe signals will be in a high-impedance state. $\overline{XHOLDA}$ is released when the $\overline{XHOLD}$ signal is released. External devices should only drive the external bus when $\overline{XHOLDA}$ is active (low).
		10	SCIRXDB - SCI-B receive. (I)
		11	MFSXB - McBSP-B transmit frame synch (I/O) This option is reserved on devices that do not have a McBSP-B port. [2]
29-28	GPIO14		Configure the GPIO14 pin as:
		00	GPIO14 - General purpose I/O 14 (default) (I/O)
		01	$\overline{TZ3}$ - Trip zone 3 or $\overline{XHOLD}$ (I). $\overline{XHOLD}$, when active (low), requests the external memory interface (XINTF) to release the external bus and place all buses and strobes into a high-impedance state. To prevent this from happening when $\overline{TZ3}$ signal goes active, disable this function by writing XINTCNF2[HOLD] = 1. If this is not done, the XINTF bus will go into high impedance anytime $\overline{TZ3}$ goes low. On the ePWM side, $\overline{TZn}$ signals are ignored by default, unless they are enabled by the code. The XINTF will release the bus when any current access is complete and there are no pending accesses on the XINTF. (I)
		10	SCITXDB - SCI-B transmit (O)
		11	MCLKXB - McBSP-B transmit clock (I/O) This option is reserved on devices that do not have a McBSP-B port. [2]
27-26	GPIO13		Configure the GPIO13 pin as:
		00	GPIO13 - General purpose I/O 13 (default) (I/O)
		01	$\overline{TZ2}$ - Trip zone 2 (I)
		10	CANRXB - eCAN-B receive. (I)
		11	MDRB - McBSP-B Data Receive (I) This option is reserved on devices that do not have a McBSP-B port. [2]
25-24	GPIO12		Configure the GPIO12 pin as:
		00	GPIO12 - General purpose I/O 12 (default) (I/O)
		01	$\overline{TZ1}$ - Trip zone 1 (I)
		10	CANTXB - eCAN-B transmit. (O)
		11	MDXB - McBSP-B, Data transmit (O) This option is reserved on devices that do not have a McBSP-B port. [2]

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

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

Table 1-50. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions (continued)

Bits	Field	Value	Description [1]
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 - SCI-B receive (I)
		11	ECAP4 - eCAP4. (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	CANRXB - eCAN-B receive (I)
		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 - SCI-B transmit (O)
		11	ECAP3 - 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	CANTXB - eCAN-B transmit (O)
		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	MCLKRA - McBSP-A Receive clock (I/O)
		11	ECAP2 - 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	MFSRA - McBSP-A Receive frame synch (I/O)
		11	ECAP1 - eCAP1 (I/O)
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. [2]
		11	Reserved. [2]
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	ECAP5 - eCAP5 (I/O)
		11	MCLKRB - McBSP-B receive clock (I/O)

Table 1-50. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions (continued)

Bits	Field	Value	Description [1]
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. [2]
		11	Reserved. [2]
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	ECAP6 - eCAP6 (I/O)
		11	MFSRB - McBSP-B Receive Frame Synch (I/O)
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. [2]
		11	Reserved. [2]

Figure 1-48. 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	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 1-51. 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 or 11	XA17 - External interface address line 17 (O)
29-28	GPIO30		Configure the GPIO30 pin as:
		00	GPIO30 (I/O) General purpose I/O 30 (default) (I/O)
		01	CANRXA - eCAN-A receive (I)
		10 or 11	XA18 - External interface address line 18
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 or 11	XA19 - External Interface address line 19 (O)
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 or 11	$\overline{\text{XZCS6}}$ - External interface zone 6 chip select (O)

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

Table 1-51. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions (continued)

Bits	Field	Value	Description [1]
23-22	GPIO27		Configure the GPIO27 pin as:
		00	GPIO27 - General purpose I/O 27 (default) (I/O)
		01	ECAP4 - eCAP4. (I/O)
		10	EQEP2S - eQEP2 strobe (I/O)
		11	MFSXB - McBSP-B Transmit Frame Synch (I/O)
21-20	GPIO26		Configure the GPIO26 pin as:
		00	GPIO26 - General purpose I/O 26 (default) (I/O)
		01	ECAP3 - eCAP3. (I/O)
		10	EQEP2I - eQEP2 index. (I/O)
		11	MCLKXB - McBSP-B Transmit Clock (I/O)
19-18	GPIO25		Configure the GPIO25 pin as:
		00	GPIO25 - General purpose I/O 25 (default) (I/O)
		01	ECAP2 - eCAP2 (I/O)
		10	EQEP2B - eQEP2 input B (I)
		11	MDRB - McBSP-B data receive (O)
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 - eQEP2 input A. (I)
		11	MDXB - McBSP-B data transmit (O)
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 - McBSP-A transmit frame synch (I/O)
		11	SCIRXDB - SCI-B receive (I/O)
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 - McBSP-A transmit clock (I/O)
		11	SCITXDB - SCI-B transmit (O)
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 - McBSP-A data receive (I)
		11	CANRXB - eCAN-B receive (I)
9-8	GPIO20		Configure the GPIO20 pin as:
		00	GPIO20 - General purpose I/O 22 (default) (I/O)
		01	EQEP1A - eQEP1 input A (I)
		10	MDXA - McBSP-A data transmit (O)
		11	CANTXB - eCAN-B transmit (O)
7-6	GPIO19		Configure the GPIO19 pin as:
		00	GPIO19 - General purpose I/O 19 (default) (I/O)
		01	$\overline{\text{SPISTEA}}$ - SPI-A slave transmit enable (I/O)
		10	SCIRXDB - SCI-B receive (I)
		11	CANTXA - eCAN-A Transmit (O)

Table 1-51. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions (continued)

Bits	Field	Value	Description [(1)]
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 - SCI-B transmit. (O)
		11	CANRXA - eCAN-A Receive (I)
3-2	GPIO17		Configure the GPIO17 pin as:
		00	GPIO17 - General purpose I/O 17 (default) (I/O)
		01	SPISOMIA - SPI-A slave-out, master-in (I/O)
		10	CANRXB eCAN-B receive (I)
		11	$\overline{TZ6}$ - Trip zone 6 (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	CANTXB - eCAN-B transmit. (O)
		11	$\overline{TZ5}$ - Trip zone 5 (I)

Figure 1-49. GPIO Port B MUX 1 (GPBMUX1) Register

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
GPIO47	GPIO46	GPIO45	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
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
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 1-52. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions

Bit	Field	Value	Description
31:30	GPIO47		Configure this pin as:
		00	GPIO 47 - general purpose I/O 47 (default)
		01	Reserved
		10 or 11	XA7 - External interface (XINTF) address line 7 (O)
29:28	GPIO46		Configure this pin as:
		00	GPIO 46 - general purpose I/O 46 (default)
		01	Reserved
		10 or 11	XA6 - External interface (XINTF) address line 6 (O)
27:26	GPIO45		Configure this pin as:
		00	GPIO 45 - general purpose I/O 45 (default)
		01	Reserved
		10 or 11	XA5 - External interface (XINTF) address line 5 (O)
25:24	GPIO44		Configure this pin:
		00	GPIO 44 - general purpose I/O 44 (default)
		01	Reserved
		10 or 11	XA4 - External interface (XINTF) address line 4 (O)

Table 1-52. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions (continued)

Bit	Field	Value	Description
23:22	GPIO43		Configure this pin as:
		00	GPIO 43 - general purpose I/O 43 (default)
		01	Reserved
		10 or 11	XA3 - External interface (XINTF) address line 3 (O)
21:20	GPIO42		Configure this pin as:
		00	GPIO 42 - general purpose I/O 42 (default)
		01	Reserved
		10 or 11	XA2 - External interface (XINTF) address line 2 (O)
19:18	GPIO41		Configure this pin as:
		00	GPIO 41 - general purpose I/O 41 (default)
		01	Reserved
		10 or 11	XA1 - External interface (XINTF) address line 1 (O)
17:16	GPIO40		Configure this pin as:
		00	GPIO 40 - general purpose I/O 40 (default)
		01	Reserved
		10 or 11	XA0/$\overline{\text{XWE1}}$ - External interface (XINTF) address line 1 or external interface write enable strobe 1 (O)
15:14	GPIO39		Configure this pin as:
		00	GPIO 39 - general purpose I/O 39 (default)
		01	Reserved
		10 or 11	XA16 - External interface (XINTF) address line 16 (O)
13:12	GPIO38		Configure this pin as:
		00	GPIO 38 - general purpose I/O 38 (default)
		01	Reserved
		10 or 11	$\overline{\text{XWE0}}$ - External interface write enable strobe 0
11:10	GPIO37		Configure this pin as:
		00	GPIO 37 - general purpose I/O 37 (default)
		01	ECAP2 - Enhanced capture input/output 2 (I/O)
		10 or 11	$\overline{\text{XZCS7}}$ - External interface zone 7 chip select (O)
9:8	GPIO36		Configure this pin as:
		00	GPIO 36 - general purpose I/O 36 (default)
		01	SCIRXDA - SCI-A receive data (I)
		10 or 11	$\overline{\text{XZCS0}}$ - External interface zone 0 chip select (O)
7:6	GPIO35		Configure this pin as:
		00	GPIO 35 - general purpose I/O 35 (default)
		01	SCITXDA - SCI-A transmit data (O)
		10 or 11	XR/$\overline{\text{W}}$ - Read Not Write Strobe. Normally held high. When low, XR/$\overline{\text{W}}$ indicates write cycle is active; when high, XR/$\overline{\text{W}}$ indicates read cycle is active.
5:4	GPIO34		Configure this pin as:
		00	GPIO 34 - general purpose I/O 34 (default)
		01	ECAPI - Enhanced capture input/output 1 (I/O)
		10 or 11	XREADY - External interface ready signal

Table 1-52. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions (continued)

Bit	Field	Value	Description
3:2	GPIO33		Configure this pin as:
		00	GPIO 33 - general purpose I/O 33 (default)
		01	SCLA - I2C 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	GPIO 32 - general purpose I/O 32 (default)
		01	SDAA - I2C 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 1-50. GPIO Port B MUX 2 (GPBMUX2) Register

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
GPIO63	GPIO62	GPIO61	GPIO60	GPIO59	GPIO58	GPIO57	GPIO56
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
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	GPIO49	GPIO48
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 1-53. GPIO Port B MUX 2 (GPBMUX2) Register Field Descriptions

Bit	Field	Value	Description
31:30	GPIO63		Configure this pin as:
		00	GPIO 63 - general purpose I/O 63 (default)
		01	SCITXDC - SCI-C transmit data (O)
		10 or 11	XD16 - External interface data line 16 (I/O)
29:28	GPIO62		Configure this pin as:
		00	GPIO 62 - general purpose I/O 62 (default)
		01	SCIRXDC - SCI-C receive data (I)
		10 or 11	XD17 - External interface data line 17 (I/O)
27:26	GPIO61		Configure this pin as:
		00	GPIO 61 - general purpose I/O 61 (default)
		01	MFSRB - McBSP-B receive frame synch (I/O)
		10 or 11	XD18 - External interface data line 18 (I/O)
25:24	GPIO60		Configure this pin as:
		00	GPIO 60 - general purpose I/O 60 (default)
		01	MCLKRB - McBSP-B receive clock (I/O)
		10 or 11	XD19 - External interface data line 19 (I/O)
23:22	GPIO59		Configure this pin as:
		00	GPIO 59 - general purpose I/O 59 (default)
		01	MFSRA - McBSP-A receive frame synch (I/O)
		10 or 11	XD20 - External interface data line 20 (I/O)

Table 1-53. GPIO Port B MUX 2 (GPBMUX2) Register Field Descriptions (continued)

Bit	Field	Value	Description
21:20	GPIO58		Configure this pin as:
		00	GPIO 58 - general purpose I/O 58 (default)
		01	MCLKRA - McBSP-A receive clock (I/O)
		10 or 11	XD21 - External interface data line 21 (I/O)
19:18	GPIO57		Configure this pin as:
		00	GPIO 57 - general purpose I/O 57 (default)
		01	$\overline{\text{SPISTEA}}$ - SPI-A slave transmit enable (I/O)
		10 or 11	XD22 - External interface data line 22 (I/O)
17:16	GPIO56		Configure this pin as:
		00	GPIO 56 - general purpose I/O 56 (default)
		01	SPICLKA - SPI-A clock input/output (I/O)
		10 or 11	XD23 - External interface data line 23 (I/O)
15:14	GPIO55		Configure this pin as:
		00	GPIO 55 - general purpose I/O 55 (default)
		01	SPISOMIA - SPI-A slave out, master in (I/O)
		10 or 11	XD24 - External interface data line 24 (I/O)
13:12	GPIO54		Configure this pin as:
		00	GPIO 54 - general purpose I/O 54 (default)
		01	SPISIMOA - SPI slave in, master out (I/O)
		10 or 11	XD25 - External interface data line 25 (I/O)
11:10	GPIO53		Configure this pin as:
		00	GPIO 53 - general purpose I/O 53 (default)
		01	EQEP1I - Enhanced QEP1 index (I/O)
		10 or 11	XD26 - External interface data line 26 (I/O)
9:8	GPIO52		Configure this pin as:
		00	GPIO 52 - general purpose I/O 52 (default)
		01	EQEP1S - Enhanced QEP1 strobe (I/O)
		10 or 11	XD27External interface data line 27 (I/O)
7:6	GPIO51		Configure this pin as:
		00	GPIO 51 - general purpose I/O 51 (default)
		01	EQEP1B - Enhanced QEP1 input B (I)
		10 or 11	XD28 - External interface data line 28 (I/O)
5:4	GPIO50		Configure this pin as:
		00	GPIO 50 - general purpose I/O 50 (default)
		01	EQEP1A - Enhanced QEP1 input A (I)
		10 or 11	XD29 - External interface data line 29 (I/O)
3:2	GPIO49		Configure this pin as:
		00	GPIO 49 - general purpose I/O 49 (default)
		01	ECAP6 - Enhanced Capture input/output 6 (I/O)
		10 or 11	XD30 - External interface data line 30 (I/O)

Table 1-53. GPIO Port B MUX 2 (GPBMUX2) Register Field Descriptions (continued)

Bit	Field	Value	Description
1:0	GPIO48		Configure this pin as:
		00	GPIO 48 - general purpose I/O 48 (default)
		01	ECAP5 - Enhanced Capture input/output 5 (I/O)
		10 or 11	XD31 - External interface data line 31 (I/O)

Figure 1-51. GPIO Port C MUX 1 (GPCMUX1) Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GPIO79		GPIO78		GPIO77		GPIO76		GPIO75		GPIO74		GPIO73		GPIO72	
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
GPIO71		GPIO70		GPIO69		GPIO68		GPIO67		GPIO66		GPIO65		GPIO64	
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 1-54. GPIO Port C MUX 1 (GPCMUX1) Register Field Descriptions

Bit	Field	Value	Description
31:30	GPIO79		Configure this pin as:
		00 or 01	GPIO 79 - general purpose I/O 79 (default)
		10 or 11	XD0 - External interface data line 0 (O)
29:28	GPIO78		Configure this pin as:
		00 or 01	GPIO 78 - general purpose I/O 78 (default)
		10 or 11	XD1 - External interface data line 1 (O)
27:26	GPIO77		Configure this pin as:
		00 or 01	GPIO 77 - general purpose I/O 77 (default)
		10 or 11	XD2 - External interface data line 2 (O)
25:24	GPIO76		Configure this pin as:
		00 or 01	GPIO 76 - general purpose I/O 76 (default)
		10 or 11	XD3 - External interface data line 3 (O)
23:22	GPIO75		Configure this pin as:
		00 or 01	GPIO 75 - general purpose I/O 75 (default)
		10 or 11	XD4 - External interface data line 4 (O)
21:20	GPIO74		Configure this pin as:
		00 or 01	GPIO 74 - general purpose I/O 74 (default)
		10 or 11	XD5 - External interface data line 5(O)
19:18	GPIO73		Configure this pin as:
		00 or 01	GPIO 73 - general purpose I/O 73 (default)
		10 or 11	XD6 - External interface data line 6 (O)
17:16	GPIO72		Configure this pin as:
		00 or 01	GPIO 72 - general purpose I/O 72 (default)
		10 or 11	XD7 - External interface data line 7 (O)
15:14	GPIO71		Configure this pin as:
		00 or 01	GPIO 71 - general purpose I/O 71 (default)
		10 or 11	XD8 - External interface data line 8 (O)
13:12	GPIO70		Configure this pin as:
		00 or 01	GPIO 70 - general purpose I/O 70 (default)
		10 or 11	XD9 - External interface data line 9 (O)

Table 1-54. GPIO Port C MUX 1 (GPCMUX1) Register Field Descriptions (continued)

Bit	Field	Value	Description
11:10	GPIO69		Configure this pin as:
		00 or 01	GPIO 69 - general purpose I/O 69 (default)
		10 or 11	XD10 - External interface data line 10 (O)
9:8	GPIO68		Configure this pin as:
		00 or 01	GPIO 68 - general purpose I/O 68 (default)
		10 or 11	XD11 - External interface data line 11 (O)
7:6	GPIO67		Configure this pin as:
		00 or 01	GPIO 67 - general purpose I/O 67 (default)
		10 or 11	XD12 - External interface data line 12 (O)
5:4	GPIO66		Configure this pin as:
		00 or 01	GPIO 66 - general purpose I/O 66 (default)
		10 or 11	XD13 - External interface data line 13 (O)
3:2	GPIO65		Configure this pin as:
		00 or 01	GPIO 65 - general purpose I/O 65 (default)
		10 or 11	XD14 - External interface data line 14 (O)
1:0	GPIO64		Configure this pin as:
		00 or 01	GPIO 64 - general purpose I/O 64 (default)
		10 or 11	XD15 - External interface data line 15 (O)

Figure 1-52. GPIO Port C MUX 2 (GPCMUX2) Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO87		GPIO86		GPIO85		GPIO84		GPIO83		GPIO82		GPIO81		GPIO80	
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 1-55. GPIO Port C MUX 2 (GPCMUX2) Register Field Descriptions

Bit	Field	Value	Description
31:16	Reserved		
15:14	GPIO87		Configure this pin as:
		00 or 01	GPIO 87 - general purpose I/O 87 (default)
		10 or 11	XA15 - External interface address line 15 (O)
13:12	GPIO86		Configure this pin as:
		00 or 01	GPIO 86 - general purpose I/O 86 (default)
		10 or 11	XA14 - External interface address line 14 (O)
11:10	GPIO85		Configure this pin as:
		00 or 01	GPIO 85 - general purpose I/O 85 (default)
		10 or 11	XA13 - External interface address line 13 (O)

Table 1-55. GPIO Port C MUX 2 (GPCMUX2) Register Field Descriptions (continued)

Bit	Field	Value	Description
9:8	GPIO84		Configure this pin as:
		00 or 01	GPIO 84 - general purpose I/O 84 (default)
		10 or 11	XA12 - External interface address line 12 (O)
7:6	GPIO83		Configure this pin as:
		00 or 01	GPIO 83 - general purpose I/O 83 (default)
		10 or 11	XA11 - External interface address line 11 (O)
5:4	GPIO82		Configure this pin as:
		00 or 01	GPIO 82 - general purpose I/O 82 (default)
		10 or 11	XA10 - External interface address line 10 (O)
3:2	GPIO81		Configure this pin as:
		00 or 01	GPIO 81 - general purpose I/O 81 (default)
		10 or 11	XA9 - External interface address line 9 (O)
1:0	GPIO80		Configure this pin as:
		00 or 01	GPIO 80 - general purpose I/O 80(default)
		10 or 11	XA8 - External interface address line 8 (O)

Figure 1-53. 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 3 or 6 samples. The sampling period is the amount of time between qualification samples relative to the period of SYSCLKOUT. The number of samples is specified in the GPxQSELn registers.

Table 1-56. 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 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
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 × $T_{SYSCLKOUT}$

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

Figure 1-54. 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 1-57. 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 GPIO63
		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 GPIO48 to GPIO55
		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 GPIO47
		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
		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 1-55. 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 1-58. 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 1-55.
		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 1-56. 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 1-59. 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 1-56.
		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 1-57. GPIO Port B Qualification Select 1 (GPBQSEL1) Register

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
GPIO47	GPIO46	GPIO45	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
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
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 1-60. GPIO Port B Qualification Select 1 (GPBQSEL1) Register Field Descriptions

Bits	Field	Value	Description [1]
31-0	GPIO47-GPIO32		Select input qualification type for GPIO32 to GPIO47. The input qualification of each GPIO input is controlled by two bits as shown in Figure 1-55.
		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 GPBCTRL register.
		10	Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPBCTRL 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 1-58. GPIO Port B Qualification Select 2 (GPBQSEL2) Register

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
GPIO63	GPIO62	GPIO61	GPIO60	GPIO59	GPIO58	GPIO57	GPIO56
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
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	GPIO49	GPIO48
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 1-61. GPIO Port B Qualification Select 2 (GPBQSEL2) Register Field Descriptions

Bits	Field	Value	Description [1]
31-0	GPIO63-GPIO48		Select input qualification type for GPIO48 to GPIO63. The input qualification of each GPIO input is controlled by two bits as shown in Figure 1-56.
		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 1-59. 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 1-62. 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 1-60. GPIO Port B Direction (GPBDIR) Register

31	30	29	28	27	26	25	24
GPIO63	GPIO62	GPIO61	GPIO60	GPIO59	GPIO58	GPIO57	GPIO56
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
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	GPIO49	GPIO48
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
GPIO47	GPIO46	GPIO45	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
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
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 1-63. GPIO Port B Direction (GPBDIR) Register Field Descriptions

Bits	Field	Value	Description [1]
31-0	GPIO63-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 1-61. GPIO Port C Direction (GPCDIR) Register

31							24
Reserved							
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
GPIO87	GPIO86	GPIO85	GPIO84	GPIO83	GPIO82	GPIO81	GPIO80
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
GPIO79	GPIO78	GPIO77	GPIO76	GPIO75	GPIO74	GPIO73	GPIO72
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
GPIO71	GPIO70	GPIO69	GPIO68	GPIO67	GPIO66	GPIO65	GPIO64
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 1-64. GPIO Port C Direction (GPCDIR) Register Field Descriptions

Bits	Field	Value	Description [1]
31-0	GPIO87-GPIO64		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.

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 1-62. 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 1-65. 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 1-63. GPIO Port B Pullup Disable (GPBPUD) Registers

31	30	29	28	27	26	25	24
GPIO63	GPIO62	GPIO61	GPIO60	GPIO59	GPIO58	GPIO57	GPIO56
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
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	GPIO49	GPIO48
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
GPIO47	GPIO46	GPIO45	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
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
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 1-66. GPIO Port B Internal Pullup Disable (GPBPUD) Register Field Descriptions

Bits	Field	Value	Description [1]
31-0	GPIO63-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.

Figure 1-64. GPIO Port C Pullup Disable (GPCPUD) Registers

31							24
Reserved							
R/W-0							
23	22	21	20	19	18	17	16
GPIO87	GPIO86	GPIO85	GPIO84	GPIO83	GPIO82	GPIO81	GPIO80
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
GPIO79	GPIO78	GPIO77	GPIO76	GPIO75	GPIO74	GPIO73	GPIO72
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
GPIO71	GPIO70	GPIO69	GPIO68	GPIO67	GPIO66	GPIO65	GPIO64
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 C Internal Pullup Disable (GPCPUD) Register Field Descriptions

Bits	Field	Value	Description [1]
31-0	GPIO87-GPIO64		Configure the internal pullup resister on the selected GPIO Port C pin. Each GPIO pin corresponds to one bit in this register.
		0	Enable the internal pullup on the specified pin.
		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 1-65. 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 1-68. 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 1-65.
		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 1-66. GPIO Port B Data (GPBDAT) Register

31	30	29	28	27	26	25	24
GPIO63	GPIO62	GPIO61	GPIO60	GPIO59	GPIO58	GPIO57	GPIO56
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
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	GPIO49	GPIO48
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
GPIO47	GPIO46	GPIO45	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
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
GPIO39	GPIO38	GPIO37	GPIO36	GPIO35	GPIO34	GPIO33	GPIO32
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 1-69. GPIO Port B Data (GPBDAT) Register Field Descriptions

Bit	Field	Value	Description
31-0	GPIO63-GPIO32		Each bit corresponds to one GPIO port B pin (GPIO32-GPIO63) as shown in Figure 1-66.
		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 1-67. GPIO Port C Data (GPCDAT) Register

31							24
Reserved							
R-0							
23	22	21	20	19	18	17	16
GPIO87	GPIO86	GPIO85	GPIO84	GPIO83	GPIO82	GPIO81	GPIO80
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
GPIO79	GPIO78	GPIO77	GPIO76	GPIO75	GPIO74	GPIO73	GPIO72
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
GPIO71	GPIO70	GPIO69	GPIO68	GPIO67	GPIO66	GPIO65	GPIO64
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 1-70. GPIO Port C Data (GPCDAT) Register Field Descriptions

Bit	Field	Value	Description
31-3	Reserved		Reserved
2-0	GPIO87-GPIO64		Each bit corresponds to one GPIO port B pin (GPIO64-GPIO87) as shown in Figure 1-67
		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 GPCMUX1 and GPCDIR 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 GPCMUX1 and GPCDIR registers; otherwise, the value is latched but not used to drive the pin.

Figure 1-68. 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 1-71. 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 1-68.
		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 1-72. 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 1-68.
		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 1-73. 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 1-68.
		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 1-69. GPIO Port B Set, Clear and Toggle (GPBSET, GPBCLEAR, GPBTOGGLE) Registers

31	30	29	28	27	26	25	24
GPIO63	GPIO62	GPIO61	GPIO60	GPIO59	GPIO58	GPIO57	GPIO56
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
GPIO55	GPIO54	GPIO53	GPIO52	GPIO51	GPIO50	GPIO49	GPIO48
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
GPIO47	GPIO46	GPIO45	GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
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
GPIO39	GPIO38	GPIO37	GPIO36	GPIO35	GPIO34	GPIO33	GPIO32
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

Table 1-74. GPIO Port B Set (GPBSET) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO63-GPIO32		Each GPIO port B pin (GPIO32-GPIO63) corresponds to one bit in this register as shown in Figure 1-69.
		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 1-75. GPIO Port B Clear (GPBCLEAR) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO63-GPIO32		Each GPIO port B pin (GPIO32-GPIO63) corresponds to one bit in this register as shown in Figure 1-69.
		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 1-76. GPIO Port B Toggle (GPBTOGGLE) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO63-GPIO32		Each GPIO port B pin (GPIO32-GPIO63) corresponds to one bit in this register as shown in Figure 1-69.
		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 1-70. GPIO Port C Set, Clear and Toggle (GPCSET, GPCCLEAR, GPCTOGGLE) Registers

31							24
Reserved							
R-0							
23	22	21	20	19	18	17	16
GPIO87	GPIO86	GPIO85	GPIO84	GPIO83	GPIO82	GPIO81	GPIO80
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
GPIO79	GPIO78	GPIO77	GPIO76	GPIO75	GPIO74	GPIO73	GPIO72
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
GPIO71	GPIO70	GPIO69	GPIO68	GPIO67	GPIO66	GPIO65	GPIO64
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 1-77. GPIO Port C Set (GPCSET) Register Field Descriptions

Bits	Field	Value	Description
31-24	Reserved		Reserved
23-0	GPIO87-GPIO64		Each GPIO port C pin (GPIO64-GPIO87) corresponds to one bit in this register as shown in Figure 1-70.
		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 1-78. GPIO Port C Clear (GPCCLEAR) Register Field Descriptions

Bits	Field	Value	Description
31-24	Reserved		Reserved
23-0	GPIO87-GPIO64		Each GPIO port C pin (GPIO64-GPIO87) corresponds to one bit in this register as shown in Figure 1-70.
		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 1-79. GPIO Port C Toggle (GPCTOGGLE) Register Field Descriptions

Bits	Field	Value	Description
31-24	Reserved		Reserved
23-0	GPIO87-GPIO64		Each GPIO port C pin (GPIO64-GPIO87) corresponds to one bit in this register as shown in Figure 1-70.
		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 1-71. GPIO XINTn, XNMI Interrupt Select (GPIOXINTnSEL, GPIOXNMISEL) 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 1-80. GPIO XINTn Interrupt Select (GPIOXINTnSEL)[1] Register Field Descriptions

Bits	Field	Value	Description [2]
15-5	Reserved		Reserved
4-0	GPIOXINTnSEL		Select the port A GPIO signal (GPIO0 - GPIO31) that will be used as the XINT1 or XINT2 interrupt source. In addition, you can configure the interrupt in the XINT1CR or XINT2CR registers described in Section 1.6.5. To use XINT2 as ADC start of conversion, enable it in the ADCTRL2 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 1-81. XINT1/XINT2 Interrupt Select and Configuration Registers

n	Interrupt	Interrupt Select Register	Configuration Register
1	XINT1	GPIOXINT1SEL	XINT1CR
2	XINT2	GPIOXINT2SEL	XINT2CR

Table 1-82. GPIO XINT3 - XINT7 Interrupt Select (GPIOXINTnSEL) Register Field Descriptions[1]

Bits	Field	Value	Description [2]
15-5	Reserved		Reserved
4-0	GPIOXINTnSEL		Select the port B GPIO signal (GPIO32 - GPIO63) that will be used as the XINTn interrupt source. In addition, you can configure the interrupt in the XINTnCR register described in Section 1.6.5.
		00000	Select the GPIO32 pin as the XINTn interrupt source (default)
		00001	Select the GPIO33 pin as the XINTn interrupt source
		. . .	. . .
		11110	Select the GPIO62 pin as the XINTn interrupt source
		11111	Select the GPIO63 pin as the XINTn interrupt source

(1) n = 3, 4, 5, 6, or 7

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

Table 1-83. XINT3 - XINT7 Interrupt Select and Configuration Registers

n	Interrupt	Interrupt Select Register	Configuration Register
3	XINT3	GPIOXINT3SEL	XINT3CR
4	XINT4	GPIOXINT4SEL	XINT4CR
5	XINT5	GPIOXINT5SEL	XINT5CR
6	XINT6	GPIOXINT6SEL	XINT6CR
7	XINT7	GPIOXINT7SEL	XINT7CR

Table 1-84. GPIO XNMI Interrupt Select (GPIOXNMISEL) Register Field Descriptions

Bits	Field	Value	Description [1]
15-5	Reserved		Reserved
4-0	GPIOSEL		Select which port A GPIO signal (GPIO0 - GPIO31) will be used as the XNMI interrupt source. In addition you can configure the interrupt in the XNMICR register described in Section 1.6.5.
		00000	Select the GPIO0 pin as the XNMI interrupt source (default)
		00001	Select the GPIO1 pin as the XNMI interrupt source
		. . .	. . .
		11110	Select the GPIO30 pin as the XNMI interrupt source
		11111	Select the GPIO31 pin as the XNMI interrupt source

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

Figure 1-72. 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 1-85. 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 1-72.
		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 section describes the peripheral frames. It also describes the device emulation registers.

1.5.1 Peripheral Frame Registers

The 2833x, 2823x devices contain 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 1-86.
- Peripheral Frame 1: These are peripherals that are mapped to the 32-bit peripheral bus. See Table 1-87.
- Peripheral Frame 2: These are peripherals that are mapped to the 16-bit peripheral bus. See Table 1-88.
- Peripheral Frame 3: McBSP registers are mapped to this. See Table 1-89.

Table 1-86. Peripheral Frame 0 Registers[1]

Name	Address Range	Size (×16)	Access Type[2]
Device Emulation Registers	0x00 0880 - 0x00 09FF	384	EALLOW protected
FLASH Registers[3]	0x00 0A80 - 0x00 0ADF	96	EALLOW protected
Code Security Module Registers	0x00 0AE0 - 0x00 0AEF	16	EALLOW protected
ADC registers (dual-mapped) (0 wait, read only)	0x00 0B00 - 0x00 0B1F	32	Not EALLOW protected
XINTF Registers	0x00 0B20 - 0x00 0B3F	32	Not EALLOW protected
CPU–TIMER0/1/2 Registers	0x00 0C00 - 0x00 0C3F	64	Not EALLOW protected
PIE Registers	0x00 0CE0 - 0x00 0CFF	32	Not EALLOW protected
PIE Vector Table	0x00 0D00 - 0x00 0DFF	256	EALLOW protected
DMA Registers	0x00 1000 - 0x00 11FF	512	EALLOW protected

(1) Registers in Frame 0 support 16-bit and 32-bit accesses.
(2) If registers are EALLOW protected, then writes cannot be performed until the EALLOW instruction is executed. The EDIS instruction disables writes to prevent stray code or pointers from corrupting register contents.
(3) The Flash Registers are also protected by the Code Security Module (CSM).

Table 1-87. Peripheral Frame 1 Registers

Name	Address Range	Size (x16)	Access Type[1]
eCANA Registers	0x6000 - 0x60FF	256	Some eCAN control registers (and selected bits in other eCAN control registers) are EALLOW-protected. eCAN control registers require 32-bit access.
eCANA Mailbox RAM	0x6100 - 0x61FF	256	Not EALLOW-protected
eCANB Registers	0x6200 - 0x62FF	256	Some eCAN control registers (and selected bits in other eCAN control registers) are EALLOW-protected. eCAN control registers require 32-bit access.
eCANB Mailbox RAM	0x6300 - 0x63FF	256	Not EALLOW-protected
ePWM1 Registers	0x6800 - 0x683F	64	Some ePWM registers are EALLOW-protected. See Section 1.5.2.
ePWM2 Registers	0x6840 - 0x687F	64	
ePWM3 Registers	0x6880 - 0x68BF	64	
ePWM4 Registers	0x68C0 - 0x68FF	64	
ePWM5 Registers	0x6900 - 0x693F	64	
ePWM6 Registers	0x6940 - 0x697F	64	

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

Table 1-87. Peripheral Frame 1 Registers (continued)

Name	Address Range	Size (x16)	Access Type[1]
eCAP1 Registers	0x6A00 - 0x6A1F	32	Not EALLOW-protected
eCAP2 Registers	0x6A20 - 0x6A3F	32	
eCAP3 Registers	0x6A40 - 0x6A5F	32	
eCAP4 Registers	0x6A60 - 0x6A7F	32	
eCAP5 Registers	0x6A80 - 0x6A9F	32	
eCAP6 Registers	0x6AA0 - 0x6ABF	32	
eQEP1 Registers	0x6B00 - 0x6B3F	64	
eQEP2 Registers	0x6B40 - 0x6B7F	64	
GPIO Control Registers	0x6F80 - 0x6FBF	128	EALLOW-protected
GPIO Data Registers	0x6FC0 - 0x6FDF	32	Not EALLOW-protected
GPIO Interrupt and LPM Select Registers	0x6FE0 - 0x6FFF	32	EALLOW-protected

Table 1-88. Peripheral Frame 2 Registers

Name	Address Range	Size (x16)	Access Type[1]
System Control Registers	0x7010 - 0x702F	32	EALLOW-protected
SPI-A Registers	0x7040 - 0x704F	16	Not EALLOW-protected
SCI-A Registers	0x7050 - 0x705F	16	Not EALLOW-protected
External Interrupt Registers	0x7070 - 0x707F	32	Not EALLOW-protected
ADC Registers	0x7100 - 0x711F	32	Not EALLOW-protected
SCI-B Registers	0x7750 - 0x775F	16	Not EALLOW-protected
SCI-C Registers	0x7770 - 0x777F	16	Not EALLOW-protected
I2C Registers	0x7900 - 0x793F	64	Not EALLOW-protected

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

Table 1-89. Peripheral Frame 3 Registers

Name	Address Range	Size (×16)	Access Type
McBSP-A Registers	0x5000 - 0x503F	64	Not EALLOW-protected
McBSP-B Registers	0x5040 - 0x507F	64	Not EALLOW-protected
EPWM1 + HRPWM1 (DMA)[1]	0x5800 - 0x583F	64	Some registers are EALLOW-protected. Refer to the device datasheet for details.
EPWM2 + HRPWM2 (DMA)	0x5840 - 0x587F	64	
EPWM3 + HRPWM3 (DMA)	0x5880 - 0x58BF	64	
EPWM4 + HRPWM4 (DMA)	0x58C0 - 0x58FF	64	
EPWM5 + HRPWM5 (DMA)	0x5900 - 0x593F	64	
EPWM6 + HRPWM6 (DMA)	0x5940 - 0x597F	64	

(1) The ePWM/HRPWM modules can be re-mapped to Peripheral Frame 3 where they can be accessed by the DMA module. To achieve this, bit 0 (MAPEPWM) of MAPCNF register (address 0x702E) must be set to 1. This register is EALLOW protected. When this bit is 0, the ePWM/HRPWM modules are mapped to Peripheral Frame 1.

Figure 1-73. MAPCNF Register (0x702E)

31	1	0
Reserved		MAPEPWM
		R/W-0

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

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 1-90.

Table 1-90. 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
- Certain eCAN Registers
- XINTF Registers

Table 1-91. EALLOW-Protected Device Emulation Registers

Name	Address	Size (x16)	Description
DEVICECNF	0x0880 0x0881	2	Device Configuration Register
PROTSTART	0x0884	1	Block Protection Start Address Register
PROTRANGE	0x0885	1	Block Protection Range Address Register

Table 1-92. 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 1-93. 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 1-94. 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	External Interrupt 13 (XINT13) or CPU-Timer 1 (for RTOS use)
INT14	0x0D1C	2	CPU-Timer 2 (for RTOS use)
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 1-95. EALLOW-Protected PLL, Clocking, Watchdog, and Low-Power Mode Registers

Name	Address	Size (x16)	Description
PLLSTS	0x7011	1	PLL Status Register
HISPCP	0x701A	1	High-Speed Peripheral Clock Prescaler Register for HSPCLK Clock
LOSPCP	0x701B	1	Low-Speed Peripheral Clock Prescaler Register for HSPCLK Clock
PCLKCR0	0x701C	1	Peripheral Clock Control Register 0
PCLKCR1	0x701D	1	Peripheral Clock Control Register 1
LPMCR0	0x701E	1	Low Power Mode Control Register 0
PCLKCR3	0x7020	1	Peripheral Clock Control Register 3
PLLCR	0x7021	1	PLL Control Register
SCSR	0x7022	1	System Control and Status Register
WDCNTR	0x7023	1	Watchdog Counter Register
WDKEY	0x7025	1	Watchdog Reset Key Register
WDCR	0x7029	1	Watchdog Control Register

Table 1-96. EALLOW-Protected GPIO MUX Registers

Name	Address	Size (x16)	Description
GPACTRL	0x6F80	2	GPIO A Control Register (GPIO0 to GPIO31)
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register (GPIO0 to GPIO15)
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register (GPIO16 to GPIO31)
GPAMUX1	0x6F86	2	GPIO A Mux 1 Register (GPIO0 to GPIO15)
GPAMUX2	0x6F88	2	GPIO A Mux 2 Register (GPIO16 to GPIO31)
GPADIR	0x6F8A	2	GPIO A Direction Register (GPIO0 to GPIO31)
GPAPUD	0x6F8C	2	GPIO A Pull Up Disable Register (GPIO0 to GPIO31)
GPBCTRL	0x6F90	2	GPIO B Control Register (GPIO32 to GPIO35)
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register (GPIO32 to GPIO35)
GPBQSEL2	0x6F94	2	Reserved
GPBMUX1	0x6F96	2	GPIO B Mux 1 Register (GPIO32 to GPIO35)
GPBMUX2	0x6F98	2	Reserved
GPBDIR	0x6F9A	2	GPIO B Direction Register (GPIO32 to GPIO35)
GPBPUD	0x6F9C	2	GPIO B Pull Up Disable Register (GPIO32 to GPIO35)
GPCMUX1	0x6FA6	2	GPIO C Mux 1 Register (GPIO64 to 79)
GPCMUX2	0x6FA8	2	GPIO C Mux 2 Register (GPIO80 to 87)
GPCDIR	0x6FAA	2	GPIO C Direction Register (GPIO64 to 87)
GPCPUD	0x6FAC	2	GPIO C Pull Up Disable Register (GPIO64 to 87)
GPIOXINT1SEL	0x6FE0	1	XINT1 GPIO Input Select Register (GPIO0 to GPIO31)
GPIOXINT2SEL	0x6FE1	1	XINT2 GPIO Input Select Register (GPIO0 to GPIO31)
GPIOXNMISEL	0x6FE2	1	XNMI GPIO Input Select Register (GPIO0 to GPIO31)
GPIOXINT3SEL	0x6FE3	1	XINT3 GPIO Input Select Register (GPIO32 to GPIO63)
GPIOXINT4SEL	0x6FE4	1	XINT4 GPIO Input Select Register (GPIO32 to GPIO63)
GPIOXINT5SEL	0x6FE5	1	XINT5 GPIO Input Select Register (GPIO32 to GPIO63)
GPIOXINT6SEL	0x6FE6	1	XINT6 GPIO Input Select Register (GPIO32 to GPIO63)
GPIOXINT7SEL	0x6FE7	1	XINT7 GPIO Input Select Register (GPIO32 to GPIO63)
GPIOLPMSEL	0x6FE8	2	LPM GPIO Select Register (GPIO0 to GPIO31)

Table 1-97. EALLOW-Protected eCAN Registers

Name	eCAN-A Address	eCAN-B Address	Size (x16)	Description
CANMC	0x6014	0x6214	2	Master Control Register(1)
CANBTC	0x6016	0x6216	2	Bit Timing Configuration Register(2)
CANGIM	0x6020	0x6220	2	Global Interrupt Mask Register(3)
CANMIM	0x6024	0x6224	2	Mailbox Interrupt Mask Register
CANTSC	0x602E	0x622E	2	Time Stamp Counter
CANTIOC	0x602A	0x622A	1	I/O Control Register for CANTXA Pin(4)
CANRIOC	0x602C	0x622C	1	I/O Control Register for CANRXA Pin(5)

(1) Only bits CANMC[15-9] and [7-6] are protected
(2) Only bits BCR[23-16] and [10-0] are protected
(3) Only bits CANGIM[17-16] , [14-8], and [2-0] are protected
(4) Only IOCONT1[3] is protected
(5) Only IOCONT2[3] is protected

Table 1-98 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 1-98. EALLOW-Protected ePWM1 - ePWM6 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

Table 1-99. XINTF Registers

Name	Address	Size (x16)	Description (1)
XTIMING0	0x0000−0B20	2	XINTF Timing Register, Zone 0
XTIMING6(2)	0x0000−0B2C	2	XINTF Timing Register, Zone 6
XTIMING7	0x0000−0B2E	2	XINTF Timing Register, Zone 7
XINTCNF2(3)	0x0000−0B34	2	XINTF Configuration Register
XBANK	0x0000−0B38	1	XINTF Bank Control Register
XREVISION	0x0000−0B3A	1	XINTF Revision Register
XRESET	0x0000 083D	1	XINTF Reset Register

(1) All XINTF registers are EALLOW protected.
(2) XTIMING1 - XTIMING5 are reserved for future expansion and are not currently used.
(3) XINTCNF1 is reserved and not currently used.

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 1-100.

Table 1-100. Device Emulation Registers

Name	Address	Size (x16)	Description
DEVICECNF	0x0880 0x0881	2	Device Configuration Register
PARTID	0x380090 0x380090	1	Part ID Register
CLASSID	0x0882	1	Class ID Register
CLASSID	0x0882	1	Class ID Register
REVID	0x0883	1	Revision ID Register
PROTSTART	0x0884	1	Block Protection Start Address Register
PROTRANGE	0x0885	1	Block Protection Range Address Register

Figure 1-74. Device Configuration (DEVICECNF) Register

31	27	26	20	19	18	16
Reserved	TRST	Reserved		ENPROT	Reserved	
R-0	R-0	R-0		R/W-1	R-111	

15	5	4	3	2	0
Reserved	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 1-101. DEVICECNF Register Field Descriptions

Bits	Field	Value	Description
31-28	Reserved		Reserved
27	TRST		Read status of $\overline{\text{TRST}}$ signal. Reading this bit gives the current satus of the $\overline{\text{TRST}}$ signal.
		0	No emulator is connected.
		1	An emulator is connected.
26:20	Reserved		
19	ENPROT		Enable Write-Read Protection Mode Bit.
		0	Disables write-read protection mode
		1	Enables write-read protection as specified by the PROTSTART and PROTRANGE registers
18-6	Reserved		Reserved
5	XRS		Reset Input Signal Status. This is connected directly to the $\overline{\text{XRS}}$ input pin.
4	Reserved		Reserved
3	VMAPS		VMAP Configure Status. This indicates the status of VMAP.
2-0	Reserved		Reserved

Figure 1-75. Part ID Register

15	8	7	0
PARTTYPE		PARTNO	
R		R	

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

Table 1-102. PARTID Register Field Descriptions

Bit	Field	Value[1]	Description
15:8	PARTTYPE		These 8 bits specify the type of device such as flash-based.
		0x00	Flash-based device
			All other values are reserved.
7:0	PARTNO		These 8 bits specify the feature set of the device as follows:
		0x00EF	F28335
		0x00EE	F28334
		0x00ED	F28332
		0x00E8	F28235
		0x00E7	F28234
		0x00E6	F28232
			All other values are reserved or used by other devices.

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

Figure 1-76. CLASSID Register

15	8	7	0
PARTTYPE		CLASSNO	
R		R	

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

Table 1-103. CLASSID Register Description

Bits	Field	Value	Description
15-8	PARTTYPE		These 8 bits specify the type of device such as flash-based.
		0x00	Flash-based device
			All other values are reserved.
7-0	CLASSNO		These 16 bits specify the class for the particular part.
		0x00EF	F28335, F28334, F28332 TMS320F2833x Floating Point Class device
		0x00E8	F28235, F28234, F28232 TMS320F2823x Fixed Point Class device

Figure 1-77. REVID Register

15	0
REVID	
R	

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

Table 1-104. 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

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

Table 1-104. REVID Register Field Descriptions (continued)

Bits	Field	Value	Description
		0x0001	Silicon Revision 1 - TMS

1.5.4 Write-Followed-by-Read Protection

The PROTSTART and PROTRANGE registers set the memory address range for which CPU "write" followed by "read" operations are protected (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
```

Table 1-105. PROTSTART and PROTRANGE Registers

Name	Address	Size	Type	Reset	Description
PROTSTART	0x0884	16	R/W	0x0100[1]	The PROTSTART register sets the starting address relative to the 16 most significant bits of the processors lower 22-bit address reach. Hence, the smallest resolution is 64 words.
PROTRANGE	0x0885	16	R/W	0x00FF[1]	The PROTRANGE register sets the block size (from the starting address), starting with 64 words and incrementing by binary multiples (64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, ..., 2M).

(1) The default values of these registers on reset are selected to cover the Peripheral Frame 1, Peripheral Frame 2, Peripheral Frame 3, and XINTF Zone 1 areas of the memory map (address range 0x4000 to 0x8000).

Table 1-106. PROTSTART Valid Values

		Register Bits [1]															
Start Address	Register Value	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00 0000	0x0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00 0040	0x0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0x00 0080	0x0002	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0x00 00C0	0x0003	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0x3F FF00	0xFFFC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0x3F FF40	0xFFFD	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0x3F FF80	0xFFFE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0x3F FFC0	0xFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(1) The quickest way to calculate register value is to divide the desired block starting address by 64.

Table 1-107. PROTRANGE Valid Values

Block Size	Register Value	Register Bits [1]															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
64	0x0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0x0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
256	0x0003	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
512	0x0007	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
1K	0x000F	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
256K	0x0FFF	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
512K	0x1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1M	0x3FFF	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2M	0x7FFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4M	0xFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(1) Not all register values are valid. The PROTSTART address value must be a multiple of the range value. For example: if the block size is set to 4K, then the start address can only be at any 4K boundary.

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

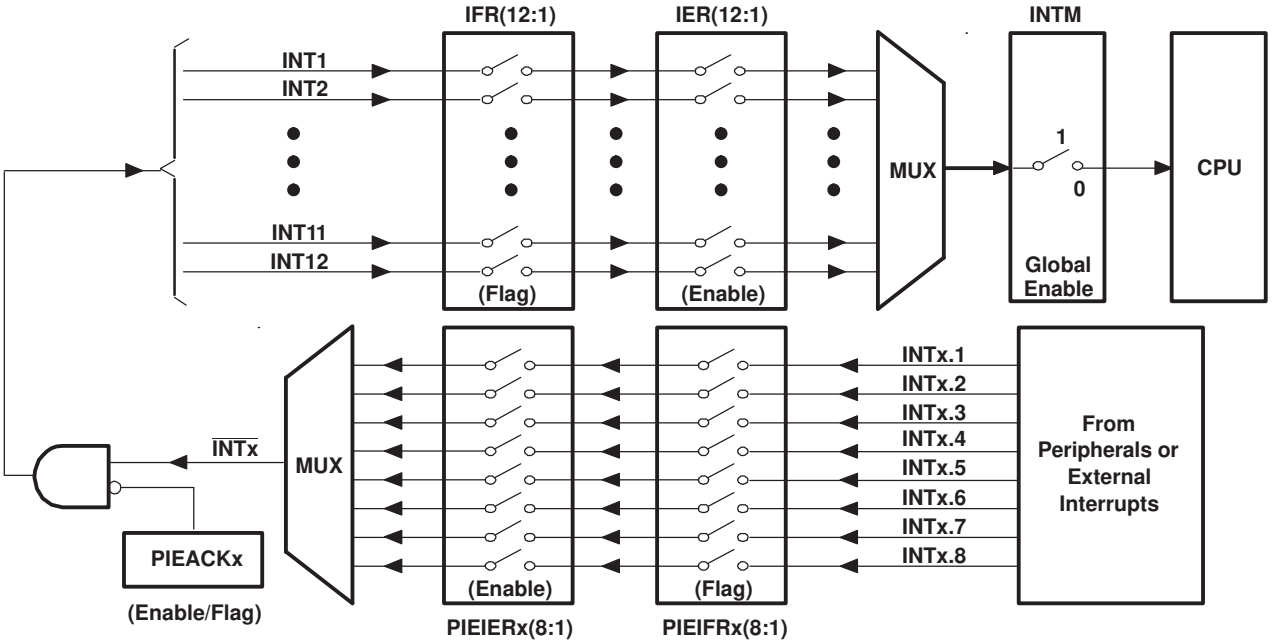
The 28x CPU supports one nonmaskable interrupt (NMI) and 16 maskable prioritized interrupt requests (INT1-INT14, RTOSINT, and DLOGINT) at the CPU level. The 28x devices have many peripherals and each peripheral is capable of generating one or more interrupts in response to many events at the peripheral 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 1-78 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 1-78. 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.

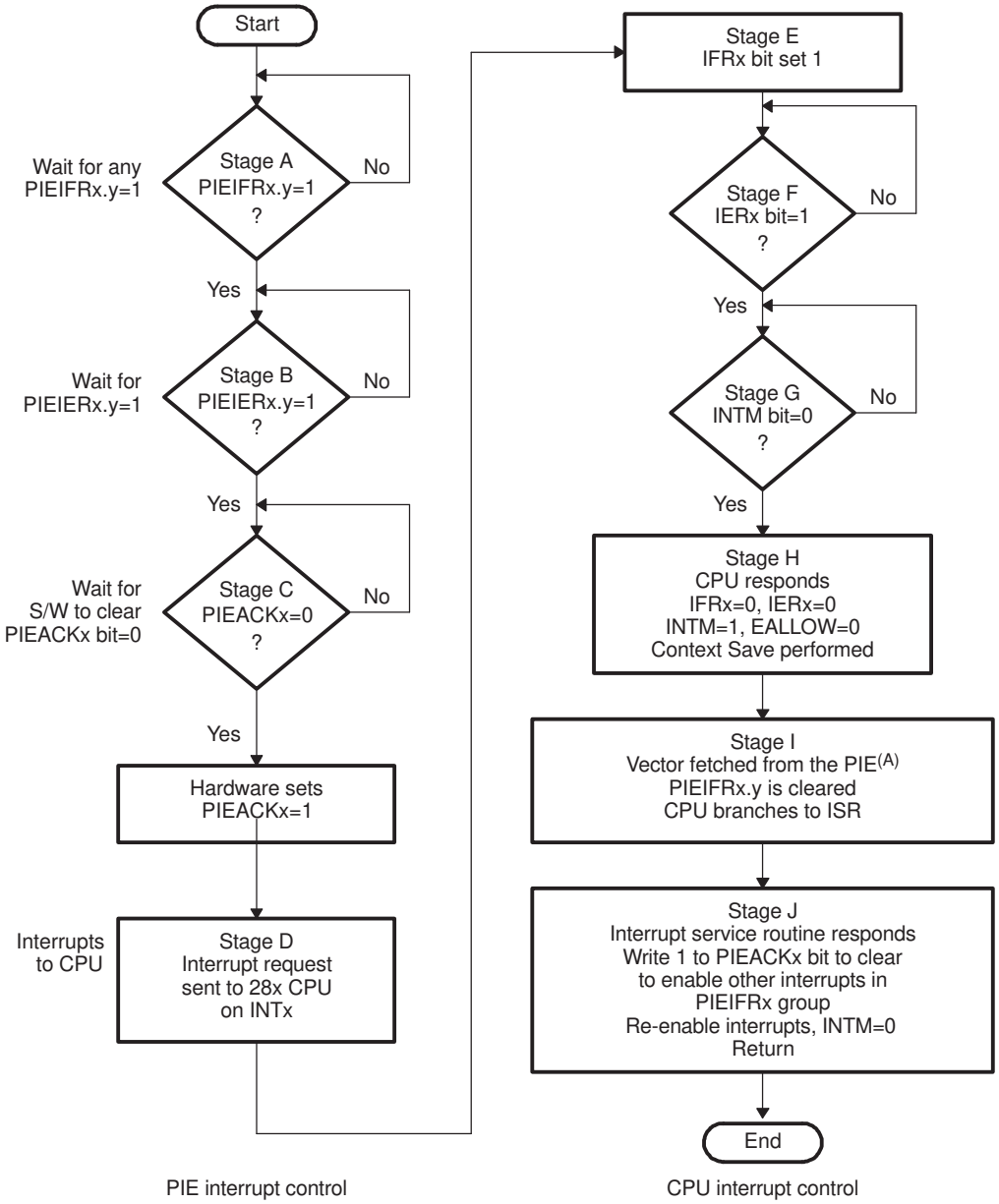
  For multiplexed interrupt sources, each interrupt group in the PIE block has an associated flag register (PIEIFRx) and enable (PIEIERx) register (x = PIE group 1 - PIE group 12). Each bit, referred to as y, corresponds to one of the 8 MUXed interrupts within the group. Thus PIEIFRx.y and PIEIERx.y correspond 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 1-79 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 1.6.3 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 1-79. Typical PIE/CPU Interrupt Response - INTx.y



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 1-108, 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 1-108. 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 *TMS320C28x 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 1-109.

Table 1-109. 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 1-110.

Table 1-110. Vector Table Mapping After Reset Operation

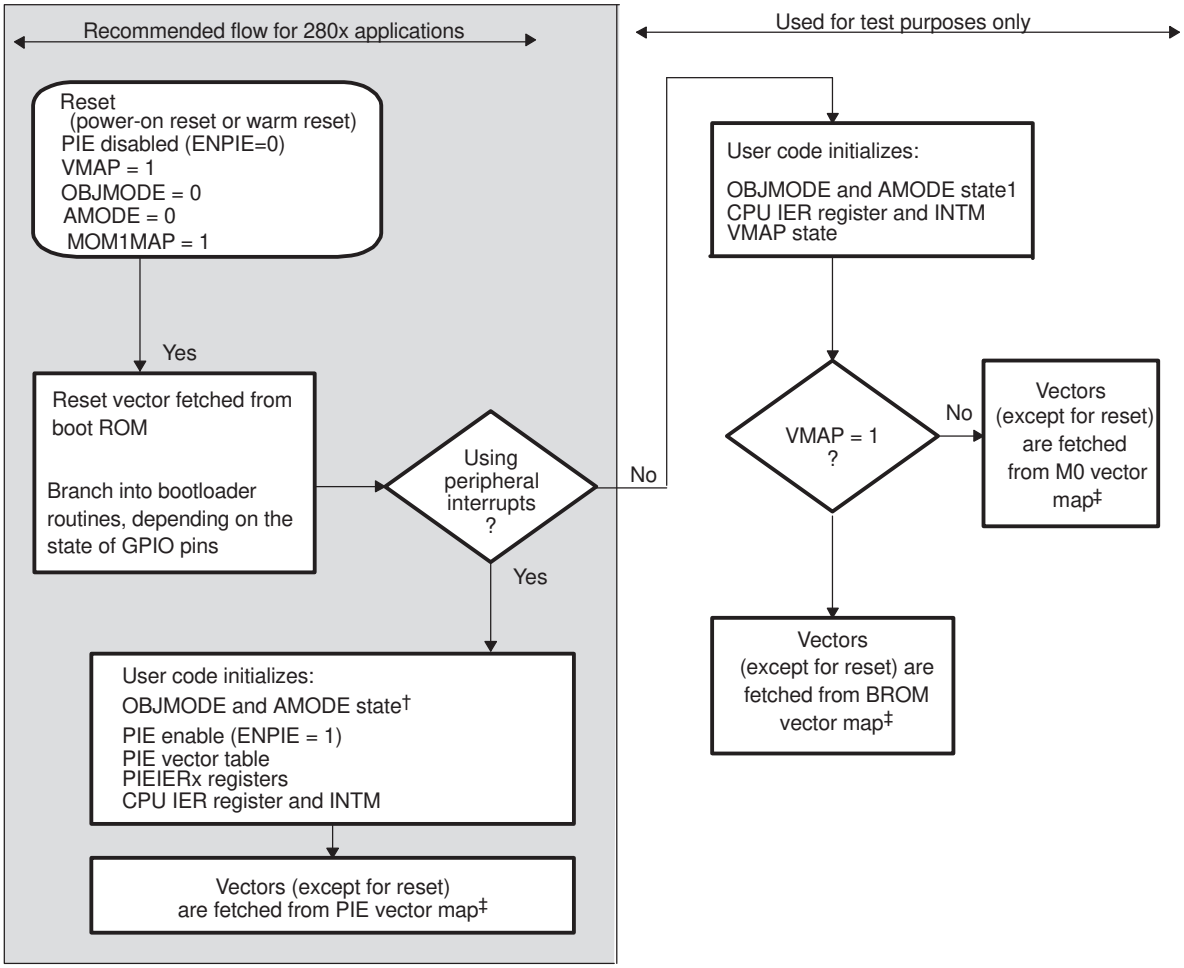
Vector MAPS	Reset Fetched From	Address Range	VMAP (1)	M0M1MAP (1)	ENPIE (1)
BROM Vector (2)	Boot ROM Block	0x3FFFC0 - 0x3FFFFF	1	1	0

(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 BROM as shown in Table 1-110. After a reset the PIE vector table is always disabled.

Figure 1-80 illustrates the process by which the vector table mapping is selected.

Figure 1-80. Reset Flow Diagram



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):

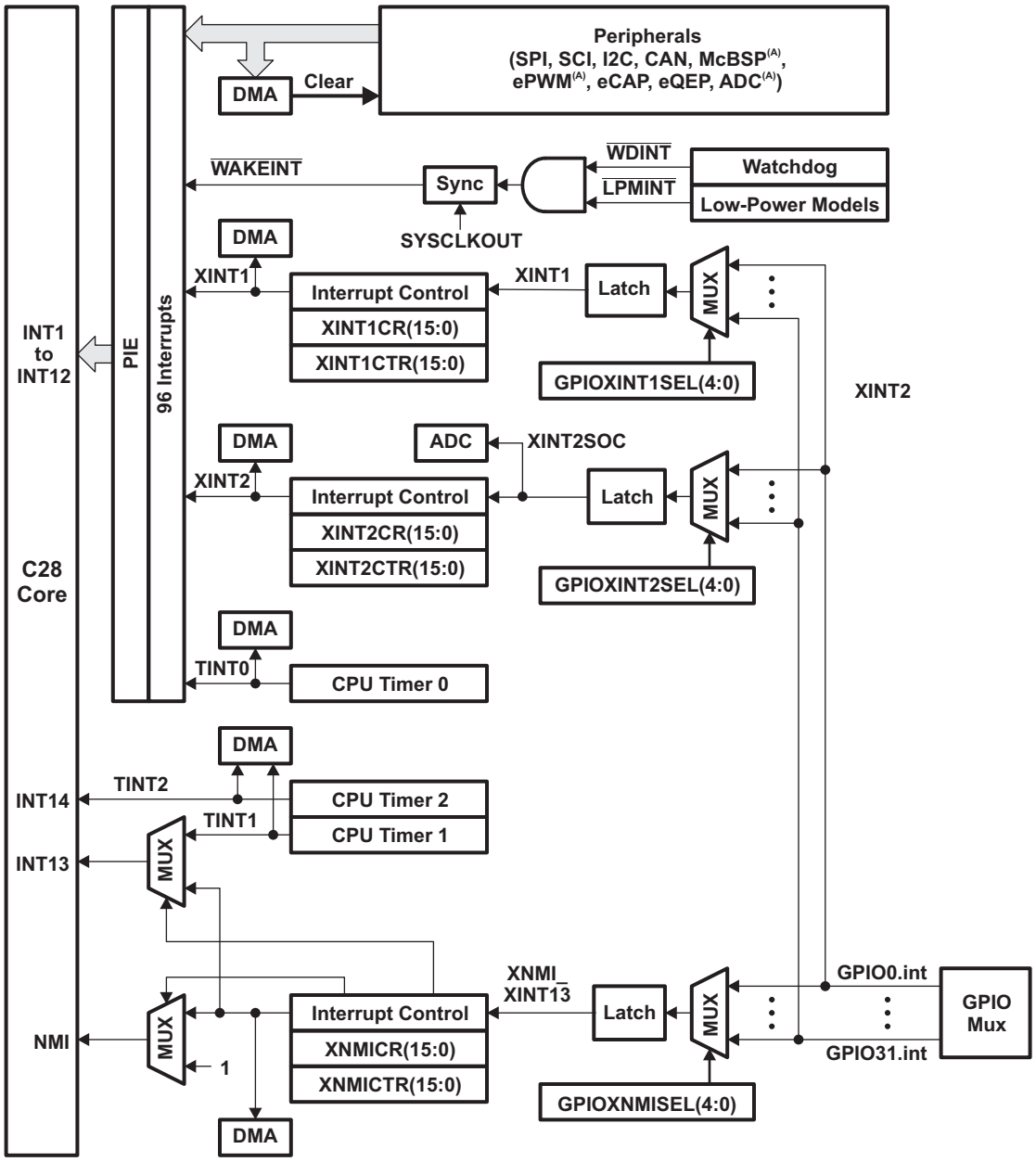
Operating Mode	OBJMODE	AMODE	
C28x Mode	1	0	
24x/240x Source-Compatible	1	1	
C27x Object-Compatible	0	0	(Default at reset)

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

1.6.3 Interrupt Sources

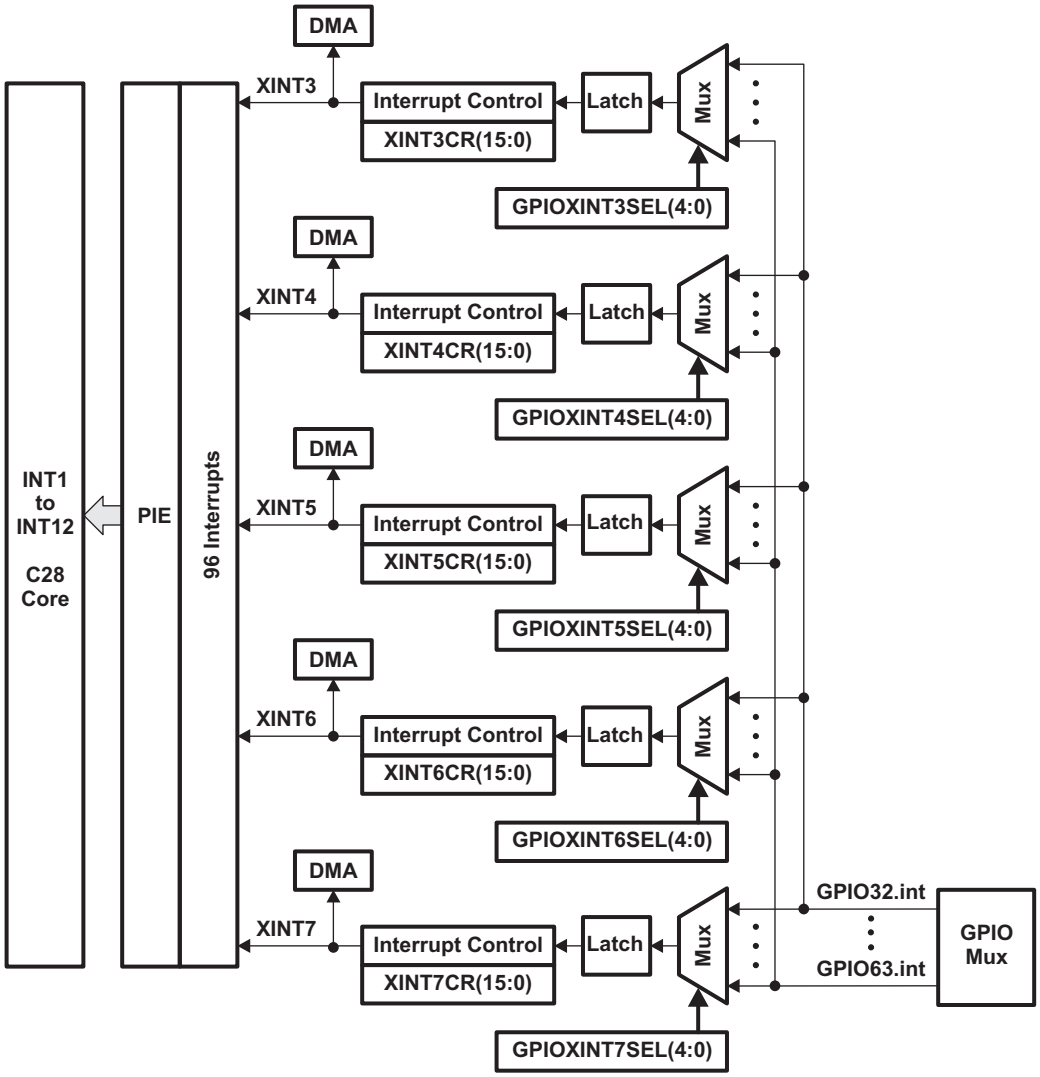
Figure 1-81 shows how the various interrupt sources are multiplexed within the devices. This multiplexing (MUX) scheme may not be exactly the same on all 28x devices. See the data manual of your particular device for details.

Figure 1-81. PIE Interrupt Sources and External Interrupts XINT1/XINT2



A DMA accessible

Figure 1-82. PIE Interrupt Sources and External Interrupts (XINT3 – XINT7)



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 C2000Ware .

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. C2000Ware 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.

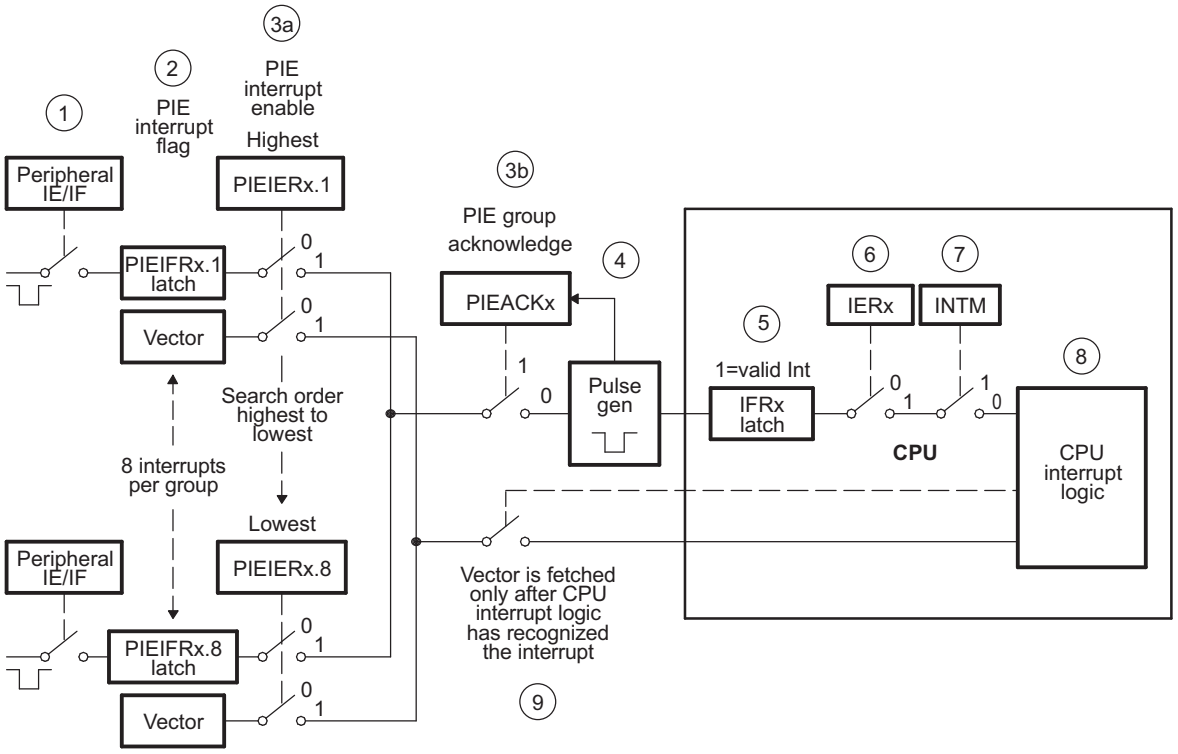
To perform a software reset of a peripheral interrupt and clear the associated flag in the PIEIFRx register and CPU IFR register, the following procedure should be followed:

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 1-83 shows the flow with the steps shown in circled numbers. Following the diagram, the steps are described.

Figure 1-83. 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:

1. The proper enable bit must be set (PIEIERx.y = 1) and
2. 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 1-112) 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 1-111, 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 1-111. 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 1-112.

Table 1-111. 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*	*ADCINT*	*XINT2*	*XINT1*	Reserved	*SEQ2INT*	*SEQ1INT*
	(LPM/WD)	(TIMER 0)	(ADC)	Ext. int. 2	Ext. int. 1	-	(ADC)	(ADC)
	0xD4E	0xD4C	0xD4A	0xD48	0xD46	0xD44	0xD42	0xD40
INT2.y	Reserved	Reserved	*EPWM6_ TZINT*	*EPWM5_TZINT*	*EPWM4_TZINT*	*EPWM3_TZINT*	*EPWM2_TZINT*	*EPWM1_TZINT*
	-	-	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD5E	0xD5C	0xD5A	0xD58	0xD56	0xD54	0xD52	0xD50
INT3.y	Reserved	Reserved	*EPWM6_ INT*	*EPWM5_INT*	*EPWM4_INT*	*EPWM3_INT*	*EPWM2_INT*	*EPWM1_INT*
	-	-	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD6E	0xD6C	0xD6A	0xD68	0xD66	0xD64	0xD62	0xD60
INT4.y	Reserved	Reserved	*ECAP6_INT*	*ECAP5_INT*	*ECAP4_INT*	*ECAP3_INT*	*ECAP2_INT*	*ECAP1_INT*
	-	-	(eCAP6)	(eCAP5)	(eCAP4)	(eCAP3)	(eCAP2)	(eCAP1)
	0xD7E	0xD7C	0xD7A	0xD78	0xD76	0xD74	0xD72	0xD70
INT5.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	*EQEP2_INT*	*EQEP1_INT*
	-	-	-	-	-	-	(eQEP2)	(eQEP1)
	0xD8E	0xD8C	0xD8A	0xD88	0xD86	0xD84	0xD82	0xD80
INT6.y	Reserved	Reserved	*MXINTA*	*MRINTA*	*MXINTB*	*MRINTB*	*SPITXINTA*	*SPIRXINTA*
	-	-	(McBSP-A)	(McBSP-A)	(McBSP-B)	(McBSP-B)	(SPI-A)	(SPI-A)
	0xD9E	0xD9C	0xD9A	0xD98	0xD96	0xD94	0xD92	0xD90
INT7.y	Reserved	Reserved	*DINTCH6*	*DINTCH5*	*DINTCH4*	*DINTCH3*	*DINTCH2*	*DINTCH1*
	-	-	(DMA6)	(DMA5)	(DMA4)	(DMA3)	(DMA2)	(DMA1)
	0xDAE	0xDAC	0xDAA	0xDA8	0xDA6	0xDA4	0xDA2	0xDA0
INT8.y	Reserved	Reserved	*SCITXINTC*	*SCIRXINTC*	Reserved	Reserved	*I2CINT2A*	*I2CINT1A*
	-	-	(SCI-C)	(SCI-C)	-	-	(I2C-A)	(I2C-A)
	0xDBE	0xDBC	0xDBA	0xDB8	0xDB6	0xDB4	0xDB2	0xDB0
INT9.y	*ECAN1INTB*	*ECAN0INTB*	*ECAN1INTA*	*ECAN0INTA*	*SCITXINTB*	*SCIRXINTB*	*SCITXINTA*	*SCIRXINTA*
	(CAN-B)	(CAN-B)	(CAN-A)	(CAN-A)	(SCI-B)	(SCI-B)	(SCI-A)	(SCI-A)
	0xDCE	0xDCC	0xDCA	0xDC8	0xDC6	0xDC4	0xDC2	0xDC0
INT10.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	-	-	-	-	-	-	-	-
	0xDDE	0xDDC	0xDDA	0xDD8	0xDD6	0xDD4	0xDD2	0xDD0
INT11.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	-	-	-	-	-	-	-	-
	0xDEE	0xDEC	0xDEA	0xDE8	0xDE6	0xDE4	0xDE2	0xDE0
INT12.y	*LUF*	*LVF*	Reserved	*XINT7*	*XINT6*	*XINT5*	*XINT4*	*XINT3*
	(FPU)	(FPU)	-	Ext. Int. 7	Ext. Int. 6	Ext. Int. 5	Ext. Int. 4	Ext. Int. 3
	0xDFE	0xDFC	0xDFA	0xDF8	0xDF6	0xDF4	0xDF2	0xDF0

Table 1-112. PIE Vector Table

Name	VECTOR ID[1]	Address[2]	Size (x16)	Description[3]		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)	-
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	SEQ1INT	(ADC)	5	1 (highest)
INT1.2	33	0x0000 0D42	2	SEQ2INT	(ADC)	5	2
INT1.3	34	0x0000 0D44	2	Reserved	-	5	3
INT1.4	35	0x0000 0D46	2	XINT1	Ext. Int. 1	5	4
INT1.5	36	0x0000 0D48	2	XINT2	Ext. Int. 2	5	5
INT1.6	37	0x0000 0D4A	2	ADCINT	(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)

(1) The VECTOR ID is used by DSP/BIOS.
(2) Reset is always fetched from location 0x003F FFC0 in Boot ROM.
(3) All the locations within the PIE vector table are EALLOW protected.

Table 1-112. PIE Vector Table (continued)

Name	VECTOR ID[1]	Address[2]	Size (x16)	Description[3]		CPU Priority	PIE Group Priority
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	Reserved	-	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	Reserved	-	7	7
INT3.8	55	0x0000 0D6E	2	Reserved	-	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
INT4.3	58	0x0000 0D74	2	ECAP3_INT	(eCAP3)	8	3
INT4.4	59	0x0000 0D76	2	ECAP4_INT	(eCAP4)	8	4
INT4.5	60	0x0000 0D78	2	ECAP5_INT	(eCAP5)	8	5
INT4.6	61	0x0000 0D7A	2	ECAP6_INT	(eCAP6)	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	Reserved	-	9	4
INT5.5	68	0x0000 0D88	2	Reserved	-	9	5
INT5.6	69	0x0000 0D8A	2	Reserved	-	9	6
INT5.7	70	0x0000 0D8C	2	Reserved	-	9	7
INT5.8	71	0x0000 0D8E	2	Reserved	-	9	8 (lowest)
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	MRINTB	(McBSP-B)	10	3
INT6.4	75	0x0000 0D96	2	MXINTB	(McBSP-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)

Table 1-112. PIE Vector Table (continued)

Name	VECTOR ID[1]	Address[2]	Size (x16)	Description[3]		CPU Priority	PIE Group Priority
PIE Group 7 Vectors - MUXed into CPU INT7							
INT7.1	80	0x0000 0DA0	2	DINTCH1	(DMA1)	11	1 (highest)
INT7.2	81	0x0000 0DA2	2	DINTCH2	(DMA2)	11	2
INT7.3	82	0x0000 0DA4	2	DINTCH3	(DMA3)	11	3
INT7.4	83	0x0000 0DA6	2	DINTCH4	(DMA4)	11	4
INT7.5	84	0x0000 0DA8	2	DINTCH5	(DMA5)	11	5
INT7.6	85	0x0000 0DAA	2	DINTCH6	(DMA6)	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	(I2C-A)	12	1 (highest)
INT8.2	89	0x0000 0DB2	2	I2CINT2A	(I2C-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	SCIRXINTC	(SCI-C)	12	5
INT8.6	93	0x0000 0DBA	2	SCITXINTC	(SCI-C)	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	SCITXINTB	(SCI-B)	13	4
INT9.5	100	0x0000 0DC8	2	ECAN0INTA	(eCAN-A)	13	5
INT9.6	101	0x0000 0DCA	2	ECAN1INTA	(eCAN-A)	13	6
INT9.7	102	0x0000 0DCC	2	ECAN0INTB	(eCAN-B)	13	7
INT9.8	103	0x0000 0DCE	2	ECAN1INTB	(eCAN-B)	13	8 (lowest)
PIE Group 10 Vectors - MUXed into CPU INT10							
INT10.1	104	0x0000 0DD0	2	Reserved	-	14	1 (highest)
INT10.2	105	0x0000 0DD2	2	Reserved	-	14	2
INT10.3	106	0x0000 0DD4	2	Reserved	-	14	3
INT10.4	107	0x0000 0DD6	2	Reserved	-	14	4
INT10.5	108	0x0000 0DD8	2	Reserved	-	14	5
INT10.6	109	0x0000 0DDA	2	Reserved	-	14	6
INT10.7	110	0x0000 0DDC	2	Reserved	-	14	7
INT10.8	111	0x0000 0DDE	2	Reserved	-	14	8 (lowest)
PIE Group 11 Vectors - MUXed into CPU INT11							
INT11.1	112	0x0000 0DE0	2	Reserved	-	15	1 (highest)
INT11.2	113	0x0000 0DE2	2	Reserved	-	15	2
INT11.3	114	0x0000 0DE4	2	Reserved	-	15	3
INT11.4	115	0x0000 0DE6	2	Reserved	-	15	4
INT11.5	116	0x0000 0DE8	2	Reserved	-	15	5
INT11.6	117	0x0000 0DEA	2	Reserved	-	15	6
INT11.7	118	0x0000 0DEC	2	Reserved	-	15	7
INT11.8	119	0x0000 0DEE	2	Reserved	-	15	8 (lowest)
PIE Group 12 Vectors - Muxed into CPU INT12							
INT12.1	120	0x0000 0DF0	2	XINT3	Ext. Int. 3	16	1 (highest)

Table 1-112. PIE Vector Table (continued)

Name	VECTOR ID[1]	Address[2]	Size (x16)	Description[3]		CPU Priority	PIE Group Priority
INT12.2	121	0x0000 0DF2	2	XINT4	Ext. Int. 4	16	2
INT12.3	122	0x0000 0DF4	2	XINT5	Ext. Int. 5	16	3
INT12.4	123	0x0000 0DF6	2	XINT6	Ext. Int. 6	16	4
INT12.5	124	0x0000 0DF8	2	XINT7	Ext. Int. 7	16	5
INT12.6	125	0x0000 0DFA	2	Reserved	-	16	6
INT12.7	126	0x0000 0DFC	2	LVF	(FPU)	16	7
INT12.8	127	0x0000 0DFE	2	LUF	(FPU)	16	8 (lowest)

1.6.4 PIE Configuration and Control Registers

The registers controlling the functionality of the PIE block are shown in Table 1-113.

Table 1-113. 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

1.6.4.1 PIE Control Register (PIECTRL)

Figure 1-84. PIE Control Register (PIECTRL) (Address CE0)

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

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

Table 1-114. PIE Control Register (PIECTRL) 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.

1.6.4.2 PIE Interrupt Acknowledge Register (PIEACK)

Figure 1-85. PIE Interrupt Acknowledge Register (PIEACK) (Address CE1)

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 1-115. 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 $\overline{\text{INT12}}$
		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 $\overline{\text{INT12}}$

1.6.4.3 PIE Interrupt Enable Registers

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

Figure 1-86. PIE Interrupt Enable Register (PIEIERx, 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 1-116. PIE Interrupt Enable Register (PIEIERx) 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.

1.6.4.4 PIE Interrupt Flag Registers

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

Figure 1-87. PIE Interrupt Flag Register (PIEIFRx, 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 1-117. PIE Interrupt Flag Register (PIEIFRx) Field Descriptions

Bits	Field	Description
15-8	Reserved	Reserved
7	INTx.8	These register bits indicate whether an interrupt is currently active. They behave very much like the CPU interrupt flag register. When an interrupt is active, the respective register bit is set. The bit is cleared when the interrupt is serviced or by writing a 0 to the register bit. This register can also be read to determine which interrupts are active or pending. x = 1 to 12. INTx means CPU INT1 to INT12
6	INTx.7	
5	INTx.6	
4	INTx.5	The PIEIFR register bit is cleared during the interrupt vector fetch portion of the interrupt processing.
3	INTx.4	Hardware has priority over CPU accesses to the PIEIFR registers.
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 1.6.3.1 for a method to clear flagged interrupts.

1.6.4.5 Interrupt Flag Register (IFR) — CPU Register

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 1-88. 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 1-118. 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

Table 1-118. Interrupt Flag Register (IFR) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
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
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

1.6.4.6 Interrupt Enable Register (IER) — CPU Register

The IER is a 16-bit CPU register. The IER contains enable bits for all the maskable CPU interrupt levels (INT1-INT14, RTOSINT and DLOGINT). Neither NMI nor XRS is included in the IER; thus, IER has no effect on these interrupts.

You can read the IER to identify enabled or disabled interrupt levels, and you can write to the IER to enable or disable interrupt levels. To enable an interrupt level, set its corresponding IER bit to one using the OR IER instruction. To disable an interrupt level, set its corresponding IER bit to zero using the AND IER instruction. When an interrupt is disabled, it is not acknowledged, regardless of the value of the INTM bit. When an interrupt is enabled, it is acknowledged if the corresponding IFR bit is one and the INTM bit is zero.

When using the OR IER and AND IER instructions to modify IER bits make sure they do not modify the state of bit 15 (RTOSINT) unless a real-time operating system is present.

When a hardware interrupt is serviced or an INTR instruction is executed, the corresponding IER bit is cleared automatically. When an interrupt is requested by the TRAP instruction the IER bit is not cleared automatically. In the case of the TRAP instruction if the bit needs to be cleared it must be done by the interrupt service routine.

At reset, all the IER bits are cleared to 0, disabling all maskable CPU level interrupts.

The IER register is shown in Figure 1-89, and descriptions of the bits follow the figure.

Figure 1-89. 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 1-119. 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	Real-time operating system interrupt is disabled
		1	Real-time operating system interrupt is enabled
14	DLOGINT		Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt.
		0	Data logging interrupt is disabled
		1	Data logging interrupt 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

Table 1-119. Interrupt Enable Register (IER) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
		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
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

1.6.4.7 Debug Interrupt Enable Register (DBGIER) — CPU Register

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 1-90. 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 1-120. 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	Real-time operating system interrupt is disabled
		1	Real-time operating system interrupt is enabled
14	DLOGINT	.	Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt
		0	Data logging interrupt is disabled
		1	Data logging interrupt 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 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

Table 1-120. Debug Interrupt Enable Register (DBGIER) — CPU Register Field Descriptions (continued)

Bits	Field	Value	Description
		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

1.6.5 External Interrupt Control Registers

Seven external interrupts, XINT1 –XINT7 are supported. XINT13 is multiplexed with one non-maskable interrupt XNMI. Each of these external interrupts can be selected for negative or positive edge triggered and can also be enabled or disabled (including XNMI). 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.

XINT1CR through XINT7CR are identical except for the interrupt number; therefore, Figure 1-91 and Table 1-121 represent registers for external interrupts 1 through 7 as XINT*n*CR where *n* = the interrupt number.

1.6.5.1 External Interrupt *n* Control Register (XINT*n*CR)

Figure 1-91. 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 1-121. 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 is 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

1.6.5.2 External NMI Interrupt Control Register (XNMICR) (Address 7077h)

Figure 1-92. External NMI Interrupt Control Register (XNMICR) — Address 7077h

15	4	3	2	1	0
Reserved		Polarity		Select	Enable
R-0		R/W-0		R-0	R/W-0

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

Table 1-122. External NMI Interrupt Control Register (XNMICR) 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 the 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 is 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	Select		Select the source for INT13
		0	Timer 1 connected To INT13
		1	XNMI_XINT13 connected To INT13
0	Enable		This read/write bit enables or disables external interrupt NMI
		0	Disable XNMI interrupt
		1	Enable XNMI interrupt

The XNMI Control Register (XNMICR) can be used to enable or disable the NMI interrupt to the CPU. In addition, you can select the source for the INT13 CPU interrupt. The source of the INT13 interrupt can be either the internal CPU Timer1 or the external GPIO signal assigned to XNMI.

The INT13 interrupt can be connected to XNMI_XINT13 for customer use.

Table 1-123 shows the relationship between the XNMICR Register settings and the interrupt sources to the 28x CPU.

Table 1-123. XNMICR Register Settings and Interrupt Sources

XNMICR	Register Bits	28x CPU Interrupt		Timestamp
ENABLE	SELECT	NMI Source	INT13 Source	(XNMICTR)
0	0	Disabled	CPU Timer 1	None
0	1	Disabled	XNMI	None
1	0	XNMI	CPU Timer 1	XNMI
1	1	Disabled	XNMI	XNMI

1.6.5.3 External Interrupt 1 Counter (XINT1CTR) (Address 7078h)

For XINT1 and XINT2, 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.

Figure 1-93. External Interrupt 1 Counter (XINT1CTR) (Address 7078h)

15	0
INTCTR[15-0]	
R-0	

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

Table 1-124. External Interrupt 1 Counter (XINT1CTR) 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.

1.6.5.4 External Interrupt 2 Counter (XINT2CTR) (Address 7079h)

For XINT1 and XINT2, 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.

Figure 1-94. External Interrupt 2 Counter (XINT2CTR) (Address 7079h)

15	0
INTCTR[15-0]	
R-0	

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

Table 1-125. External Interrupt 2 Counter (XINT2CTR) 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.

1.6.5.5 External NMI Interrupt Counter (XNMICTR) (Address 707Fh)

Figure 1-95. External NMI Interrupt Counter (XNMICTR) (Address 707Fh)

15	0
INTCTR[15-0]	
R-0	

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

Table 1-126. External NMI Interrupt Counter (XNMICTR) 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.

Boot ROM

The boot ROM is a block of read-only memory that is factory programmed. This chapter explains the boot procedure, the available boot modes, and the various details of the ROM code including memory maps, initializations, reset handling, and status information. The ROM source code is available under \libraries\boot_rom directory in C2000Ware.

Topic **Page**

2.1 Boot ROM Memory Map

The boot ROM is an 8K x 16 block of read-only memory located at addresses 0x3F E000 - 0x3F FFFF.

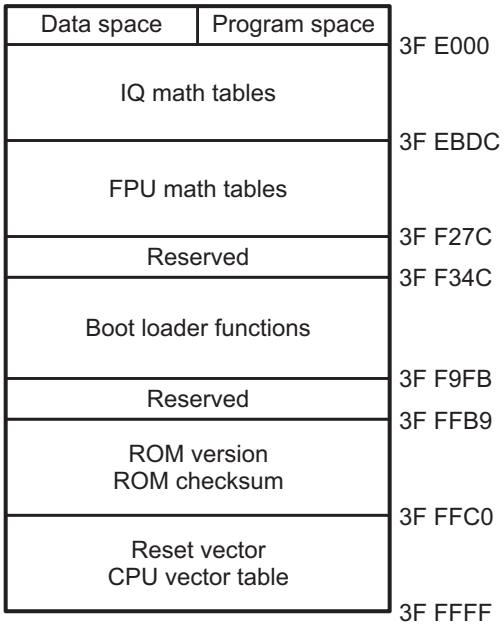
The on-chip boot ROM is factory programmed with bootload routines and math tables. These are for use with the C28x IQMath Library - A Virtual Floating Point Engine.

This document describes the following items:

- Bootloader functions
- Version number, release date and checksum
- Reset vector
- Illegal trap vector (ITRAP)
- CPU vector table (used for test purposes only)
- IQmath Tables
- Floating-point unit (FPU) math tables

Figure 2-1 shows the memory map of the on-chip boot ROM. The memory block is 8Kx16 in size and is located at 0x3F E000 - 0x3F FFFF in both program and data space.

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



2.1.1 On-Chip Boot ROM IQmath Tables

Approximately 4K of the boot ROM is reserved for floating-point and IQmath 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 C28x Floating Point Unit fastRTS Library. 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.

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

```
MEMORY
{
    PAGE 0 :
    ...
    FPUTABLES : origin = 0x3FEBDC, length = 0x0006A0
    ...
}
    SECTIONS
{
    ...
    FPUmathTables : > FPUTABLES, PAGE = 0, TYPE = NOLOAD
    ...
}
```

The following floating-point 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

The fixed-point math tables included in the boot ROM are used by the C28x IQMath Library - A Virtual Floating Point Engine). The 28x IQmath Library is a collection of highly optimized and high precision mathematical functions for C/C++ programmers to seamlessly port a floating-point algorithm into fixed-point code on 28x devices.

These routines are typically used in computationally-intensive, real-time applications where optimal execution speed and high accuracy is critical. By using these routines, you can achieve execution speeds that are considerably faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use high precision functions, the TI IQmath Library can significantly shorten the development time.

The IQmath library accesses the tables through the IQmathTables and the IQmathTablesRam linker sections. 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 lookup tables without actually loading the section to the target.

Example 2-2. Example 1: Linker Command File to Access IQ Tables

```
MEMORY
{
    PAGE 0 :
    ...
    IQTABLES : origin = 0x3FE000, length = 0x000b50
    IQTABLES2 : origin = 0x3FEB50, length = 0x00008c ...
}
SECTIONS
{
    ...
    IQmathTables : load = IQTABLES, type = NOLOAD, PAGE = 0
    IQmathTables2 > IQTABLES2, type = NOLOAD, PAGE = 0
    {
      IQmath.lib<IQNexpTable.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

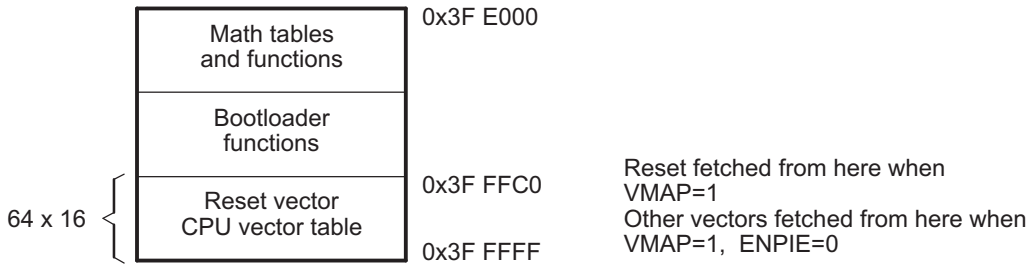
  This table is used as an initial estimate in the Arctan iterative algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.
- **Rounding and Saturation Table, IQ Math Table**
  - Table size: 360 words
  - Q format: Q30
  - Contents: 32-bit rounding and saturation 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

2.1.2 CPU Vector Table

A CPU vector table, Figure 2-2, resides in boot ROM memory from address 0x3F E000 - 0x3F FFFF. This vector table is active after reset when VMAP = 1, ENPIE = 0 (PIE vector table disabled).

Figure 2-2. Vector Table Map



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

(2) 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 bootload process. A series of checking operations is performed on select General-Purpose I/O (GPIO) pins to determine which boot mode to use. This boot mode selection is described in Section 2.2.9 of this document.

The remaining vectors in the boot ROM are not used during normal operation. After the boot process is complete, you should initialize the Peripheral Interrupt Expansion (PIE) vector table and enable the PIE block. From that point on, all vectors, except reset, will be fetched from the PIE module and not the CPU vector table shown in Table 2-1.

For TI silicon debug and test purposes the vectors located in the boot ROM memory point to locations in the M0 SARAM block as 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 (that is, points to)	Vector	Location in Boot ROM	Contents (that is, points to)
RESET	0x3F FFC0	InitBoot	RTOSINT	0x3F FFE0	0x00 0060
INT1	0x3F FFC2	0x00 0042	Reserved	0x3F FFE2	0x00 0062
INT2	0x3F FFC4	0x00 0044	NMI	0x3F FFE4	0x00 0064
INT3	0x3F FFC6	0x00 0046	ILLEGAL	0x3F FFE6	ITRAPIsr
INT4	0x3F FFC8	0x00 0048	USER1	0x3F FFE8	0x00 0068
INT5	0x3F FFCA	0x00 004A	USER2	0x3F FFEA	0x00 006A
INT6	0x3F FFCC	0x00 004C	USER3	0x3F FFEC	0x00 006C
INT7	0x3F FFCE	0x00 004E	USER4	0x3F FFEE	0x00 006E
INT8	0x3F FFD0	0x00 0050	USER5	0x3F FFF0	0x00 0070
INT9	0x3F FFD2	0x00 0052	USER6	0x3F FFF2	0x00 0072
INT10	0x3F FFD4	0x00 0054	USER7	0x3F FFF4	0x00 0074
INT11	0x3F FFD6	0x00 0056	USER8	0x3F FFF6	0x00 0076
INT12	0x3F FFD8	0x00 0058	USER9	0x3F FFF8	0x00 0078
INT13	0x3F FFDA	0x00 005A	USER10	0x3F FFFA	0x00 007A
INT14	0x3F FFDC	0x00 005C	USER11	0x3F FFFC	0x00 007C
DLOGINT	0x3F FFDE	0x00 005E	USER12	0x3F FFFE	0x00 007E

2.2 Bootloader Features

This section describes in detail the boot mode selection process, as well as the specifics of the bootloader operation.

2.2.1 Bootloader Functional Operation

The bootloader is the program located in the on-chip boot ROM that is executed following a reset.

The bootloader is used to transfer code from an external source into internal memory following power up. This allows the on-chip flash memory to be programmed via a serial port, for example.

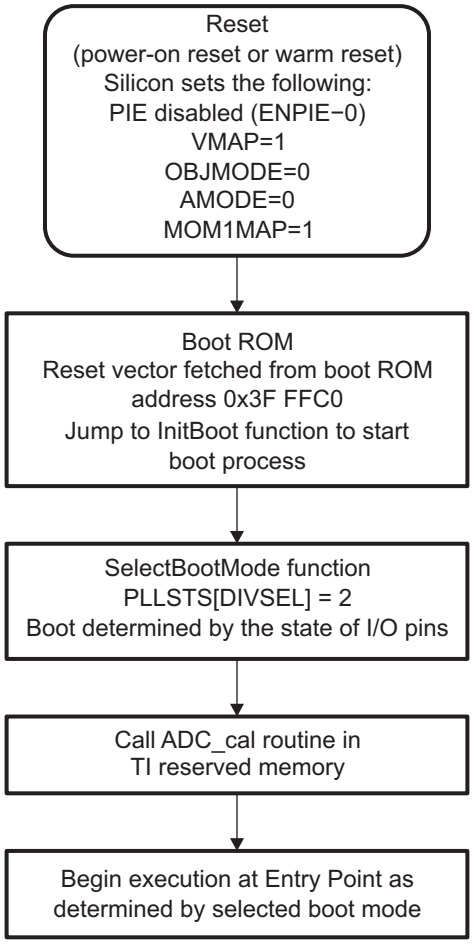
The bootloader provides a variety of different ways to download code to accommodate different system requirements. The bootloader uses various GPIO signals to determine which boot mode to use. The boot mode selection process as well as the specifics of each bootloader are described in the remainder of this document. Figure 2-3 shows the basic bootloader flow.

The reset vector in boot ROM redirects program execution to the InitBoot function. After performing device initialization the bootloader will check the state of GPIO pins to determine which boot mode you want to execute. Options include: jump to flash, jump to OTP, jump to SARAM, jump to XINTF, or call one of the on-chip boot loading routines.

After the selection process and if the required bootloading is complete, the processor will continue execution at an entry point determined by the boot mode selected. If a bootloader was called, then the input stream loaded by the peripheral determines this entry address. This data stream is described in Section 2.2.10. If, instead, you choose to boot directly to Flash, OTP, XINTF or SARAM, the entry address is predefined for each of these memory blocks.

The following sections discuss in detail the different boot modes available and the process used for loading data code into the device.

Figure 2-3. Bootloader Flow Diagram



2.2.2 Bootloader Device Configuration

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

When booting from the internal boot ROM, the device is configured for 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(1)

	C27x Mode (Reset)	28x Mode	C2xLP Source Compatible Mode
OBJMODE	0	1	1
AMODE	0	0	1
PAGE0	0	0	0
M0M1MAP(2)	1	1	1
Other Settings			SXM = 1, C = 1, SPM = 0

(1) C27x refers to the TMS320C27x family of processors. C2xLP refers to the TMS320F24x/TMS320LF240xA family of devices that incorporate the C2xLP core. The information in the table above is for reference only and is not applicable for the typical user development. For more information on the C2xLP core, refer to SPRU430.

(2) 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:

- **XINTF parallel loader:**
  - PLLCR and PLLSTS[DIVSEL] are specified by the user as part of the incoming data stream.
- **All other boot modes:**
  - PLLCR is not modified. PLLSTS[DIVSEL] is set to 2 for SYSCLKOUT = CLKIN/2 . 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.

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/2 to improve performance of the loaders. PLLSTS[DIVSEL] is left in this state when the boot ROM exits and it is up to the application to change it before configuring the PLLCR register.

2.2.4 Watchdog Module

When branching directly to Flash, OTP, or M0 single-access RAM (SARAM) or external interface (XINTF) 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.

2.2.5 Taking an ITRAP Interrupt

If an illegal opcode is fetched, the device will take an ITRAP (illegal trap) interrupt. During the boot process, the interrupt vector used by the ITRAP is within the CPU vector table of the boot ROM. The ITRAP vector points to an interrupt service routine (ISR) within the boot ROM named ITRAPIsr(). This interrupt service routine attempts to enable the watchdog and then loops forever until the processor is reset. This ISR will be used for any ITRAP until the user's application initializes and enables the peripheral interrupt expansion (PIE) block. Once the PIE is enabled, the ITRAP vector located within the PIE vector table will be used.

2.2.6 Internal Pullup Circuit

Each GPIO pin has an internal pullup circuit that can be enabled or disabled in software. The pins that are read by the boot mode selection code to determine the boot mode selection have pull-ups enabled after reset by default. In noisy conditions it is still recommended to configure each of the boot mode selection pins externally.

The peripheral bootloaders all enable the pullup circuit for the pins that are used for control and data transfer. The bootloader leaves the circuit enabled for these pins when it exits. For example, the SCI-A bootloader enables the pullup circuit on the SCITXA and SCIRXA pins. It is the user's responsibility to disable them, if desired, after the bootloader exits.

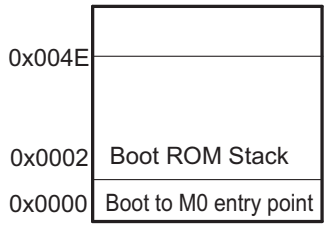
2.2.7 PIE Configuration

The boot modes do not enable the PIE. It is left in its default state, which is disabled.

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. This is a change for this boot loader.

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.

2.2.9 Bootloader Modes

To accommodate different system requirements, the boot ROM offers a variety of boot modes. This section describes the different boot modes and gives a brief summary of their functional operation. The states of four GPIO pins are used to determine the desired boot mode as shown in Table 2-3 and Figure 2-5.

Table 2-3. Boot Mode Selection(1)

MODE	GPIO87/XA15	GPIO86/XA14	GPIO85/XA13	GPIO84/XA12	MODE
F	1	1	1	1	Jump to Flash
E	1	1	1	0	SCI-A boot
D	1	1	0	1	SPI-A boot
C	1	1	0	0	I2C-A boot
B	1	0	1	1	eCAN-A boot
A	1	0	1	0	McBSP-A boot
9	1	0	0	1	Jump to XINTF x16
8	1	0	0	0	Jump to XINTF x32
7	0	1	1	1	Jump to OTP
6	0	1	1	0	Parallel GPIO I/O boot
5	0	1	0	1	Parallel XINTF boot
4	0	1	0	0	Jump to SARAM
3	0	0	1	1	Branch to check boot mode
2	0	0	1	0	Branch to Flash, skip ADC calibration
1	0	0	0	1	Branch to SARAM, skip ADC calibration
0	0	0	0	0	Branch to SCI, skip ADC calibration

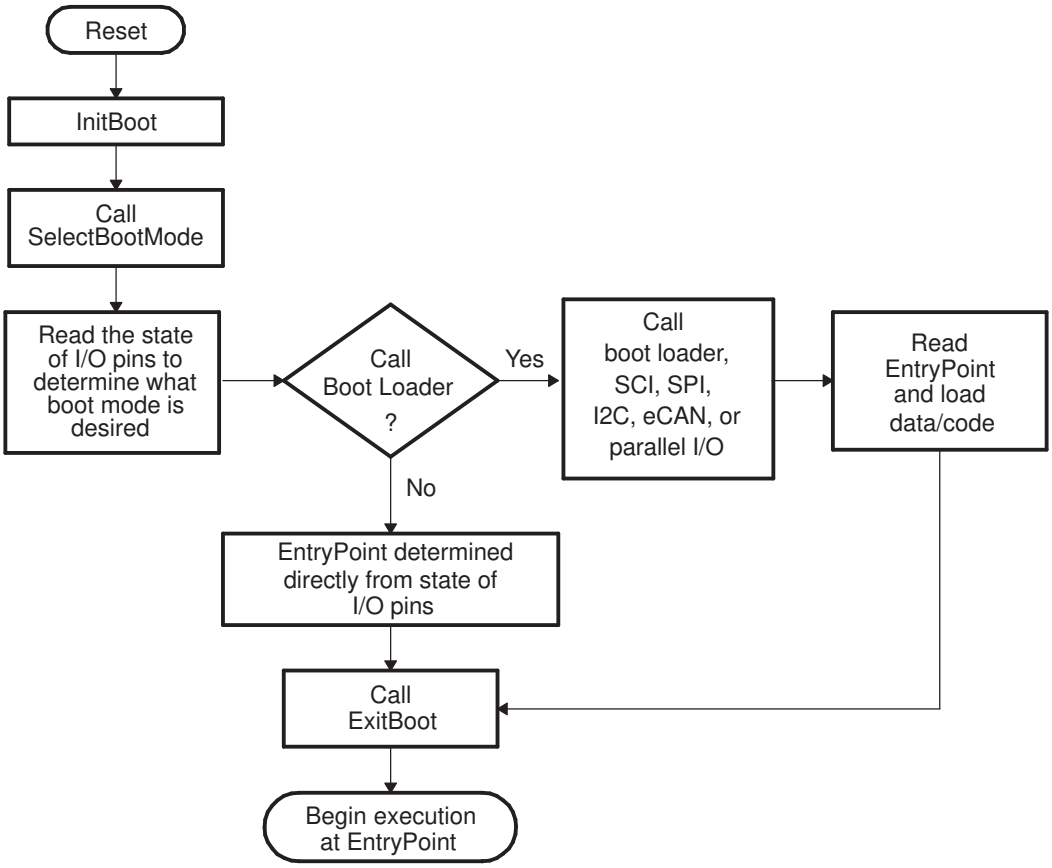
(1) All four GPIO pins have an internal pullup.

NOTE: Boot modes 0, 1, and 2 shown in Table 2-3 bypass the ADC calibration function call. These boot modes are for TI debug only.

The ADC calibration function initializes the ADCREFSEL and ADCOFFTRIM registers. If these registers are not properly initialized the ADC will operate out of specification. For more information on the ADC calibration function, refer to Section 7.2.5.1.

Figure 2-5 shows an overview of the boot process. Each step is described in greater detail in following sections.

Figure 2-5. Boot ROM Function Overview



The following boot mode is used for debug purposes:

- **Branch to check boot mode**

  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:

  – 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.

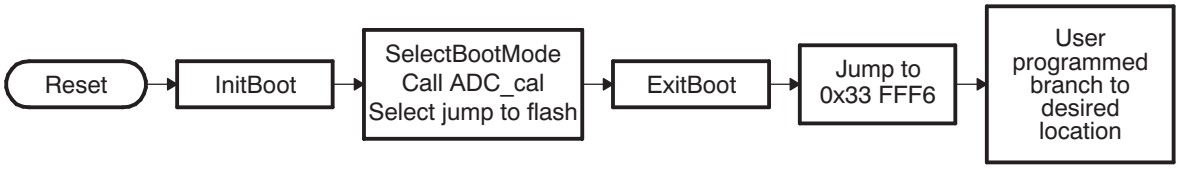
  – The second option is to use the “Branch to check boot mode” boot option. This will sit in a loop and continuously poll the boot mode select pins. The user can select this boot mode and then exit this mode once the emulator is connected by re-mapping the PC to another address or by changing the boot mode selection pin to the desired boot mode.

The following boot modes do not call a bootloader. Instead, they jump to a predefined location in memory:

- **Jump to branch instruction in flash memory**

  In this mode, the boot ROM software configures the device for 28x operation and branches directly to location 0x33 FFF6. 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 0x33 FFF6 that will redirect code execution to either a custom boot-loader or the application code.

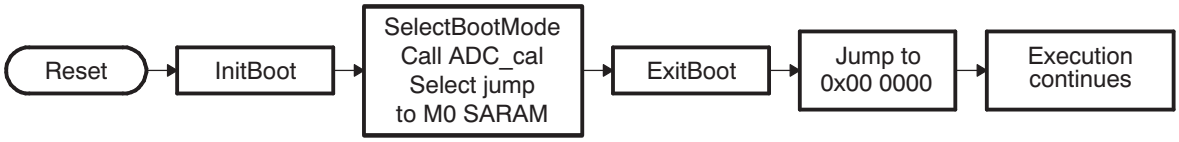
Figure 2-6. Jump-to-Flash Flow Diagram



- **Jump to M0 SARAM**

  In this mode, the boot ROM software configures the device for 28x operation and branches directly to 0x00 0000. This is the first address in the M0 SARAM memory block.

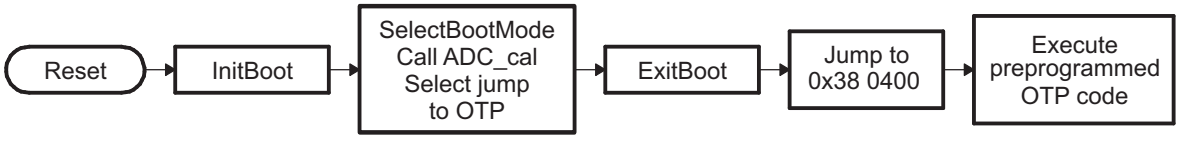
Figure 2-7. Flow Diagram of Jump to M0 SARAM



- **Jump to OTP memory**

  In this mode, the boot ROM software configures the device for 28x operation and branches to 0x38 0400. This is the first address in the OTP memory block.

Figure 2-8. Flow Diagram of Jump-to-OTP Memory



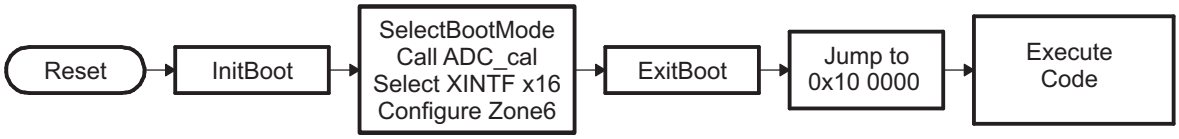
- **Jump to XINTF Zone 6 Configured for 16-bit Data**

  The boot ROM configures XINTF zone 6 for 16 bit wide memory, maximum wait states, and sample XREADY in asynchronous mode. This is the default values list after reset:

  – XTIMCLK = ½ SYSCLKOUT
  – XCLKOUT = ½ XTIMCLK
  – XRDLEAD = XWRLEAD = 3
  – XRDACTIVE = XWRACTIVE = 7
  – XRDTRAIL = XWRACTIVE = 3
  – XSIZE = 16-bit wide
  – X2TIMING = 1. Timing values are 2:1.
  – USEREADY = 1, READYMODE = 1 (XREADY sampled asynchronous mode)

  The boot ROM will then jump to the first location within zone 6 at address 0x10 0000.

Figure 2-9. Flow Diagram of Jump to XINTF x16

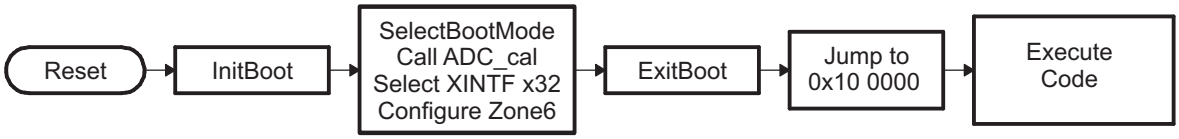


- **Jump to XINTF Zone 6 Configured for 32-bit Data**

  In this mode, the boot ROM configures XINTF zone 6 for 32-bit wide memory, maximum wait states, and sample XREADY in asynchronous mode. This is the default mode after reset except the bus width is x32. The boot ROM will then jump to the first location within zone 6 at address 0x10 0000.

  – XTIMCLK = ½ SYSCLKOUT
  – XCLKOUT = ½ XTIMCLK
  – XRDLEAD = XWRLEAD = 3
  – XRDACTIVE = XWRACTIVE = 7
  – XRDTRAIL = XWRACTIVE = 3
  – XSIZE = 32-bit wide
  – X2TIMING = 1. Timing values are 2:1.
  – USEREADY = 1, READYMODE = 1 (XREADY sampled asynchronous mode)

Figure 2-10. Flow Diagram of Jump to XINTF x32



The following boot modes call a boot load routine that loads a data stream from the peripheral into memory:

- **Standard serial boot mode (SCI-A)**

  In this mode, the boot ROM will load code to be executed into on-chip memory via the SCI-A port.

- **SPI EEPROM or Flash boot mode (SPI-A)**

  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)**

  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. The EEPROM must adhere to conventional I2C EEPROM protocol with a 16-bit base address architecture.
- **eCAN Boot Mode (eCAN-A)**

  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.
- **McBSP Boot Mode (McBSP-A)**

  Synchronously transfers code from McBSP-A to internal memory. McBSP-A is configured for slave mode operation, that is, it receives the frame sync and clock from the host. Upon receiving a word, the McBSP echoes the data back to the host.
- **Boot from GPIO Port (Parallel Boot from GPIO0-GPIO15)**

  In this mode, the boot ROM uses GPIO port A pins GPIO0-GPIO15 to load code and data from an external source. This mode supports both 8-bit and 16-bit data streams. Since this mode requires a number of GPIO pins, it is typically used to download code for flash programming when the device is connected to a platform explicitly for flash programming and not a target board.
- **Boot From XINTF (Parallel Boot From XD[15:0])**

  This mode is similar to the GPIO parallel boot mode except the boot ROM uses XINTF data lines XD[15:0] to load code and data from an external source instead of GPIO pins. This mode supports both 8-bit and 16-bit data streams. The user can specify the PLL configuration as well as XINTF timing through the input data stream.

2.2.10 Bootloader Data Stream Structure

Table 2-4 and Example 2-3 show the structure of the incoming data stream 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. In this case, the entry point for the flash memory (0x33 7FF6) will be used.

The next eight words are used to initialize register values or otherwise enhance the bootloader by passing values to it. If a bootloader does not use these values then they are reserved for future use and the bootloader simply reads the value and then discards it. Only the SPI and I2C and parallel XINTF bootloaders use these words to initialize registers.

The tenth and eleventh words comprise the 22-bit entry point address. This address is used to initialize the PC after the boot load is complete. This address is most likely the entry point of the program downloaded by the bootloader.

The twelfth word in the data stream is the size of the first data block to be transferred. The size of the block is defined for 8-bit and 16-bit data stream formats as the number of 16-bit words in the block. For example, to transfer a block of twenty 8-bit data values from an 8-bit data stream, the block size would be 0x000A to indicate ten 16-bit words.

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

This pattern of block size/destination address repeats for each block of data to be transferred. Once all the blocks have been transferred, a block size of 0x0000 signals to the loader that the transfer is complete. At this point the loader will return the entry point address to the calling routine which in turn will cleanup and exit. Execution will then continue at the entry point address as determined by the input data stream contents.

Table 2-4. General Structure of Source Program Data Stream in 16-Bit Mode

Word	Contents
1	10AA (KeyValue for memory width = 16 bits)
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	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+	Block size of 0000h - indicates end of the source program

Example 2-3. 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 2-5 and Example 2-4 show the structure of the incoming data stream to the bootloader.

Table 2-5. 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

Table 2-5. LSB/MSB Loading Sequence in 8-bit Data Stream (continued)

Byte		Contents	
		LSB (First Byte of 2)	MSB (Second Byte of 2)
.	.	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 2-4. Data Stream Structure 8-bit

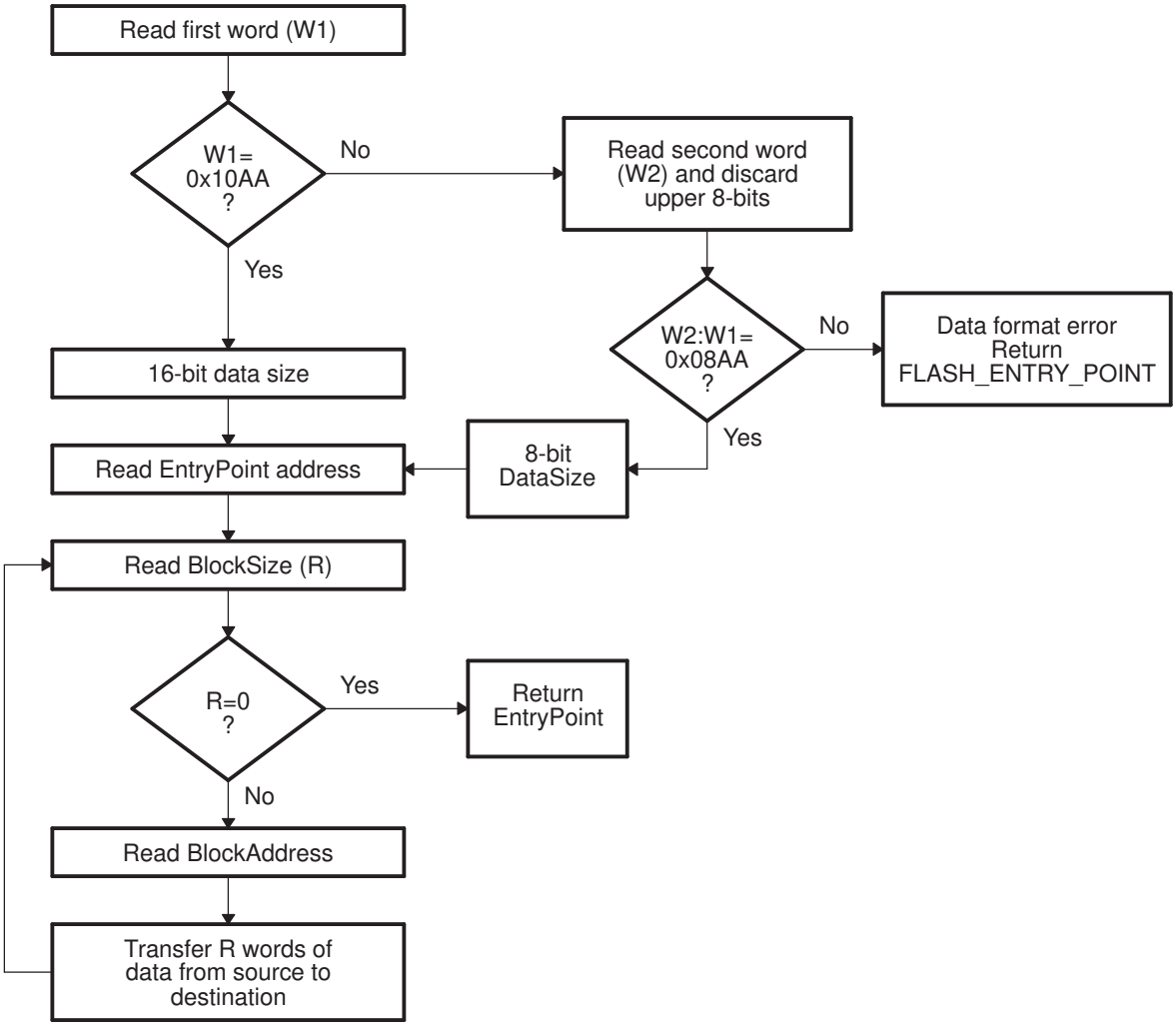
```
AA 08          ; 0x08AA 8-bit key value
00 00 00 00    ; 8 reserved words
00 00 00 00
00 00 00 00
00 00 00 00
3F 00 00 80    ; 0x003F8000 EntryAddr, starting point after boot load completes
05 00          ; 0x0005 - First block consists of 5 16-bit words
3F 00 10 90    ; 0x003F9010 - First block 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
```

2.2.11 Basic Transfer Procedure

Figure 2-11 illustrates the basic process a bootloader uses to transfer data and start program execution. This process occurs after the bootloader determines the valid boot mode selected by the state of the 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.

In this case, the loader will return the entry point address for the flash to the calling routine.

Figure 2-11. Bootloader Basic Transfer Procedure



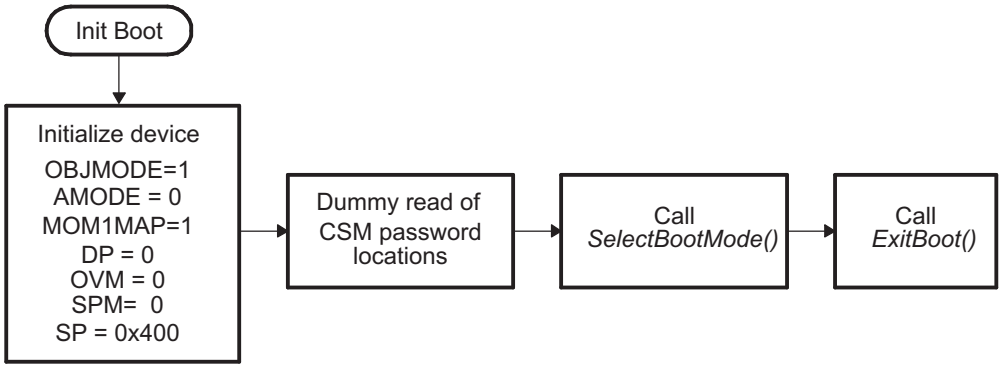
NOTE: See the info specific to a particular bootloader for any limitations. In 8-bit mode, the LSB of the 16-bit word is read first followed by the MSB.

2.2.12 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 certain GPIO pins. This process is described in Section 2.2.13. Once the boot is complete, the *SelectBootMode* function passes back the entry point address (*EntryAddr*) to the *InitBoot* function. The *EntryAddr* is the location where code execution will begin after the bootloader exits. *InitBoot* then calls the *ExitBoot* routine that then restores CPU registers to their reset state and exits to the *EntryAddr* that was determined by the boot mode.

Figure 2-12. Overview of InitBoot Assembly Function



2.2.13 SelectBootMode Function

To determine the desired boot mode, the *SelectBootMode* function examines the state of 4 GPIO pins as shown in Table 2-3.

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.

The *SelectBootMode* function checks the missing clock detect bit (MCLKSTS) in the PLLSTS register to determine if the PLL is operating in limp mode. If the PLL is operating in limp mode, the boot mode select function takes an appropriate action depending on the boot mode selected:

- **Boot to flash, OTP, SARAM, I2C-A, SPI-A, XINTF, and the parallel I/O**

  These modes behave as normal. The user's software must check for missing clock status and take the appropriate action if the MCLKSTS bit is set.
- **SCI-A boot**

  The SCI bootloader will be called. Depending on the requested baud rate, however, the device may not be able to autobaud lock. In this case the boot ROM software will loop in the autobaud lock function indefinitely. Should the SCI-A boot complete, the user's software must check for a missing clock status and take the appropriate action.
- **eCAN-A boot**

  The eCAN bootloader will not be called. Instead the boot ROM will loop indefinitely.
- **McBSP boot**

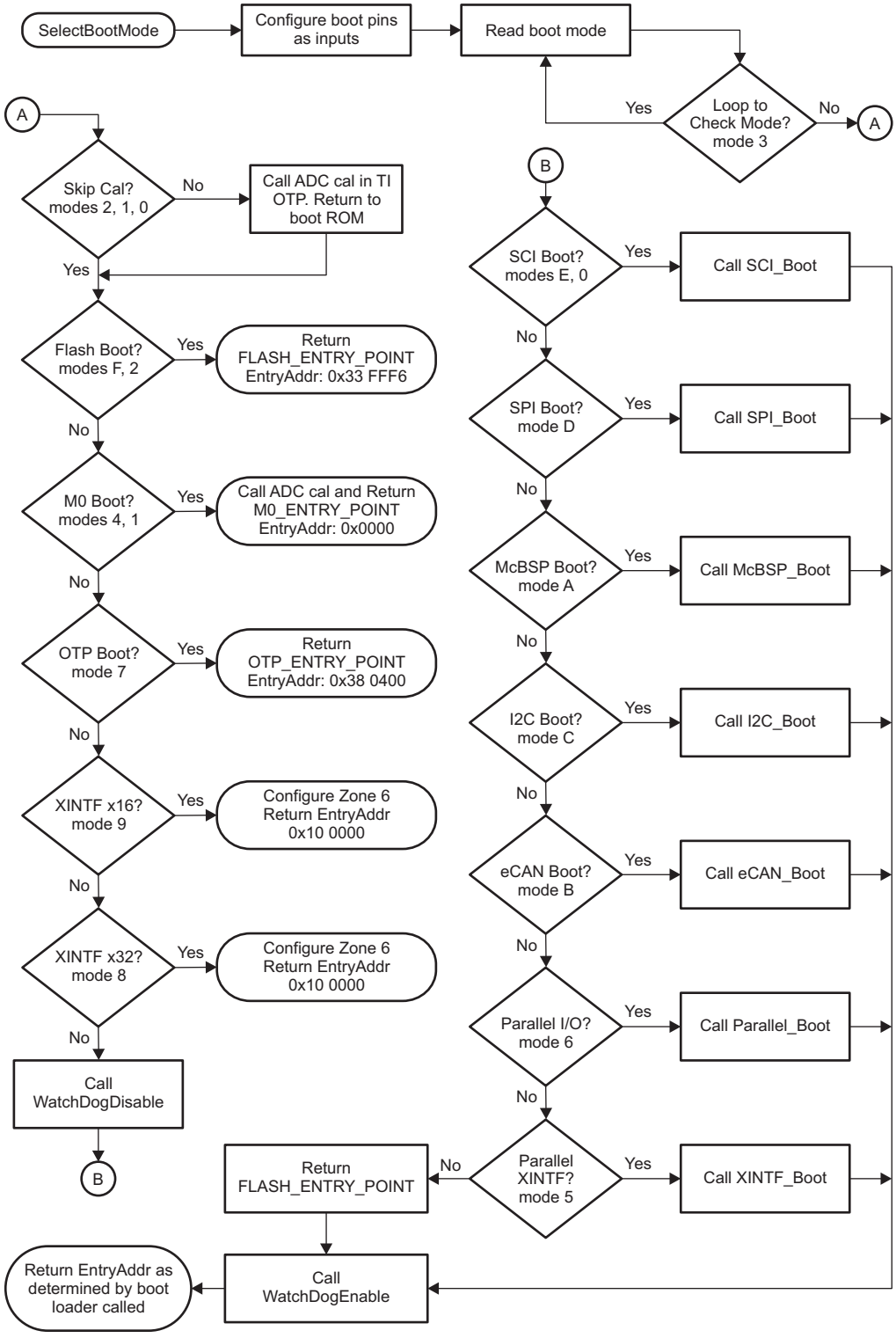
  The McBSP loader will not be called. Instead, the boot ROM will loop indefinitely.

NOTE: The SelectBootMode routine disables the watchdog before calling the SCI, I2C, eCAN , SPI, McBSP , or parallel bootloaders. The bootloaders do not service the watchdog and assume that it is disabled. Before exiting, the SelectBootMode routine will re-enable the watchdog and reset its timer.

If a bootloader is not going to be called, then the watchdog is left untouched.

When selecting a boot mode, the pins should be pulled low or high through a weak pulldown or weak pull-up such that the device can drive them to a new state when required.

Figure 2-13. Overview of the SelectBootMode Function



2.2.14 ADC_cal Assembly Routine

The ADC_cal() routine is programmed into TI reserved OTP memory by the factory. The boot ROM automatically calls the ADC_cal() routine to initialize the ADCREFSEL and ADCOFFTRIM registers 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 ADCREFSEL and ADCOFFTRIM must be initialized by the application. For working examples, see the ADC initialization in C2000Ware.

NOTE: Failure to initialize these registers will cause the ADC to function out of specification. The following three steps describe how to call the ADC_cal routine from an application.

Step 1. Add the ADC_cal assembly function to your project. The source is included in C2000Ware. Example 2-5 shows the contents of the ADC_cal function. The values 0xAAAA and 0xBBBB are place holders. The actual values programmed by TI are device specific.

Step 2. Add the .adc_cal section to your linker command file as shown in Example 2-6.

Step 3. Call the ADC_cal function before using the ADC. The ADC clocks must be enabled before making this call. See Example 2-7.

Example 2-5. Step 1: Add ADC_cal.asm to the Project

```
;---------------------------------------------
;This is the ADC cal routine. This routine is
;programmed into reserved memory by the factory.
;0xAAAA and 0xBBBB are place holders. The actual
;values programmed by TI are device specific.
;
;The ADC clock must be enabled before calling
;this function.
;---------------------------------------------
  .def _ADC_cal
  .asg "0x711C", ADCREFSEL_LOC
  .sect ".adc_cal"
_ADC_cal
  MOVW DP, #ADCREFSEL_LOC >> 6
  MOV @28, #0xAAAA
  MOV @29, #0xBBBB
  LRETR
```

Example 2-6. Step 2: Modify the Application Linker Command file to Access ADC_cal()

```
MEMORY
{
  PAGE 0 :
  ...
  ADC_CAL : origin = 0x380080, length = 0x000009
  ...
}
  SECTIONS
{
  ...
  .adc_cal : load = ADC_CAL, PAGE = 0, TYPE = NOLOAD
  ...
}
```

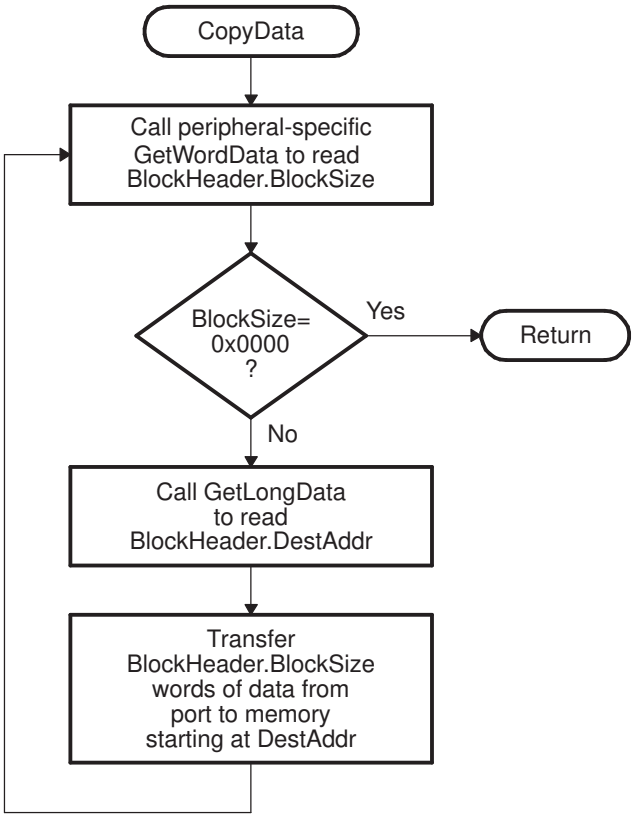
Example 2-7. Step 3: Call the ADC_cal() Function

```
extern void ADC_cal(void)           ;
…
EALLOW;
SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
ADC_cal();
SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0;
EDIS;
```

2.2.15 CopyData Function

All bootloaders use the same function to copy data from the port to the device's SARAM. This function is the *CopyData()* function. This function uses a pointer to a *GetWordData* function that is initialized by each of the loaders to properly read data from that port. For example, when the SPI loader is evoked, the *GetWordData* function pointer is initialized to point to the *SPI-specific SPI_GetWordData* function. Thus when the *CopyData()* function is called, the correct port is accessed. The flow of the *CopyData* function is shown in Figure 2-14.

Figure 2-14. Overview of CopyData Function



2.2.16 McBSP_Boot Function

The McBSP bootloader synchronously transfers code from McBSP-A to internal memory. McBSP-A is configured for slave mode operation. that is, it receives the frame sync and clock from the host. Upon receiving a word, the McBSP echoes the data back to the host. The host could use this feature to ensure that the previous word was received and copied by the McBSP before transmitting the next word. The host can download a kernel to reconfigure the McBSP if higher data throughput is desired. This can be done by choosing a faster PLL multiplier and also by choosing the /1 divider for the PLL output.

NOTE: Version 1 of the McBSP loader does not echo back data to the host. This feature is new as of Version 2 included on Rev A devices.

The McBSP-A loader uses pins shown in Table 2-6.

Table 2-6. Pins Used by the McBSP Loader

C28x Slave	Device Pin Number	Host Signal
MDXA	GPIO20	MDR
MDRA	GPIO21	MDX
MCLKXA	GPIO22	CLKX
MFSXA	GPIO23	FSR
MCLKRA	GPIO7	CLKX
MFSXA	GPIO5	FSXA

The bit rates achieved for different XCLKIN values as shown in Table 2-7. The SYSCLKOUT values shown are for the default PLLCR of 0 and PLLSTS[DIVSEL] set to 2.

Table 2-7. Bit-Rate Values for Different XCLKIN Values

XCLKIN	SYSCLKOUT	LSPCLK	CLKG
30 MHz	15 MHz	3.75 MHz	1.875 MHz
15 MHz	7.5 MHz	1.875 MHz	937.5 KHz

The host should transmit MSB first and LSB next. For example, to transmit the word 0x10AA to the device, transmit 10 first, followed by AA. The program flow of the McBSP bootloader is identical to the SCI bootloader, with the exception that 16-bit data is used. The data sequence for the McBSP bootloader follows the 16-bit data stream and is shown in Table 2-8.

Table 2-8. McBSP 16-Bit Data Stream

Word	Contents	Description
1	10AA	10AA (KeyValue for memory width = 16bits)
2	0000	8 reserved words (words 2-9)
...	...	...
9	0000	Last reserved word
10	AABB	Entry point PC[22:16]
11	CCDD	Entry point PC[15:0] (PC = 0xAABBCCDD)
12	MMNN	Block size (number of words) of the first block of data to load = 0xMMNN words
13	AABB	Destination address of first block Addr[31:16]
14	CCDD	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
15	XXXX	First word of the first block in the source being loaded
...		...
...		... Data for this section.

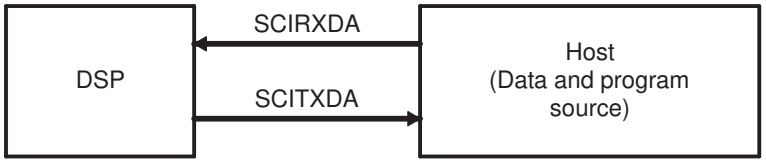
Table 2-8. McBSP 16-Bit Data Stream (continued)

Word	Contents	Description
.	XXXX	Last word of the first block of the source being loaded
.	MMNN	Block size of the 2nd block to load = 0xMMNN words
.	AABB	Destination address of second block Addr[31:16]
.	CCDD	Destination address of second block Addr[15:0]
.	XXXX	First word of the second block in the source being loaded
.		…
n	XXXX	Last word of the last block of the source being loaded
n+1	0000	Block size of 0000h - indicates end of the source program

2.2.17 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 2-4.

Figure 2-15. Overview of SCI Bootloader Operation



The SCI-A loader uses following pins:

- SCIRXDA on GPIO28
- SCITXDA on GPIO29

The 28x device communicates with the external host device through the SCI-A peripheral. The autobaud feature of the SCI port is used to lock baud rates with the host. For this reason the SCI loader is very flexible and you can use a number of different baud rates to communicate with the device.

After each data transfer, the 28x will echo back the 8-bit character received to the host. In this manner, the host can perform checks that each character was received by the 28x.

At higher baud rates, the slew rate of the incoming data bits can be affected by transceiver and connector performance. While normal serial communications may work well, this slew rate may limit reliable auto-baud detection at higher baud rates (typically beyond 100kbaud) and cause the auto-baud lock feature to fail. To avoid this, the following is recommended:

1. Achieve a baud-lock between the host and 28x SCI bootloader using a lower baud rate.
2. Load the incoming 28x application or custom loader at this lower baud rate.
3. The host may then handshake with the loaded 28x application to set the SCI baud rate register to the desired high baud rate.

Figure 2-16. Overview of SCI_Boot Function

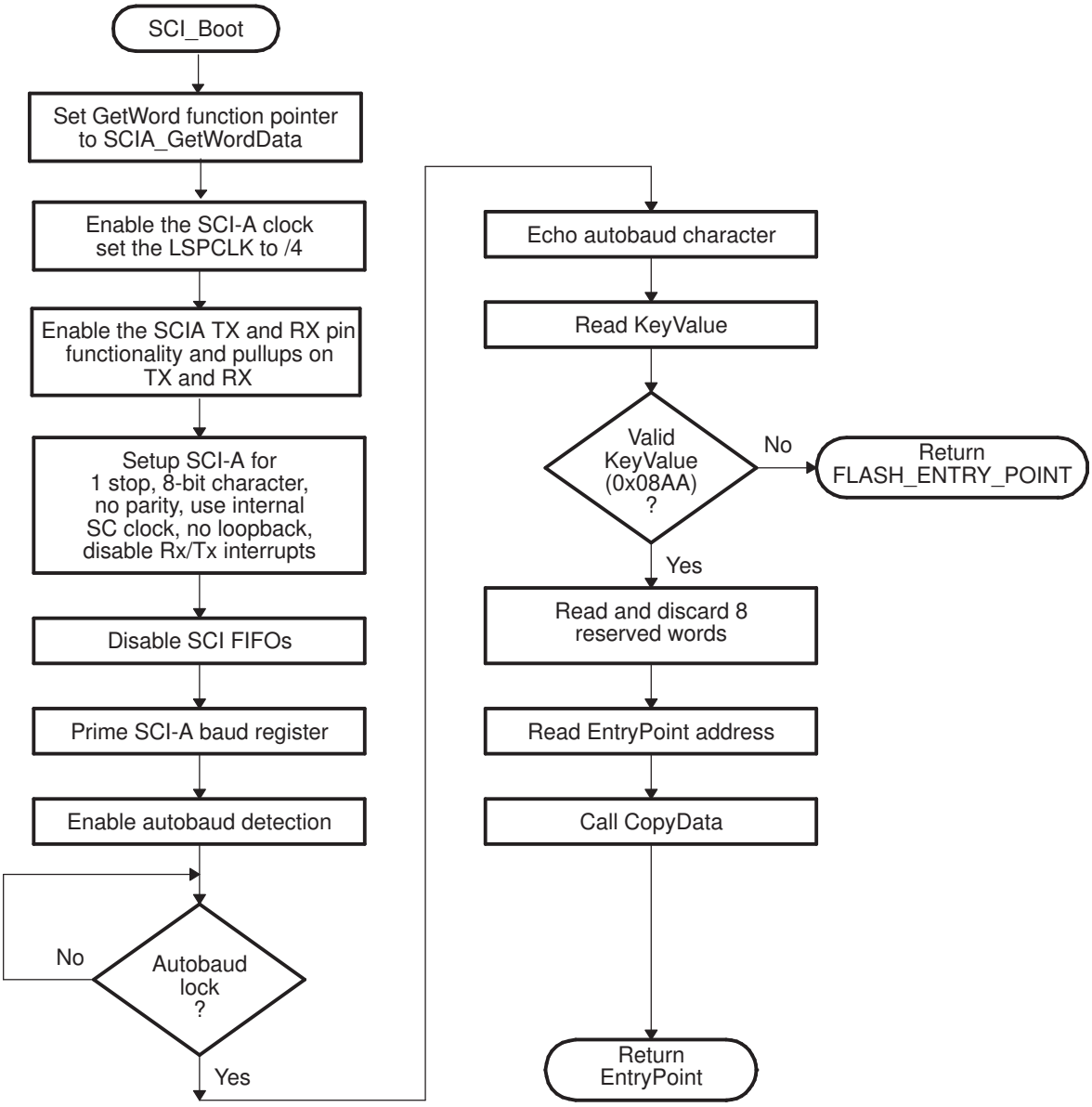
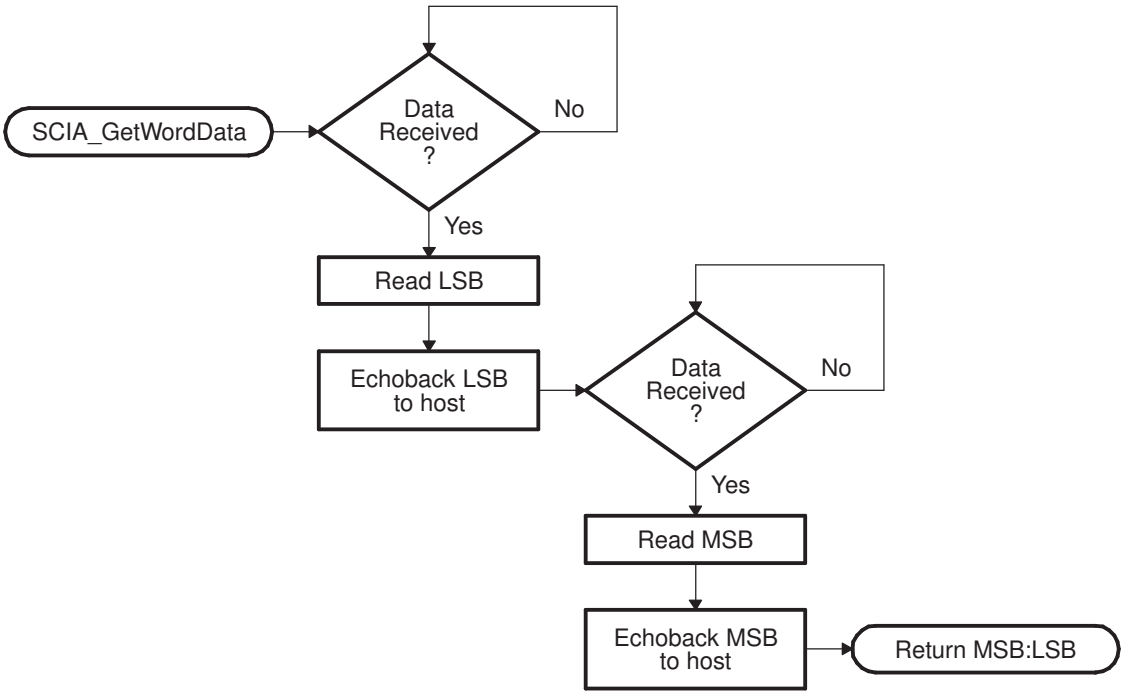


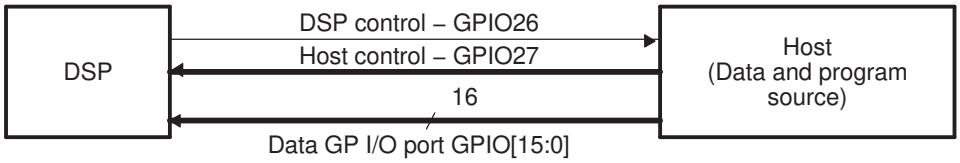
Figure 2-17. Overview of SCI_GetWordData Function



2.2.18 Parallel_Boot Function (GPIO)

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from GPIO0-GPIO15 to internal memory. Each value can be 16 bits or 8 bits long and follows the same data flow as outlined in Section 2.2.10.

Figure 2-18. Overview of Parallel GPIO Bootloader Operation



The parallel GPIO loader uses following pins:

- Data on GPIO[15:0] or GPIO[7:0]
- 28x Control on GPIO26
- Host Control on GPIO27

The 28x communicates with the external host device by polling/driving the GPIO27 and GPIO26 lines. The handshake protocol shown in Figure 2-19 must be used to successfully transfer each word via GPIO [15:0]. This protocol is very robust and allows for a slower or faster host to communicate with the device.

If the 8-bit mode is selected, two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of GPIO[7:0] ignoring the higher byte .

The 16-bit data stream is shown in Table 2-9 and the 8-bit data stream is shown in Table 2-10.

Table 2-9. Parallel GPIO Boot 16-Bit Data Stream

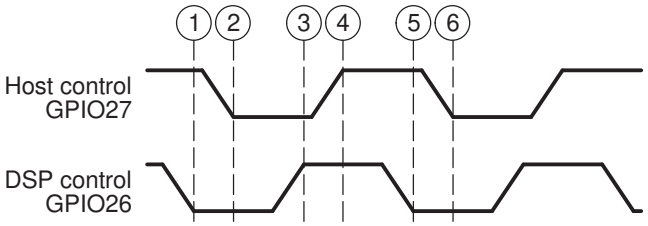
Word	GPIO[15:0]	Description
1	10AA	10AA (KeyValue for memory width = 16bits)
2	0000	8 reserved words (words 2 - 9)
...	...	...
9	0000	Last reserved word
10	AABB	Entry point PC[22:16]
11	CCDD	Entry point PC[15:0] (PC = 0xAABBCCDD)
12	MMNN	Block size (number of words) of the first block of data to load = 0xMMNN words
13	AABB	Destination address of first block Addr[31:16]
14	CCDD	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
15	XXXX	First word of the first block in the source being loaded
...		...
...		Data for this section. ...
.	XXXX	Last word of the first block of the source being loaded
.	MMNN	Block size of the 2nd block to load = 0xMMNN words
.	AABB	Destination address of second block Addr[31:16]
.	CCDD	Destination address of second block Addr[15:0]
.	XXXX	First word of the second block in the source being loaded
.		…
n	XXXX	Last word of the last block of the source being loaded (More sections if required)
n+1	0000	Block size of 0000h - indicates end of the source program

Table 2-10. Parallel GPIO Boot 8-Bit Data Stream

Bytes		GPIO[7 :0] (Byte 1 of 2)	GPIO[7 :0] (Byte 2 of 2)	Description
1	2	AA	08	0x08AA (KeyValue for memory width = bits)
3	4	00	00	8 reserved words (words 2 - 9)
...	...	...	...	...
17	18	00	00	Last reserved word
19	20	BB	00	Entry point PC[22:16]
21	22	DD	CC	Entry point PC[15:0] (PC = 0x00BBCCDD)
23	24	NN	MM	Block size of the first block of data to load = 0xMMNN words
25	26	BB	AA	Destination address of first block Addr[31:16]
27	28	DD	CC	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
29	30	BB	AA	First word of the first block in the source being loaded = 0xAABB
...				...
...				Data for this section. ...
.		BB	AA	Last word of the first block of the source being loaded = 0xAABB
.		NN	MM	Block size of the 2nd block to load = 0xMMNN words
.		BB	AA	Destination address of second block Addr[31:16]
.		DD	CC	Destination address of second block Addr[15:0]
.		BB	AA	First word of the second block in the source being loaded
.				…
n	n+1	BB	AA	Last word of the last block of the source being loaded (More sections if required)
n+2	n+3	00	00	Block size of 0000h - indicates end of the source program

The 28x device first signals the host that it is ready to begin data transfer by pulling the GPIO26 pin low. The host load then initiates the data transfer by pulling the GPIO27 pin low. The complete protocol is shown in Figure 2-19.

Figure 2-19. Parallel GPIO Boot Loader Handshake Protocol



1. The 28x device indicates it is ready to start receiving data by pulling the GPIO26 pin low.
2. The bootloader waits until the host puts data on GPIO [15:0]. The host signals to the 28x device that data is ready by pulling the GPIO27 pin low.
3. The 28x device reads the data and signals the host that the read is complete by pulling GPIO26 high.
4. The bootloader waits until the host acknowledges the 28x device by pulling GPIO27 high.
5. The 28x device again indicates it is ready for more data by pulling the GPIO26 pin low.

This process is repeated for each data value to be sent.

Figure 2-20 shows an overview of the Parallel GPIO bootloader flow.

Figure 2-20. Parallel GPIO Mode Overview

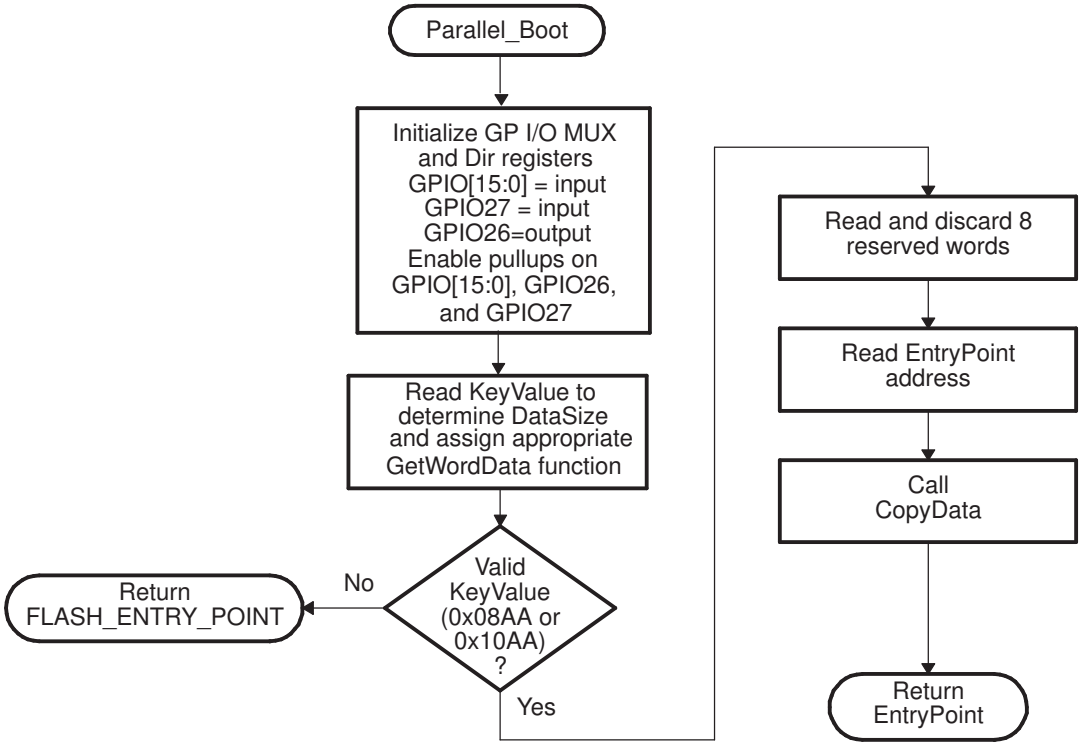


Figure 2-21 shows the transfer flow from the host side. The operating speed of the CPU and host are not critical in this mode as the host will wait for the 28x device, and the 28x device will in turn wait for the host. In this manner the protocol will work with both a host running faster and a host running slower than the 28x device.

Figure 2-21. Parallel GPIO Mode - Host Transfer Flow

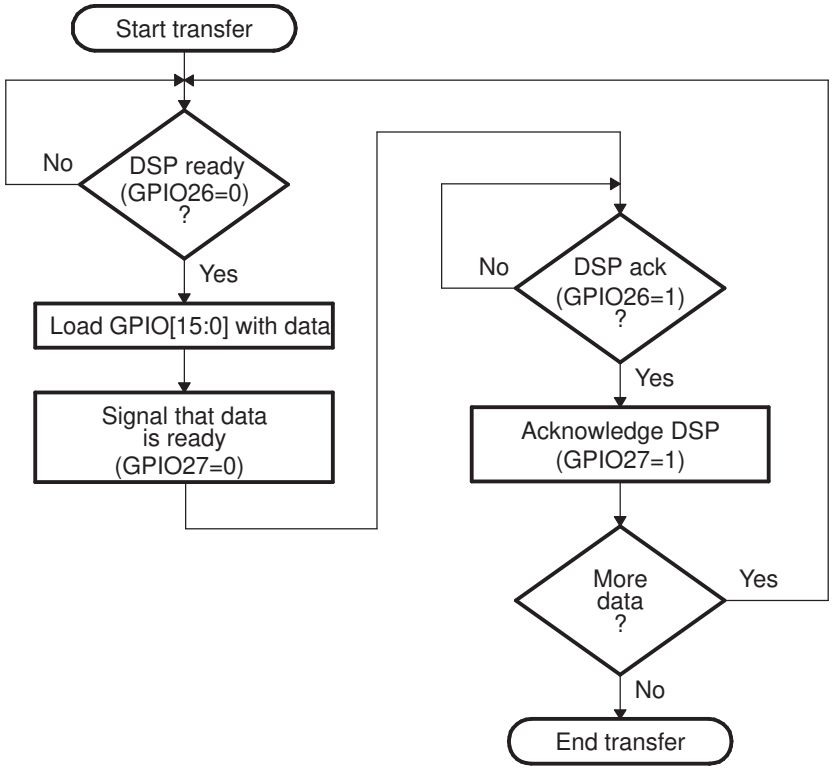


Figure 2-22 and Figure 2-23 shows the flow used to read a single word of data from the parallel port. The loader uses the method shown in Figure 2-11 to read the key value and to determine if the incoming data stream width is 8-bit or 16-bit. A different GetWordData function is used by the parallel loader depending on the data size of the incoming data stream.

- **16-bit data stream**

  For an 16-bit data stream, the function Parallel_GetWordData16bit is used. This function reads all 16-bits at a time. The flow of this function is shown in Figure 2-22.

- **8-bit data stream**

  The 8-bit routine, shown in Figure 2-23, discards the upper 8 bits of the first read from the port and treats the lower 8 bits as the least significant byte (LSB) of the word to be fetched. The routine will then perform a second read to fetch the most significant byte (MSB). It then combines the MSB and LSB into a single 16-bit value to be passed back to the calling routine.

Figure 2-22. 16-Bit Parallel GetWord Function

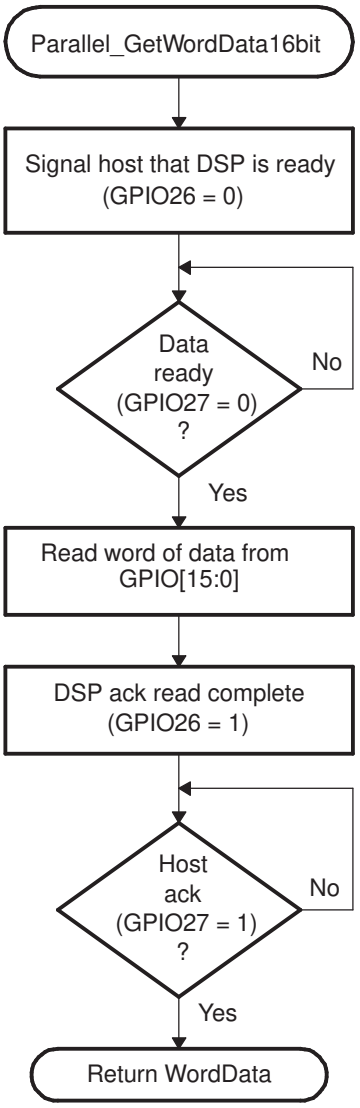
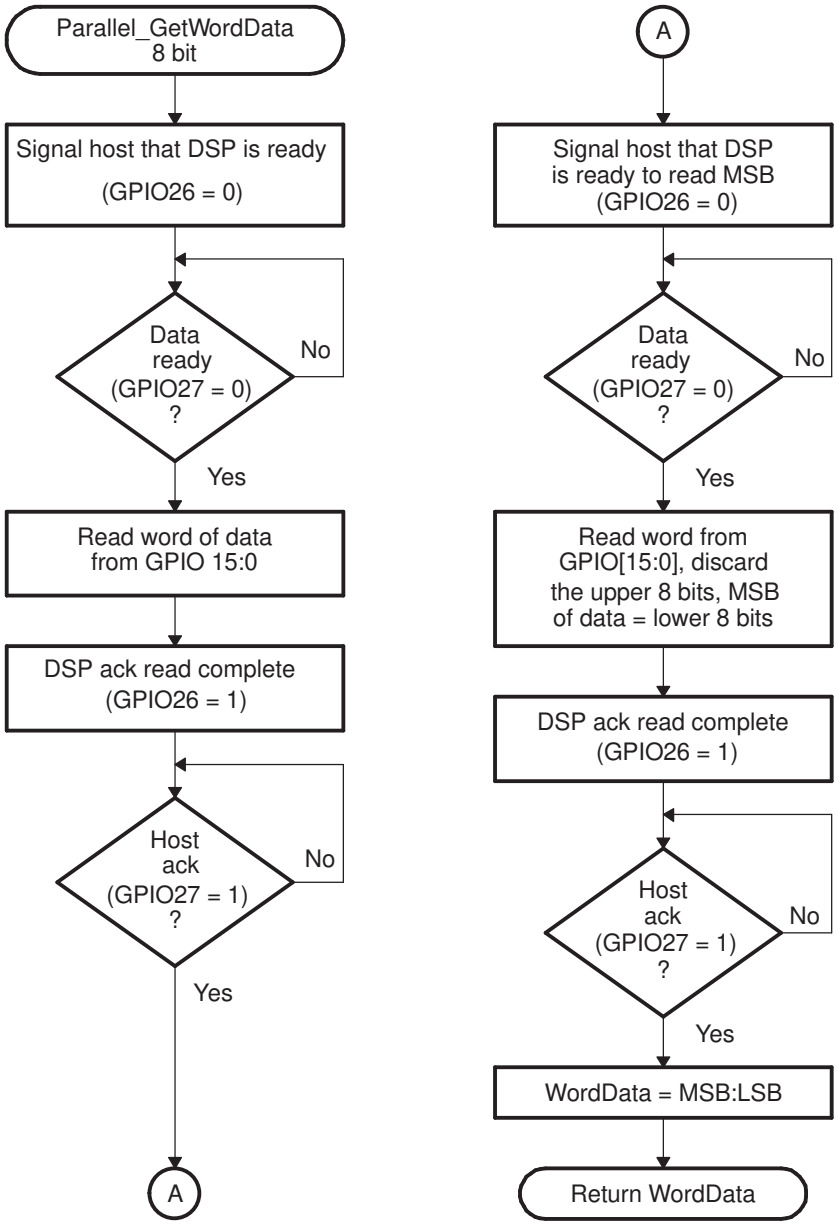


Figure 2-23. 8-Bit Parallel GetWord Function



2.2.19 XINTF_Parallel_Boot Function

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from XD[15:0] to internal memory. Each value can be 16 bits or 8 bits long and follows the same data flow as outlined in Section 2.2.10. The each word or byte of data is read from address 0x100000 in XINTF zone 6.

NOTE: This mode loads a stream of data into the SARAM of the device using XINTF resources. If you instead want to configure and jump to the XINTF then use the "Jump to XINTF x16" or "Jump to XINTF x32" boot mode.

The parallel XINTF loader uses following pins:

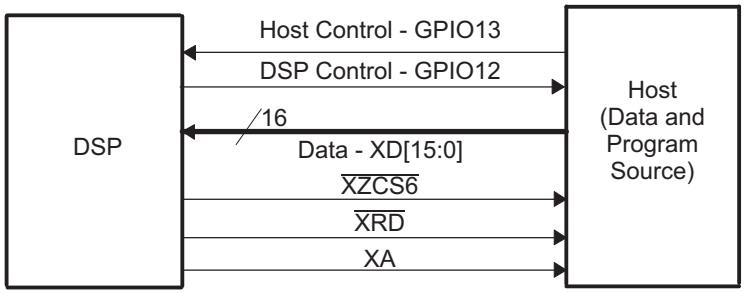
- Data on XD[15:0] or XD[7:0]
- 28x Control on GPIO13
- Host Control on GPIO12

The 28x communicates with the external host device by polling/driving the GPIO13 and GPIO12 lines. The handshake protocol shown in Figure 2-19 must be used to successfully transfer each word via XD[15:0]. This protocol is very robust and allows for a slower or faster host to communicate with the device.

If the 8-bit mode is selected, two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of XD[7:0] ignoring the higher byte.

The device first signals the host that the device is ready to begin data transfer by pulling the GPIO12 pin low. The host load then initiates the data transfer by pulling the GPIO13 pin low. The complete protocol is shown in Figure 2-24.

Figure 2-24. Overview of the Parallel XINTF Boot Loader Operation



The device communicates with the external host device by polling/driving the GPIO13 and GPIO12 lines. The handshake protocol shown below must be used to successfully transfer each word via the first address location within XINTF zone 6. This protocol is very robust and allows for a slower or faster host to communicate with the device.

If the 8-bit mode is selected, two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of XD[7:0] ignoring the higher byte.

To begin the transfer, the device will use the default XINTF timing for zone 6. This is the maximum wait states, slowest XINTF timing available. That is:

1. XTIMCLK = ½ SYSCLKOUT
2. XCLKOUT = ½ XTIMCLK
3. XRDLEAD = XWRLEAD = 3
4. XRDACTIVE = XWRACTIVE = 7
5. XRDTRAIL = XWRACTIVE = 3
6. XSIZE = 3 for 16-bit wide
7. X2TIMING = 1. Timing values are 2:1.
8. USEREADY = 1, READYMODE = 1 (XREADY sampled asynchronous mode)

The first 7 words of the data stream are read at this slow timing. Words 2 – 7 include configuration information that will be used to adjust the PLLCR/PLLSTS and XINTF XTIMING6. The rest of the data stream is read using the new configuration.

The 16-bit data stream is shown in Table 2-11 and the 8-bit data stream is shown in Table 2-12.

Table 2-11. XINTF Parallel Boot 16-Bit Data Stream

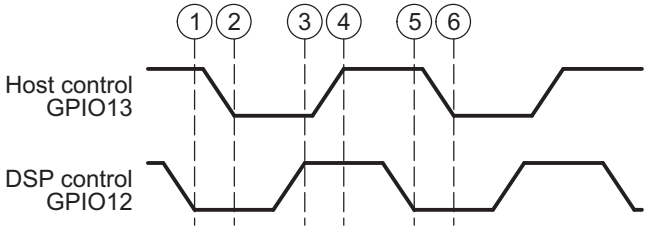
Word	XD[15:0]	Description
1	10AA	10AA (KeyValue for memory width = 16bits)
2	AABB	PLLCR register = 0xAABB
3	000B	PLLSTS[DIVSEL] bits = 0xB
4	AABB	XTIMING6[31:16]
5	CCDD	XTIMING6[15:0] (XTIMING6 = 0xAABBCCDD)
6	EEFF	XINTCNF2[31:16]
7	GGHH	XINTCNF2[15:0] (XINTCNF2 = 0xEEFFGGHH)
8	0000	reserved
9	0000	reserved
10	AABB	Entry point PC[22:16]
11	CCDD	Entry point PC[15:0] (PC = 0xAABBCCDD)
12	MMNN	Block size (number of words) of the first block of data to load = 0xMMNN words
13	AABB	Destination address of first block Addr[31:16]
14	CCDD	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
15	XXXX	First word of the first block in the source being loaded
...		...
...		Data for this section.
		...
.	XXXX	Last word of the first block of the source being loaded
.	MMNN	Block size of the 2nd block to load = 0xMMNN words
.	AABB	Destination address of second block Addr[31:16]
.	CCDD	Destination address of second block Addr[15:0]
.	XXXX	First word of the second block in the source being loaded
.		…
n	XXXX	Last word of the last block of the source being loaded (More sections if required)
n+1	0000	Block size of 0000h - indicates end of the source program

Table 2-12. XINTF Parallel Boot 8-Bit Data Stream

Bytes		XD[7:0] (Byte 1 of 2)	XD[7:0] (Byte 2 of 2)	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 8bits)
3	4	BB	AA	PLLCR register = 0xAABB
5	6	0B	00	PLLSTS[DIVSEL] bits = 0xB
7	8	BB	AA	XTIMING6[31:16]
9	10	DD	CC	XTIMING6[15:0] (XTIMING6 = 0xAABBCCDD)
11	12	FF	EE	XINTCNF2[31:16]
13	14	HH	GG	XINTCNF2[15:0] (XINTCNF2 = 0xEEFFGGHH)
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 = 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

Figure 2-25 shows an overview of the XINTF parallel boot loader handshake protocol.

Figure 2-25. XINTF_Parallel Boot Loader Handshake Protocol



1. The 28x device indicates it is ready to start receiving data by pulling the GPIO12 pin low.
2. The bootloader waits until the host puts data on XD[15:0]. The host signals to the 28x device that data is ready by pulling the GPIO13 pin low.
3. The 28x device reads the data and signals the host that the read is complete by pulling GPIO12 high.
4. The bootloader waits until the host acknowledges the 28x by pulling GPIO13 high.
5. The 28x device again indicates it is ready for more data by pulling the GPIO12 pin low.

This process is repeated for each data value to be sent.

Figure 2-26 shows an overview of the XINTF parallel bootloader flow.

Figure 2-26. XINTF Parallel Mode Overview

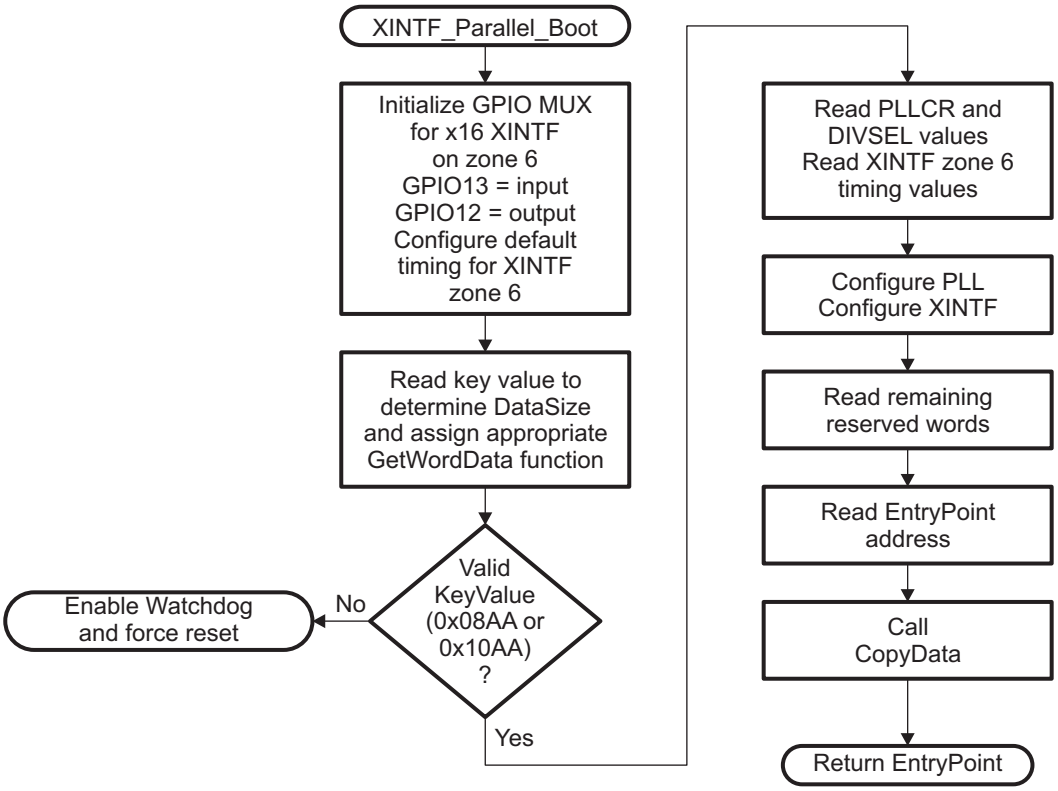


Figure 2-27 shows the transfer flow from the host side. The operating speed of the CPU and host are not critical in this mode as the host will wait for the 28x and the 28x will in turn wait for the host. In this manner the protocol will work with both a host running faster and a host running slower then the 28x device.

Figure 2-27. XINTF Parallel Mode - Host Transfer Flow

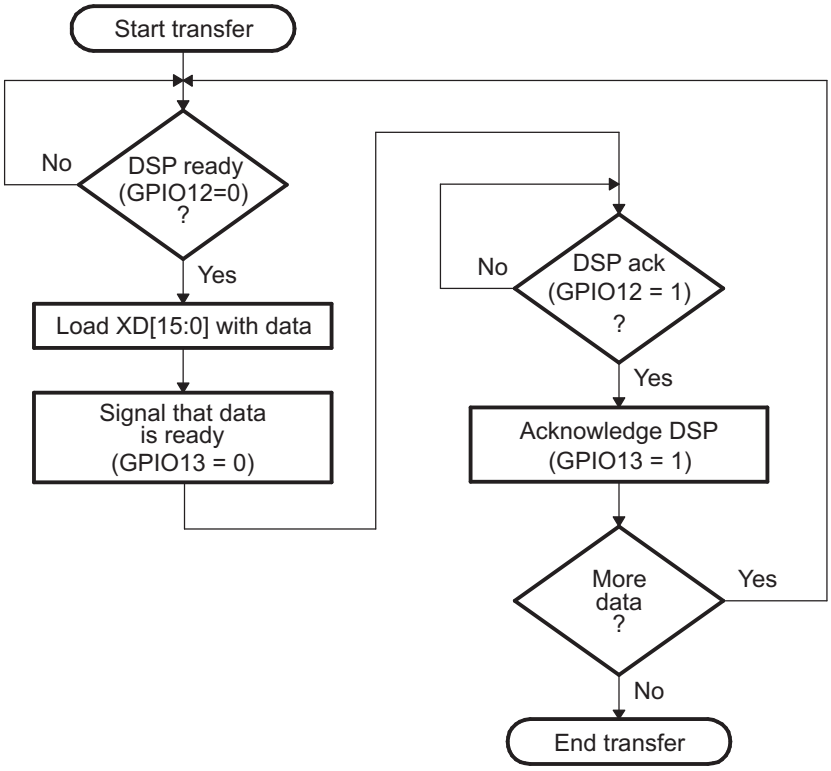


Figure 2-28 and Figure 2-29 show the flow used to read a single word of data from the parallel port. The loader uses the method shown in Figure 2-11 to read the key value and to determine if the incoming data stream width is 8-bit or 16-bit. A different GetWordData function is used by the parallel loader depending on the data size of the incoming data stream.

- **16-bit data stream**

  For an 16-bit data stream, the function XINTF_Parallel_GetWordData16bit is used. This function reads all 16-bits at a time. The flow of this function is shown in Figure 2-28.

- **8-bit data stream**

  For an 8-bit data stream, the function XINTF_Parallel_GetWordData8bit is used. The 8-bit routine, shown in Figure 2-29, discards the upper 8 bits of the first read from the port and treats the lower 8 bits as the least significant byte (LSB) of the word to be fetched. The routine will then perform a second read to fetch the most significant byte (MSB). It then combines the MSB and LSB into a single 16-bit value to be passed back to the calling routine.

Figure 2-28. 16-Bit Parallel GetWord Function

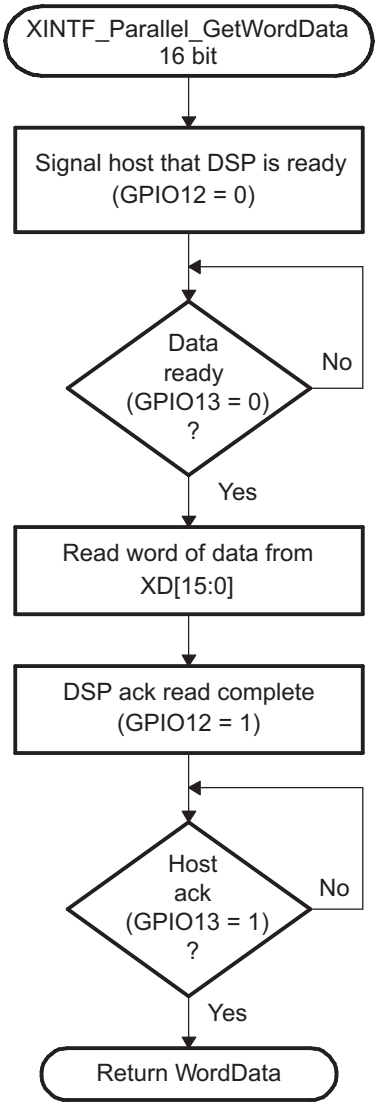
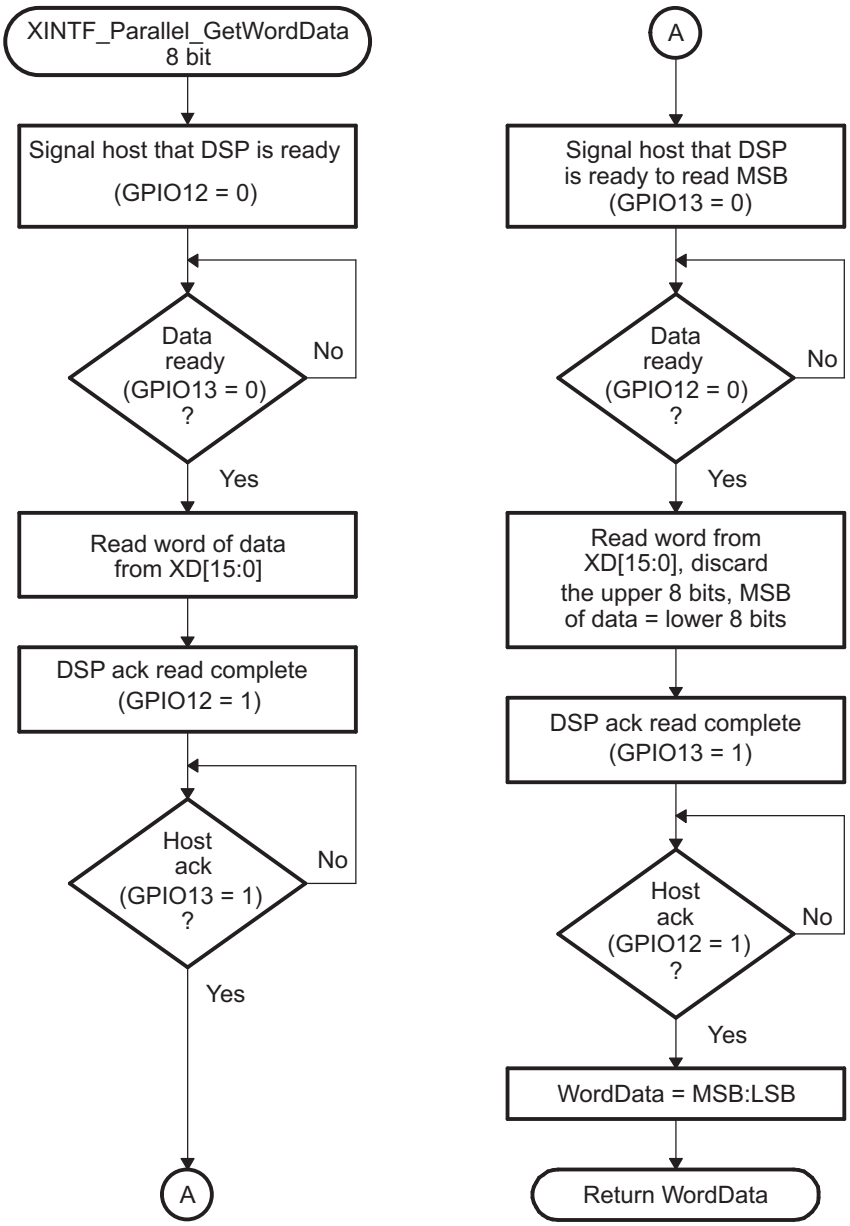


Figure 2-29. 8-Bit Parallel GetWord Function



2.2.20 SPI_Boot Function

The SPI boot ROM loader initializes the SPI module to interface to a SPI-compatible, 16-bit or 24-bit addressable serial EEPROM or flash device.

It expects such a device to be present on the SPI-A pins as indicated in Figure 2-30. The SPI bootloader supports an 8-bit data stream. It does not support a 16-bit data stream.

Figure 2-30. SPI Loader

The SPI-A loader uses following pins:

- SPISIMOA on GPIO16
- SPISOMIA on GPIO17
- SPICLKA on GPIO18
- SPISTEA on GPIO19

The SPI boot ROM loader initializes the SPI with the following settings: FIFO enabled, 8-bit character, internal SPICLK master mode and talk mode, clock phase = 1, polarity = 0, using the slowest baud rate.

If the download is to be performed from an SPI port on another device, then that device must be setup to operate in the slave mode and mimic a serial SPI EEPROM. Immediately after entering the SPI_Boot function, the pin functions for the SPI pins are set to primary and the SPI is initialized. The initialization is done at the slowest speed possible. Once the SPI is initialized and the key value read, you could specify a change in baud rate or low speed peripheral clock.

Table 2-13. 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 2-13. 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 entry point for the flash (0x33 7FF6) is returned to the calling routine.

Step 6. The next two bytes fetched can be used to change the value of the low speed peripheral clock register (LOSPCP) and the SPI baud rate register (SPIBRR). The first byte read is the LOSPCP value and the second byte read is the SPIBRR value. The next 7 words are reserved for future enhancements. The SPI bootloader reads these 7 words and discards them.

Step 7. The next two words makeup the 32-bit entry point address where execution will continue after the boot load process is complete. This is typically the entry point for the program being downloaded through the SPI port.

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 2-31. Data Transfer From EEPROM Flow

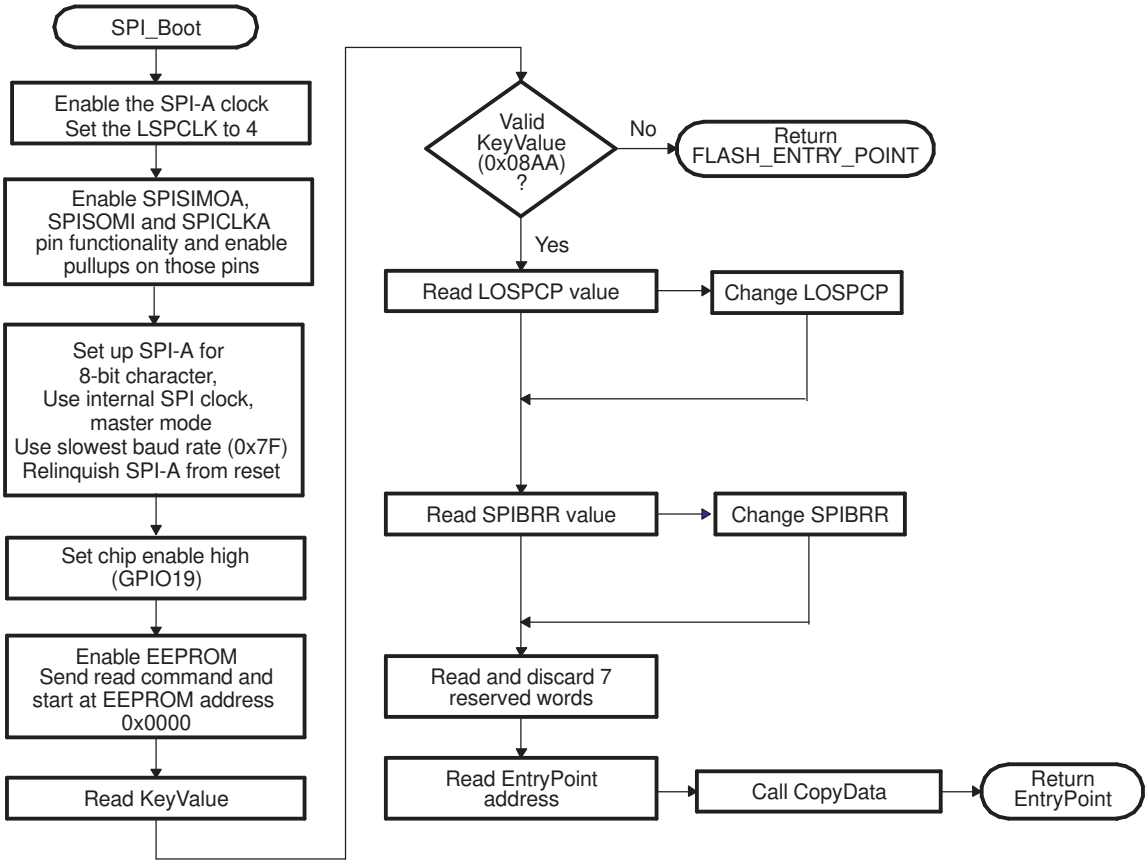
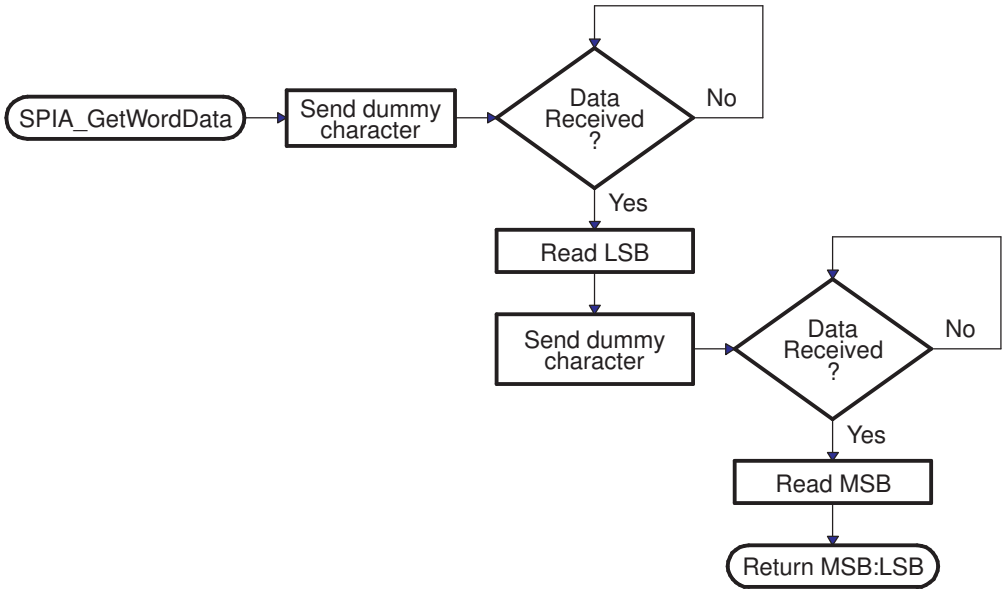


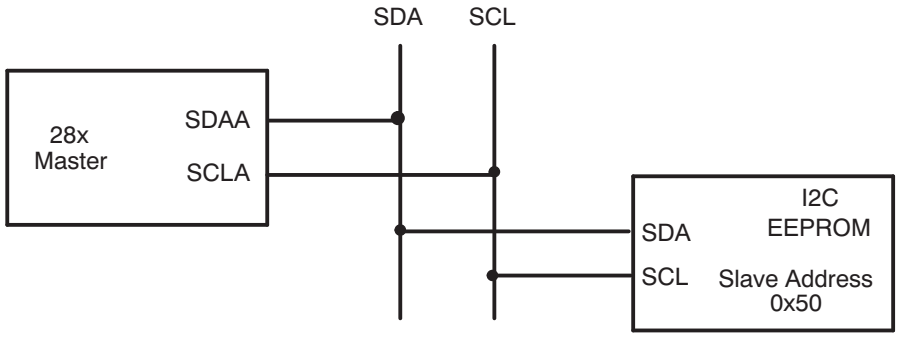
Figure 2-32. Overview of SPIA_GetWordData Function



2.2.21 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 2-33. The EEPROM must adhere to conventional I2C EEPROM protocol, as described in this section, with a 16-bit base address architecture.

Figure 2-33. EEPROM Device at Address 0x50



The I2C loader uses following pins:

- SDAA on GPIO32
- SCLA on GPIO33

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:

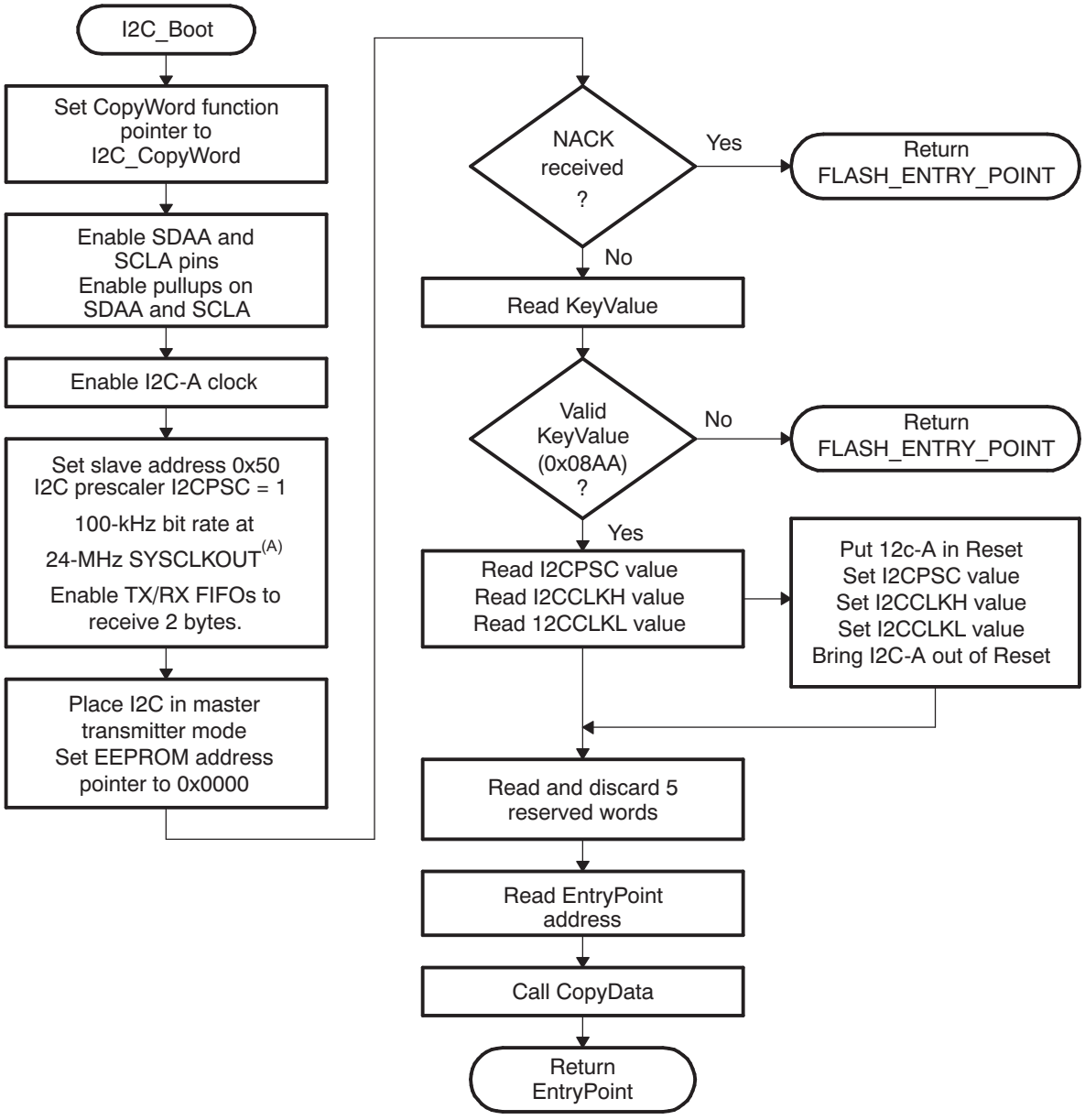
- The input frequency to the device must be in the appropriate range.
- The EEPROM must be at slave address 0x50.

To use the I2C-A bootloader, the input clock frequency to the device must be between 28 MHz and 48 MHz. This input clock frequency will result in a default 14 MHz to 24 MHz system clock (SYSCLKOUT). By default, the bootloader sets the I2CPSC prescale value to 1 so that the I2C clock will be divided down from SYSCLKOUT. This results in an I2C clock between 7 MHz and 12 MHz, which meets the I2C peripheral clocking specification. The I2CPSC value can be modified after receiving the first few bytes from the EEPROM, but it is not advisable to do this, because this can cause the I2C to operate out of the required specification.

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 12 MHz. These registers can be modified after receiving the first few bytes from the EEPROM. This allows the communication to be increased up to a 400-kHz bit rate (fast I2C mode) during the remaining data reads.

Arbitration, bus busy, and slave signals are not checked. Therefore, no other master is allowed to control the bus during this initialization phase. If the application requires another master during I2C boot mode, that master must be configured to hold off sending any I2C messages until the application software signals that it is past the bootloader portion of initialization.

Figure 2-34. Overview of I2C_Boot Function



A During device boot, SYSCLKOUT will be the device input frequency divided by two.

The nonacknowledgment bit is checked only during the first message sent to initialize the EEPROM base address. This is to make sure that an EEPROM is present at address 0x50 before continuing. If an EEPROM is not present, code will jump to the flash entry point. The nonacknowledgment bit is not checked during the address phase of the data read messages (I2C_Get Word). If a non acknowledgment is received during the data read messages, the I2C bus will hang. Table 2-14 shows the 8-bit data stream used by the I2C.

Table 2-14. I2C 8-Bit Data Stream

Byte	Contents
1	LSB: AA (KeyValue for memory width = 8 bits)
2	MSB: 08h (KeyValue for memory width = 8 bits)
3	LSB: I2CPSC[7:0]
4	reserved
5	LSB: I2CCLKH[7:0]
6	MSB: I2CCLKH[15:8]
7	LSB: I2CCLKL[7:0]
8	MSB: I2CCLKL[15:8]
...	...
...	Data for this section.
17	LSB: Reserved for future use
18	MSB: Reserved for future use
19	LSB: Upper half of entry point PC
20	MSB: Upper half of entry point PC[22:16] (Note: Always 0x00)
21	LSB: Lower half of entry point PC[15:8]
22	MSB: Lower half of entry point PC[7:0]
	...
...	Data for this section.
...	...
	Blocks of data in the format size/destination address/data as shown in the generic data stream description.
...	...
...	Data for this section.
LSB: 00h	
n+1	MSB: 00h - indicates the end of the source

The I2C EEPROM protocol required by the I2C bootloader is shown in Figure 2-35 and Figure 2-36. The first communication, which sets the EEPROM address pointer to 0x0000 and reads the KeyValue (0x08AA) from it, is shown in Figure 2-35. All subsequent reads are shown in Figure 2-36 and are read two bytes at a time.

Figure 2-35. Random Read

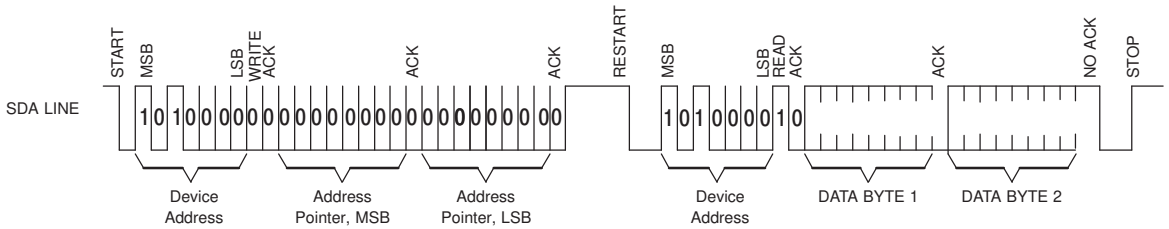
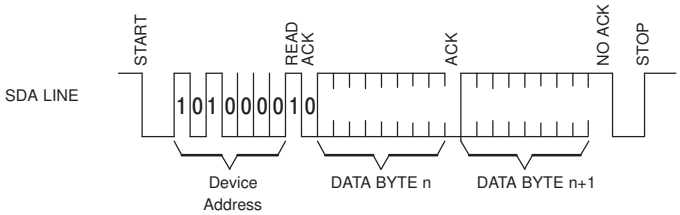


Figure 2-36. Sequential Read



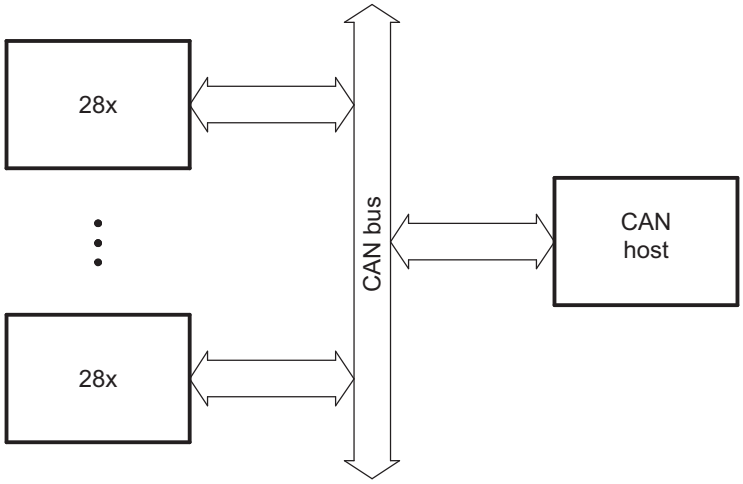
2.2.22 eCAN Boot Function

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 the following pins:

- CANRXA on GPIO30
- CANTXA on GPIO31

Figure 2-37. Overview of eCAN-A Bootloader Operation



The bit-timing registers are programmed in such a way that a valid bit-rate is achieved for different XCLKIN values as shown in Table 2-15.

Table 2-15. Bit-Rate Values for Different XCLKIN Values

XCLKIN	SYSCLKOUT	Bit Rate
30 MHz	15 MHz	500 kbps
15 MHz	7.5 MHz	250 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 15, 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 2-16.

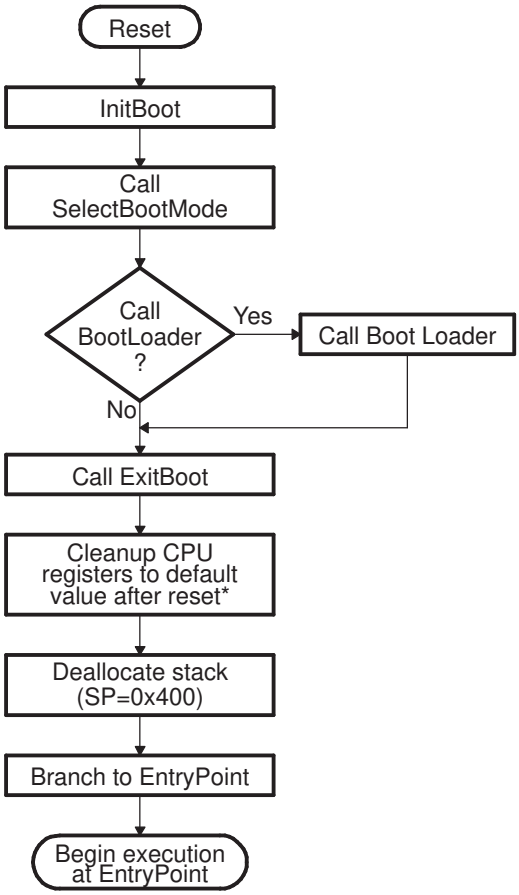
Table 2-16. eCAN 8-Bit Data Stream

Bytes		Byte 1 of 2	Byte 2 of 2	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 8bits)
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

2.2.23 ExitBoot Assembly Routine

The Boot ROM includes an ExitBoot routine that restores the CPU registers to their default state at reset. This is performed on all registers with one exception. The OBJMODE bit in ST1 is left set so that the device remains configured for C28x operation. This flow is detailed in Figure 2-38:

Figure 2-38. ExitBoot Procedure Flow



The following CPU registers are restored to their default values:

- ACC = 0x0000 0000
- RPC = 0x0000 0000
- P = 0x0000 0000
- XT = 0x0000 0000
- ST0 = 0x0000
- ST1 = 0x0A0B
- XAR0 = XAR7 = 0x0000 0000

After the ExitBoot routine completes and the program flow is redirected to the entry point address, the CPU registers will have the following values as shown in Table 2-17.

Table 2-17. 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 2.2.10. 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 .

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.

Table 2-18 summarizes the hex conversion utility options available for the bootloader. See the *TMS320C28x Assembly Language Tools v18.1.0.LTS User's Guide* for more information abut the compiling and linking process, and for a detailed description of the hex2000 operations used to generate a boot table. Updates will be made to support the I2C boot. See the Codegen release notes for the latest information.

Table 2-18. Bootloader Options

Option	Description
-boot	Convert all sections into bootable form (use instead of a SECTIONS directive)
-sci8	Specify the source of the bootloader table as the SCI-A port, 8-bit mode
-spi8	Specify the source of the bootloader table as the SPI-A port, 8-bit mode
-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 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* 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 device in order for the code to execute properly. In this case, the codestart, ramfuncs, .cinit, myreset and .text sections need to be loaded. The other sections are uninitialized and will not be included in the loading process. The map file also indicates the size of each section and the starting address. For example, the .text section has 0x155 words and starts at 0x3FA000.

Example 2-8. 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. The checksum is intended for TI internal use and will change based on the silicon and boot ROM version.

Table 2-19. Bootloader Revision and Checksum Information

Address	Contents
0x3F FFB9	Flash API silicon compatibility check. This location is read by some versions of the flash API to make sure it is running on a compatible silicon version.
0x3F FFBA	Boot ROM Version Number
0x3F FFBB	MM/YY of release (in decimal)
0x3F FFBC	Least significant word of checksum
0x3F FFBD	. . .
0x3F FFBE	. . .
0x3F FFBF	Most significant word of checksum

Table 2-20 shows the boot ROM revision details.

Table 2-20. Bootloader Revision Per Device

Device(s)	Silicon REVID (Address 0x883)	Boot ROM Revision
2823x, 2833x	0 (First silicon)	Version 1
2823x, 2833x	1	Version 2

2.4.2 Bootloader Code Revision History

- **Version: 2, Released: March 2008:**
  The following changes were made:
  - Corrected the GPIO configuration for boot to XINTF x16, x32 and parallel XINTF modes.
  - Updated the McBSP bootloader to echo back data received. Version 1 did not echo back the data.
  - The eCAN loader leaves the SAM bit in its default state (0). Version 1 changed SAM to 1.
- **Version: 1, Released: June 2007:**
  The initial release of the boot ROM.

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 is applicable for ePWM type 0. See the *C2000 Real-Time Control Peripherals Reference Guide* for a list of all devices with an ePWM module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

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 High-Resolution Pulse Width Modulator (HRPWM) chapter. See the datasheet 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.

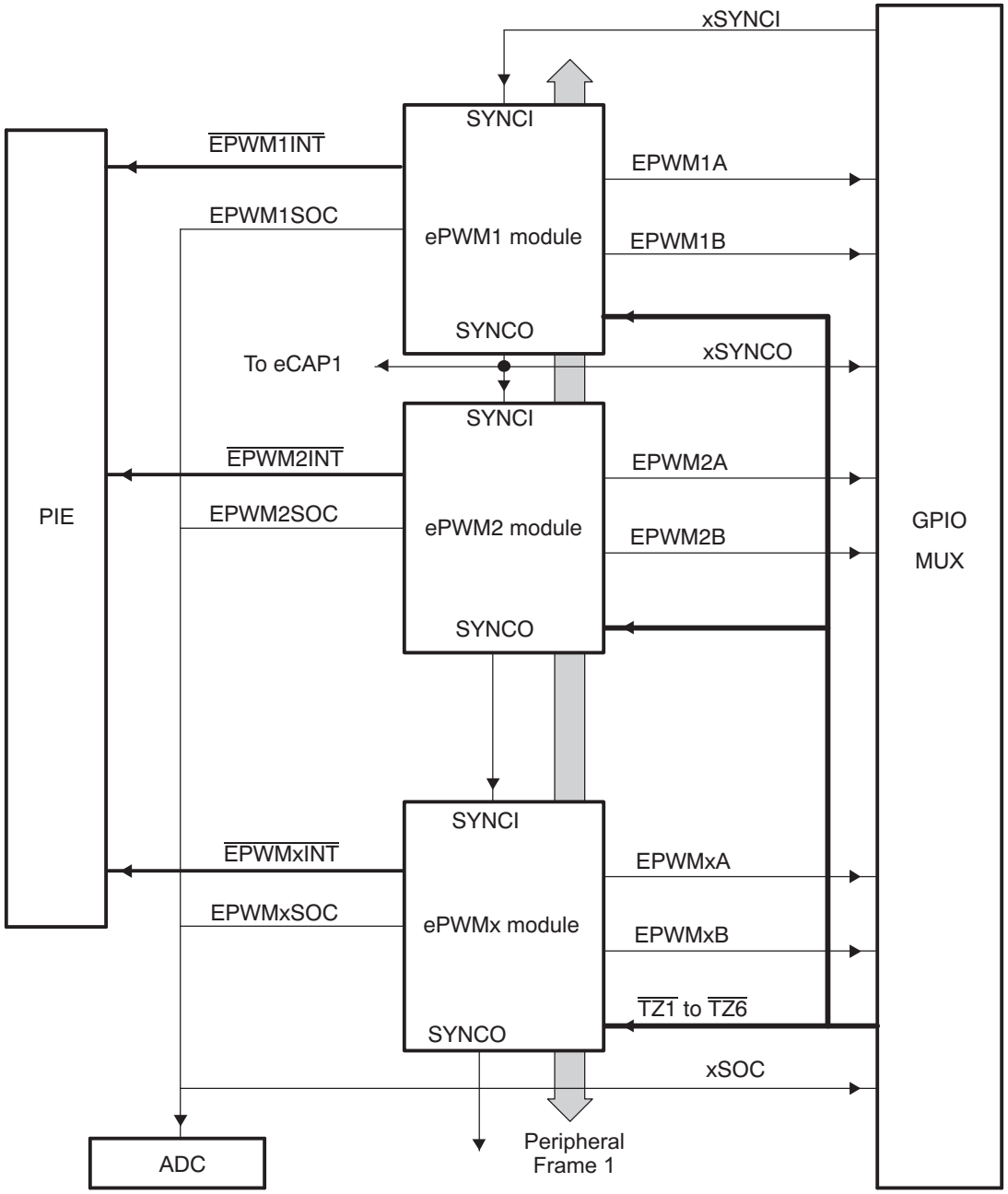
The ePWM modules are chained together via a clock synchronization scheme that allows them to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral modules (eCAP). The number of modules is device-dependent and based on target application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



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 seven 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

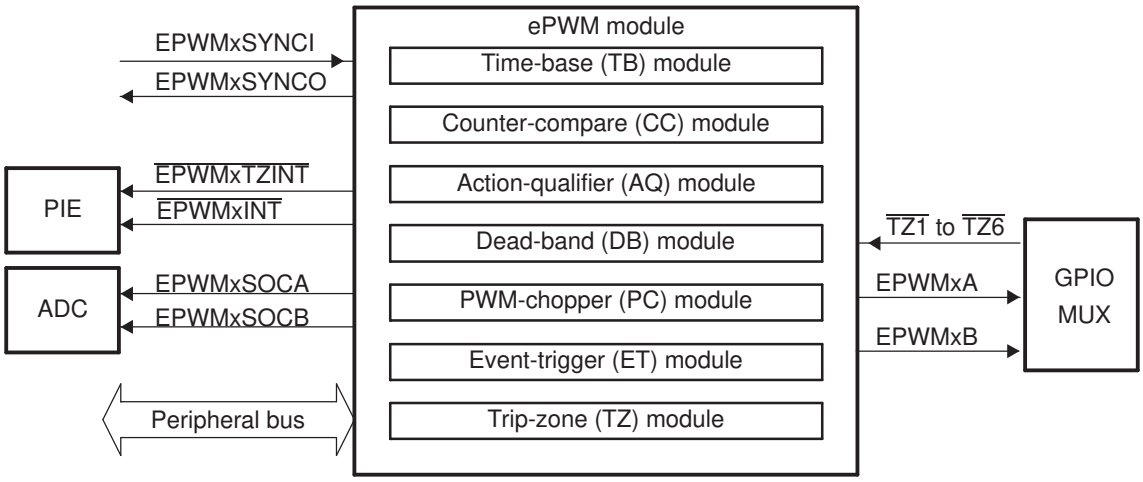


Figure 3-3 shows more internal details of a single ePWM module. The main signals used by the ePWM module are:

- **PWM output signals (EPWMxA and EPWMxB).**

  The PWM output signals are made available external to the device through the GPIO peripheral described in the system control and interrupts guide for your device.
- **Trip-zone signals ($\overline{TZ1}$ to $\overline{TZ6}$).**

  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 $\overline{TZ1}$ to $\overline{TZ6}$ trip-zone signals can be configured as asynchronous inputs through the GPIO peripheral.
- **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 (one for each sequencer). Any ePWM module can trigger a start of conversion for either sequencer. Which event triggers the start of conversion is configured in the Event-Trigger submodule of the ePWM.
- **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

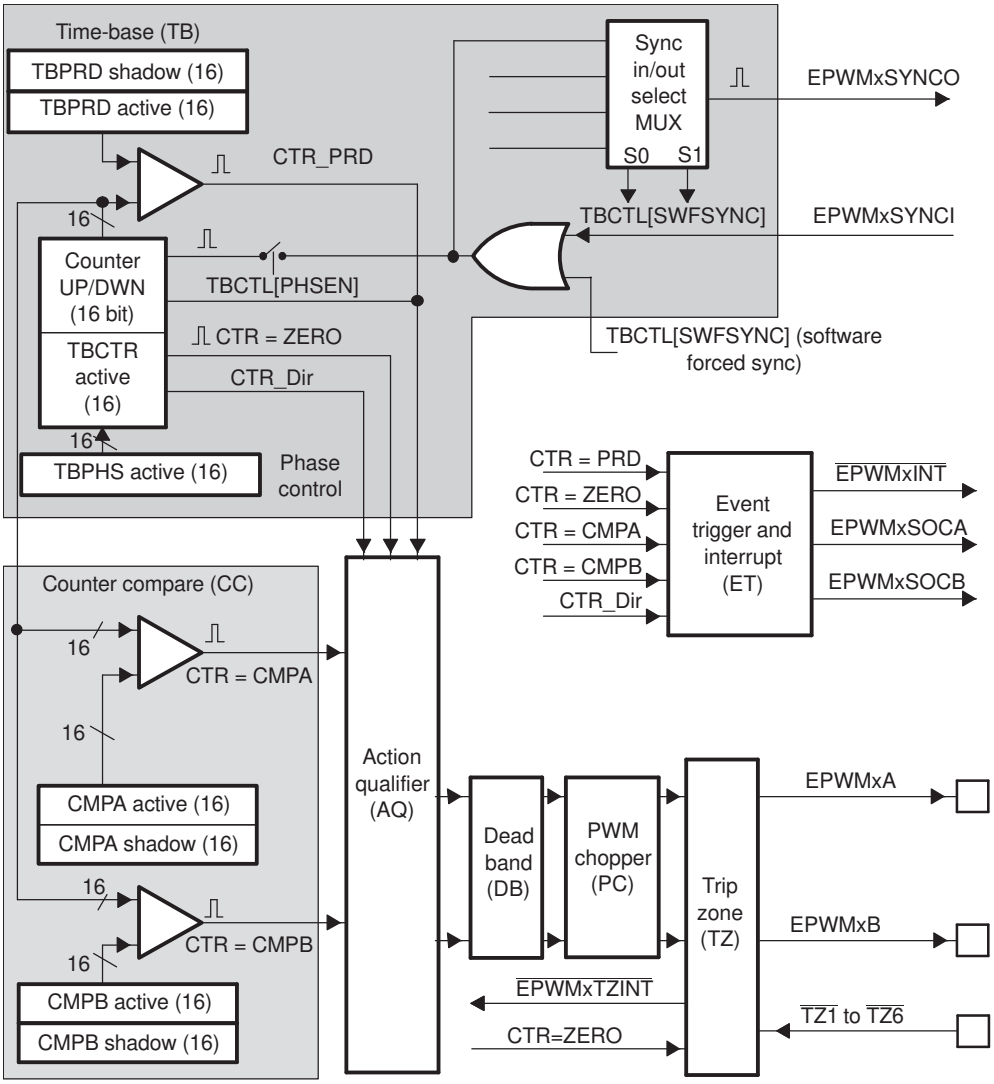


Figure 3-3 also shows the key internal submodule interconnect signals. Each submodule is described in detail in its respective section.

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 datasheet.

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
					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
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]

(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) chapter. See the datasheet to determine which instances include the HRPWM.

(3) EALLOW protected registers as described in the specific device version of the *System Control and Interrupts chapter* .

3.2 ePWM Submodules

Seven 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 seven 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.
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 pins . • 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 pins : – 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

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 which can be found in C2000Ware.

Example 3-1. Constant Definitions Used in the Code Examples

```
// TBCTL (Time-Base Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// TBCTR MODE bits
#define    TB_COUNT_UP        0x0
#define    TB_COUNT_DOWN      0x1
#define    TB_COUNT_UPDOWN    0x2
#define    TB_FREEZE          0x3
// PHSEN bit
#define    TB_DISABLE         0x0
#define    TB_ENABLE          0x1
// PRDLD bit
#define    TB_SHADOW          0x0
#define    TB_IMMEDIATE       0x1
// SYNCOSEL bits
#define    TB_SYNC_IN         0x0
#define    TB_CTR_ZERO        0x1
#define    TB_CTR_CMPB        0x2
#define    TB_SYNC_DISABLE    0x3
// HSPCLKDIV and CLKDIV bits
#define    TB_DIV1            0x0
#define    TB_DIV2            0x1
#define    TB_DIV4            0x2
// PHSDIR bit
#define    TB_DOWN            0x0
#define    TB_UP              0x1
// CMPCTL (Compare Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// LOADAMODE and LOADBMODE bits
#define    CC_CTR_ZERO        0x0
#define    CC_CTR_PRD         0x1
#define    CC_CTR_ZERO_PRD    0x2
#define    CC_LD_DISABLE      0x3
// SHDWAMODE and SHDWBMODE bits
#define    CC_SHADOW          0x0
#define    CC_IMMEDIATE       0x1
// AQCTLA and AQCTLB (Action-qualifier Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// ZRO, PRD, CAU, CAD, CBU, CBD bits
#define    AQ_NO_ACTION       0x0
#define    AQ_CLEAR           0x1
#define    AQ_SET             0x2
#define    AQ_TOGGLE          0x3
// DBCTL (Dead-Band Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// MODE bits
#define    DB_DISABLE         0x0
#define    DBA_ENABLE         0x1
#define    DBB_ENABLE         0x2
#define    DB_FULL_ENABLE     0x3
// POLSEL bits
#define    DB_ACTV_HI         0x0
#define    DB_ACTV_LOC        0x1
#define    DB_ACTV_HIC        0x2

#define    DB_ACTV_LO         0x3
// PCCTL (chopper control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// CHPEN bit
#define    CHP_ENABLE         0x0
#define    CHP_DISABLE        0x1
// CHPFREQ bits
#define    CHP_DIV1           0x0
#define    CHP_DIV2           0x1
#define    CHP_DIV3           0x2
```

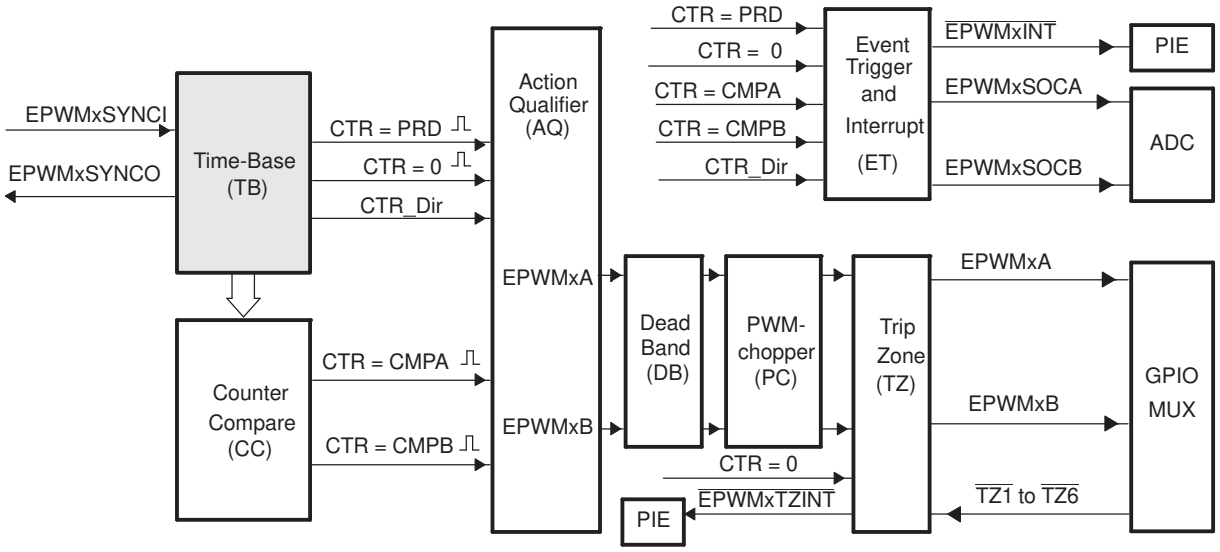
Example 3-1. Constant Definitions Used in the Code Examples (continued)

```
#define    CHP_DIV4          0x3
#define    CHP_DIV5          0x4
#define    CHP_DIV6          0x5
#define    CHP_DIV7          0x6
#define    CHP_DIV8          0x7
// CHPDUTY bits
#define    CHP1_8TH          0x0
#define    CHP2_8TH          0x1
#define    CHP3_8TH          0x2
#define    CHP4_8TH          0x3
#define    CHP5_8TH          0x4
#define    CHP6_8TH          0x5
#define    CHP7_8TH          0x6
// TZSEL (Trip-zone Select)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// CBCn and OSHTn bits
#define    TZ_ENABLE         0x0
#define    TZ_DISABLE        0x1
// TZCTL (Trip-zone Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// TZA and TZB bits
#define    TZ_HIZ            0x0
#define    TZ_FORCE_HI       0x1
#define    TZ_FORCE_LO       0x2
#define    TZ_DISABLE        0x3
// ETSEL (Event-trigger Select)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// INTSEL, SOCASEL, SOCBSEL bits
#define    ET_CTR_ZERO       0x1
#define    ET_CTR_PRD        0x2
#define    ET_CTRU_CMPA      0x4
#define    ET_CTRD_CMPA      0x5
#define    ET_CTRU_CMPB      0x6
#define    ET_CTRD_CMPB      0x7
// ETPS (Event-trigger Prescale)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// INTPRD, SOCAPRD, SOCBPRD bits
#define    ET_DISABLE        0x0
#define    ET_1ST            0x1
#define    ET_2ND            0x2
#define    ET_3RD            0x3
```

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

(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 High-Resolution Pulse Width Modulator (HRPWM) chapter. See the datasheet to determine which ePWM instances include this feature.

The block diagram in Figure 3-5 shows the critical signals and registers of the time-base submodule. Table 3-4 provides descriptions of the key signals associated with the time-base submodule.

Figure 3-5. Time-Base Submodule Signals and Registers

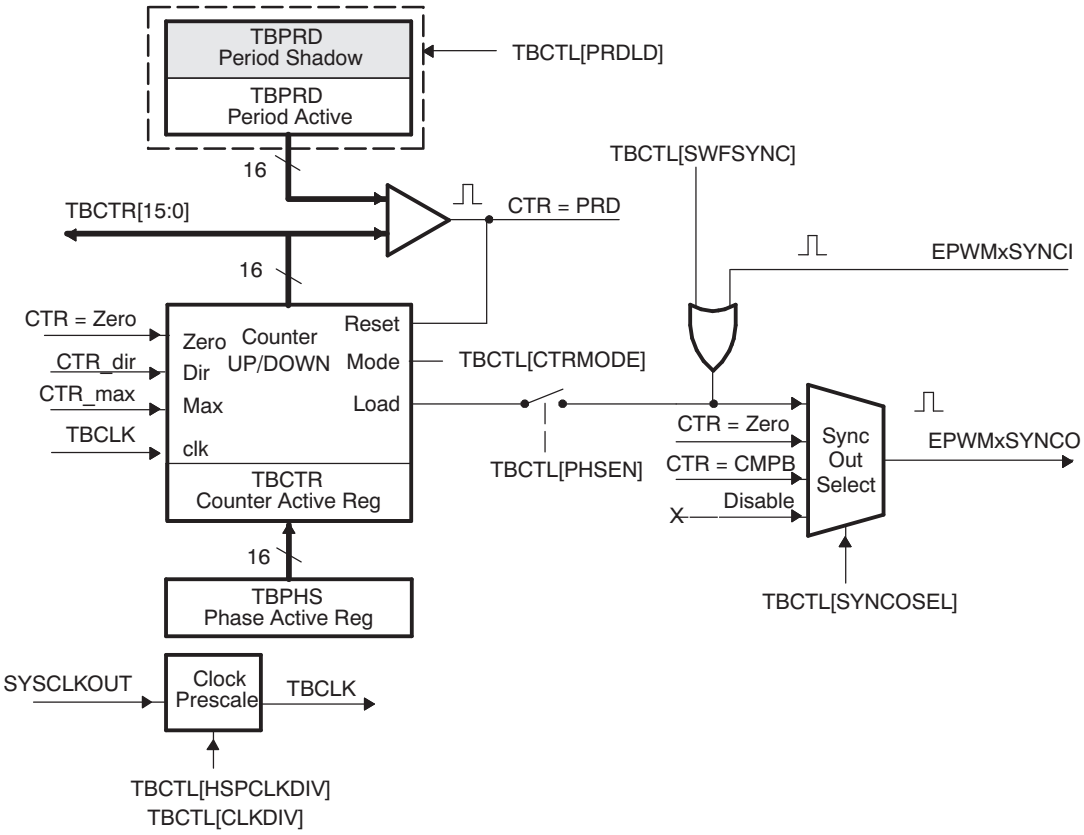


Table 3-4. Key Time-Base Signals

Signal	Description
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.

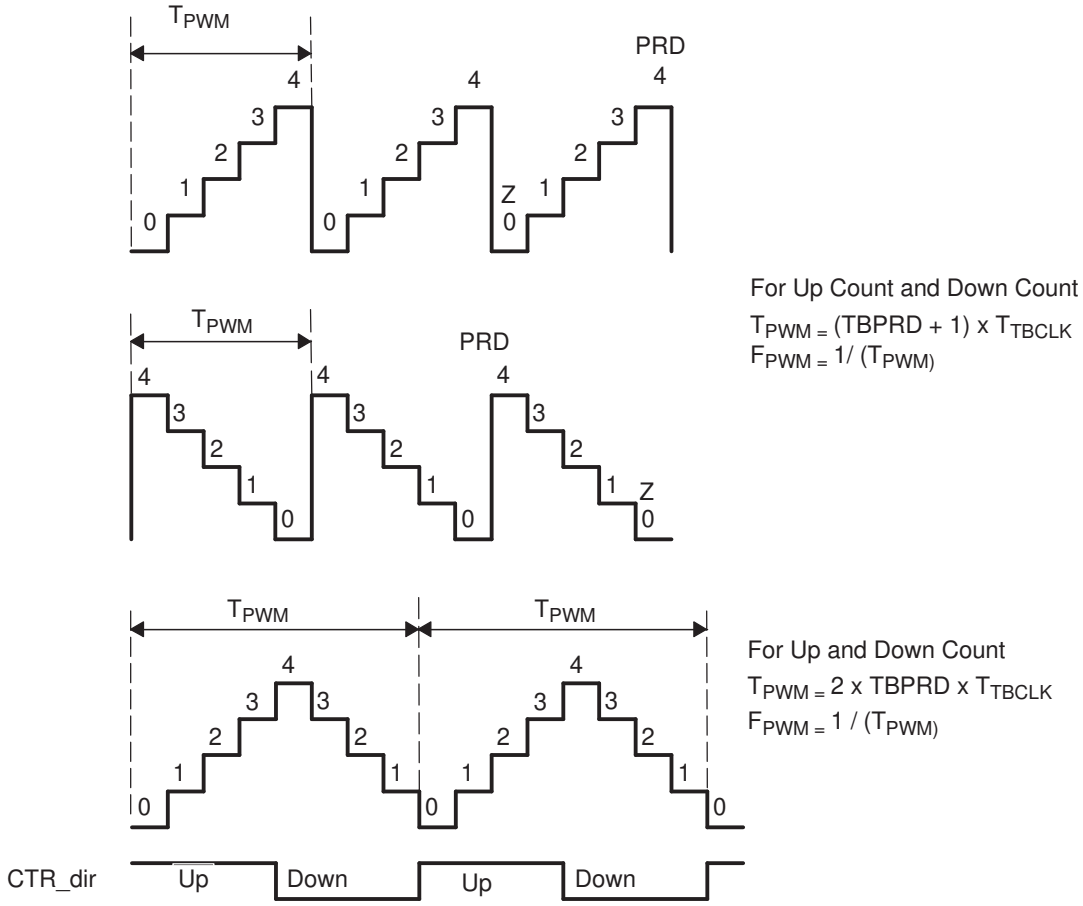
- **Up-Count Mode:**

  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



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.

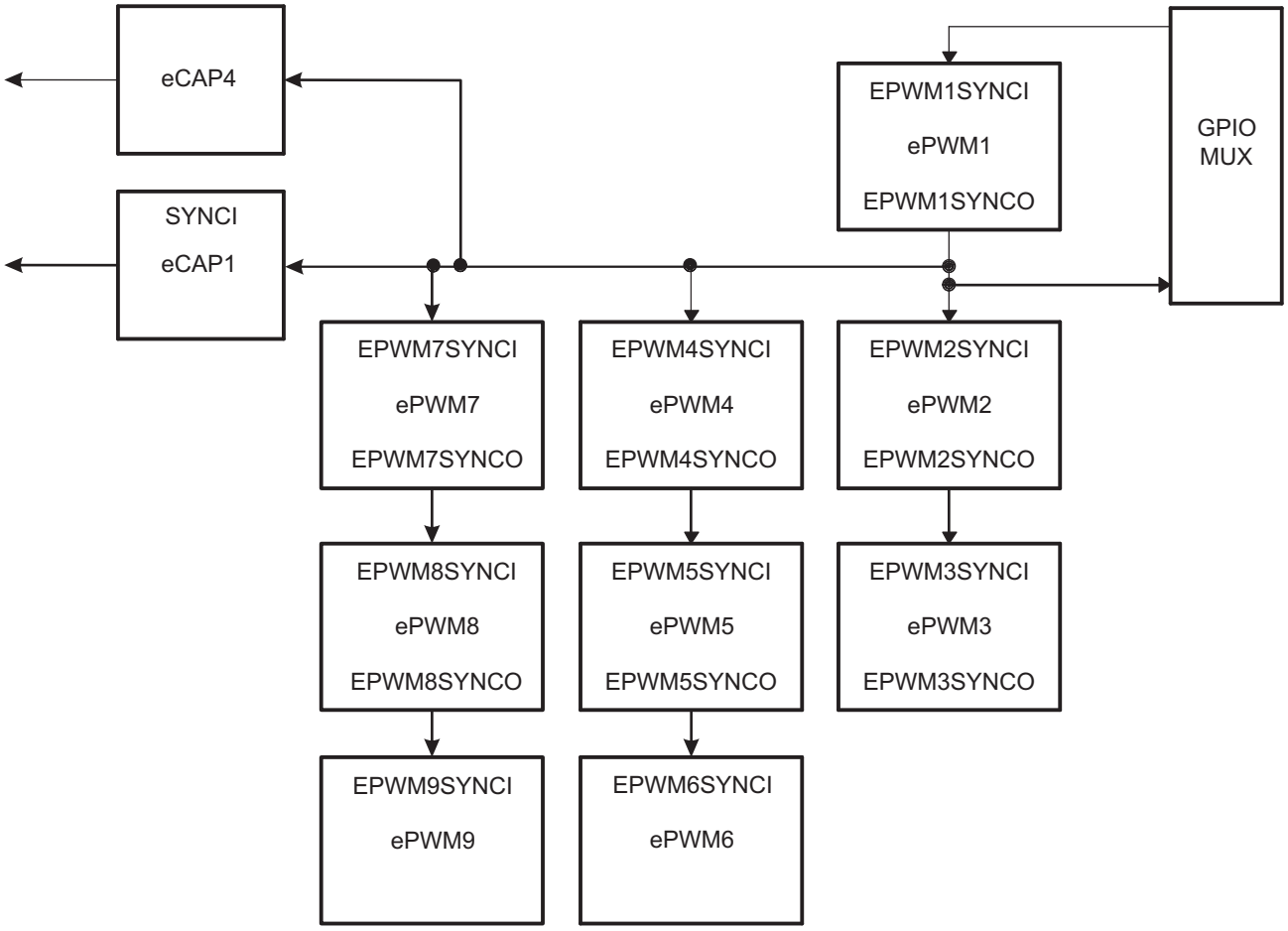
The proper procedure for enabling ePWM clocks is as follows:

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 synchronization connections for the remaining ePWM modules are shown in Figure 3-7

Figure 3-7. Time-Base Counter Synchronization Scheme



NOTE: All modules shown in the synchronization schemes may not be available on all devices. Please refer to the datasheet 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.

- 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 3-8 through Figure 3-11 for examples.

Clearing the TBCTL[PHSEN] bit configures the ePWM to ignore the synchronization input pulse. The synchronization pulse can still be allowed to flow-through to the EPWMxSYNCO and be used to synchronize other ePWM modules. In this way, you can set up a master time-base (for example, ePWM1) and downstream modules (ePWM2 - ePWMx) may elect to run in synchronization with the master. See 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 specific device version of the *System Control and Interrupts chapter*. 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 specific device version of the *System Control and Interrupts chapter* .
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 3-8. Time-Base Up-Count Mode Waveforms

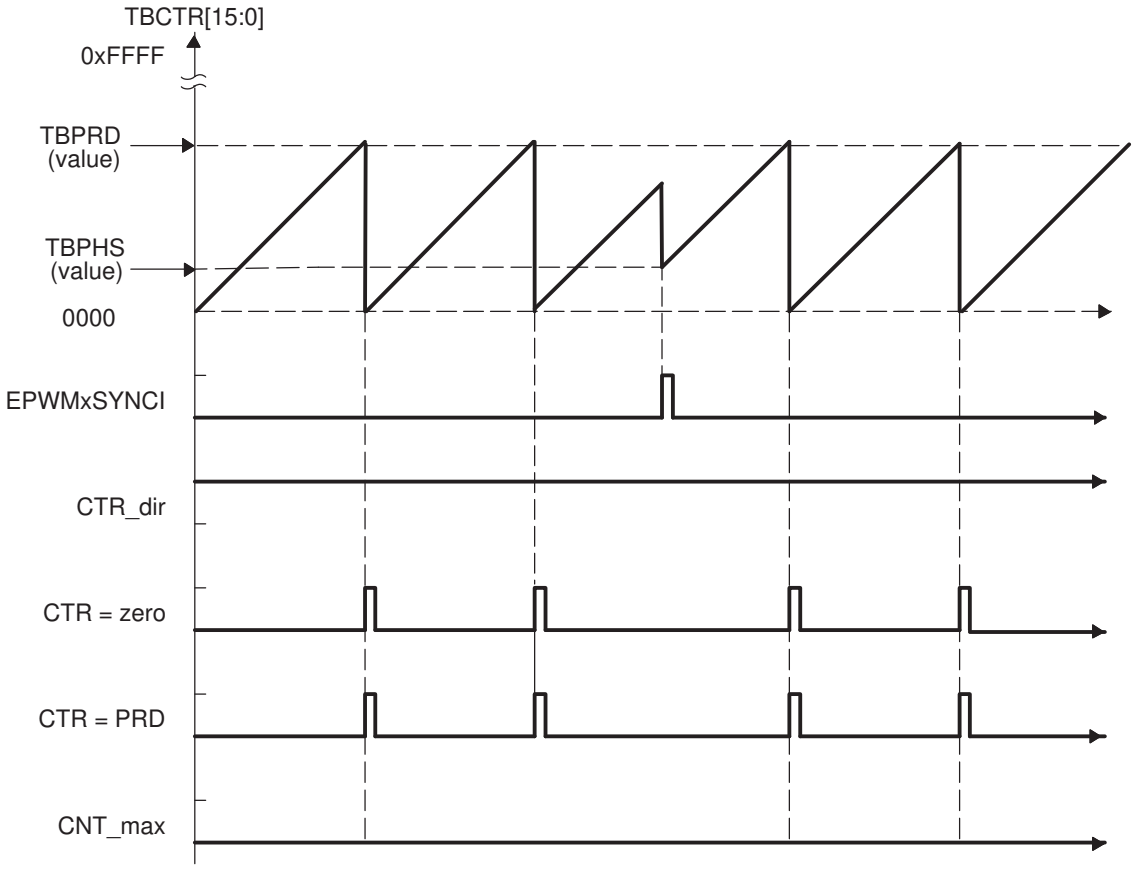


Figure 3-9. Time-Base Down-Count Mode Waveforms

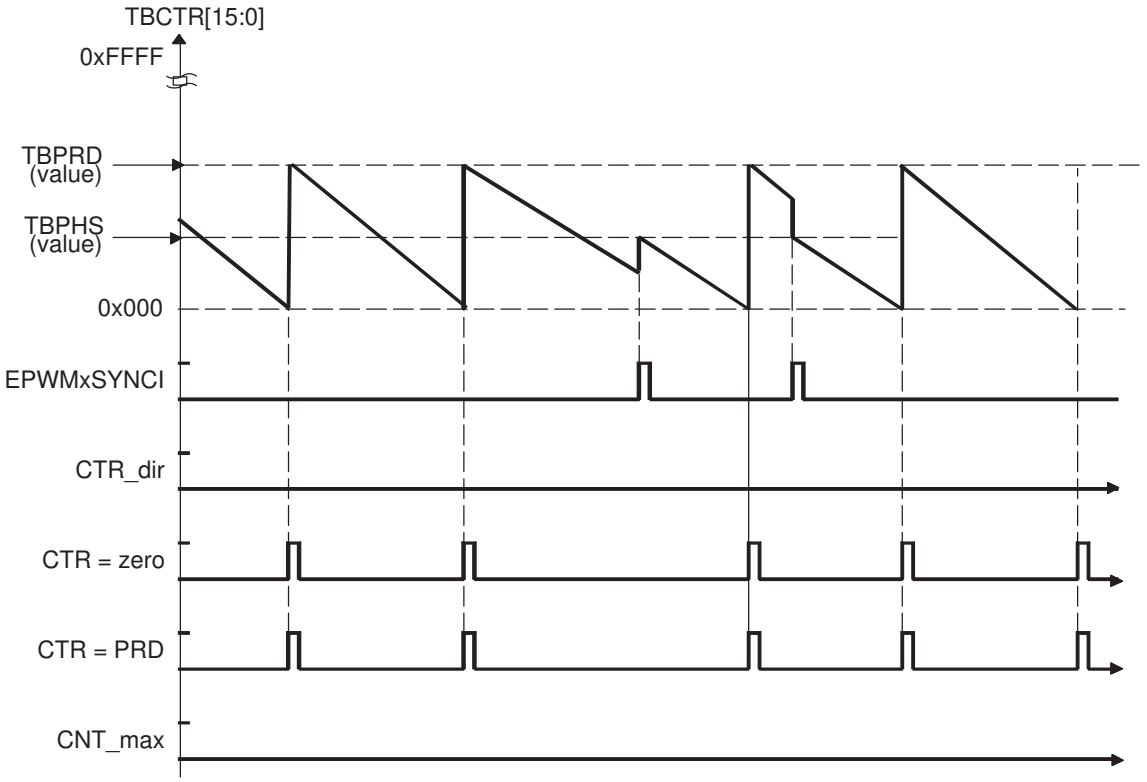


Figure 3-10. Time-Base Up-Down-Count Waveforms, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

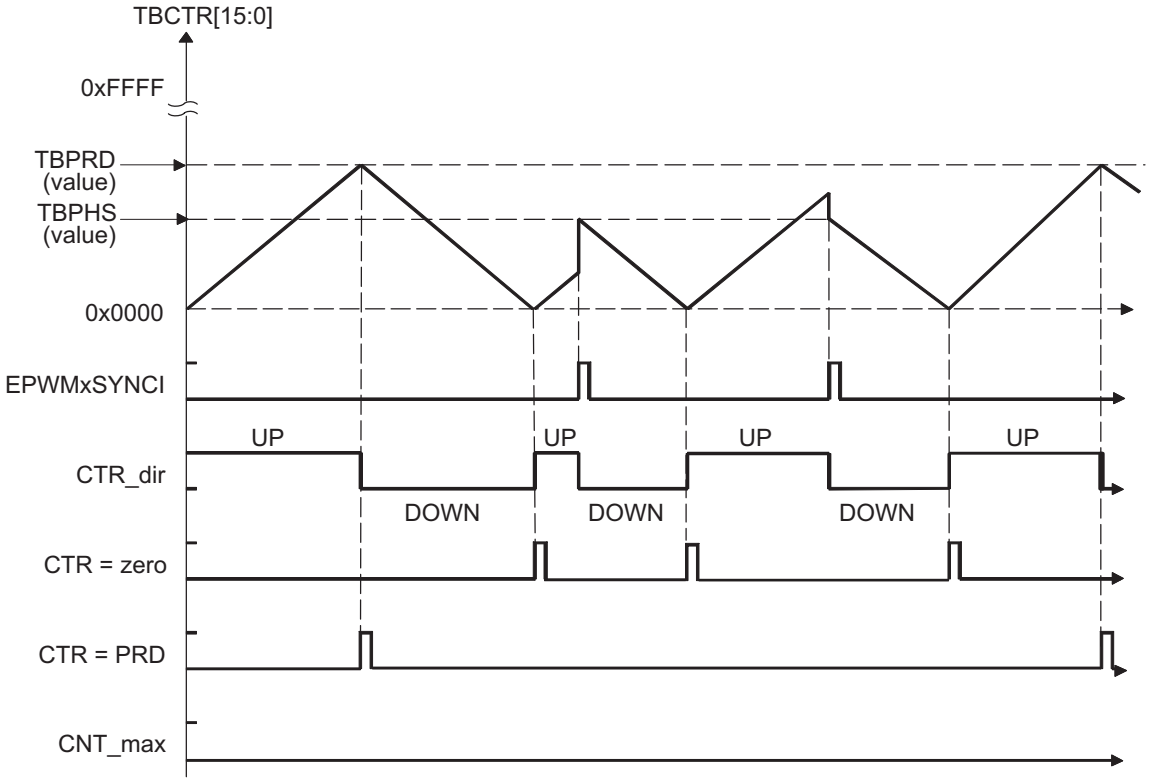
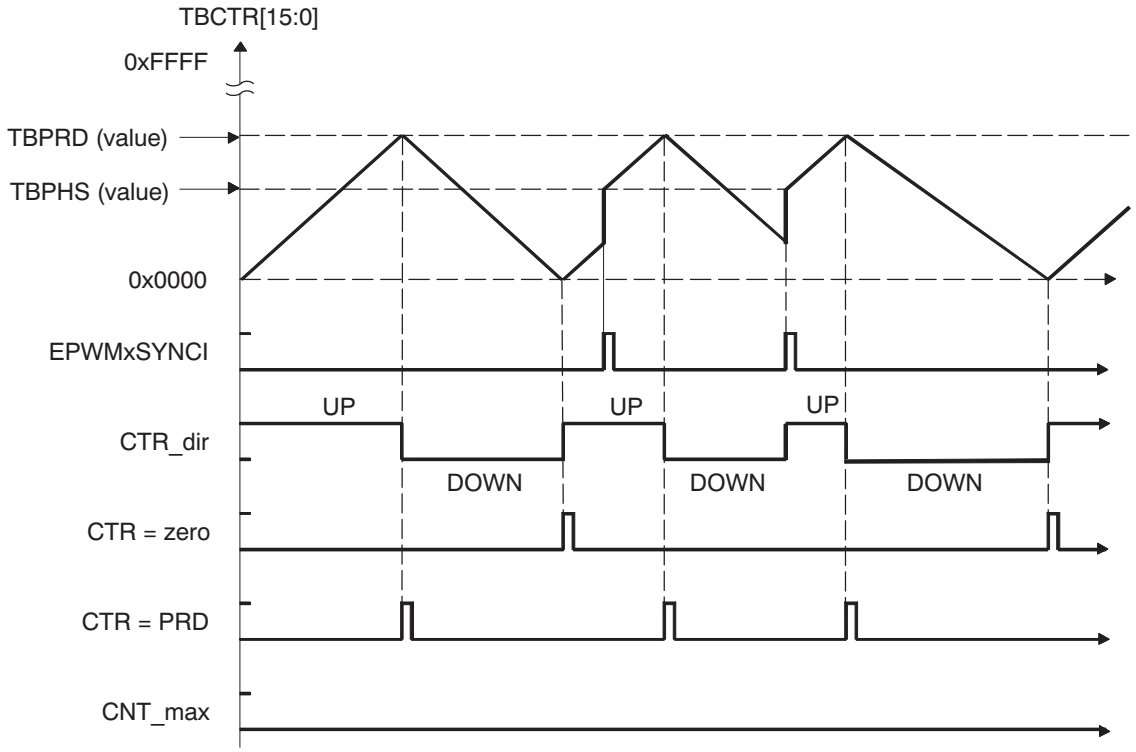


Figure 3-11. Time-Base Up-Down Count Waveforms, TBCTL[PHSDIR = 1] Count Up On Synchronization Event



3.2.3 Counter-Compare (CC) Submodule

Figure 3-12 illustrates the counter-compare submodule within the ePWM.

Figure 3-12. Counter-Compare Submodule

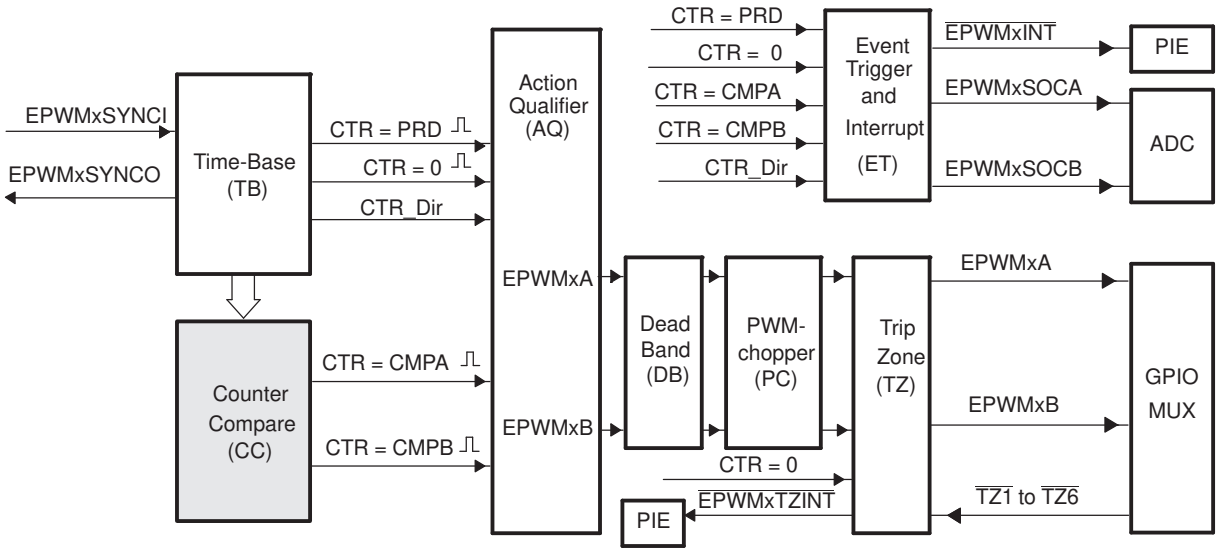


Figure 3-13 shows the basic structure of the counter-compare submodule.

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

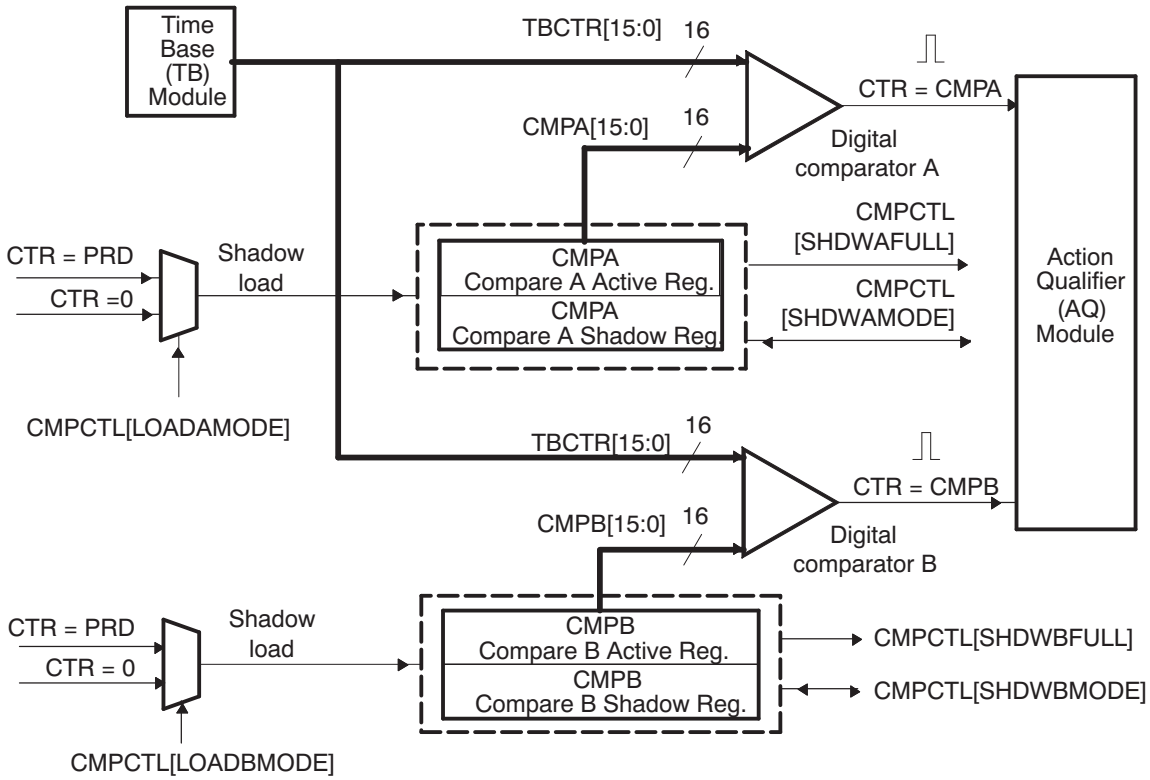
The counter-compare submodule operation is controlled and monitored by the registers shown in Table 3-5:

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

(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 High-Resolution Pulse Width Modulator (HRPWM) chapter. Refer to the datasheet to determine which ePWM instances include this feature.

Figure 3-13. 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.

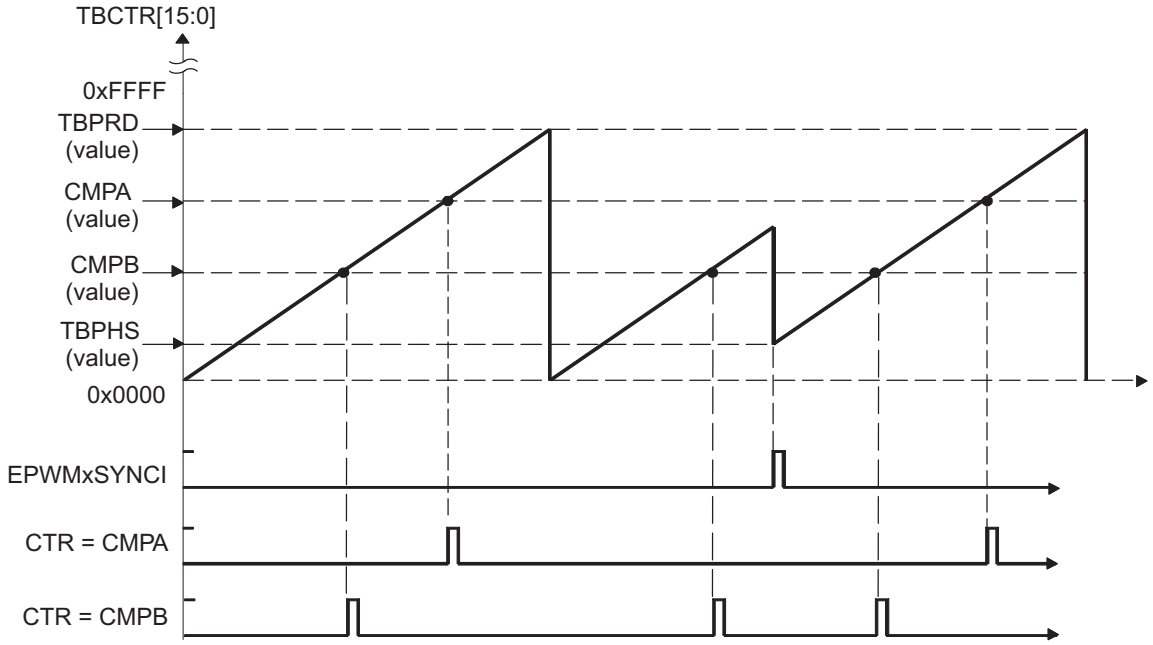
3.2.3.4 Count Mode Timing Waveforms

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 3-14 through Figure 3-17 show when events are generated and how the EPWMxSYNCI signal interacts.

Figure 3-14. 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 3-15. Counter-Compare Events in Down-Count Mode

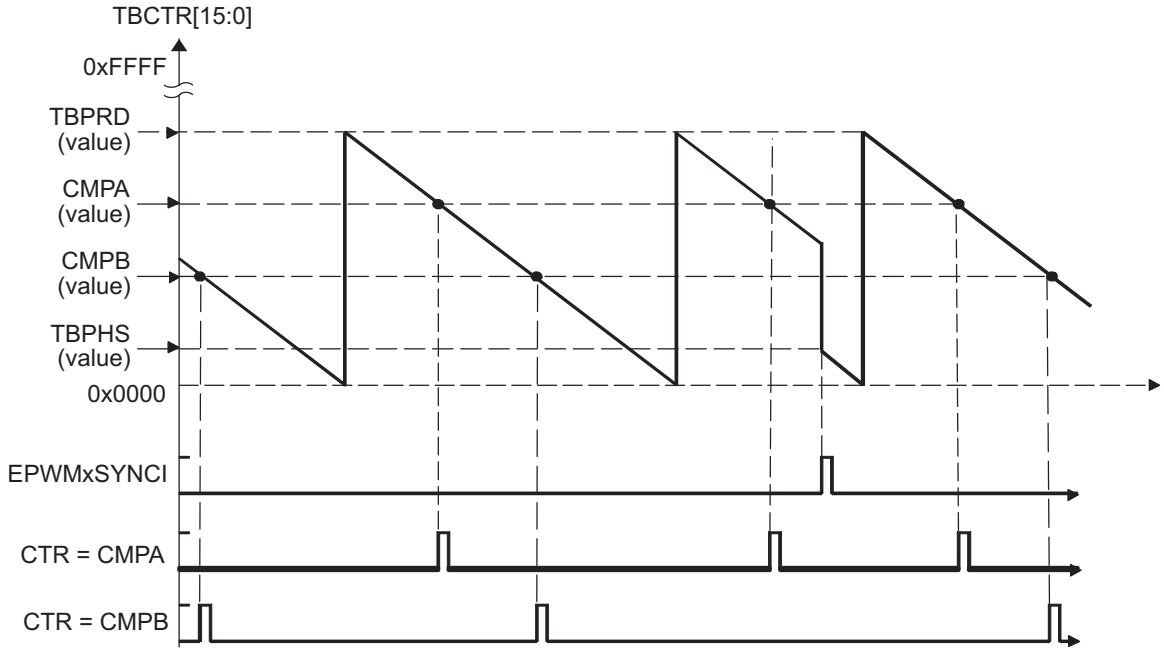


Figure 3-16. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

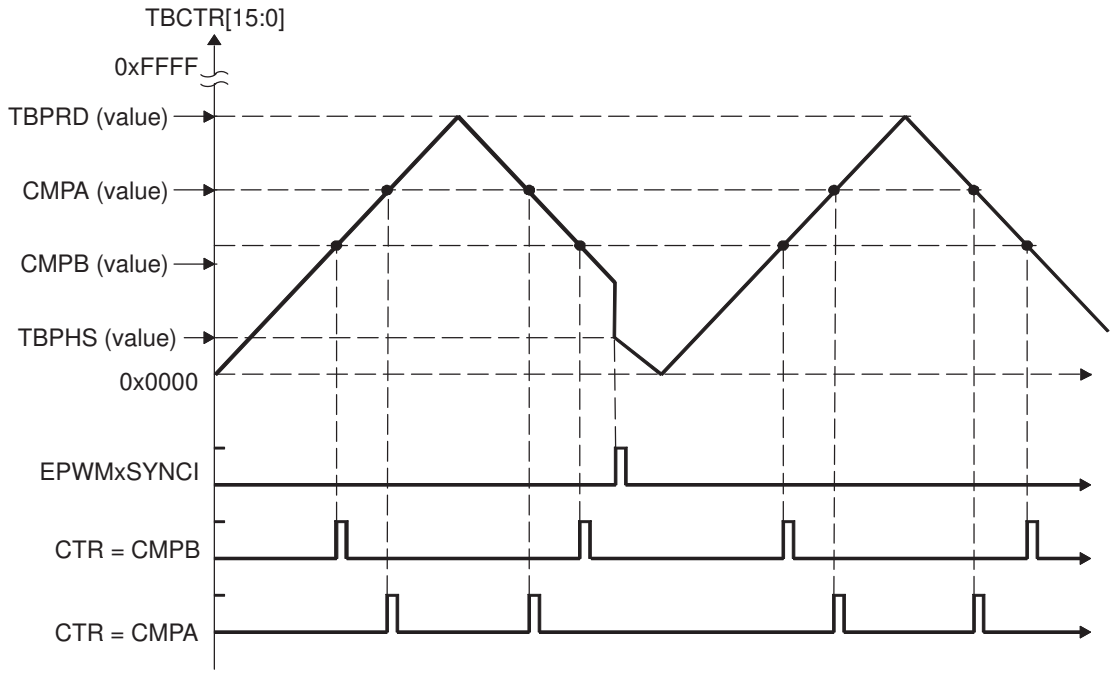
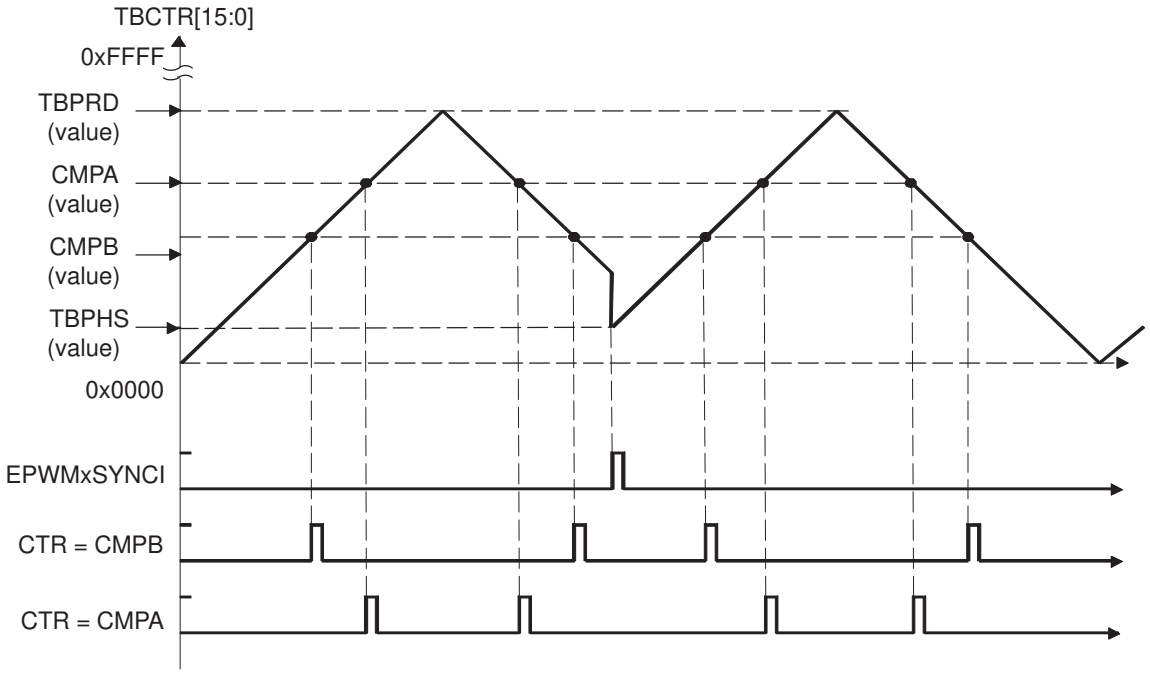


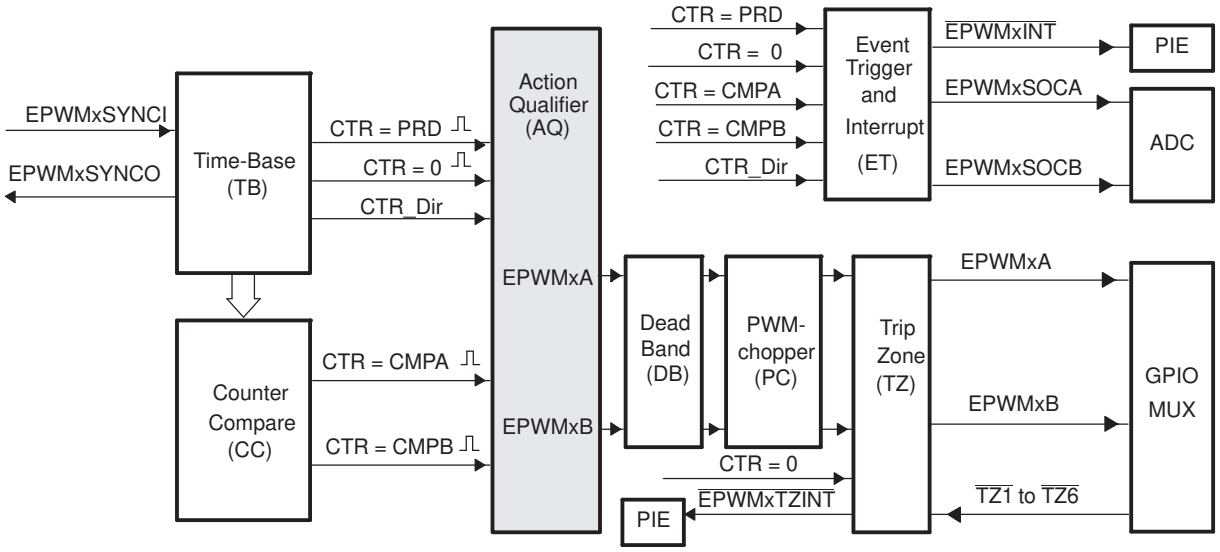
Figure 3-17. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 1] Count Up On Synchronization Event



3.2.4 Action-Qualifier (AQ) Submodule

Figure 3-18 shows the action-qualifier (AQ) submodule (see shaded block) in the ePWM system.

Figure 3-18. 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

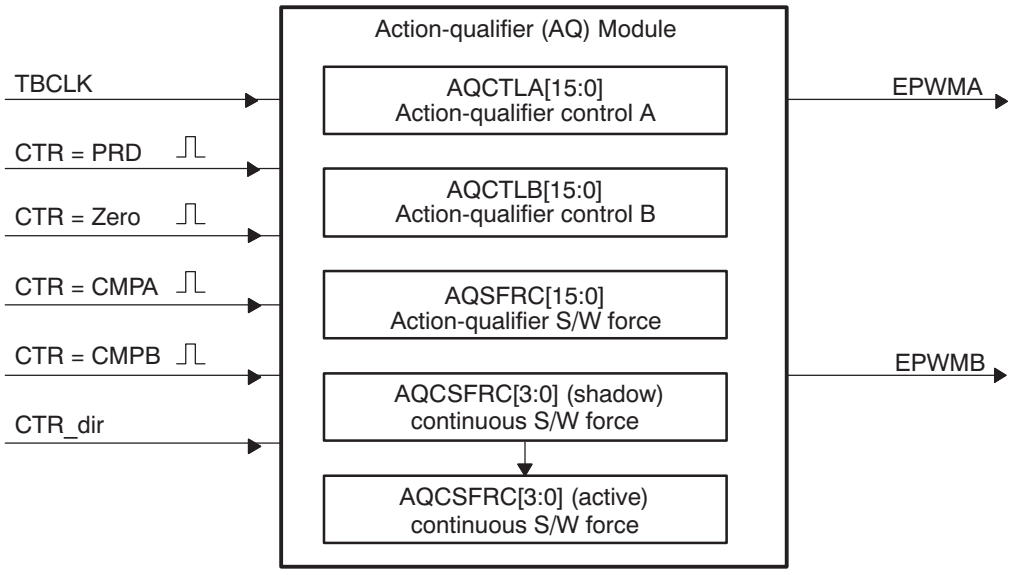
The action-qualifier submodule operation is controlled and monitored via the registers in Table 3-7.

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 3-19. Action-Qualifier Submodule Inputs and Outputs



For convenience, the possible input events are summarized again in Table 3-8.

Table 3-8. Action-Qualifier Submodule Possible Input Events

Signal	Description	Registers Compared
CTR = PRD	Time-base counter equal to the period value	TBCTR = TBPRD
CTR = Zero	Time-base counter equal to zero	TBCTR = 0x0000
CTR = CMPA	Time-base counter equal to the counter-compare A	TBCTR = CMPA
CTR = CMPB	Time-base counter equal to the counter-compare B	TBCTR = CMPB
Software forced event	Asynchronous event initiated by software	

The software forced action is a useful asynchronous event. This control is handled by registers AQSFRC and AQCSFRC.

The action-qualifier submodule controls how the two outputs EPWMxA and EPWMxB behave when a particular event occurs. The event inputs to the action-qualifier submodule are further qualified by the counter direction (up or down). This allows for independent action on outputs on both the count-up and count-down phases.

The possible actions imposed on outputs EPWMxA and EPWMxB are:

- **Set High:**

  Set output EPWMxA or EPWMxB to a high level.
- **Clear Low:**

  Set output EPWMxA or EPWMxB to a low level.
- **Toggle:**

  If EPWMxA or EPWMxB is currently pulled high, then pull the output low. If EPWMxA or EPWMxB is currently pulled low, then pull the output high.
- **Do Nothing:**

  Keep outputs EPWMxA and EPWMxB at same level as currently set. Although the "Do Nothing" option prevents an event from causing an action on the EPWMxA and EPWMxB outputs, this event can still trigger interrupts and ADC start of conversion. See the Event-trigger Submodule description in Section 3.2.8 for details.

Actions are specified independently for either output (EPWMxA or EPWMxB). Any or all events can be configured to generate actions on a given output. For example, both CTR = CMPA and CTR = CMPB can operate on output EPWMxA. All qualifier actions are configured via the control registers found at the end of this section.

For clarity, the drawings in this document use a set of symbolic actions. These symbols are summarized in Figure 3-20. Each symbol represents an action as a marker in time. Some actions are fixed in time (zero and period) while the CMPA and CMPB actions are moveable and their time positions are programmed via the counter-compare A and B registers, respectively. To turn off or disable an action, use the "Do Nothing option"; it is the default at reset.

Figure 3-20. 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 CAU/CBU	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 CAU/CBU	Compare on Down-Count Event CAD/CBD
Down-Count Mode	Never occurs.	If CMPA/CMPB < TBPRD, the event will occur on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event will occur on a period match (TBCTR=TBPRD).
Up-Down-Count Mode	If CMPA/CMPB < TBPRD and the counter is incrementing, the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB is ≥ TBPRD, the event will occur on a period match (TBCTR = TBPRD).	If CMPA/CMPB < TBPRD and the counter is decrementing, the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event occurs on a period match (TBCTR=TBPRD).

3.2.4.4 Waveforms for Common Configurations

NOTE: The waveforms in this document show the ePWMs behavior for a static compare register value. In a running system, the active compare registers (CMPA and CMPB) are typically updated from their respective shadow registers once every period. The user specifies when the update will take place; either when the time-base counter reaches zero or when the time-base counter reaches period. There are some cases when the action based on the new value can be delayed by one period or the action based on the old value can take effect for an extra period. Some PWM configurations avoid this situation. These include, but are not limited to, the following:

Use up-down-count mode to generate a symmetric PWM:

- If you load CMPA/CMPB on zero, then use CMPA/CMPB values greater than or equal to 1.
- If you load CMPA/CMPB on period, then use CMPA/CMPB values less than or equal to TBPRD-1.

  This means there will always be a pulse of at least one TBCLK cycle in a PWM period which, when very short, tend to be ignored by the system.

Use up-down-count mode to generate an asymmetric PWM:

- To achieve 50%-0% asymmetric PWM use the following configuration: Load CMPA/CMPB on period and use the period action to clear the PWM and a compare-up action to set the PWM. Modulate the compare value from 0 to TBPRD to achieve 50%-0% PWM duty.

When using up-count mode to generate an asymmetric PWM:

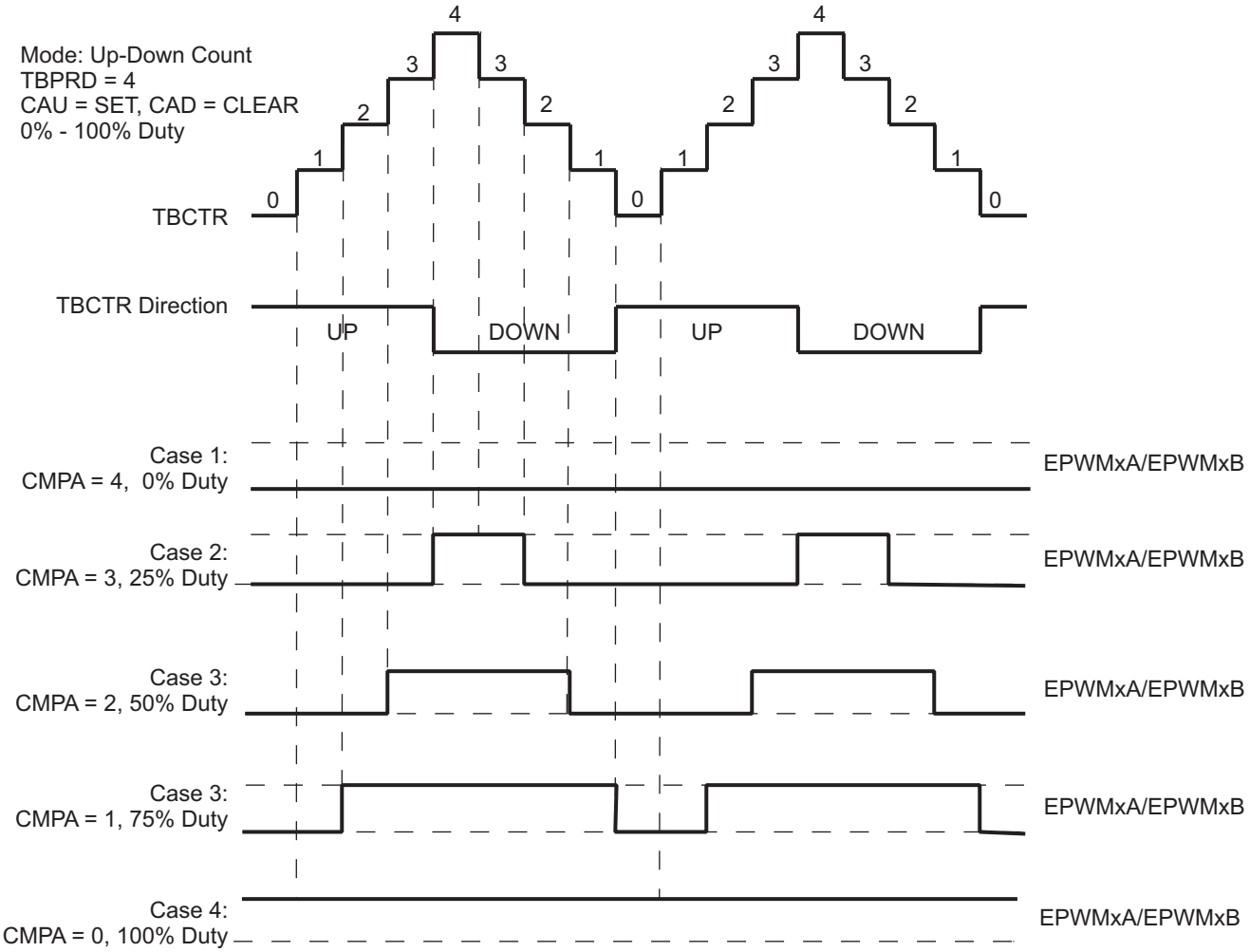
- To achieve 0-100% asymmetric PWM use the following configuration: Load CMPA/CMPB on TBPRD. Use the Zero action to set the PWM and a compare-up action to clear the PWM. Modulate the compare value from 0 to TBPRD+1 to achieve 0-100% PWM duty.

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

Figure 3-21 shows how a symmetric PWM waveform can be generated using the up-down-count mode of the TBCTR. In this mode 0%-100% DC modulation is achieved by using equal compare matches on the up count and down count portions of the waveform. In the example shown, CMPA is used to make the comparison. When the counter is incrementing the CMPA match will pull the PWM output high. Likewise, when the counter is decrementing the compare match will pull the PWM signal low. When CMPA = 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.

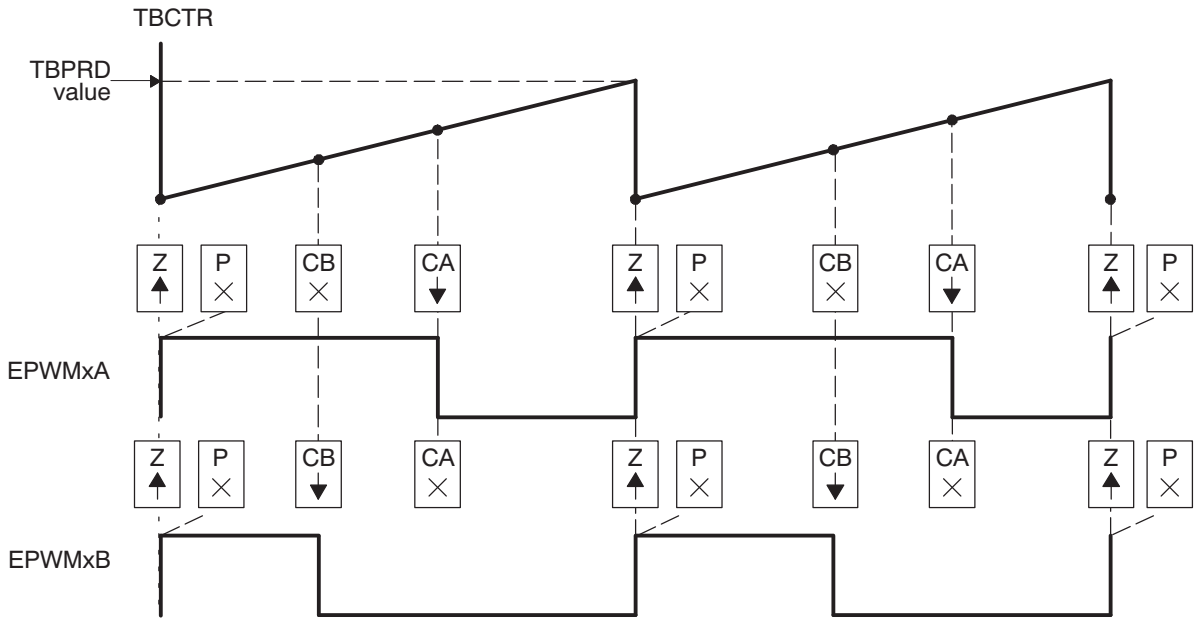
Figure 3-21. Up-Down-Count Mode Symmetrical Waveform



The PWM waveforms in Figure 3-22 through Figure 3-27 show some common action-qualifier configurations. The C-code samples in Example 3-2 through Example 3-7 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 3-22. 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 3-2 contains a code sample showing initialization and run time for the waveforms in Figure 3-22.

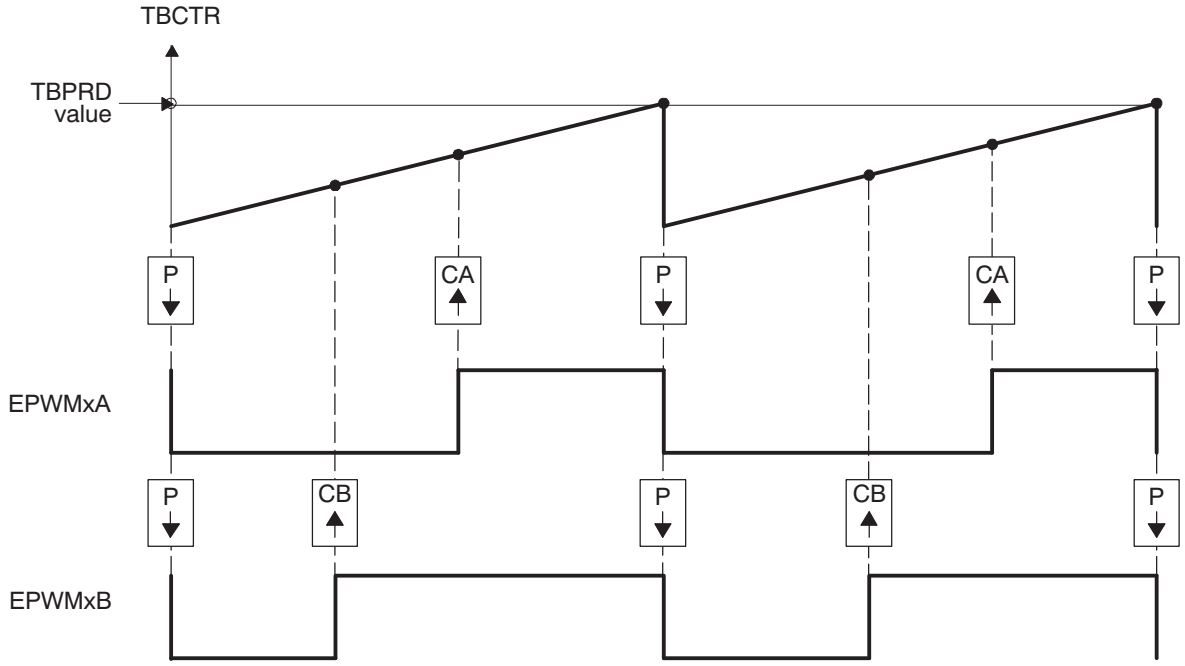
Example 3-2. Code Sample for Figure 3-22

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                          // Period = 601 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350;                 // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 200;                           // Compare B = 200 TBCLK counts
EPwm1Regs.TBPHS = 0;                            // Set Phase register to zero
EPwm1Regs.TBCTR = 0;                            // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;         // Phase loading disabled
EPwm1Regs.TBCTL.bit.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
```

Example 3-2. Code Sample for Figure 3-22 (continued)

```
EPwm1Regs.CMPB = Duty1B;                    // adjust duty for output EPWM1B
```

Figure 3-23. 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).

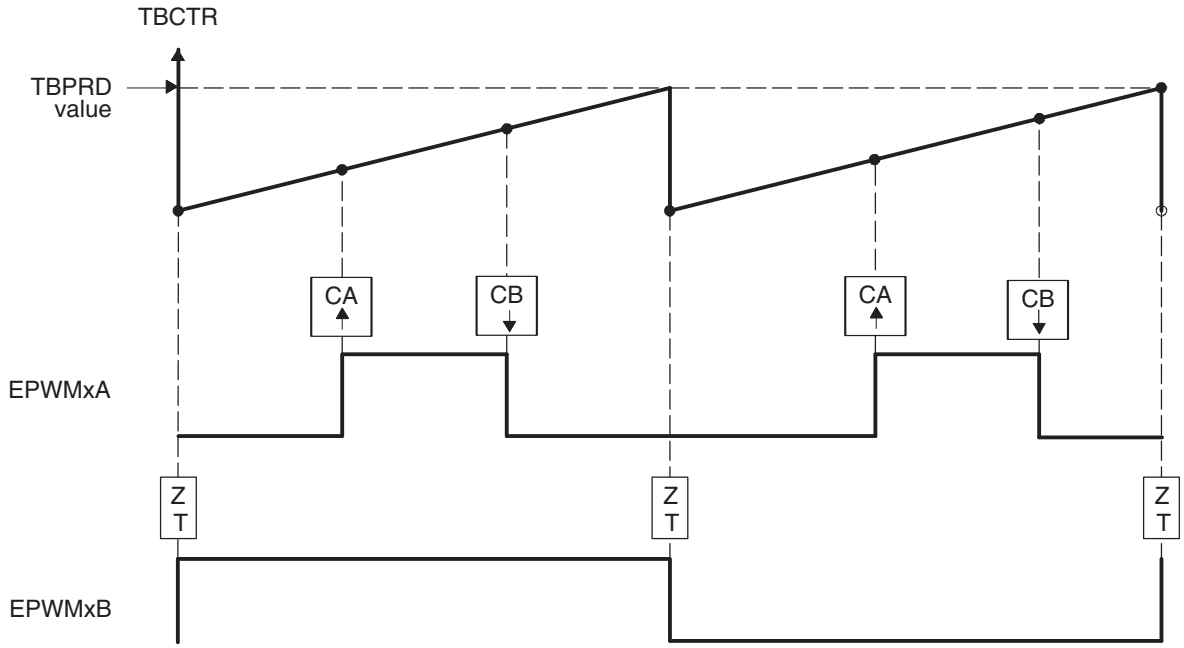
D Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

Example 3-3 contains a code sample showing initialization and run time for the waveforms in Figure 3-23.

Example 3-3. Code Sample for Figure 3-23

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                        // Period = 601 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350;               // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 200;                         // Compare B = 200 TBCLK counts
EPwm1Regs.TBPHS = 0;                          // Set Phase register to zero
EPwm1Regs.TBCTR = 0;                          // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Phase loading disabled
EPwm1Regs.TBCTL.bit.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 3-24. 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)

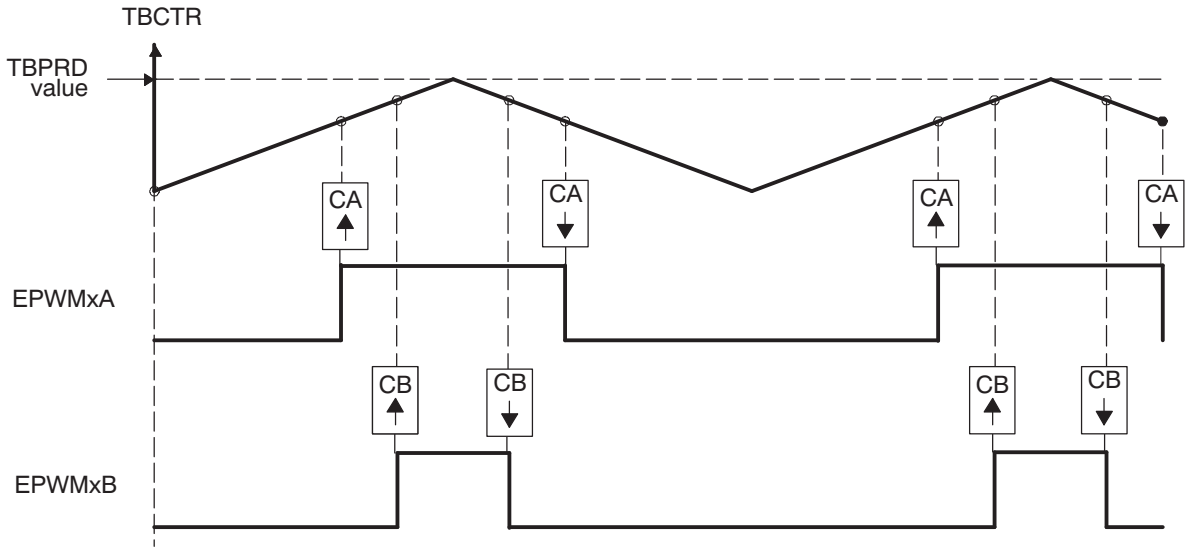
D EPWMxB can be used to generate a 50% duty square wave with frequency = ⁄ × ((TBPRD + 1) × TBCLK)

Example 3-4 contains a code sample showing initialization and run time for the waveforms Figure 3-24. Use the code in Example 3-1 to define the headers.

Example 3-4. Code Sample for Figure 3-24

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                        // Period = 601 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 200;               // Compare A = 200 TBCLK counts
EPwm1Regs.CMPB = 400;                         // Compare B = 400 TBCLK counts
EPwm1Regs.TBPHS = 0;                          // Set Phase register to zero
EPwm1Regs.TBCTR = 0;                          // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Phase loading disabled
EPwm1Regs.TBCTL.bit.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 3-25. 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).

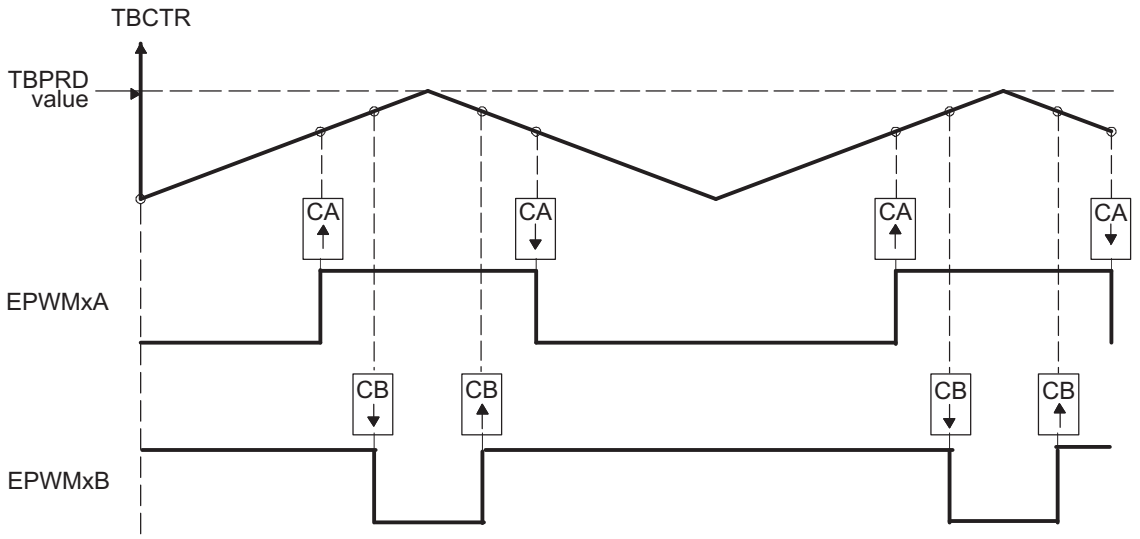
D Outputs EPWMxA and EPWMxB can drive independent power switches

Example 3-5 contains a code sample showing initialization and run time for the waveforms in Figure 3-25. Use the code in Example 3-1 to define the headers.

Example 3-5. Code Sample for Figure 3-25

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                          // Period = 2´600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 400;                 // Compare A = 400 TBCLK counts
EPwm1Regs.CMPB = 500;                           // Compare B = 500 TBCLK counts
EPwm1Regs.TBPHS = 0;                            // Set Phase register to zero
EPwm1Regs.TBCTR = 0;                            // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetric
xEPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Phase loading disabled
xEPwm1Regs.TBCTL.bit.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 3-26. 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

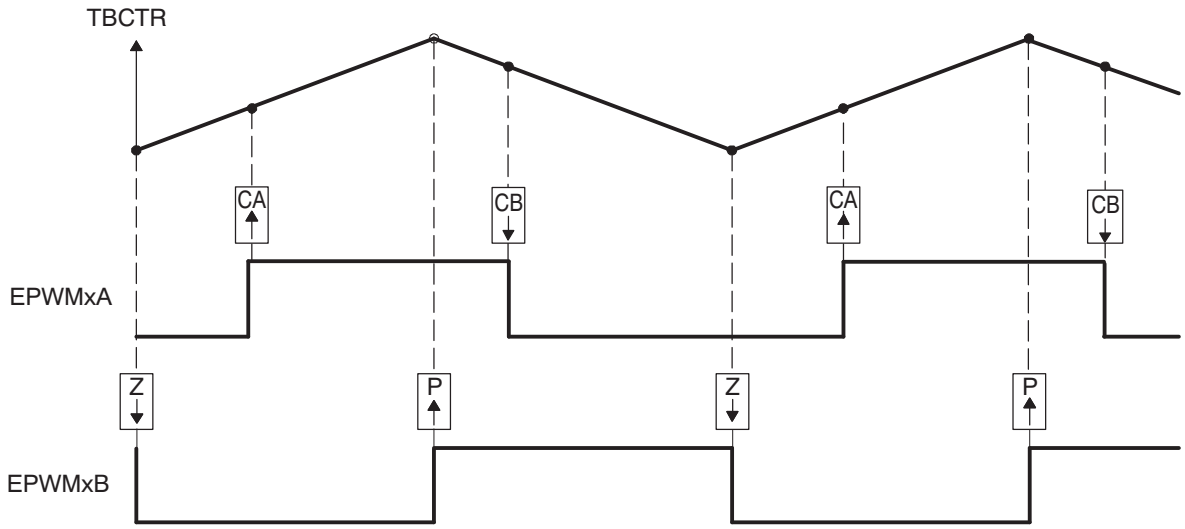
E Dead-band = CMPB - CMPA (fully programmable edge placement by software). Note the dead-band module is also available if the more classical edge delay method is required.

Example 3-6 contains a code sample showing initialization and run time for the waveforms in Figure 3-26. Use the code in Example 3-1 to define the headers.

Example 3-6. Code Sample for Figure 3-26

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                          // Period = 2´600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350;                 // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 400;                           // Compare B = 400 TBCLK counts
EPwm1Regs.TBPHS = 0;                            // Set Phase register to zero
EPwm1Regs.TBCTR = 0;                            // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetric
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Phase loading disabled
EPwm1Regs.TBCTL.bit.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 3-27. 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 3-7 contains a code sample showing initialization and run time for the waveforms in Figure 3-27. Use the code in Example 3-1 to define the headers.

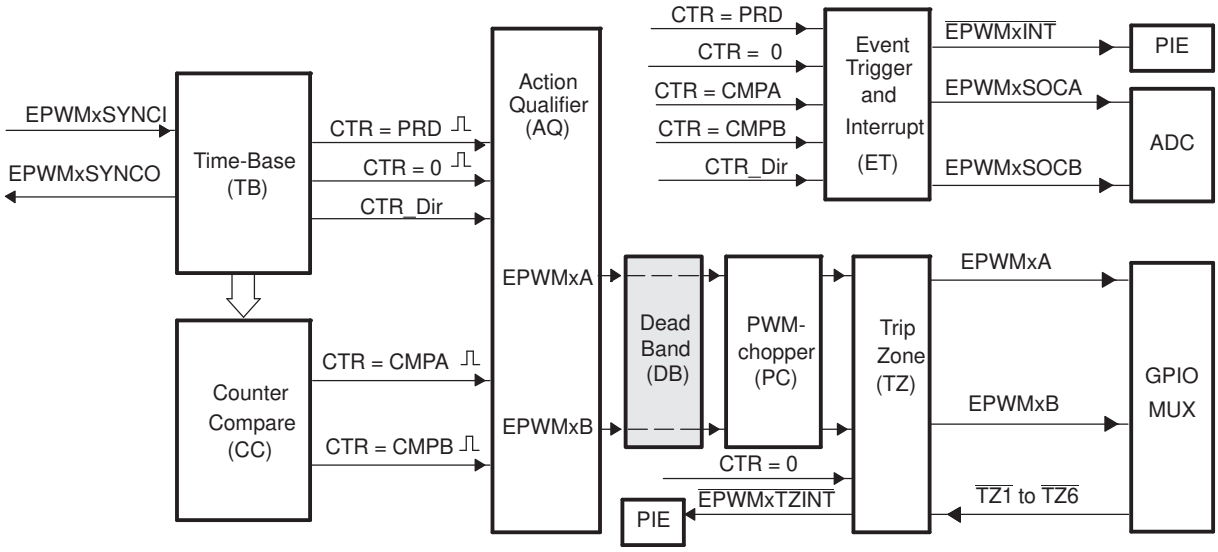
Example 3-7. Code Sample for Figure 3-27

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                          //  Period = 2 ´ 600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 250;                 //  Compare A = 250 TBCLK counts
EPwm1Regs.CMPB = 450;                           //  Compare B = 450 TBCLK counts
EPwm1Regs.TBPHS = 0;                            //  Set Phase register to zero
EPwm1Regs.TBCTR = 0;                            //  clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; //  Symmetric
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;         //  Phase loading disabled
EPwm1Regs.TBCTL.bit.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 3-28 illustrates the dead-band submodule within the ePWM module.

Figure 3-28. Dead_Band Submodule



3.2.5.1 Purpose of the Dead-Band Submodule

The "Action-qualifier (AQ) Module" section discussed how it is possible to generate the required dead-band by having full control over edge placement using both the CMPA and CMPB resources of the ePWM module. However, if the more classical edge delay-based dead-band with polarity control is required, then the dead-band submodule described here should be used.

The key functions of the dead-band module are:

- Generating appropriate signal pairs (EPWMxA and EPWMxB) with dead-band relationship from a single EPWMxA input
- Programming signal pairs for:
  - Active high (AH)
  - Active low (AL)
  - Active high complementary (AHC)
  - Active low complementary (ALC)
- Adding programmable delay to rising edges (RED)
- Adding programmable delay to falling edges (FED)
- Can be totally bypassed from the signal path (note dotted lines in diagram)

3.2.5.2 Controlling and Monitoring the Dead-Band Submodule

The dead-band submodule operation is controlled and monitored via the following registers:

Table 3-13. Dead-Band Generator Submodule Registers

Register 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 3-29.

- **Input Source Selection:**

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

  – EPWMxA In is the source for both falling-edge and rising-edge delay. This is the default mode.
  – EPWMxA In is the source for falling-edge delay, EPWMxB In is the source for rising-edge delay.
  – EPWMxA In is the source for rising edge delay, EPWMxB In is the source for falling-edge delay.
  – EPWMxB In is the source for both falling-edge and rising-edge delay.

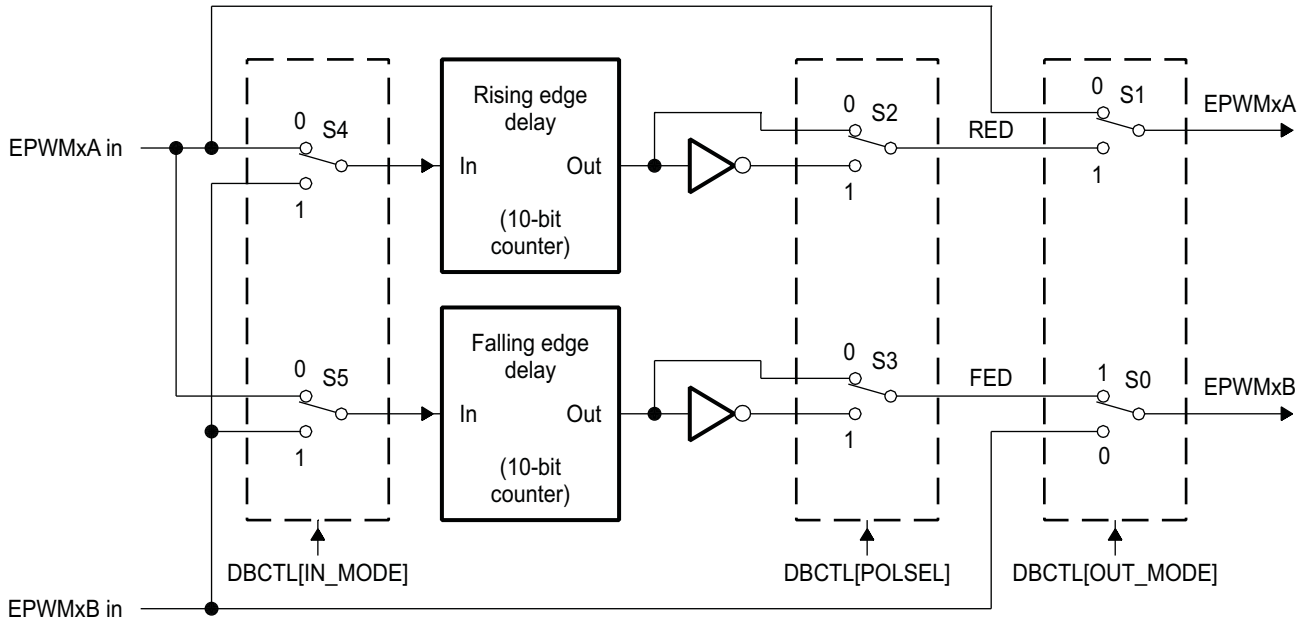
- **Output Mode Control:**

  The output mode is configured by way of the DBCTL[OUT_MODE] bits. These bits determine if the falling-edge delay, rising-edge delay, neither, or both are applied to the input signals.

- **Polarity Control:**

  The polarity control (DBCTL[POLSEL]) allows you to specify whether the rising-edge delayed signal and/or the falling-edge delayed signal is to be inverted before being sent out of the dead-band submodule.

Figure 3-29. Configuration Options for the Dead-Band Submodule



Although all combinations are supported, not all are typical usage modes. Table 3-14 documents some classical dead-band configurations. These modes assume that the DBCTL[IN_MODE] is configured such that EPWMxA In is the source for both falling-edge and rising-edge delay. Enhanced, or non-traditional modes can be achieved by changing the input signal source. The modes shown in Table 3-14 fall into the following categories:

- **Mode 1: Bypass both falling-edge delay (FED) and rising-edge delay (RED)**

  Allows you to fully disable the dead-band submodule from the PWM signal path.

- **Mode 2-5: Classical Dead-Band Polarity Settings:**

  These represent typical polarity configurations that should address all the active high/low modes required by available industry power switch gate drivers. The waveforms for these typical cases are shown in Figure 3-30. Note that to generate equivalent waveforms to Figure 3-30, configure the action-

qualifier submodule to generate the signal as shown for EPWMxA.

- **Mode 6: Bypass rising-edge-delay and Mode 7: Bypass falling-edge-delay**

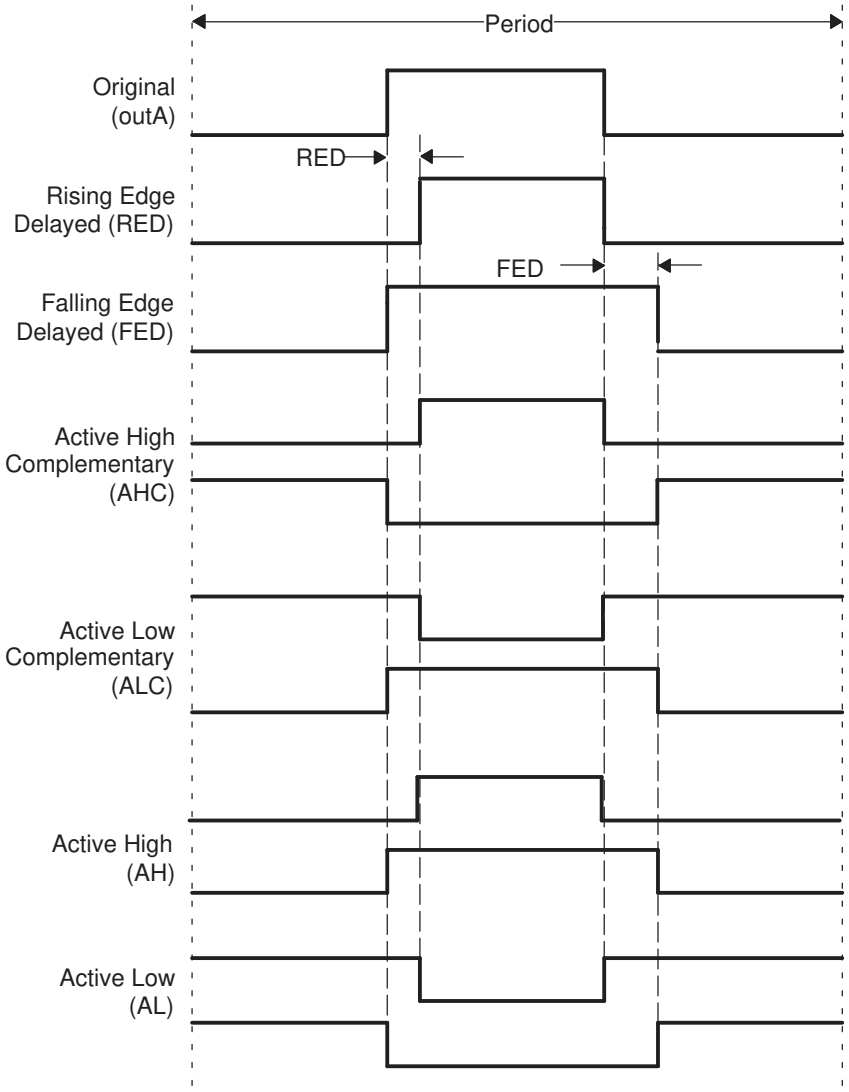
  Finally the last two entries in Table 3-14 show combinations where either the falling-edge-delay (FED) or rising-edge-delay (RED) blocks are bypassed.

Table 3-14. Classical Dead-Band Operating Modes

Mode	Mode Description	DBCTL[POLSEL]		DBCTL[OUT_MODE]	
		S3	S2	S1	S0
1	EPWMxA and EPWMxB Passed Through (No Delay)	X	X	0	0
2	Active High Complementary (AHC)	1	0	1	1
3	Active Low Complementary (ALC)	0	1	1	1
4	Active High (AH)	0	0	1	1
5	Active Low (AL)	1	1	1	1
6	EPWMxA Out = EPWMxA In (No Delay) 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 3-30 shows waveforms for typical cases where 0% < duty < 100%.

Figure 3-30. Dead-Band Waveforms for Typical Cases (0% < Duty < 100%)



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.

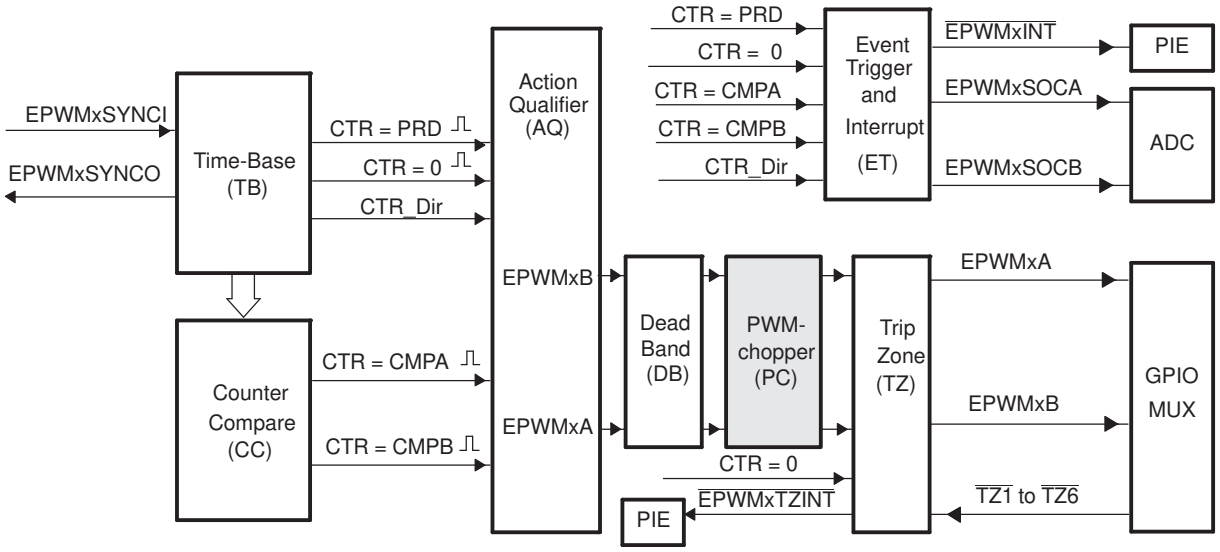
Table 3-15. Dead-Band Delay Values in µS as a Function of DBFED and DBRED

Dead-Band Value	Dead-Band Delay in µS		
DBFED, DBRED	TBCLK = SYSCLKOUT/1	TBCLK = SYSCLKOUT /2	TBCLK = SYSCLKOUT/4
1	0.01 µS	0.02 µS	0.04 µS
5	0.05 µS	0.10 µS	0.20 µS
10	0.10 µS	0.20 µS	0.40 µS
100	1.00 µS	2.00 µS	4.00 µS
200	2.00 µS	4.00 µS	8.00 µS
300	3.00 µS	6.00 µS	12.00 µS
400	4.00 µS	8.00 µS	16.00 µS
500	5.00 µS	10.00 µS	20.00 µS
600	6.00 µS	12.00 µS	24.00 µS
700	7.00 µS	14.00 µS	28.00 µS
800	8.00 µS	16.00 µS	32.00 µS
900	9.00 µS	18.00 µS	36.00 µS
1000	10.00 µS	20.00 µS	40.00 µS

3.2.6 PWM-Chopper (PC) Submodule

Figure 3-31 illustrates the PWM-chopper (PC) submodule within the ePWM module.

Figure 3-31. 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.

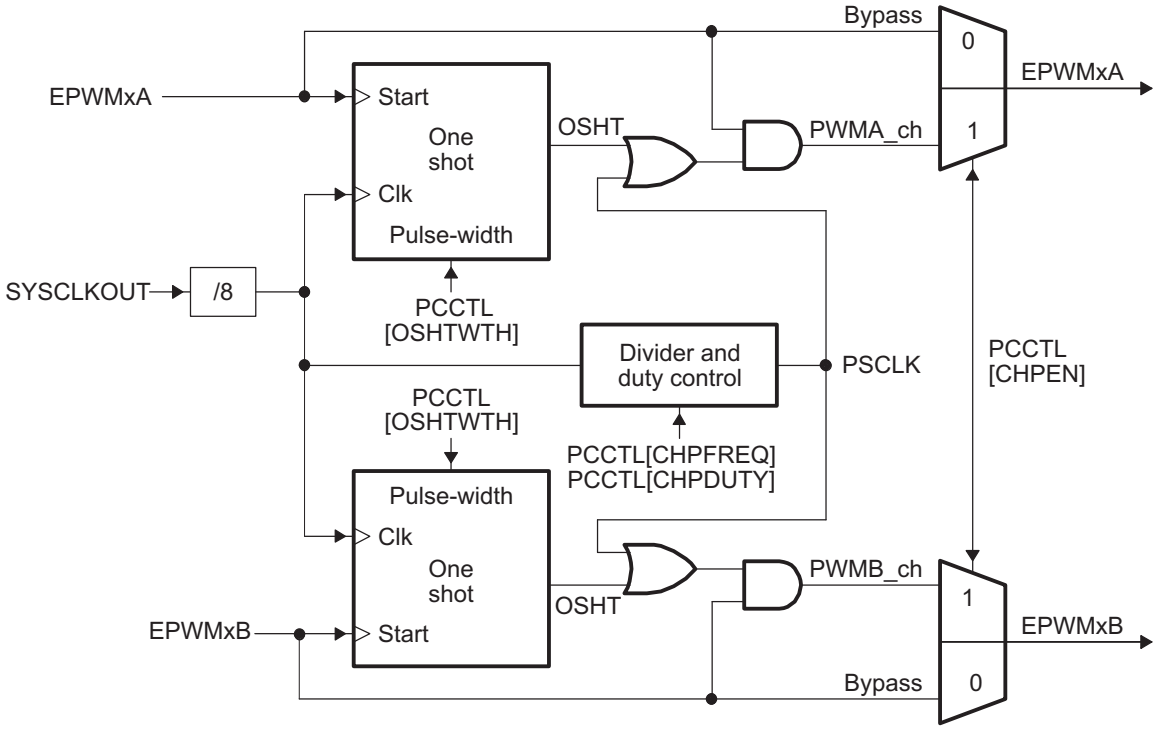
Table 3-16. PWM-Chopper Submodule Registers

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

3.2.6.3 Operational Highlights for the PWM-Chopper Submodule

Figure 3-32 shows the operational details of the PWM-chopper submodule. The carrier clock is derived from SYSCLKOUT. Its frequency and duty cycle are controlled via the CHPFREQ and CHPDUTY bits in the PCCTL register. The one-shot block is a feature that provides a high energy first pulse to ensure hard and fast power switch turn on, while the subsequent pulses sustain pulses, ensuring the power switch remains on. The one-shot width is programmed via the OSHTWTH bits. The PWM-chopper submodule can be fully disabled (bypassed) via the CHPEN bit.

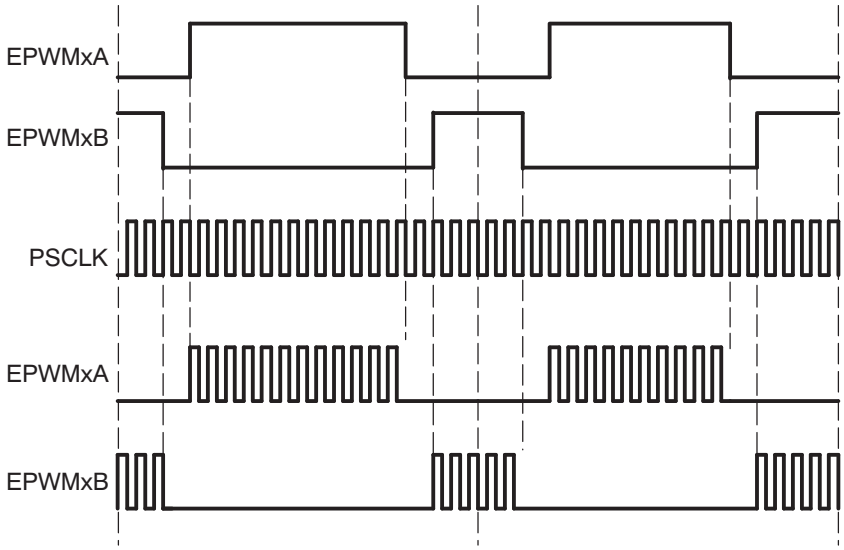
Figure 3-32. PWM-Chopper Submodule Operational Details



3.2.6.4 Waveforms

Figure 3-33 shows simplified waveforms of the chopping action only; one-shot and duty-cycle control are not shown. Details of the one-shot and duty-cycle control are discussed in the following sections.

Figure 3-33. Simple PWM-Chopper Submodule Waveforms Showing Chopping Action Only



3.2.6.4.1 One-Shot Pulse

The width of the first pulse can be programmed to any of 16 possible pulse width values. The width or period of the first pulse is given by:

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

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

Figure 3-34 shows the first and subsequent sustaining pulses and Table 7.3 gives the possible pulse width values for a SYSCLKOUT = 100 MHz.

Figure 3-34. PWM-Chopper Submodule Waveforms Showing the First Pulse and Subsequent Sustaining Pulses

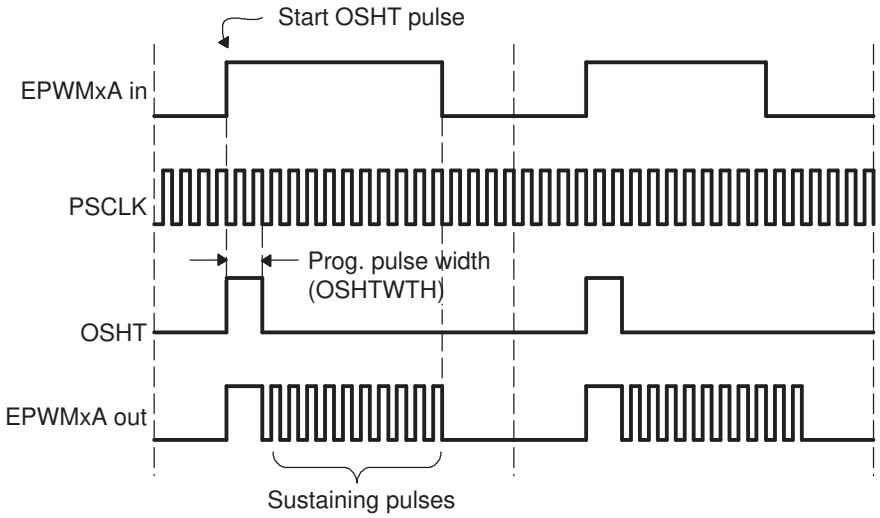


Table 3-17. Possible Pulse Width Values for SYSCLKOUT = 100 MHz

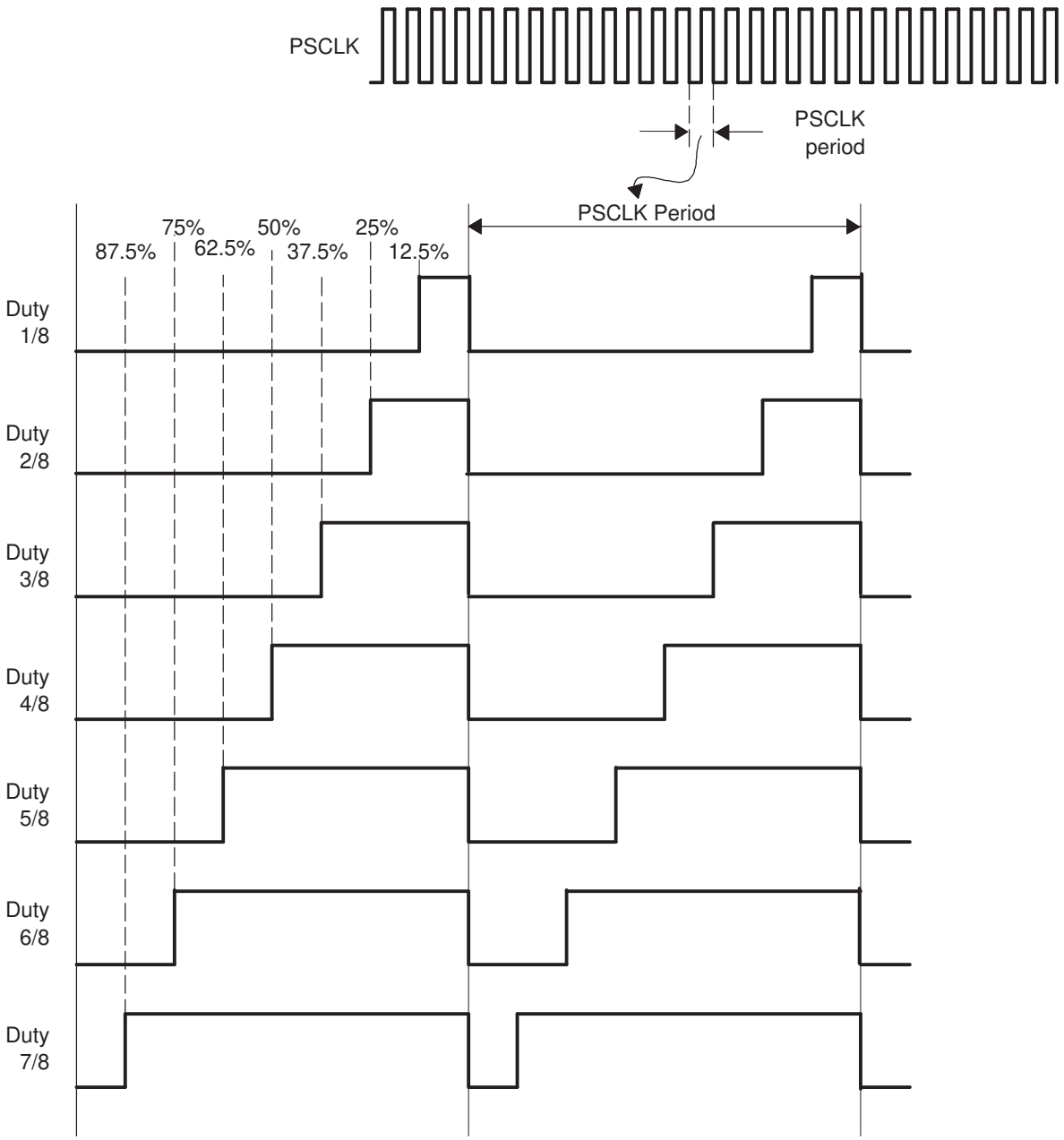
OSHTWTHz (hex)	Pulse Width (nS)
0	80
1	160
2	240
3	320
4	400
5	480
6	560
7	640
8	720
9	800
A	880
B	960
C	1040
D	1120
E	1200
F	1280

3.2.6.4.2 Duty Cycle Control

Pulse transformer-based gate drive designs need to comprehend the magnetic properties or characteristics of the transformer and associated circuitry. Saturation is one such consideration. To assist the gate drive designer, the duty cycles of the second and subsequent pulses have been made programmable. These sustaining pulses ensure the correct drive strength and polarity is maintained on the power switch gate during the on period, and hence a programmable duty cycle allows a design to be tuned or optimized via software control.

Figure 3-35 shows the duty cycle control that is possible by programming the CHPDUTY bits. One of seven possible duty ratios can be selected ranging from 12.5% to 87.5%.

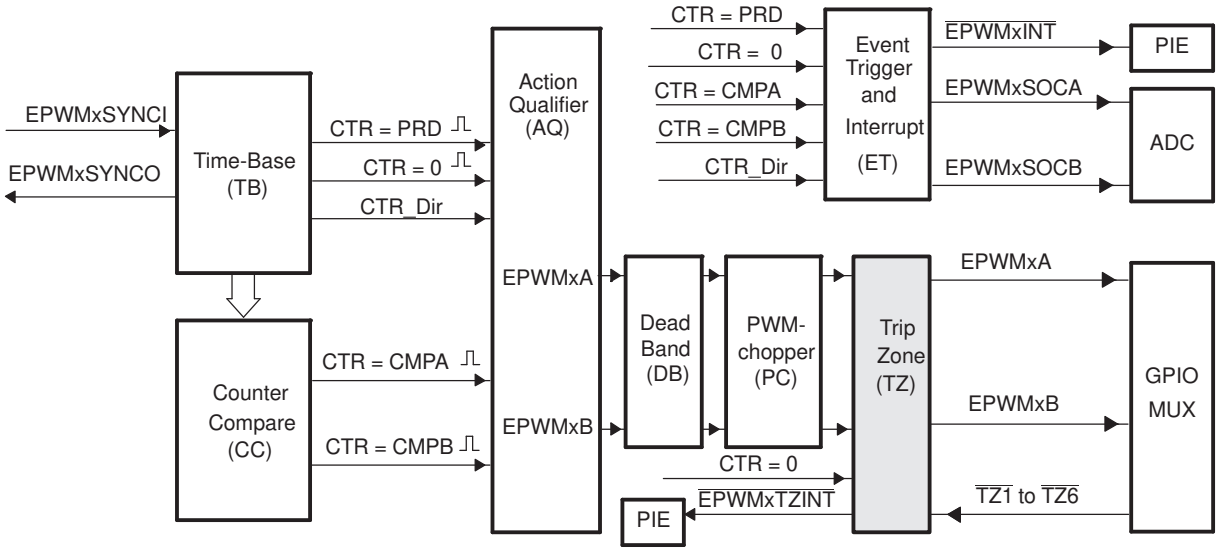
Figure 3-35. PWM-Chopper Submodule Waveforms Showing the Pulse Width (Duty Cycle) Control of Sustaining Pulses



3.2.7 Trip-Zone (TZ) Submodule

Figure 3-36 shows how the trip-zone (TZ) submodule fits within the ePWM module.

Figure 3-36. Trip-Zone Submodule



Each ePWM module is connected to six TZn signals (TZ1 to TZ6) that are sourced from the GPIO MUX. 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 TZ1 to 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.
- Each trip-zone input pin can be allocated to either one-shot or cycle-by-cycle operation.
- Interrupt generation is possible on any trip-zone pin .
- 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
reserved	0x0013		
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 System Control and Interrupts chapter.

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 at pins $\overline{TZ1}$ to $\overline{TZ6}$ (also collectively referred to as $\overline{TZn}$) are active low input signals. When one of these pins goes low, it indicates that a trip event has occurred. Each ePWM module can be individually configured to ignore or use each of the trip-zone pins . Which trip-zone pins 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 1 SYSCLKOUT low pulse on $\overline{TZn}$ inputs is sufficient to trigger a fault condition in the ePWM module. The asynchronous trip makes sure that if clocks are missing for any reason, the outputs can still be tripped by a valid event present on $\overline{TZn}$ inputs, providing the GPIO is appropriately configured . For more information, see the GPIO section of the *System Control and Interrupts chapter*.

Each $\overline{TZn}$ input can be individually configured to provide either a cycle-by-cycle or one-shot trip event for an ePWM module. This configuration is determined by the TZSEL[CBCn], and TZSEL[OSHTn] control bits (where n corresponds to the trip pin) respectively.

- **Cycle-by-Cycle (CBC):**

  When a cycle-by-cycle trip event occurs, the action specified in the TZCTL register 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.

  The specified condition on the pins 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 register 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.

The action taken when a trip event occurs can be configured individually for each of the ePWM output pins by way of the TZCTL[TZA] and TZCTL[TZB] register bits 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[TZA] and/or TZCTL[TZB]	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 3-8. Trip-Zone Configurations

Scenario A:
A one-shot trip event on $\overline{TZ1}$ pulls both EPWM1A, EPWM1B low and also forces EPWM2A and EPWM2B high.

- Configure the ePWM1 registers as follows:
  - TZSEL[OSHT1] = 1: enables $\overline{TZ1}$ as a one-shot event source for ePWM1
  - TZCTL[TZA] = 2: EPWM1A will be forced low on a trip event.
  - TZCTL[TZB] = 2: EPWM1B will be forced low on a trip event.
- Configure the ePWM2 registers as follows:
  - TZSEL[OSHT1] = 1: enables $\overline{TZ1}$ as a one-shot event source for ePWM2
  - TZCTL[TZA] = 1: EPWM2A will be forced high on a trip event.
  - TZCTL[TZB] = 1: EPWM2B will be forced high on a trip event.

Scenario B:
A cycle-by-cycle event on $\overline{TZ5}$ pulls both EPWM1A, EPWM1B low.
A one-shot event on $\overline{TZ1}$ or $\overline{TZ6}$ puts EPWM2A into a high impedance state.

- Configure the ePWM1 registers as follows:
  - TZSEL[CBC5] = 1: enables $\overline{TZ5}$ as a one-shot event source for ePWM1
  - TZCTL[TZA] = 2: EPWM1A will be forced low on a trip event.
  - TZCTL[TZB] = 2: EPWM1B will be forced low on a trip event.
- Configure the ePWM2 registers as follows:
  - TZSEL[OSHT1] = 1: enables $\overline{TZ1}$ as a one-shot event source for ePWM2
  - TZSEL[OSHT6] = 1: enables $\overline{TZ6}$ as a one-shot event source for ePWM2
  - TZCTL[TZA] = 0: EPWM2A will be put into a high-impedance state on a trip event.
  - TZCTL[TZB] = 3: EPWM2B will ignore the trip event.

3.2.7.4 Generating Trip Event Interrupts

Figure 3-37 and Figure 3-38 illustrate the trip-zone submodule control and interrupt logic, respectively.

Figure 3-37. Trip-Zone Submodule Mode Control Logic

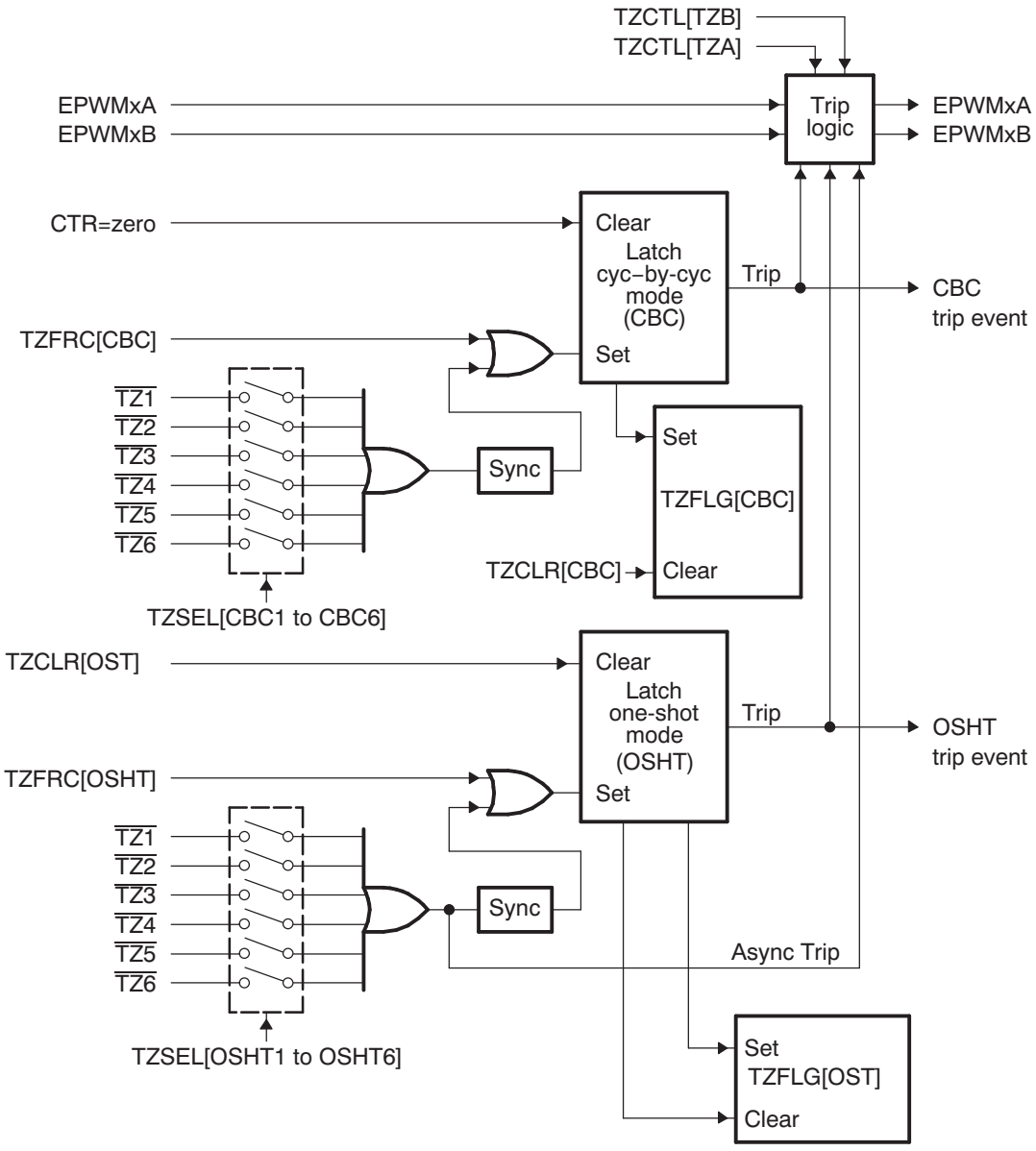
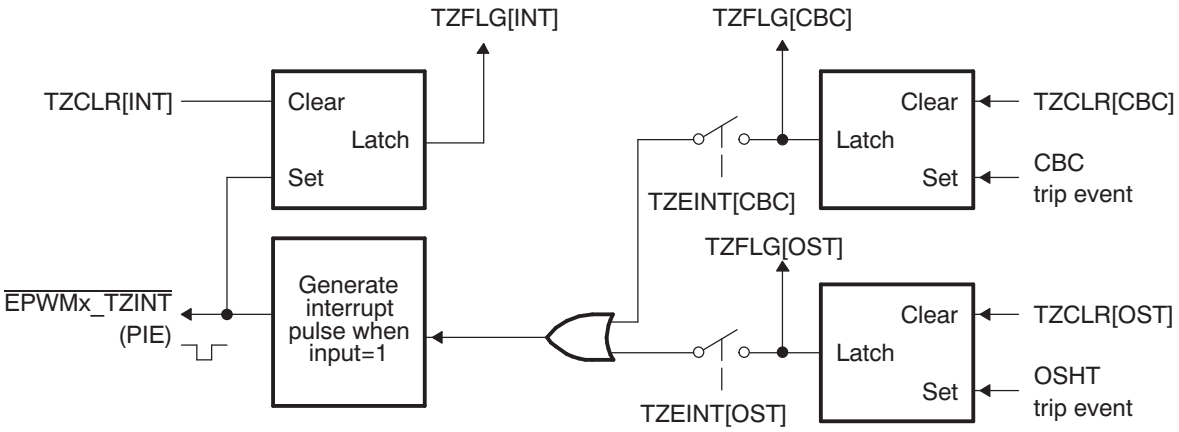


Figure 3-38. Trip-Zone Submodule Interrupt Logic



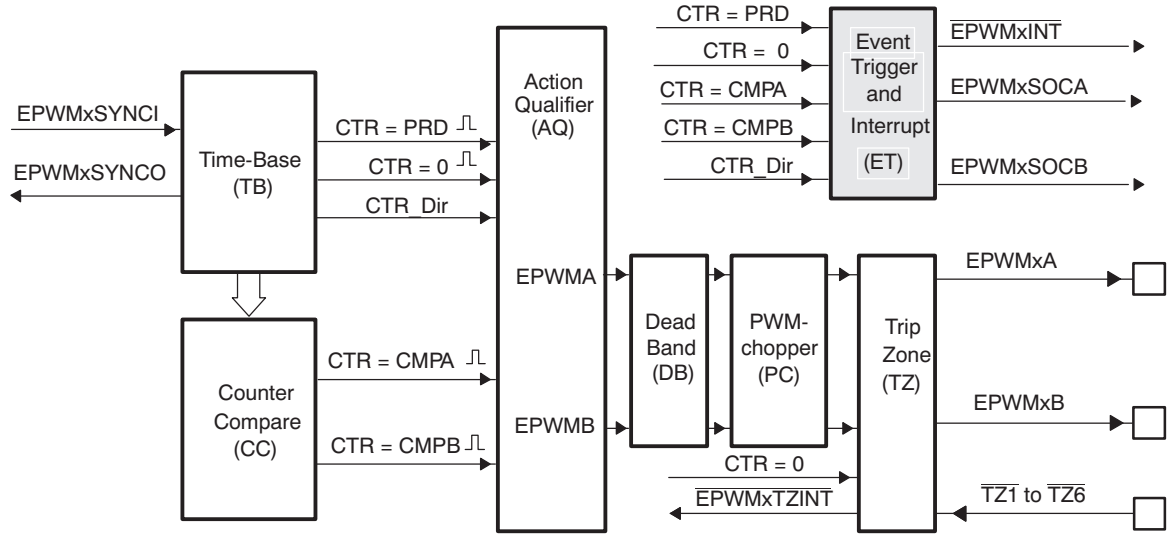
3.2.8 Event-Trigger (ET) Submodule

The key functions of the event-trigger submodule are:

- Receives event inputs generated by the time-base and counter-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 and the counter-compare submodule to generate an interrupt to the CPU and/or a start of conversion pulse to the ADC when a selected event occurs. Figure 3-39 illustrates where the event-trigger submodule fits within the ePWM system.

Figure 3-39. Event-Trigger Submodule

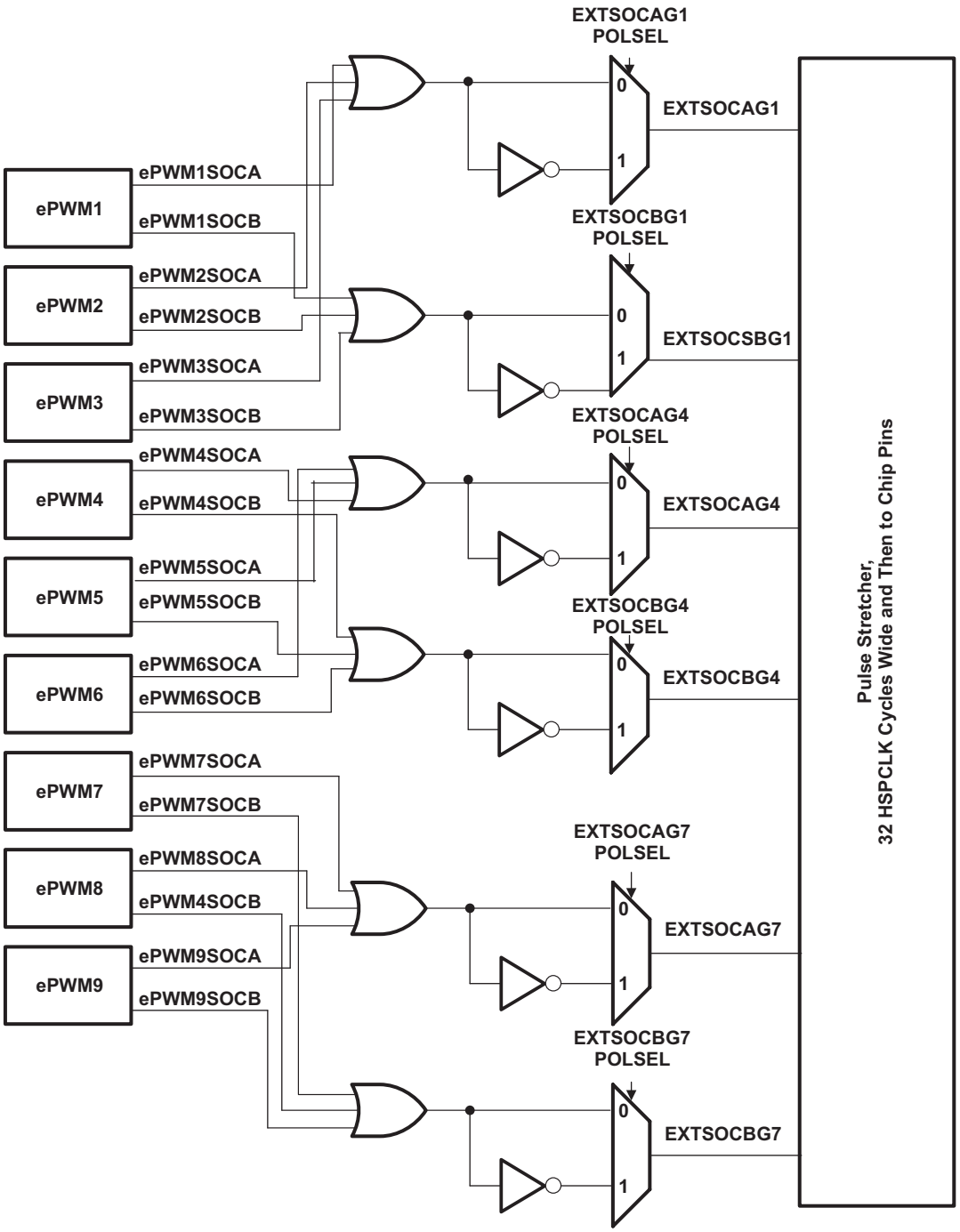


3.2.8.1 Operational Overview of the Event-Trigger Submodule

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 (one for each sequencer) connected to the ADC module. As shown in Figure 3-40, ADC start of conversion for all ePWM modules are ORed together and hence multiple modules can initiate an ADC start of conversion. If two requests occur on one start of conversion line, then only one will be recognized by the ADC.

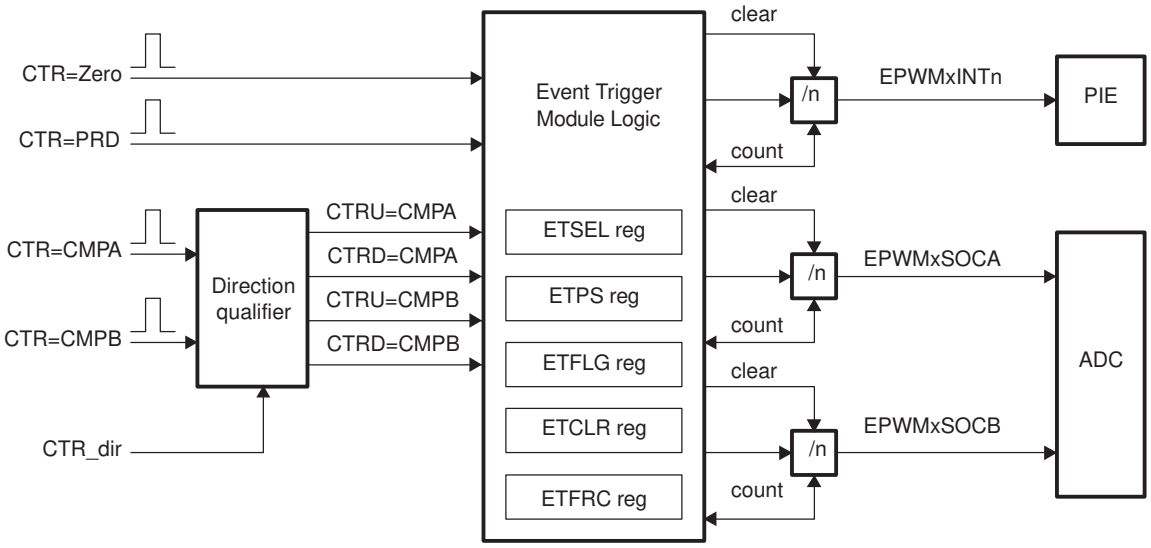
Figure 3-40. Event-Trigger Submodule Inter-Connectivity of ADC Start of Conversion



The event-trigger submodule monitors various event conditions (the left side inputs to event-trigger submodule shown in Figure 3-41) and can be configured to prescale these events before issuing an Interrupt request or an ADC start of conversion. The event-trigger prescaling logic can issue Interrupt requests and ADC start of conversion at:

- Every event
- Every second event
- Every third event

Figure 3-41. Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs



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 3-42, Figure 3-43, and Figure 3-44.

Figure 3-42 shows the event-trigger's interrupt generation logic. The interrupt-period (ETPS[INTPRD]) bits specify the number of events required to cause an interrupt pulse to be generated. The choices available are:

- Do not generate an interrupt.
- Generate an interrupt on every event
- Generate an interrupt on every second event
- Generate an interrupt on every third event

Which event can cause an interrupt is configured by the interrupt selection (ETSEL[INTSEL]) bits. The event can be one of the following:

- Time-base counter equal to zero (TBCTR = 0x0000).
- Time-base counter equal to period (TBCTR = TBPRD).
- Time-base counter equal to 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 3-42. Event-Trigger Interrupt Generator

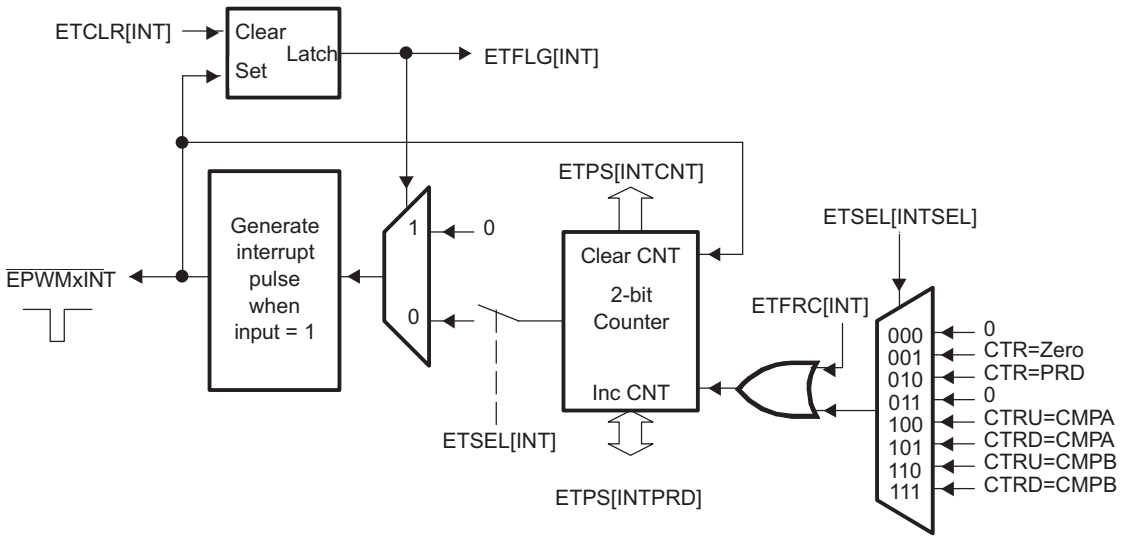


Figure 3-43 shows the operation of the event-trigger's start-of-conversion-A (SOCA) pulse generator. The ETPS[SOCACNT] counter and ETPS[SOCAPRD] period values behave similarly to the interrupt generator except that the pulses are continuously generated. That is, the pulse flag ETFLG[SOCA] is latched when a pulse is generated, but it does not stop further pulse generation. The enable/disable bit ETSEL[SOCAEN] stops pulse generation, but input events can still be counted until the period value is reached as with the interrupt generation logic. The event that will trigger an SOCA and SOCB pulse can be configured separately in the ETSEL[SOCASEL] and ETSEL[SOCBSEL] bits. The possible events are the same events that can be specified for the interrupt generation logic.

Figure 3-43. Event-Trigger SOCA Pulse Generator

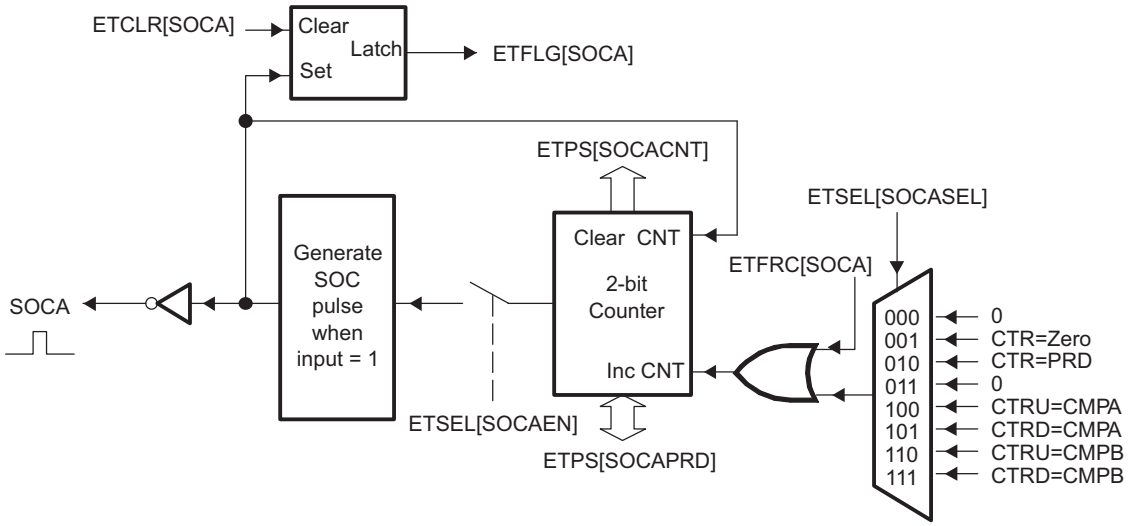
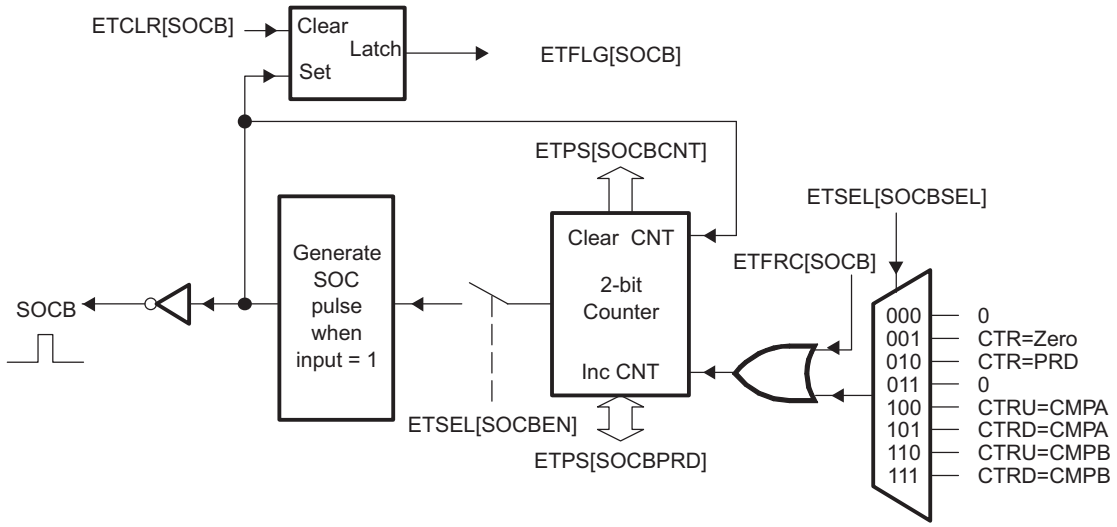


Figure 3-44 shows the operation of the event-trigger's start-of-conversion-B (SOCB) pulse generator. The event-trigger's SOCB pulse generator operates the same way as the SOCA.

Figure 3-44. Event-Trigger SOCB Pulse Generator



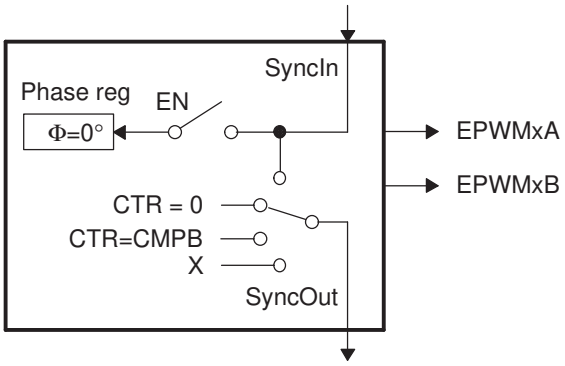
3.3 Applications to Power Topologies

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 3-45. 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 3-45. Simplified ePWM Module



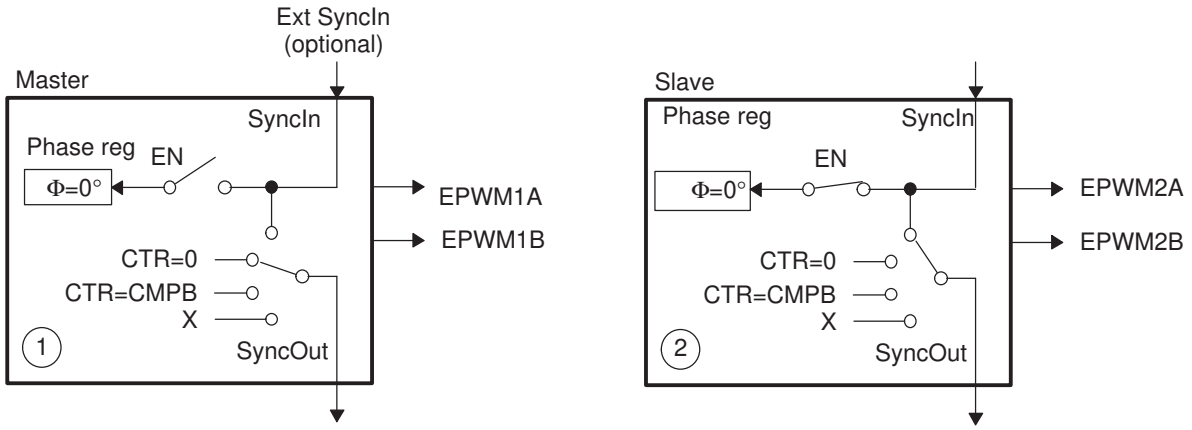
3.3.2 Key Configuration Capabilities

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
- 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 3-46.

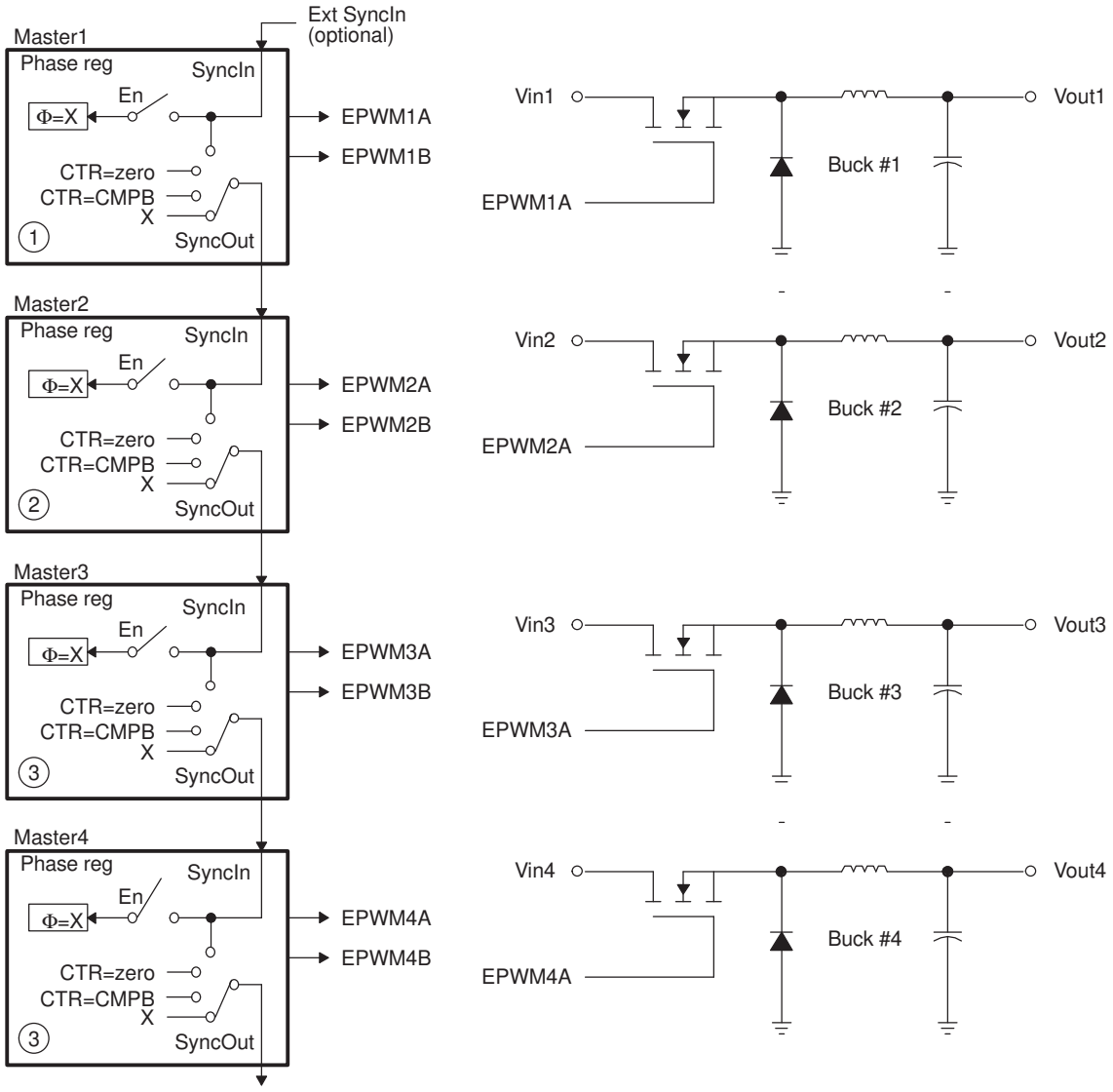
Figure 3-46. EPWM1 Configured as a Typical Master, EPWM2 Configured as a Slave



3.3.3 Controlling Multiple Buck Converters With Independent Frequencies

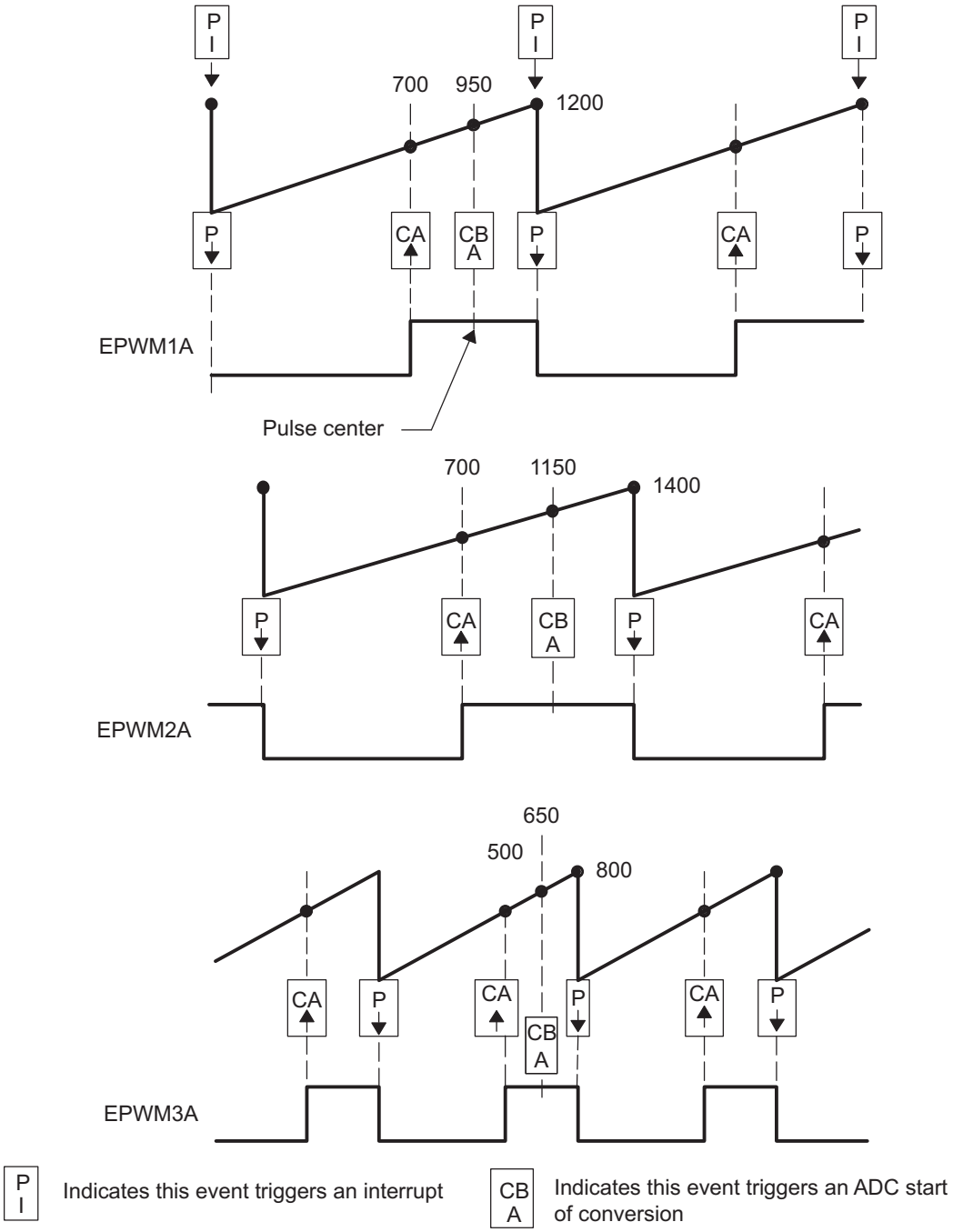
One of the simplest power converter topologies is the buck. A single ePWM module configured as a master can control two buck stages with the same PWM frequency. If independent frequency control is required for each buck converter, then one ePWM module must be allocated for each converter stage. Figure 3-47 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 3-48 shows the waveforms generated by the setup shown in Figure 3-47; note that only three waveforms are shown, although there are four stages.

Figure 3-47. 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 3-48. Buck Waveforms for Figure 3-47 (Note: Only three bucks shown here)



Example 3-9. Configuration for Example in Figure 3-48

```
//=====================================================================
// (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 3-49 shows such a configuration; Figure 3-50 shows the waveforms generated by the configuration.

Figure 3-49. Control of Four Buck Stages. (Note: F_{PWM2} = N x F_{PWM1})

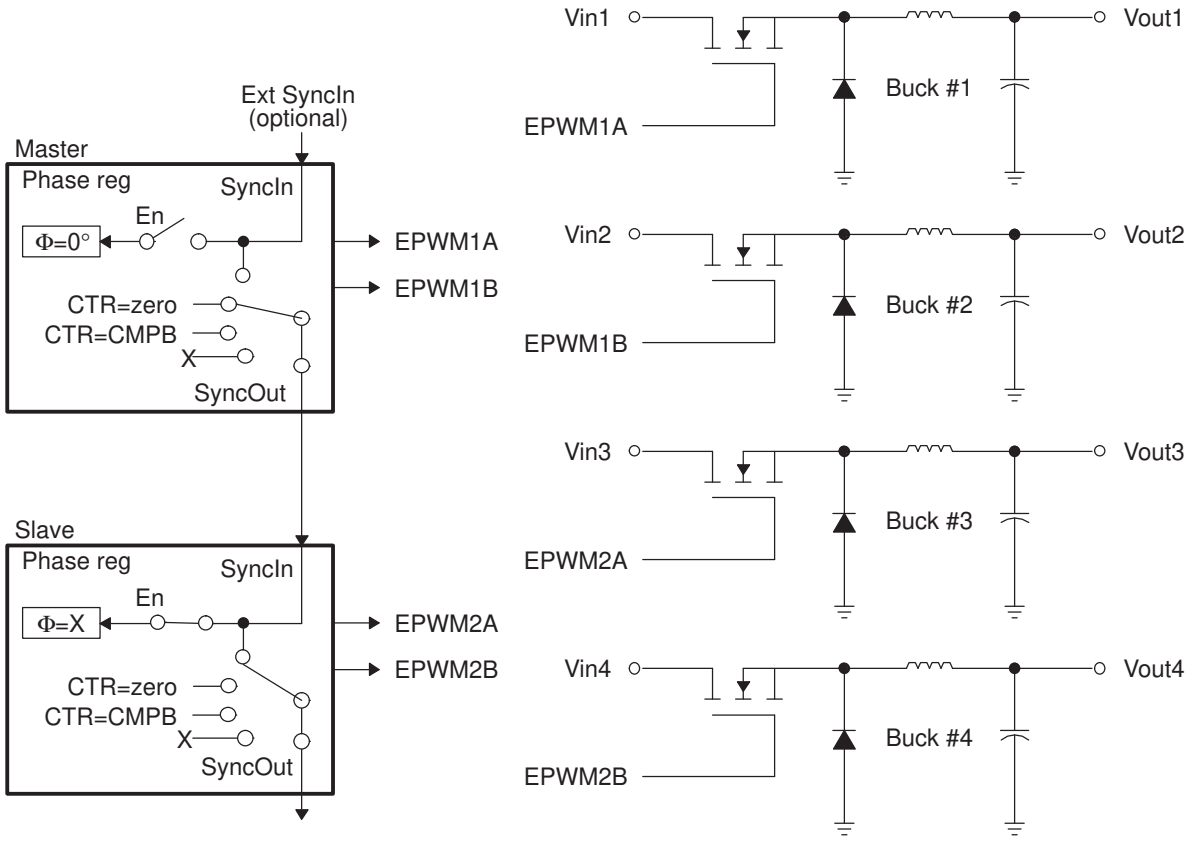
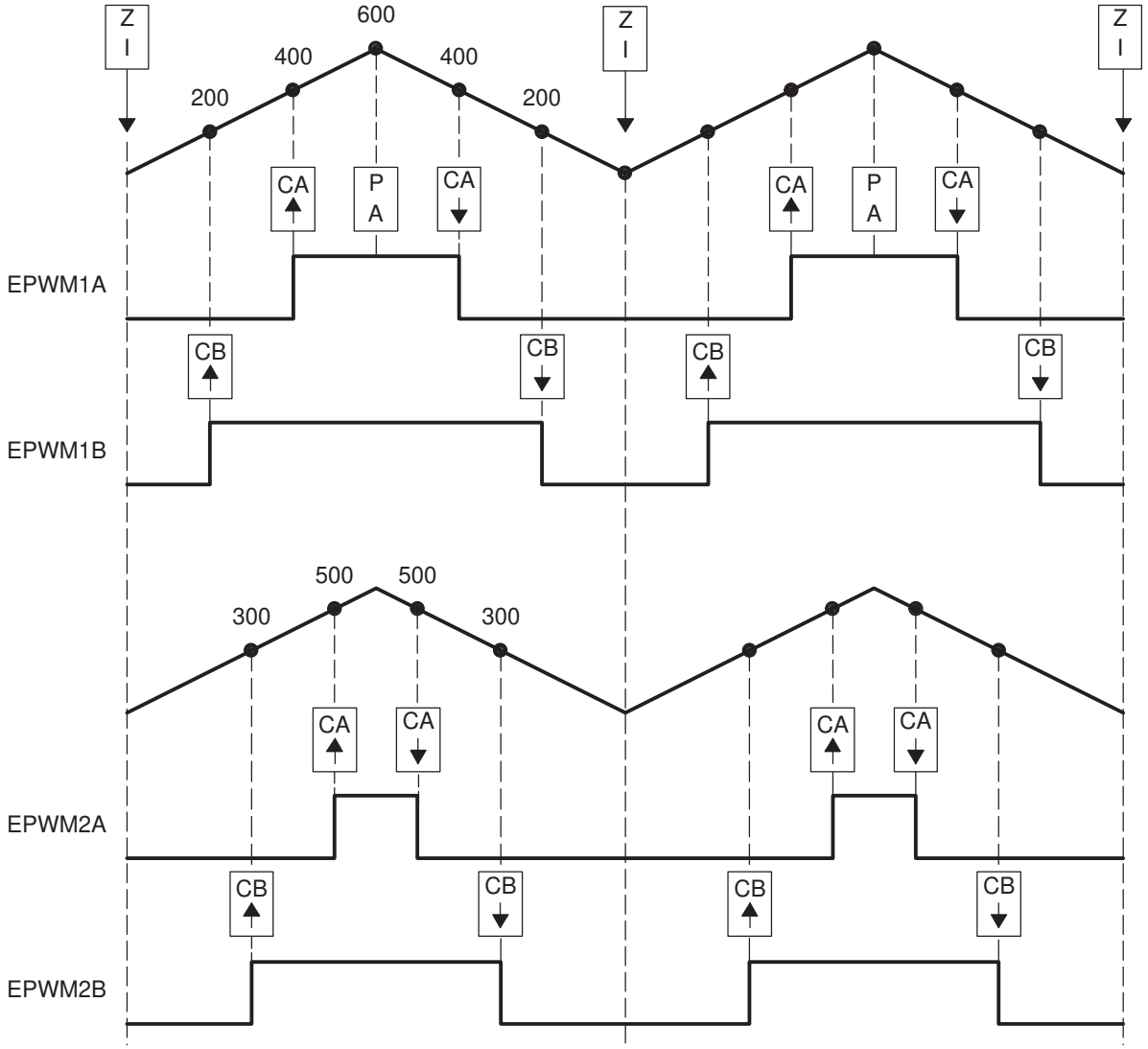


Figure 3-50. Buck Waveforms for Figure 3-49 (Note: $F_{PWM2} = F_{PWM1}$)



Example 3-10. Code Snippet for Configuration in Figure 3-49

```
//=======================
// 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 3-51 shows control of two synchronized Half-H bridge stages where stage 2 can operate at integer multiple (N) frequencies of stage 1. Figure 3-52 shows the waveforms generated by the configuration shown in Figure 3-51.

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 3-51. Control of Two Half-H Bridge Stages (F_{PWM2} = N x F_{PWM1})

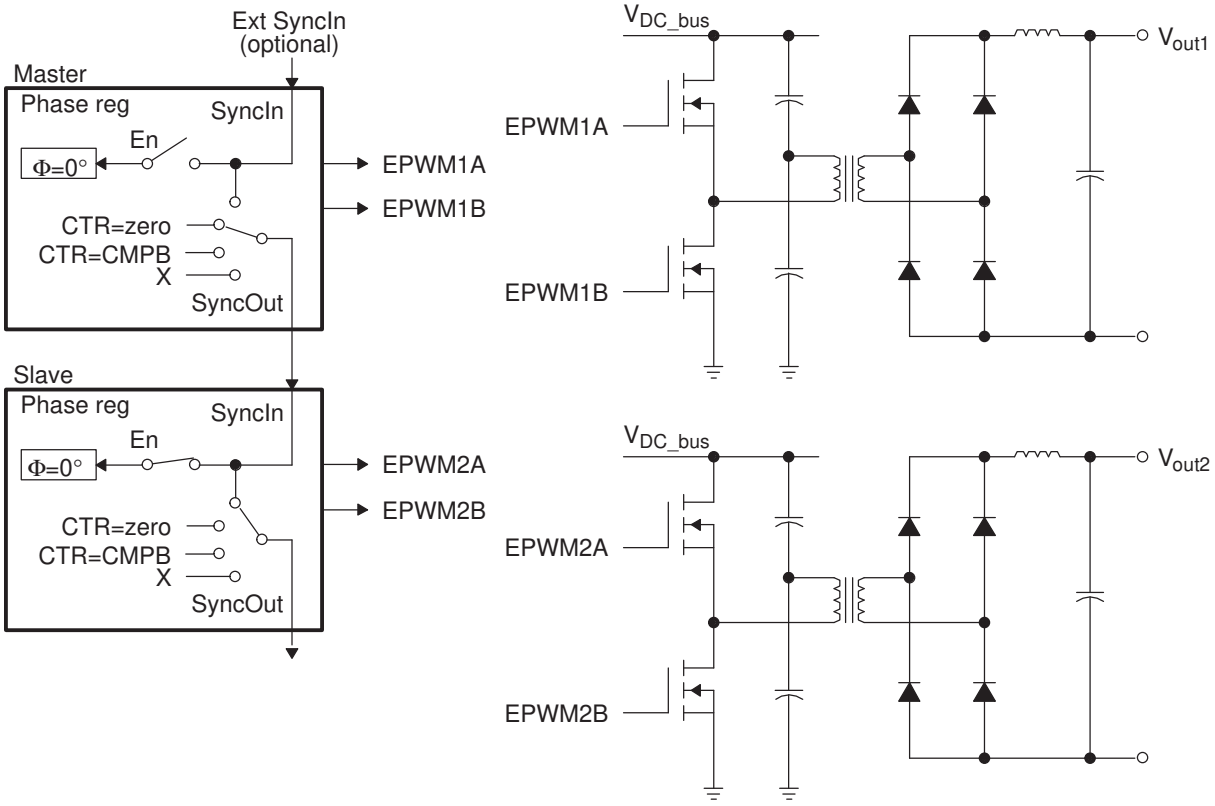
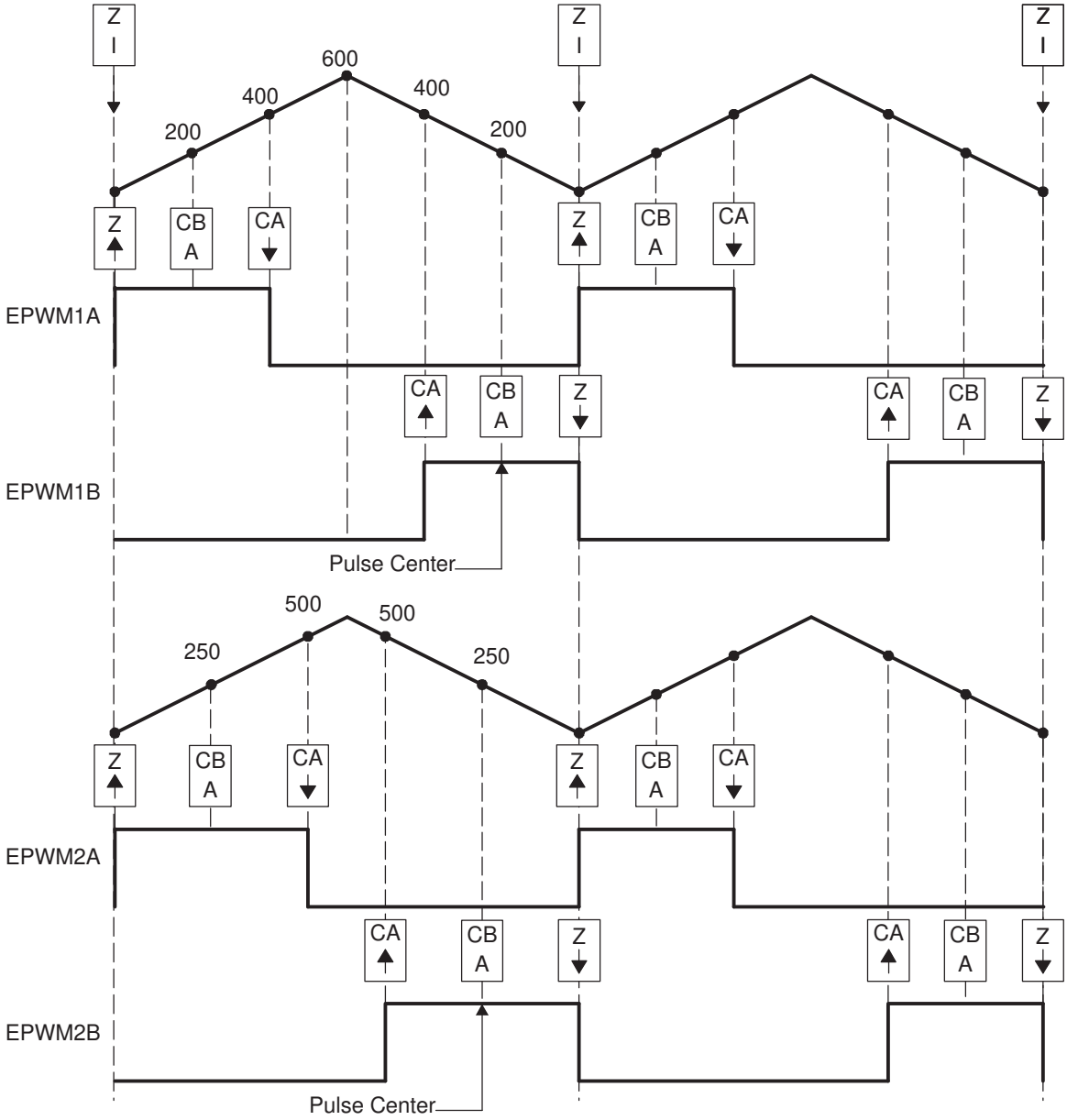


Figure 3-52. Half-H Bridge Waveforms for Figure 3-51 (Note: Here $F_{PWM2} = F_{PWM1}$)



Example 3-11. Code Snippet for Configuration in Figure 3-51

```
//=====================================================================
// 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 3-53 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 3-53), 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 3-53. Control of Dual 3-Phase Inverter Stages as Is Commonly Used in Motor Control

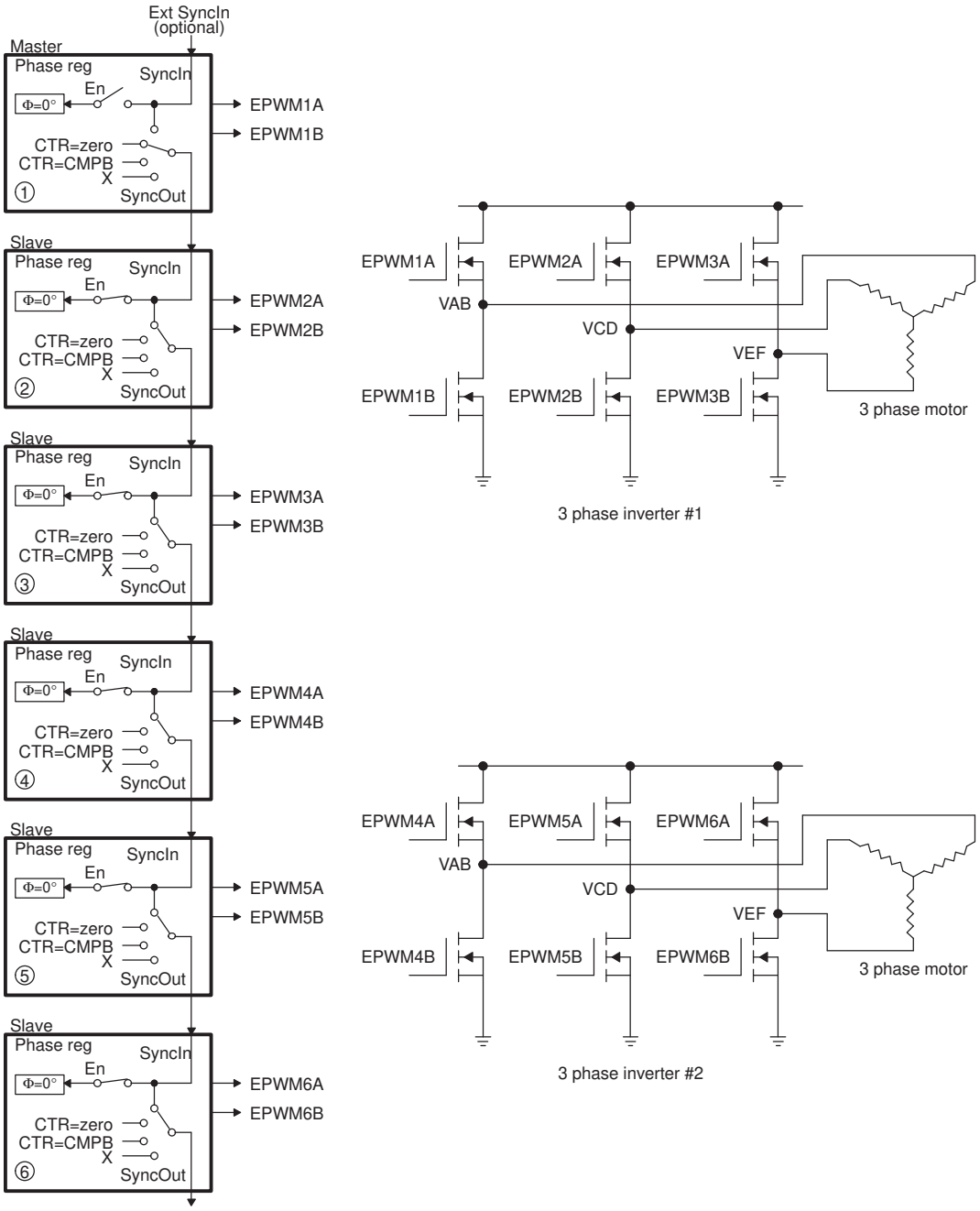
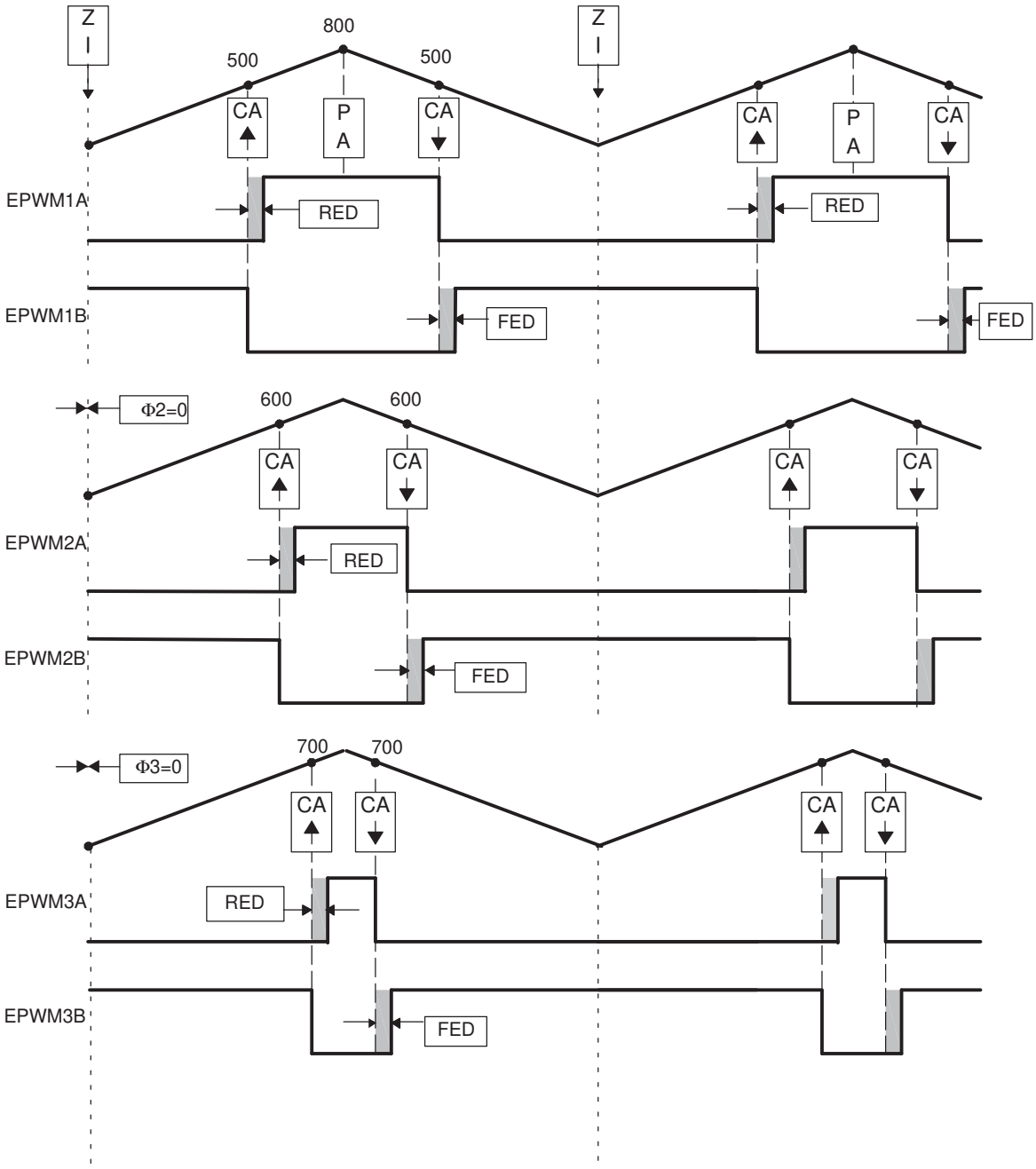


Figure 3-54. 3-Phase Inverter Waveforms for Figure 3-53 (Only One Inverter Shown)



Example 3-12. Code Snippet for Configuration in Figure 3-53

```
//=============================================================================
// 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.bit.OUT_MODE = DB_FULL_ENABLE;       // enable Dead-band module
     EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;        // Active Hi complementary
     EPwm3Regs.DBFED = 50;                            // FED = 50 TBCLKs
     EPwm3Regs.DBRED = 50;                            // RED = 50 TBCLKs
// Run Time (Note: Example execution of one run-time instant)
//=====================================================
     EPwm1Regs.CMPA.half.CMPA = 500;                  // adjust duty for output EPWM1A
     EPwm2Regs.CMPA.half.CMPA = 600;                  // adjust duty for output EPWM2A
     EPwm3Regs.CMPA.half.CMPA = 700;                  // adjust duty for output EPWM3A
```

3.3.7 Practical Applications Using Phase Control Between PWM Modules

So far, none of the examples have made use of the phase register (TBPHS). It has either been set to zero or its value has been a don't care. However, by programming appropriate values into TBPHS, multiple PWM modules can address another class of power topologies that rely on phase relationship between legs (or stages) for correct operation. As described in the TB module section, a PWM module can be configured to allow a SyncIn pulse to cause the TBPHS register to be loaded into the TBCTR register. To illustrate this concept, Figure 3-55 shows a master and slave module with a phase relationship of 120°, i.e., the slave leads the master.

Figure 3-55. Configuring Two PWM Modules for Phase Control

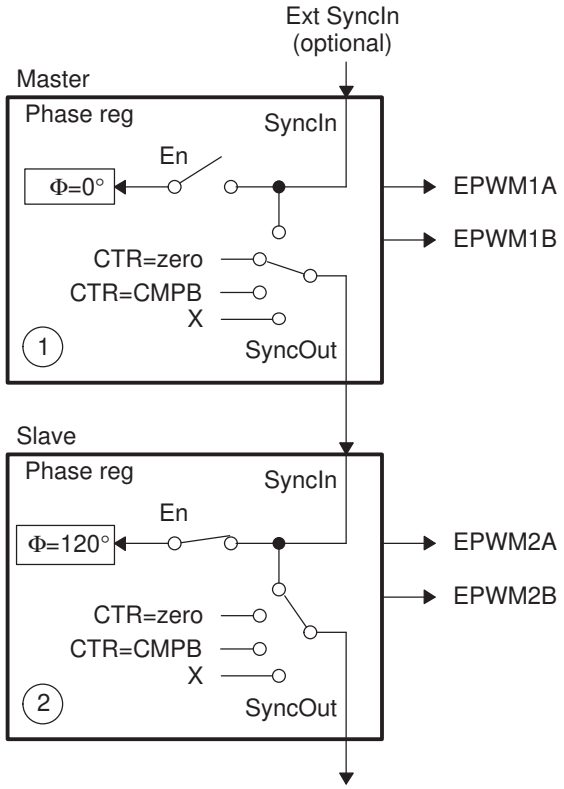
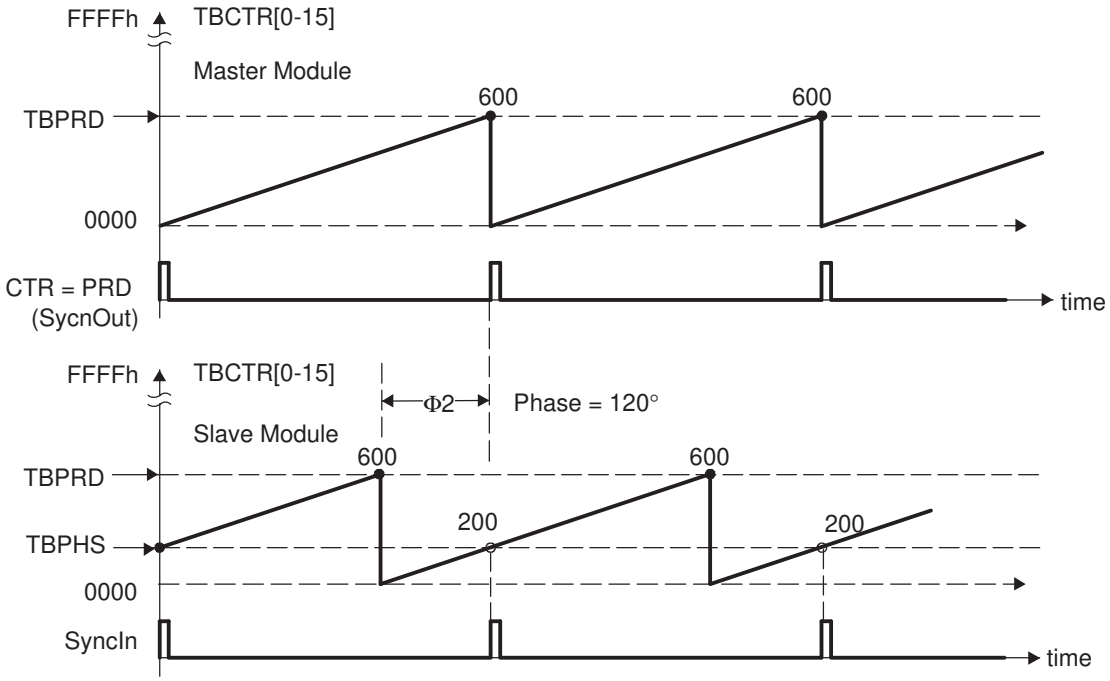


Figure 3-56 shows the associated timing waveforms for this configuration. Here, TBPRD = 600 for both master and slave. For the slave, TBPHS = 200 (i.e., 200/600 X 360° = 120°). Whenever the master generates a SyncIn pulse (CTR = PRD), the value of TBPHS = 200 is loaded into the slave TBCTR register so the slave time-base is always leading the master's time-base by 120°.

Figure 3-56. 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 3-57. 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 (M—1)

Where:

N = number of phases

M = PWM module number

For example, for the 3-phase case (N=3), TBPRD = 600,

TBPHS(3,2) = (600/3) 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 3-58 shows the waveforms for the configuration in Figure 3-57.

Figure 3-57. Control of a 3-Phase Interleaved DC/DC Converter

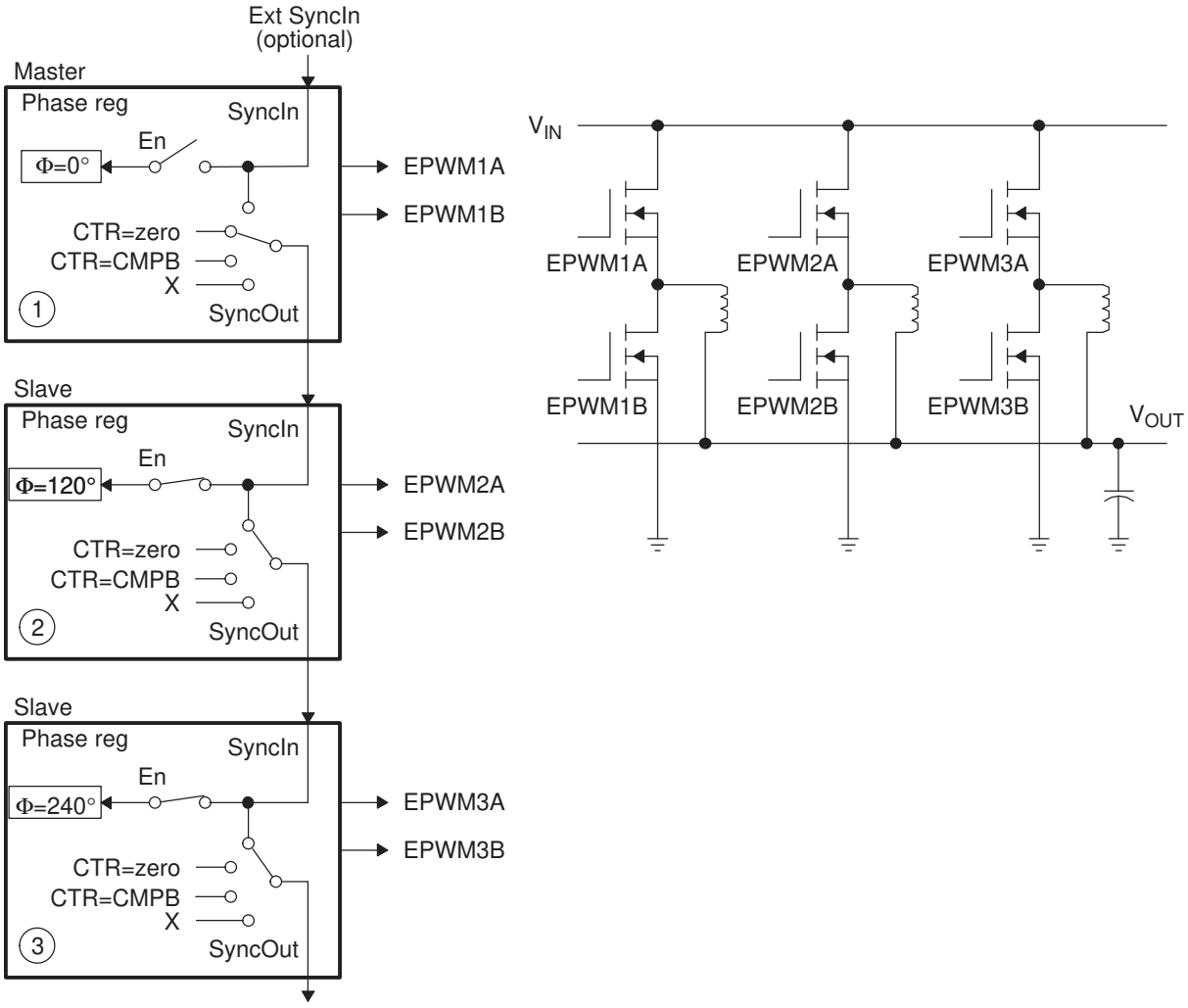
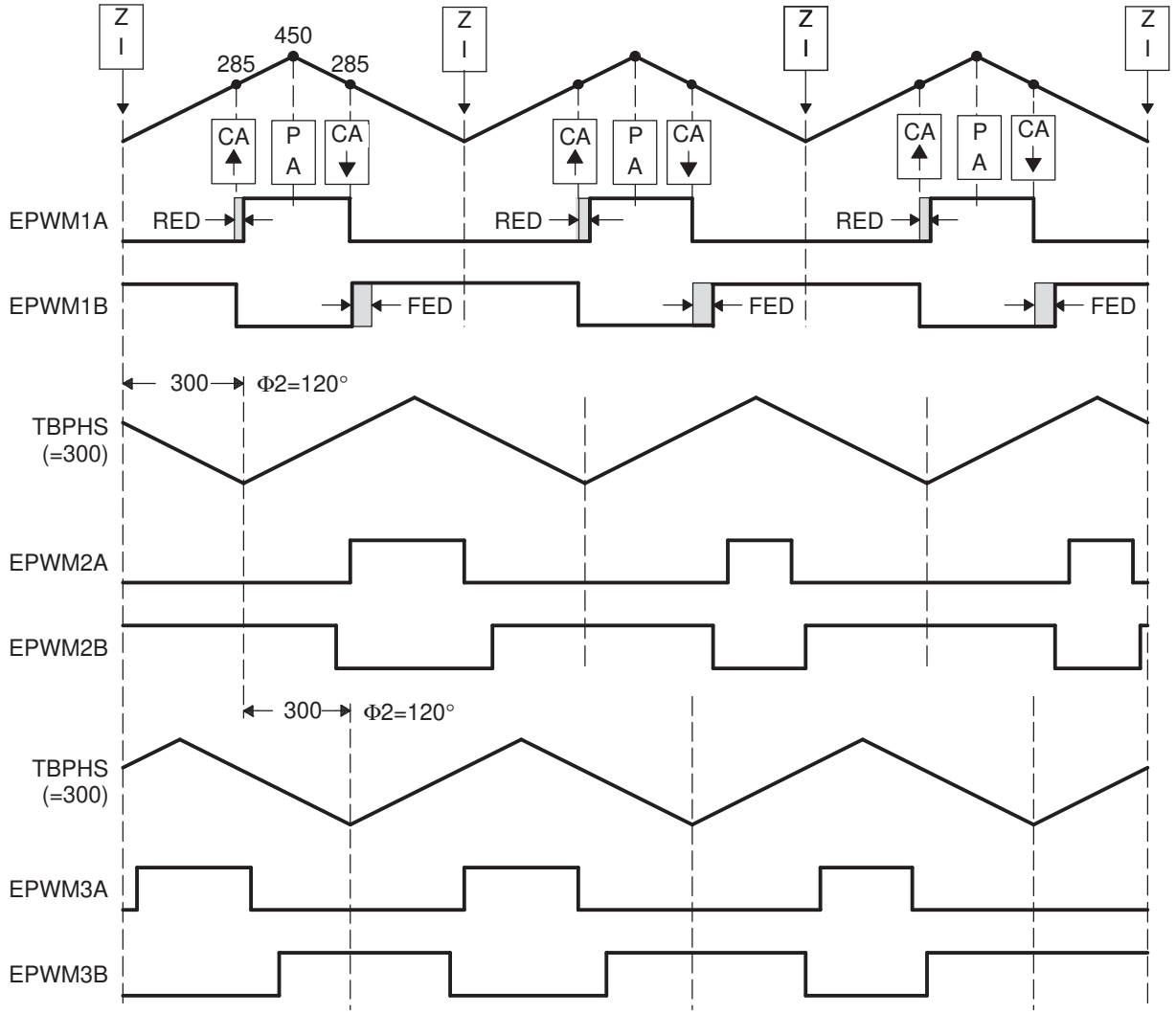


Figure 3-58. 3-Phase Interleaved DC/DC Converter Waveforms for Figure 3-57



Example 3-13. Code Snippet for Configuration in Figure 3-57

```
//=============================================================================
// 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 3-59 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 3-60 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 3-59. Controlling a Full-H Bridge Stage (F_{PWM2} = $F_{PWM1)}$

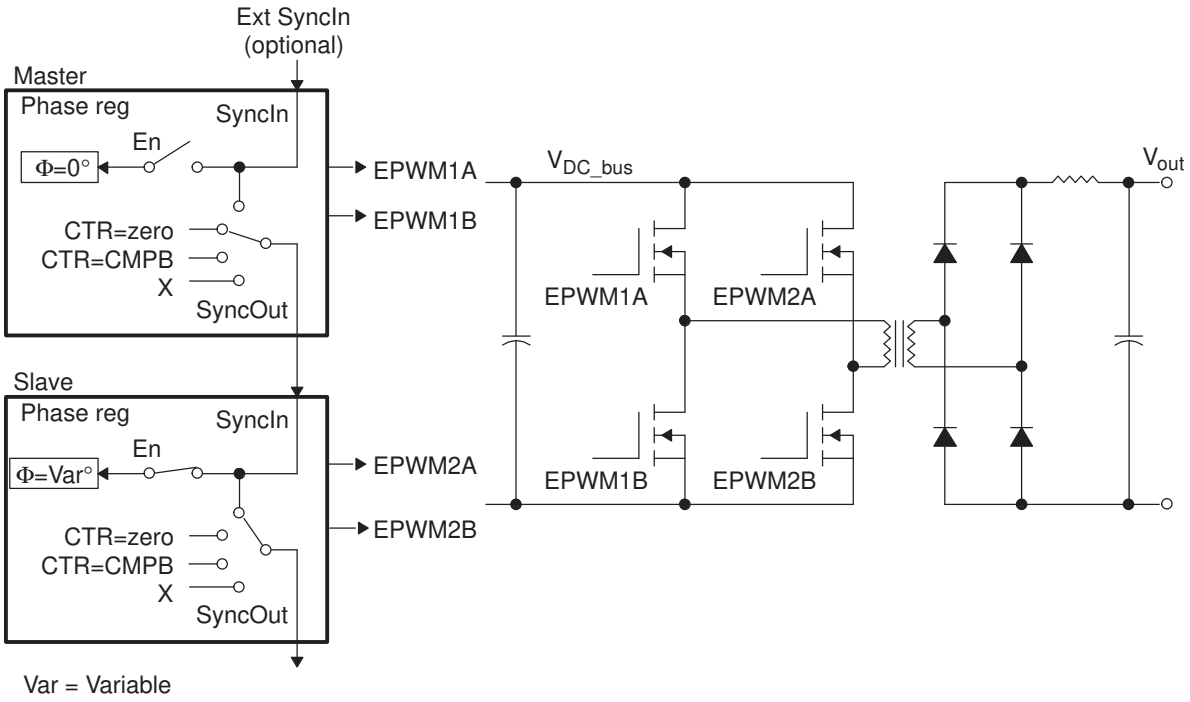
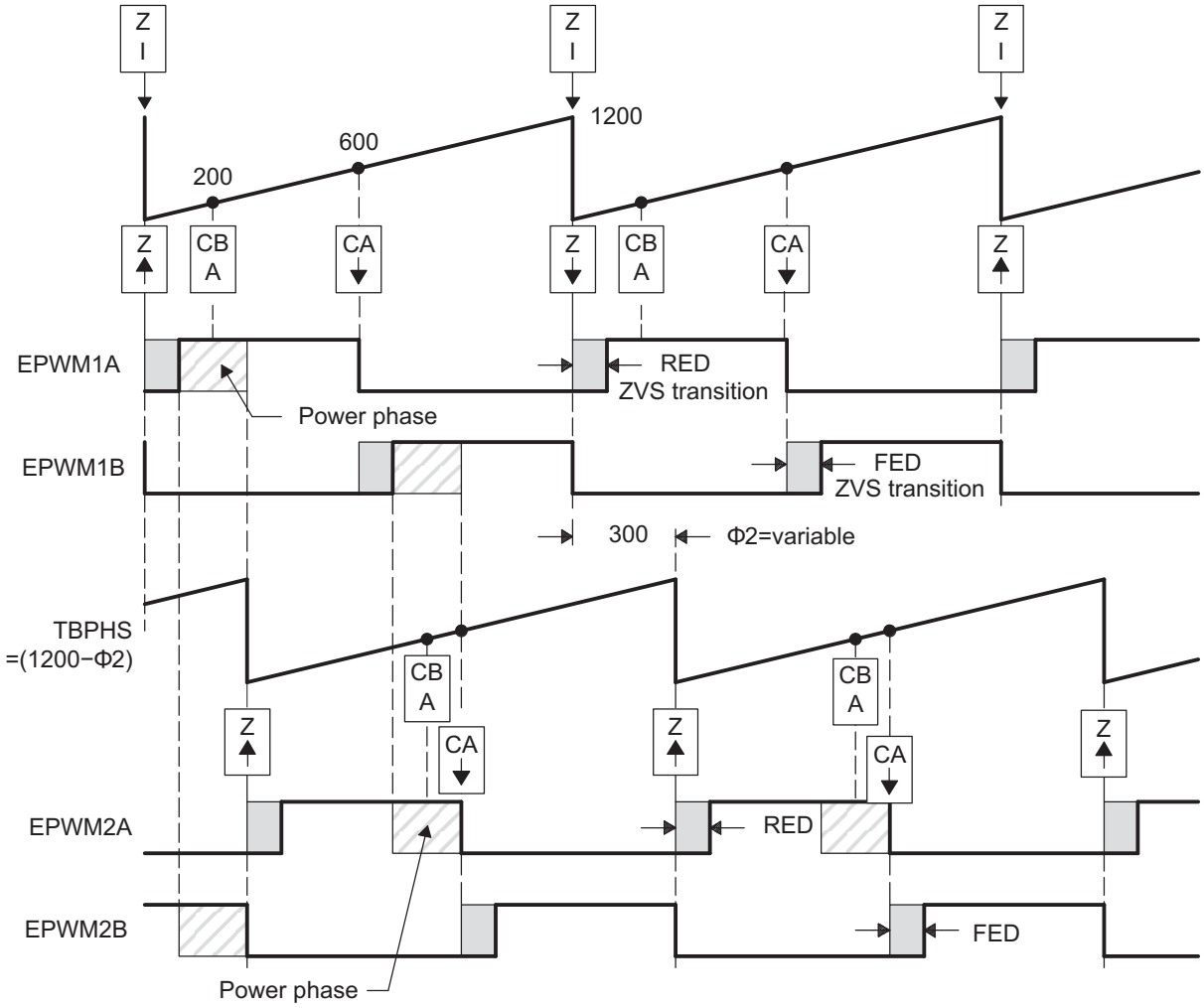


Figure 3-60. ZVS Full-H Bridge Waveforms



Example 3-14. Code Snippet for Configuration in Figure 3-59

```
//=============================================================================
// 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
```

3.4 Registers

This section includes the register layouts and bit description for the submodules.

3.4.1 Time-Base Submodule Registers

Figure 3-61 through Figure 3-65 and Table 3-21 through Table 3-25 provide the time-base register definitions.

Figure 3-61. Time-Base Period Register (TBPRD)

15	0
TBPRD	
R/W-0	

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

Table 3-21. 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 3-62. Time-Base Phase Register (TBPHS)

15	0
TBPHS	
R/W-0	

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

Table 3-22. 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 3-63. Time-Base Counter Register (TBCTR)

15	0
TBCTR	
R/W-0	

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

Table 3-23. 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 3-64. 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 3-24. 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
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

Table 3-24. Time-Base Control Register (TBCTL) Field Descriptions (continued)

Bit	Field	Value	Description
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.
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
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 3-65. 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 3-25. 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.

3.4.2 Counter-Compare Submodule Registers

Figure 3-66 through Figure 3-69 and Table 3-26 through Table 3-28 illustrate the counter-compare submodule control and status registers.

Figure 3-66. Counter-Compare A Register (CMPA)

15	0
CMPA	
R/W-0	

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

Table 3-26. 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 3-67. Counter-Compare B Register (CMPB)

15	0
CMPB	
R/W-0	

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

Table 3-27. 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 3-68. 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 3-28. 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 3-69. 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 3-29. 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

3.4.3 Action-Qualifier Submodule Registers

Figure 3-70 through Figure 3-73 and Table 3-30 through Table 3-33 provide the action-qualifier submodule register definitions.

Figure 3-70. Action-Qualifier Output A Control Register (AQCTLA)

15	14	13	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 3-30. 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.
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 3-71. Action-Qualifier Output B Control Register (AQCTLB)

15	14	13	12	11	10	9	8
Reserved				CBD		CBU	
R-0				R/W-0		R/W-0	

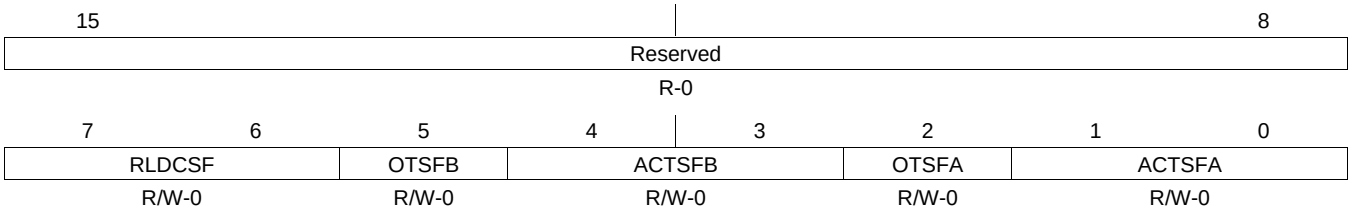
7	6	5	4	3	2	1	0
CAD		CAU		PRD		ZRO	
R/W-0		R/W-0		R/W-0		R/W-0	

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

Table 3-31. 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.
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 3-72. Action-Qualifier Software Force Register (AQSFRC)



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

Table 3-32. 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)
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 3-73. 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 3-33. 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	Software forcing is disabled and 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	Software forcing is disabled and 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 3-74 through Figure 3-76 and Table 3-34 through Table 3-36 provide the register definitions.

Figure 3-74. Dead-Band Generator Control Register (DBCTL)

15							8
Reserved							
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 3-34. Dead-Band Generator Control Register (DBCTL) Field Descriptions

Bits	Name	Value	Description
15-6	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 3-29. 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 3-29. 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 3-34. 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 3-29. 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 3-75. 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 3-35. 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 3-76. 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 3-36. 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 3-77 and Table 3-37 provide the definitions for the PWM-chopper submodule control register.

Figure 3-77. 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 3-37. 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, = 12.5 MHz at 100 MHz SYSCLKOUT)
		001	Divide by 2 (6.25 MHz at 100 MHz SYSCLKOUT)
		010	Divide by 3 (4.16 MHz at 100 MHz SYSCLKOUT)
		011	Divide by 4 (3.12 MHz at 100 MHz SYSCLKOUT)
		100	Divide by 5 (2.50 MHz at 100 MHz SYSCLKOUT)
		101	Divide by 6 (2.08 MHz at 100 MHz SYSCLKOUT)
		110	Divide by 7 (1.78 MHz at 100 MHz SYSCLKOUT)
		111	Divide by 8 (1.56 MHz at 100 MHz SYSCLKOUT)

Table 3-37. PWM-Chopper Control Register (PCCTL) Bit Descriptions (continued)

Bits	Name	Value	Description
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)
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 3-78 through Figure 3-83 and Table 3-38 through Table 3-43 provide the definitions for the trip-zone submodule control and status registers.

Figure 3-78. Trip-Zone Select Register (TZSEL)

15	14	13	12	11	10	9	8
Reserved	Reserved	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
Reserved	Reserved	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 3-38. Trip-Zone 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 (Table 3-39) 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 ().			
15	Reserved		
14	Reserved		
13	OSHT6		Trip-zone 6 ($\overline{TZ6}$) Select
		0	Disable $\overline{TZ6}$ as a one-shot trip source for this ePWM module.
		1	Enable $\overline{TZ6}$ as a one-shot trip source for this ePWM module.
12	OSHT5		Trip-zone 5 ($\overline{TZ5}$) Select
		0	Disable $\overline{TZ5}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{TZ5}$ as a one-shot trip source for this ePWM module
11	OSHT4		Trip-zone 4 ($\overline{TZ4}$) Select
		0	Disable $\overline{TZ4}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{TZ4}$ as a one-shot trip source for this ePWM module
10	OSHT3		Trip-zone 3 ($\overline{TZ3}$) Select
		0	Disable $\overline{TZ3}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{TZ3}$ as a one-shot trip source for this ePWM module
9	OSHT2		Trip-zone 2 ($\overline{TZ2}$) Select
		0	Disable $\overline{TZ2}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{TZ2}$ as a one-shot trip source for this ePWM module
8	OSHT1		Trip-zone 1 ($\overline{TZ1}$) Select
		0	Disable $\overline{TZ1}$ as a one-shot trip source for this ePWM module
		1	Enable $\overline{TZ1}$ 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 (Table 3-39) 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	Reserved		
6	Reserved		
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

Table 3-38. Trip-Zone Select Register (TZSEL) Field Descriptions (continued)

Bits	Name	Value	Description
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 3-79. Trip-Zone Control Register (TZCTL)

15					8
Reserved					
R-0					

7	4	3	2	1	0
Reserved		TZB		TZA	
R-0		R/W-0		R/W-0	

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

Table 3-39. Trip-Zone Control Register (TZCTL) Field Descriptions

Bits	Name	Value	Description
15–4	Reserved		Reserved
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 3-80. Trip-Zone Enable Interrupt Register (TZEINT)

15				8
Reserved				
R -0				
7	3	2	1	0
Reserved		OST	CBC	Reserved
R-0		R/W-0	R/W-0	R-0

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

Table 3-40. Trip-Zone Enable Interrupt Register (TZEINT) Field Descriptions

Bits	Name	Value	Description
15-3	Reserved		Reserved
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.
		1	Enable interrupt generation; a cycle-by-cycle trip event will cause an EPWMx_TZINT PIE interrupt.
0	Reserved		Reserved

Figure 3-81. Trip-Zone Flag Register (TZFLG)

15				8
Reserved				
R-0				
7	3	2	1	0
Reserved		OST	CBC	INT
R-0		R-0	R-0	R-0

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

Table 3-41. Trip-Zone Flag Register (TZFLG) Field Descriptions

Bits	Name	Value	Description
15-3	Reserved		Reserved
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 pin 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 pins 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 pins 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 3-82. Trip-Zone Clear Register (TZCLR)

15				8
Reserved				
R-0				

7	3	2	1	0
Reserved		OST	CBC	INT
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. Trip-Zone Clear Register (TZCLR) Field Descriptions

Bits	Name	Value	Description
15-3	Reserved		Reserved
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 3-83. Trip-Zone Force Register (TZFRC)

15				8
Reserved				
R-0				
7	**3**	**2**	**1**	**0**
Reserved		OST	CBC	Reserved
R-0		R/W-0	R/W-0	R- 0

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

Table 3-43. Trip-Zone Force Register (TZFRC) Field Descriptions

Bits	Name	Value	Description
15- 3	Reserved		Reserved
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

3.4.7 Event-Trigger Submodule Registers

Figure 3-84 through Figure 3-88 and Table 3-44 through Table 3-48 provide the definitions for the event-trigger submodule registers.

Figure 3-84. 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 3-44. 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	Reserved
		001	Enable event time-base counter equal to zero. (TBCTR = 0x0000)
		010	Enable event time-base counter equal to period (TBCTR = TBPRD)
		011	Reserved
		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	Reserved
		001	Enable event time-base counter equal to zero. (TBCTR = 0x0000)
		010	Enable event time-base counter equal to period (TBCTR = TBPRD)
		011	Reserved
		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

Table 3-44. Event-Trigger Selection Register (ETSEL) Field Descriptions (continued)

Bits	Name	Value	Description
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	Reserved
		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 3-85. 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 3-45. 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 3-45. 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 3-86. 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 3-46. 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 3-42.

Figure 3-87. 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 3-47. 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 3-88. 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 3-48. 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.

3.4.8 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

High-Resolution Pulse Width Modulator (HRPWM)

This chapter should be used in conjunction with the *Enhanced Pulse Width Modulator (ePWM) Module* chapter.

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. This occurs at PWM frequencies greater than ~200 kHz when using a CPU/system clock of 100 MHz. The key features of HRPWM are:

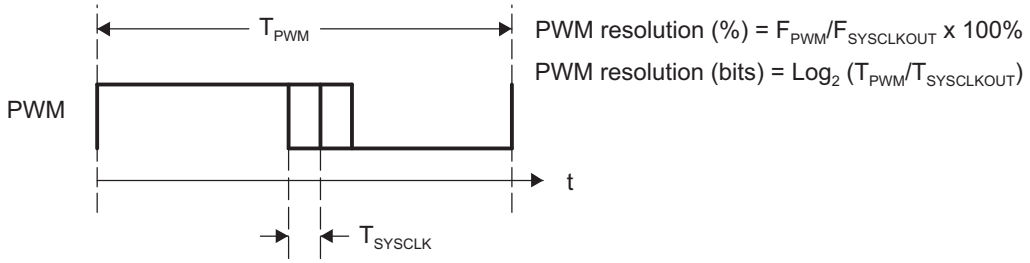
- Extended time resolution capability
- Used in both duty cycle and phase-shift control methods
- Finer time granularity control or edge positioning using extensions to the Compare A and Phase registers
- Implemented using the A signal path of PWM, that is, on the EPWMxA output. EPWMxB output has conventional PWM capabilities
- Self-check diagnostics software mode to check if the micro edge positioner (MEP) logic is running optimally

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, where $T_{SYSCLKOUT}$ = 10 ns (that is, 100 MHz clock), the effective resolution for conventionally generated PWM is a function of PWM frequency (or period) and system clock frequency.

Figure 4-1. Resolution Calculations for Conventionally Generated PWM



If the required PWM operating frequency does not offer sufficient resolution in PWM mode, you may want to consider HRPWM. As an example of improved performance offered by HRPWM, Table 4-1 shows resolution in bits for various PWM frequencies. These values assume a 100 MHz SYSCLK frequency and 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 (kHz)	Regular Resolution (PWM)		High Resolution (HRPWM)	
	Bits	%	Bits	%
20	12.3	0.0	***18.1***	0.000
50	11.0	0.0	***16.8***	0.001
100	10.0	0.1	***15.8***	0.002
150	9.4	0.2	***15.2***	0.003
200	9.0	0.2	***14.8***	0.004
250	8.6	0.3	***14.4***	0.005
500	7.6	0.5	***13.8***	0.007
1000	6.6	1.0	***12.4***	0.018
1500	6.1	1.5	***11.9***	0.027
2000	5.6	2.0	***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:

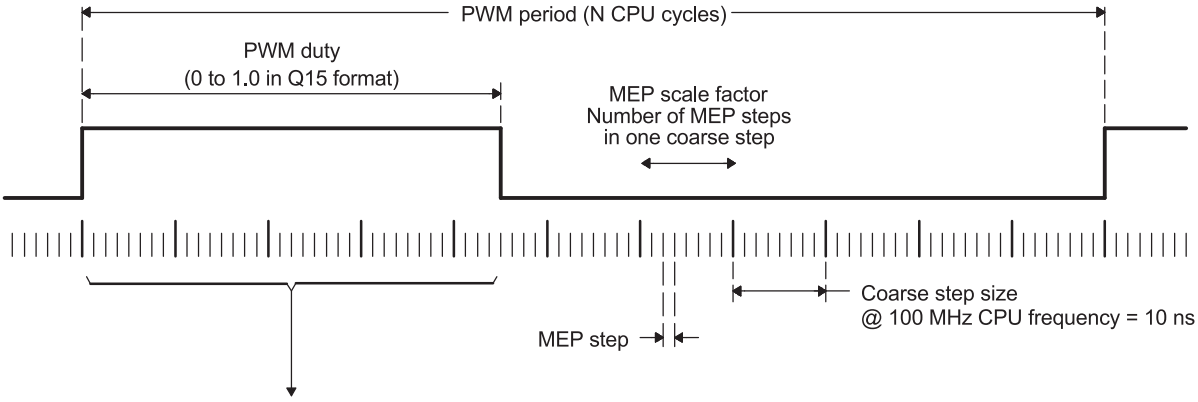
- Single-phase buck, boost, and flyback
- Multi-phase buck, boost, and flyback
- Phase-shifted full bridge
- Direct modulation of D-Class power amplifiers

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



Number of coarse steps = integer(PWMduty * PWMperiod)
Number of MEP steps = fraction(PWMduty * PWMperiod) * (MEPScaleFactor)

16−bit CMPA register value	= number of coarse steps
16−bit CMPAHR register value	[(number of MEP steps) +1.5†]<< 8

† For MEP range and rounding adjustment (0x0180 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 chapter can be found in the *Enhanced Pulse Width Modulator (ePWM)* chapter.

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)
CMPAHR	0x0008	Yes	Extension Register for HRPWM Duty (8 bits)
HRCNFG(1)	0x0020	No	HRPWM Configuration Register

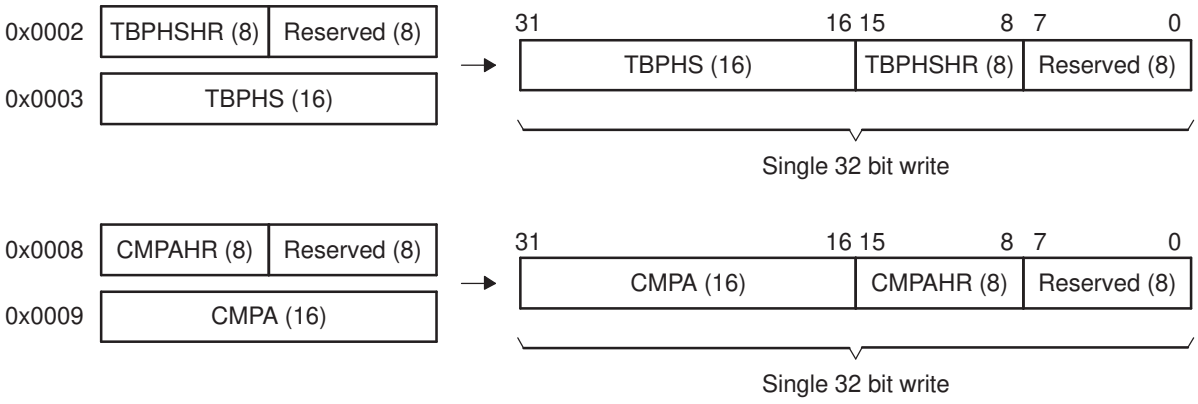
(1) This register is EALLOW protected.

4.2.1 Controlling the HRPWM Capabilities

The MEP of the HRPWM is controlled by two extension registers, each 8-bits wide. These two HRPWM registers are concatenated with the 16-bit TBPHS and CMPA registers used to control PWM operation.

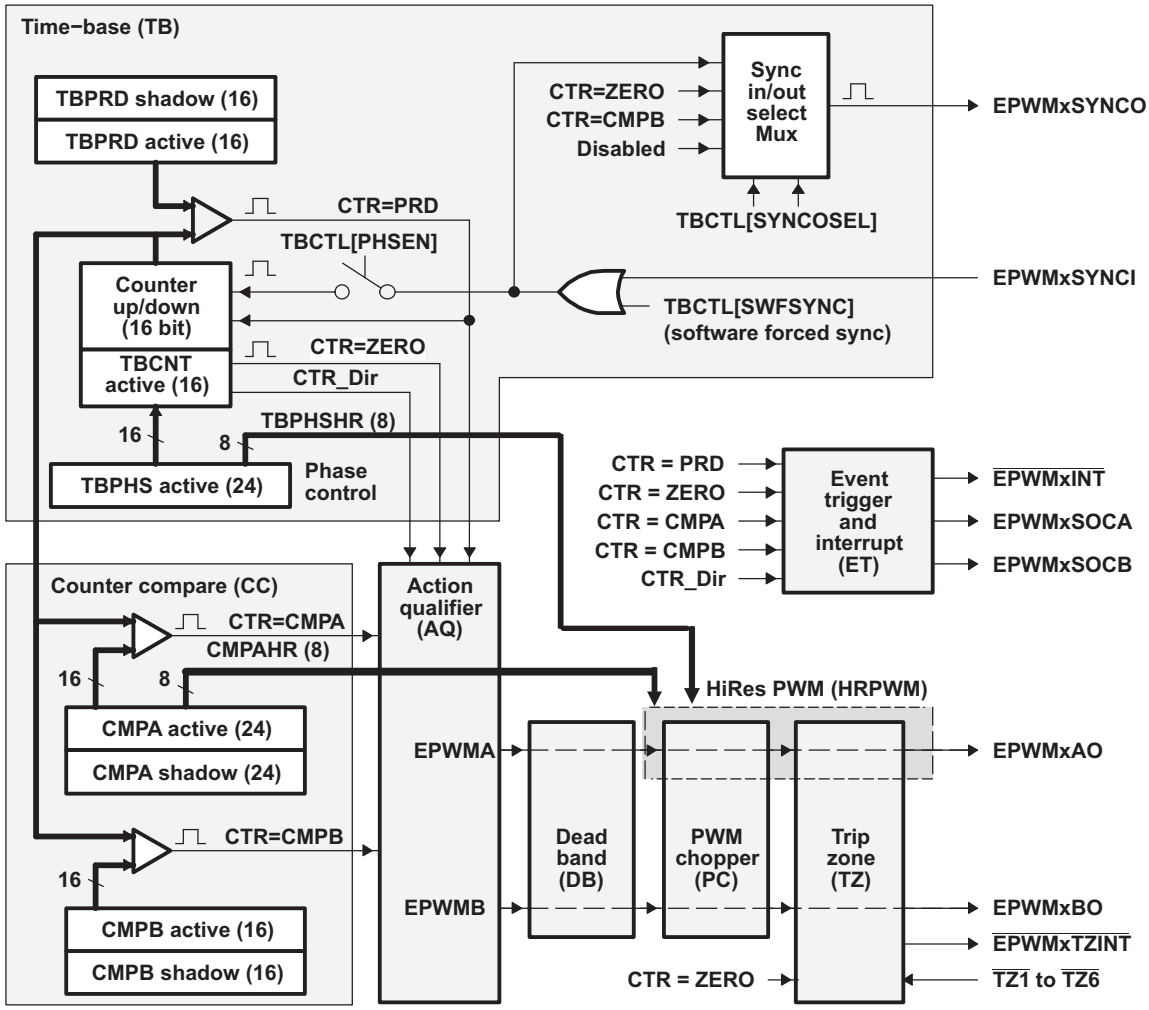
- TBPHSHR - Time Base Phase High Resolution Register
- CMPAHR - Counter Compare A High Resolution Register

Figure 4-3. HRPWM Extension Registers and Memory Configuration



HRPWM capabilities are controlled using the Channel A PWM signal path. Figure 4-4 shows how the HRPWM interfaces with the 8-bit extension registers.

Figure 4-4. HRPWM System Interface



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 configuration options for the following key operating modes :

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, while BE is used for topologies requiring phase shifting, e.g., phase shifted full bridge.

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.

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 register 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.

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 Table 4-3shows 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)
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, that is, CPU clock. TBCLK =SYSCLKOUT.
(2) Table data based on a MEP time resolution of 180 ps (this is an example value, see the device-specific data sheet for MEP limits.
(3) MEP steps applied = $T_{SYSCLKOUT}$/180 ps in this example.
(4) PWM minimum frequency is based on a maximum period value, that is, TBPRD = 65535. PWM mode is asymmetrical up-count.
(5) Resolution in bits is given for the maximum PWM frequency stated.

4.2.3.1 Edge Positioning

In a typical power control loop (e.g., switch modes, digital motor control [DMC], uninterruptible power supply [UPS]), a digital controller (PID, 2pole/2zero, lag/lead, etc.) issues a duty command, usually expressed in a per unit or percentage terms. Assume that for a particular operating point, the demanded duty cycle is 0.405 or 40.5% on time and the required converter PWM frequency is 1.25 MHz. In conventional PWM generation with a system clock of 100 MHz, the duty cycle choices are in the vicinity of 40.5%. In Figure 4-5, a compare value of 32 counts (that is, duty = 40%) is the closest to 40.5% that you can attain. This is equivalent to an edge position of 320 ns instead of the desired 324 ns. This data is shown in Table 4-4.

By utilizing the MEP, you can achieve an edge position much closer to the desired point of 324 ns. Table 4-4 shows that in addition to the CMPA value, 22 steps of the MEP (CMPAHR register) will position the edge at 323.96 ns, resulting in almost zero error. In this example, it is assumed that the MEP has a step resolution of 180 ps.

Figure 4-5. Required PWM Waveform for a Requested Duty = 40.5%

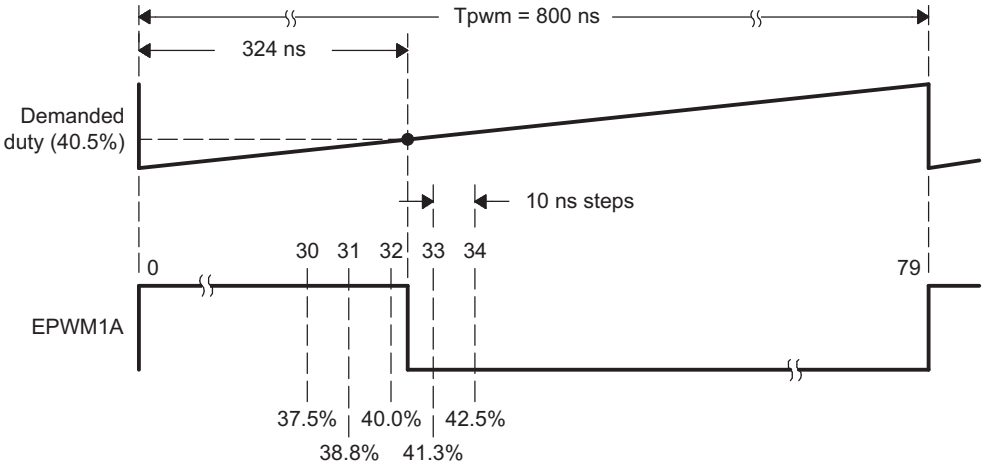


Table 4-4. CMPA vs Duty (left) and [CMPA:CMPAHR] vs Duty (right)

CMPA[1] [2] [3] (count)	DUTY %	High Time (ns)	CMPA (count)	CMPAHR (count)	Duty (%)	High Time (ns)
28	***35.0***	280	32	18	***40.405***	323.24
29	***36.3***	290	32	19	***40.428***	323.42
30	***37.5***	300	32	20	***40.450***	323.60
31	***38.8***	310	32	21	***40.473***	323.78
32	***40.0***	320	32	22	***40.495***	323.96
33	***41.3***	330	32	23	***40.518***	324.14
34	***42.5***	340	32	24	***40.540***	324.32
			32	25	***40.563***	324.50
Required			32	26	***40.585***	324.68
32.40	***40.5***	324	32	27	***40.608***	324.86

(1) System clock, SYSCLKOUT and TBCLK = 100 MHz, 10 ns
(2) For a PWM Period register value of 80 counts, PWM Period = 80 x 10 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.

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	=	10 ns (100 MHz)
PWM frequency	=	1.25 MHz (1/800 ns)
Required PWM duty cycle, **PWMDuty**	=	0.405 (40.5%)
PWM period in terms of coarse steps, **PWMperiod** (800 ns/10 ns)	=	80
Number of MEP steps per coarse step at 180 ps (10 ns /180 ps), **MEP_ScaleFactor**	=	55
Value to keep CMPAHR within the range of 1-255 and fractional rounding constant (default value)	=	1.5 (0180h in Q8 format)

Step 1: Percentage Integer Duty value conversion for CMPA register

CMPA register value	=	int(**PWMDuty*****PWMperiod**); int means integer part
	=	int(0.405*80)
	=	int(32.4)
CMPA register value	=	32 (20h)

Step 2: Fractional value conversion for CMPAHR register

CMPAHR register value	=	(frac(**PWMDuty*****PWMperiod**)***MEP_ScaleFactor** +1.5) << 8); frac means fractional part
	=	(frac(32.4) *55 + 1.5) <<8 Shift is to move the value as CMPAHR high byte
	=	(0.4 * 55 + 1.5) <<8
	=	(22 + 1.5) <<8
	=	23.5 * 256; Shifting left by 8 is the same as multiplying by 256.
	=	6016
CMPAHR value	=	1780h CMPAHR value = 1700h , lower 8 bits will be ignored by hardware.

NOTE: The MEP scale factor (MEP_ScaleFactor) varies with the system clock and device 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, that is, [CMPA:CMPAHR].

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 diagnostics are disabled
- 6 SYSCLK cycles after the period starts when SFO diagnostics are running

Duty cycle range limitations are illustrated in Figure 4-6 . This limitation imposes a lower duty cycle limit on the MEP. For example, precision edge control is not available all the way down to 0% duty cycle. Although for the first 3 or 6 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.

Figure 4-6. Low % Duty Cycle Range Limitation Example When PWM Frequency = 1 MHz

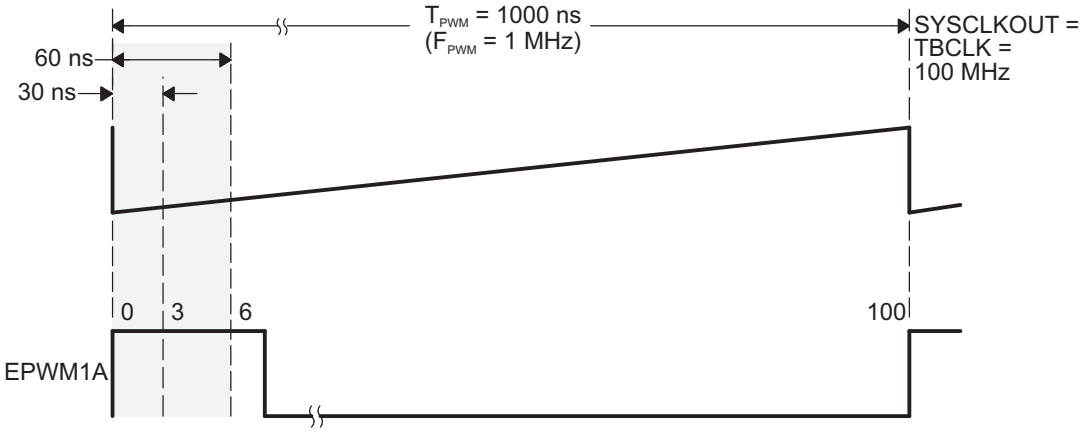


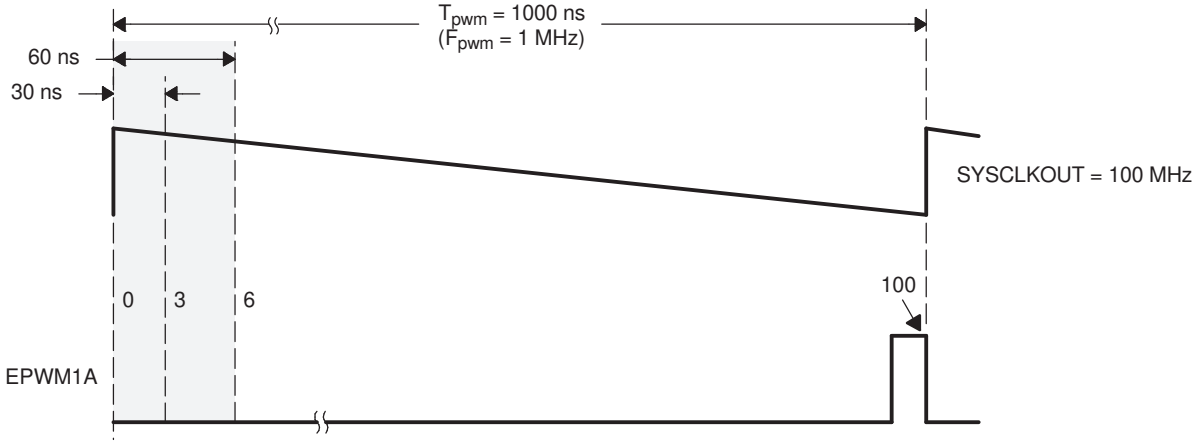
Table 4-5. Duty Cycle Range Limitation for 3 and 6 SYSCLK/TBCLK Cycles

PWM Frequency[(1)] (kHz)	3 Cycles Minimum Duty	6 Cycles SYSCLKOUT Minimum Duty
200	0.6%	1.2%
400	1.2%	2.4%
600	1.8%	3.6%
800	2.4%	4.8%
1000	3.0%	6.0%
1200	3.6%	7.2%
1400	4.2%	8.4%
1600	4.8%	9.6%
1800	5.4%	10.8%
2000	6.0%	12.0%

(1) System clock - $T_{SYSCLKOUT}$ = 10 ns
System clock = TBCLK = 100 MHz

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. This is illustrated in Figure 4-7. 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 4-7. High % Duty Cycle Range Limitation Example when PWM Frequency = 1 MHz



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 to [CMPA:CMPAHR] 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, each 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 two SFO functions that utilize this hardware and determines the optimum MEP_ScaleFactor. As such, MEP Control and Diagnostics registers are reserved for TI use.

Currently, there are two released versions of the SFO library - SFO_TI_Build.lib and SFO_TI_Build_V5.lib. Versions 2, 3, and 4 were TI Internal only. A detailed description of the SFO_TI_Build.lib software functions follows below.

NOTE: Information on the SFO_TI_Build_V5.lib software functions, which support up to 16 HRPWM channels, can be found in , along with a high-level comparison table between the two library versions.

Table 4-6 provides functional descriptions of the two SFO library routines in SFO_TI_Build.lib.

Table 4-6. SFO Library Routines

Function	Description
SFO_MepDis(n)	Scale Factor Optimizer with MEP Disabled
	This routine runs faster, as the calibration logic works when HRPWM capabilities are disabled; therefore, HRPWM capabilities cannot be run concurrently when the ePWMn is being used. If SYSCLKOUT = TBCLK = 100 MHz and assuming MEP steps size is 150 ps Typical value at 100 MHz = 66 MEP steps per unit TBCLK (10 ns) The function returns a value in the variable array: MEP_ScaleFactor[n] = Number of MEP steps/SYSCLKOUT If TBCLK is not equal to SYSCLKOUT, then the returned value must be adjusted to reflect the correct TBCLK: MEP steps per TBCLK = MEP_ScaleFactor[n] * (SYSCLKOUT/TBCLK)(1) Example: If TBCLK =SYSCLKOUT/2, MEP steps per TBCLK = MEP_ScaleFactor[n] * (100/50) =66 * 2 = 132 (1) **Constraints when using this function:** SFO_MepDis(n) can be used with SYSCLKOUT from 50 MHz to maximum SYSCLK frequency. MEP diagnostics logic uses SYSCLKOUT not TBCLK and hence SYSCLKOUT restriction is an important constraint. SFO_MepDis(n) function does not require a starting Scale Factor value. Additionally, TBCLK must equal SYSCLKOUT. **When to use** If one of the ePWM modules is not used in HRPWM mode, then it can be dedicated to run the SFO diagnostics for the modules that are running HRPWM mode. Here the single MEP_ScaleFactor value obtained can be applied to other ePWM modules. This assumes that all HRPWM module's MEP steps are similar but may not be identical. The ePWM module that is not active in HRPWM mode is still fully operational in conventional PWM mode and can be used to drive PWM pins. The SFO function only makes use of the MEP diagnostics logic. The other ePWM modules operating in HRPWM mode incur only a 3-cycle minimum duty limitation.
SFO_MepEn(n)	Scale Factor Optimizer with MEP Enabled
	This routine runs slower as the calibration logic is used concurrently while HRPWM capabilities are being used by the ePWM module. If SYSCLKOUT = TBCLK = 100 MHz and assuming MEP steps size is 150 ps Typical value at 100 MHz = 66 MEP steps per unit TBCLK (10 ns) The function returns a value in the variable array: MEP_ScaleFactor[n](1) = Number of MEP steps/SYSCLKOUT = Number of MEP steps/TBCLK **Constraints when using this function:** SFO_MepEn(n) function is restricted to be used with SYSCLKOUT of 60 MHz maximum SYSCLK frequency. MEP diagnostics logic uses SYSCLKOUT not TBCLK and hence SYSCLKOUT restriction is an important constraint. SFO_MepEn(n) function does require a starting Scale Factor value.MEP_ScaleFactor[0] needs to be initialized to a typical MEP step size value. Additionally, TBCLK must equal SYSCLKOUT. **NOTE:** SFO_MepEn(n) only supports the following HRPWM configuration: • HRCNFG[HRLOAD] = 0 (load on CTR = ZERO) • HRCNFG[EDGMODE] = 10 (falling edge MEP control) SFO_MepEn(n)_V5B.lib includes an SFO_MepEn(n)_V5(n) function which does not have this limitation. **When to use** If the application requires all ePWM modules to have HRPWM capability (that is, MEP is operational), then the SFO_MepEn(n) function can run for each of the active ePWM modules with HRPWM capability. • In the above case, a 6-cycle MEP inactivity zone exists at the start of the PWM period. See Section 4.2.3.3 on duty cycle range limitation. • It is also possible to run the SFO_MepEn(n) function for only one ePWM module and to use the SFO return value for the other modules. In this case only one ePWM module incurs the 6-cycle limitation, and remaining modules incur only a 3-cycle minimum duty limitation. See "Duty cycle limitation" section. This assumes that all HRPWM module's MEP steps are similar but may not be identical.

(1) n is the ePWM module number on which the SFO function operates.
for example, n = 1, 2, 3, or 4 for the F2808. Check your device data manual for device configurations.

Both routines can be run as background tasks in a slow loop requiring negligible CPU cycles. In most applications only one of these routines will be needed. However, if the application has free HRPWM resources then both the routines could be used. The repetition rate at which an SFO function needs to be executed depends on the applications 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 HRPWM feature with no SFO diagnostics, HRPWM logic will not be active for the first 3 TBCLK cycles of the PWM period. While running the application in this configuration, if CMPA register value is less than 3 cycles, then its CMPAHR register must be cleared to zero. This would avoid any unexpected transitions on PWM signal.

However, if SFO diagnostic function SFO_MepEn is used in the background, then HRPWM logic will not be active for the first 6 TBCLK cycles of PWM period. While using SFO_MepEn function if CMPA register value is less than 6 cycles, then its CMPAHR register must be cleared to zero. This would avoid any unexpected transitions on PWM signal. Also note that the SFO_MepDis function cannot be used concurrently with PWM signals with HRPWM enabled (see the previous section for details).

4.2.4.1 Software Usage

Software library functions SFO_MepEn(int n) and SFO_MepDis(int n) calculate the MEP scale factor for ePWMn modules, where n = 1, 2, 3, or 4. 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 array of integer variables of length 5 called MEP_ScaleFactor[5]. For example, see Table 4-7.

Table 4-7. Factor Values

Software function calls	Functional description	Updated Variable MEP_ScaleFactor[5](1)
SFO_MepDis(n)		
SFO_MepDis(1);	Returns the scale factor value to array index 1	MEP_ScaleFactor[1]
SFO_MepDis(2);	Returns the scale factor value to array index 2	MEP_ScaleFactor[2]
SFO_MepDis(3);	Returns the scale factor value to array index 3	MEP_ScaleFactor[3]
SFO_MepDis(4);	Returns the scale factor value to array index 4	MEP_ScaleFactor[4]
SFO_MepEn(n)		
SFO_MepEn(1);	Returns the scale factor value to array index 1	MEP_ScaleFactor[1]
SFO_MepEn(2);	Returns the scale factor value to array index 2	MEP_ScaleFactor[2]
SFO_MepEn(3);	Returns the scale factor value to array index 3	MEP_ScaleFactor[3]
SFO_MepEn(4);	Returns the scale factor value to array index 4	MEP_ScaleFactor[4]

(1) MEP_ScaleFactor[0] variable is a starting value and used by the SFO software functions internally

To use the HRPWM feature of the ePWMs, it is recommended that the SFO functions be used as described here.

Step 1. Add Include Files

The SFO.h file needs to be included as follows. This include file is mandatory while using the SFO library function.

Example 4-1. A Sample of How to Add Include Files

```
#include "device280x_Device.h" // device280x Headerfile
#include "device280x_EPWM_defines.h" // init defines
#include "SFO.h" // SFO lib functions (needed for HRPWM)
```

Step 2. Element Declaration

Declare a 5-element array of integer variables as follows:

Example 4-2. Declaring an Element

```
int MEP_ScaleFactor[5] = {0,0,0,0,0};  // Scale factor values for ePWM1-4
int MEP_ScaleFactor1, MEP_ScaleFactor2, MEP_ScaleFactor3, MEP_ScaleFactor4 // Not required by library
volatile struct EPWM_REGS *ePWM[] = {0, &EPwm1Regs, &EPwm2Regs, &EPwm3Regs, &EPwm4Regs};
```

Step 3. MEP_ScaleFactor Initialization

After power up, the SFO_MepEn(n) function needs a starting Scale Factor value. This value can be conveniently determined by using one of the ePWM modules to run the SFO_MepDis(n) function prior to configuring its PWM outputs for the application. SFO_MepDis(n) function does not require a starting Scale Factor value.

As part of the one-time initialization code, include the following:

Example 4-3. Initializing With a Scale Factor Value

```
// MEP_ScaleFactor variables initialized using function SFO_MepDis
while (MEP_ScaleFactor[1] == 0) SFO_MepDis(1); //SFO for HRPWM1
while (MEP_ScaleFactor[2] == 0) SFO_MepDis(2); //SFO for HRPWM2
while (MEP_ScaleFactor[3] == 0) SFO_MepDis(3); //SFO for HRPWM3
while (MEP_ScaleFactor[4] == 0) SFO_MepDis(4); //SFO for HRPWM4

// Initialize a common seed variable MEP_ScaleFactor[0] // required for all SFO functions
      MEP_ScaleFactor[0] = MEP_ScaleFactor[1];          // Common variable for SFOMepEn(n) function
```

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 C2000Ware available from the TI website.

Example 4-4. SFO Function Calls

```
main()
{
     //  User code
     //  Case1: ePWM1,2,3,4 are running in HRPWM mode
         SFO_MepEn(1); // Each of these of function enables
         SFO_MepEn(2); // the respective MEP diagnostic logic

         SFO_MepEn(3); // and returns MEP Scale factor value
         SFO_MepEn(4);

         MEP_ScaleFactor1 = MEP_ScaleFactor[1]; // used for ePWM1
         MEP_ScaleFactor2 = MEP_ScaleFactor[2]; // used for ePWM2
         MEP_ScaleFactor3 = MEP_ScaleFactor[3]; // used for ePWM3
         MEP_ScaleFactor4 = MEP_ScaleFactor[4]; // used for ePWM4

     //  Case2:ePWM1,2,3 only are running in HRPWM mode.
             One of the ePWM channel(as an example ePWM4) is used as for
             Scale factor calibration
     //  Here minimum duty cycle limitation is 3 clock cycles.
     //
     //  HRPWM 4 MEP diagnostics circuit is used to estimate the MEP steps
     //  with the assumption that all HRPWM channels behave similarly
     //  though may not be identical.

         SFO_MepDis(4); // MEP steps using ePWM4
         MEP_ScaleFactor1 = MEP_ScaleFactor[4]; // used for ePWM1
         MEP_ScaleFactor2 = MEP_ScaleFactor1    // used for ePWM2
         MEP_ScaleFactor3 = MEP_ScaleFactor1    // used for ePWM3
         MEP_ScaleFactor4 = MEP_ScaleFactor1    // used for ePWM4
```

4.2.5 HRPWM Examples Using Optimized Assembly Code

The best way to understand how to use the HRPWM capabilities is through two real examples:

1. Simple buck converter using asymmetrical PWM (that is, count-up) with active high polarity.
2. DAC function using simple R+C reconstruction filter.

The following examples all have Initialization/configuration code written in C. To make these easier to understand, the #defines shown below are used. Note, #defines can be found in C2000Ware.

Example 4-5 assumes MEP step size of 150 ps and does not use the SFO library.

Example 4-5. #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
```

4.2.5.1 Implementing a Simple Buck Converter

In this example, the PWM requirements are:

- PWM frequency = 1 MHz (that is, TBPRD = 100)
- PWM mode = asymmetrical, up-count
- Resolution = 12.7 bits (with a MEP step size of 150 ps)

Figure 4-8 and Figure 4-9 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 4-8. Simple Buck Controlled Converter Using a Single PWM

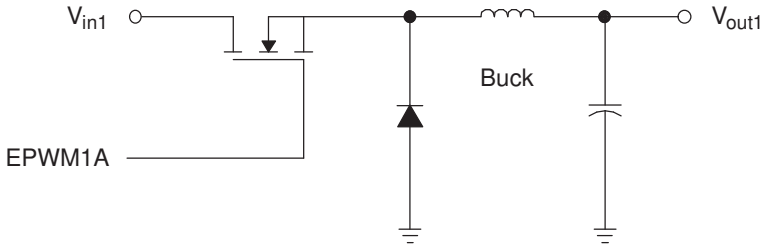
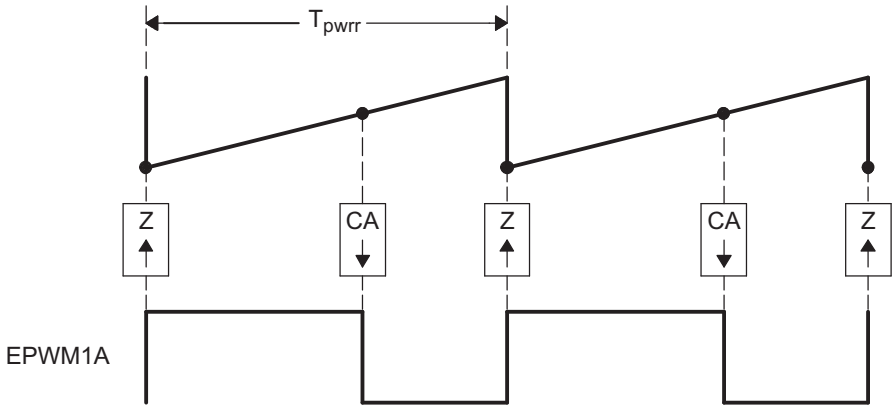


Figure 4-9. PWM Waveform Generated for Simple Buck Controlled Converter



The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

Example 4-6 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 4-6. 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 1000 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 = 66*256;                             //  Start with typical Scale Factor
                                                              //  value for 100 MHz
                                                              //  Note: Use SFO functions to update
                                                               // MEP_ScaleFactor dynamically
}
```

Example 4-7 shows an assembly example of run-time code for the HRPWM buck converter.

Example 4-7. 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, #0x180                ; MEP range and rounding adjustment
     MOVL *XAR3,ACC                 ; CMPA:CMPAHR(31:8) <= ACC
; Output for EPWM1B (Regular Res) Optional - for comparison purpose only
     MOV *+XAR3[2],AH ; Store ACCH to regular CMPB
```

4.2.5.2 Implementing a DAC function Using an R+C Reconstruction Filter

In this example, the PWM requirements are:

- PWM frequency = 400 kHz (that is, TBPRD = 250)
- PWM mode = Asymmetrical, Up-count
- Resolution = 14 bits (MEP step size = 150 ps)

Figure 4-10 and Figure 4-11 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 4-10. Simple Reconstruction Filter for a PWM Based DAC

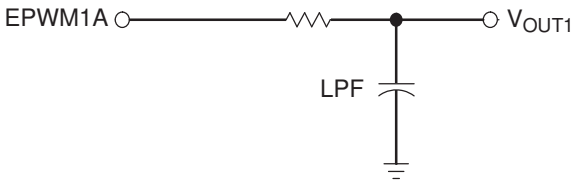
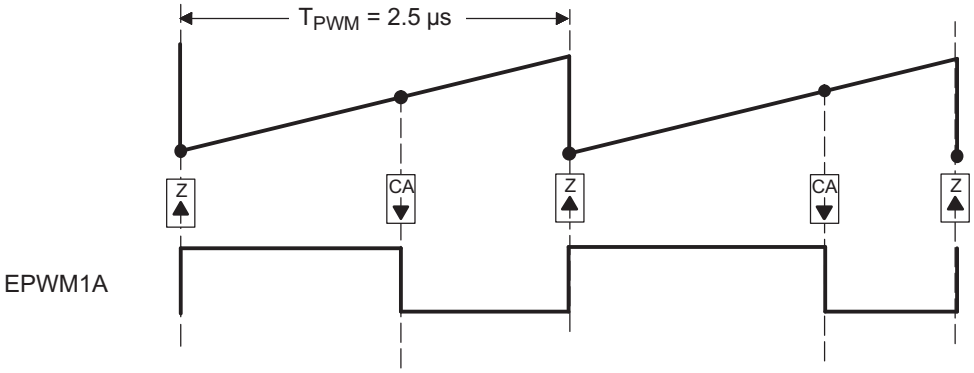


Figure 4-11. PWM Waveform Generated for the PWM DAC Function



The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

This example assumes a typical MEP_ScaleFactor and does not use the SFO library.

Example 4-8 shows the Initialization code. The first part is configured for conventional PWM. The second part sets up the HRPWM resources.

Example 4-8. PWM DAC Function Initialization Code

```
void HrPwmDacDrvCnf(void) {
         // Config for conventional PWM first
            EPwm1Regs.TBCTL.bit.PRDLD = TB_IMMEDIATE;   // Set Immediate load
            EPwm1Regs.TBPRD = 250;                      // Period set for 400 kHz PWM
            hrDAC_period = 250;                         // 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 = 66*256;                     // Start with typical Scale Factor
                                                          // value for 100 MHz.
                                                          // Use SFO functions to update
                                                          // MEP_ScaleFactor dynamically
}
```

Example 4-9 shows an assembly example of run-time code that can execute in a high-speed ISR loop.

Example 4-9. 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, #0x180                   ; 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 Registers

4.3.1 Register Summary

A summary of the registers required for the HRPWM is shown in Table 4-8.

Table 4-8. Register Descriptions

Name	Offset	Size(x16)	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
Reserved	**0x0006**	**1/0**	
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**

4.3.2 Registers and Field Descriptions

Figure 4-12. HRPWM Configuration Register (HRCNFG)

15	8
Reserved	
R-0	

7	4	3	2	1	0
Reserved		HRLOAD	CTLMODE	EDGMODE	
R-0		R/W-0	R/W-0	R/W-0	

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

Table 4-9. HRPWM Configuration Register (HRCNFG) Field Descriptions

Bit	Field	Value	Description[(1)]
15-4	Reserved		Reserved
3	HRLOAD		Shadow mode bit: Selects the time event that loads the CMPAHR shadow value into the active register:
		0	CTR = zero (counter equal zero)
		1	CTR=PRD (counter equal period)
			Note: Load mode selection is valid only if CTLMODE=0 has been selected (bit 2). You should select this event to match the selection of the CMPA load mode (that is, CMPCTL[LOADMODE] bits) in the EPWM module as follows:
			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 (should not be used with HRPWM)
			11 Freeze (no loads possible – should not be used with HRPWM)
2	CTLMODE		Control Mode Bits: Selects the register (CMP or TBPHS) that controls the MEP:
		0	CMPAHR(8) Register controls the edge position (that is, this is duty control mode). (default on reset)
		1	TBPHSHR(8) Register controls the edge position (that is, 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
		10	MEP control of falling edge
		11	MEP control of both edges

(1) This register is EALLOW protected.

Figure 4-13. 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 4-10. Counter Compare A High-Resolution Register (CMPAHR) Field Descriptions

Bit	Field	Value	Description
15-8	CMPAHR		Compare A High Resolution register bits for MEP step control. A minimum value of 0x0001 is needed to enable HRPWM capabilities. Valid MEP range of operation 1-255h.
7-0	Reserved		Any writes to these bit(s) must always have a value of 0.

Figure 4-14. Time Base 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 4-11. Time Base Phase High-Resolution Register (TBPHSHR) Field Descriptions

Bit	Field	Value	Description
15-8	TBPHSH		Time base phase high resolution bits
7-0	Reserved		Any writes to these bit(s) must always have a value of 0.

Enhanced Capture (eCAP)

This chapter describes the enhanced capture (eCAP) module, which is used in systems where accurate timing of external events is important.

Topic **Page**

5.1 Introduction

This eCAP module is a Type-0 eCAP. See the *C2000 Real-Time Control Peripheral Reference Guide* for a list of all devices with an eCAP module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

5.2 Features

Features for eCAP include:

- Speed measurements of rotating machinery (for example, 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 are dedicated to a single input pin
- When not used in capture mode, the eCAP module can be configured as a single-channel PWM output

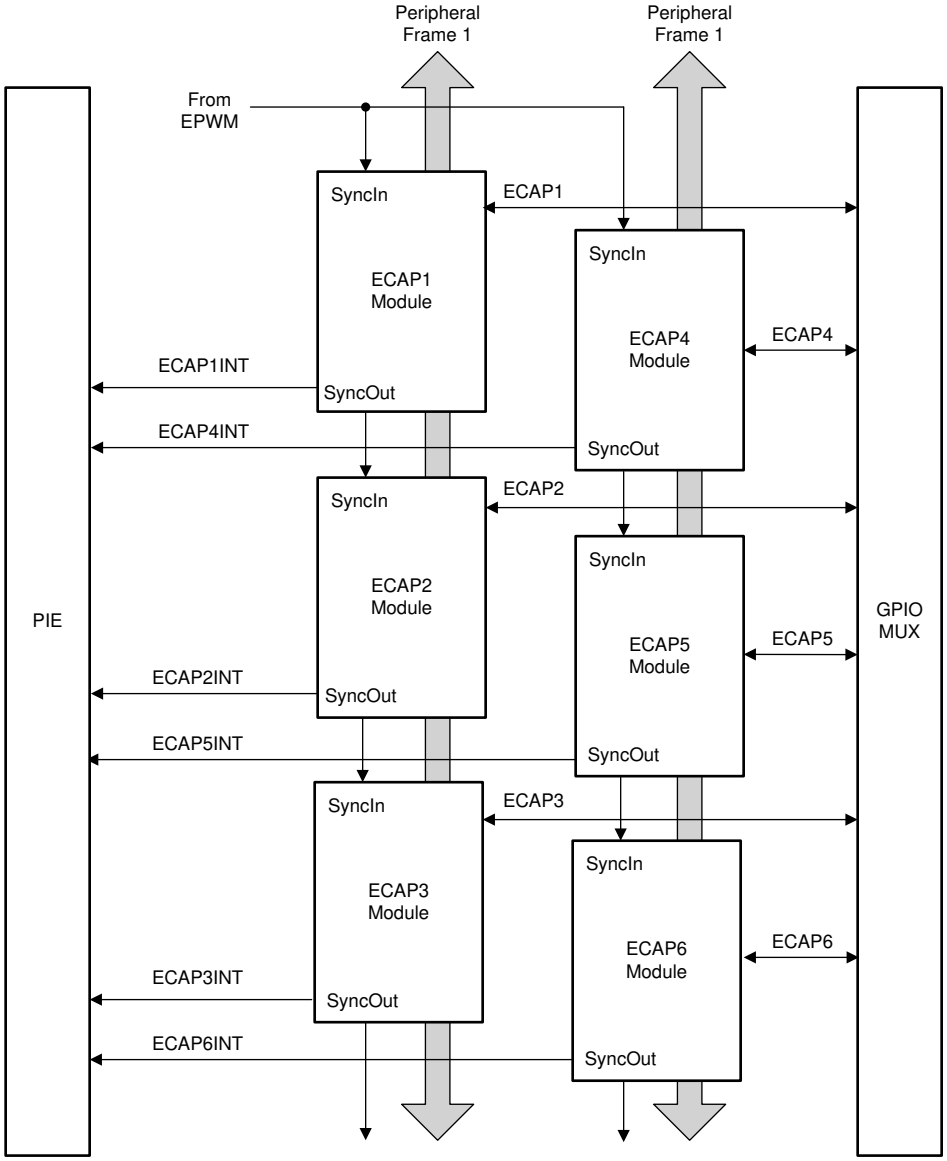
5.3 Description

The eCAP module represents one complete capture channel that can be instantiated multiple times, depending on the target device. In the context of this guide, one eCAP channel has the following independent key resources:

- Dedicated input capture pin
- 32-bit time base (counter)
- 4 x 32-bit time-stamp capture registers (CAP1-CAP4)
- Four-stage sequencer (modulo4 counter) that is synchronized to external events, eCAP pin rising/falling edges.
- Independent edge polarity (rising/falling edge) selection for all four events
- Input capture signal prescaling (from 2-62 or bypass)
- One-shot compare register (two bits) to freeze captures after 1-4 time-stamp events
- Control for continuous time-stamp captures using a four-deep circular buffer (CAP1-CAP4) scheme
- Interrupt capabilities on any of the four capture events

Multiple identical eCAP modules can be contained in a system as shown in Figure 5-1. The number of modules is device-dependent and is based on target application needs.

Figure 5-1. Multiple eCAP Modules In A C28x System

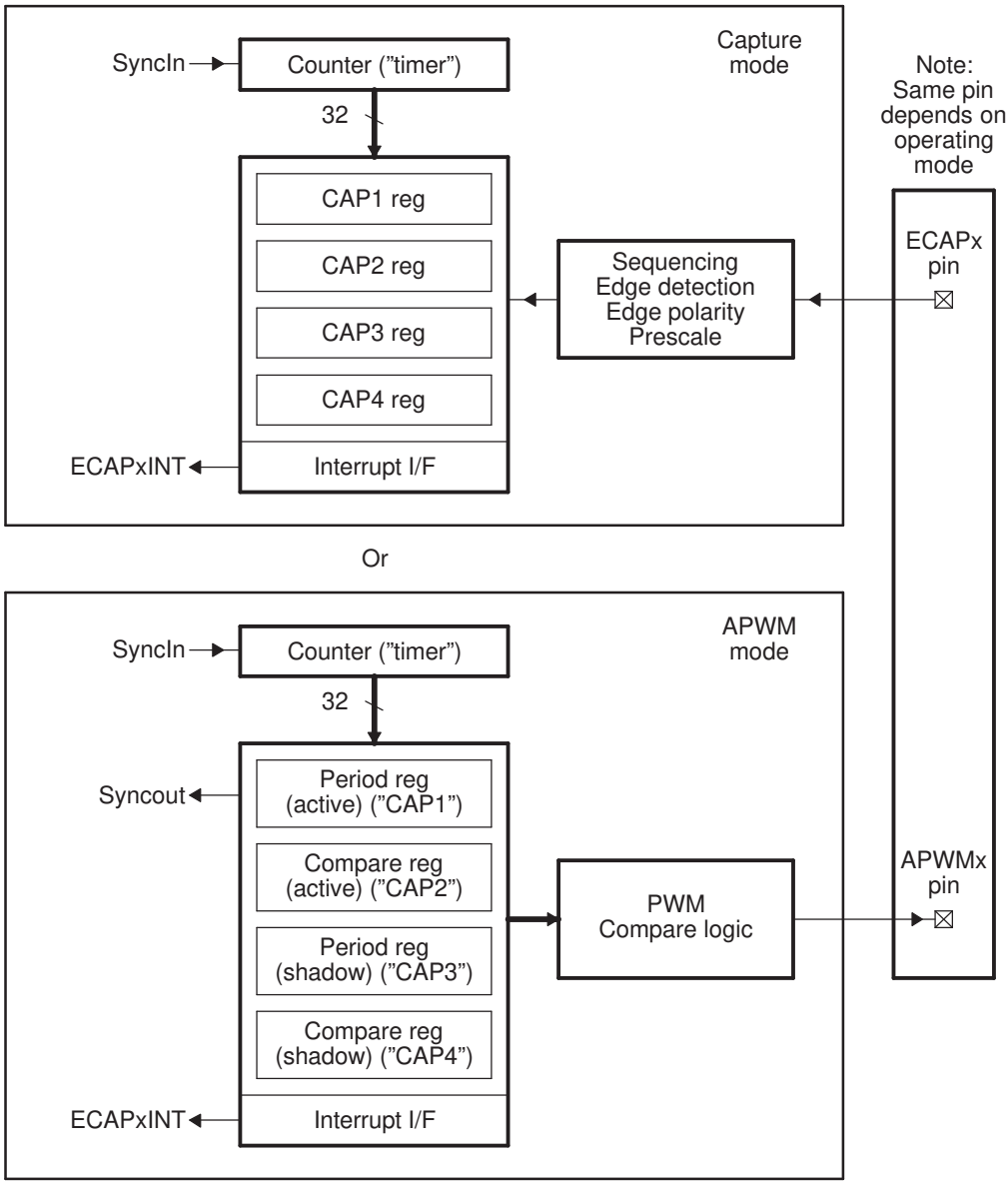


5.4 Capture and APWM Operating Mode

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 5-2 is a high-level view of both the capture and auxiliary pulse-width modulator (APWM) modes of operation.

Figure 5-3 further descries the output of the eCAP in APWM mode based on the CMP and PRD values.

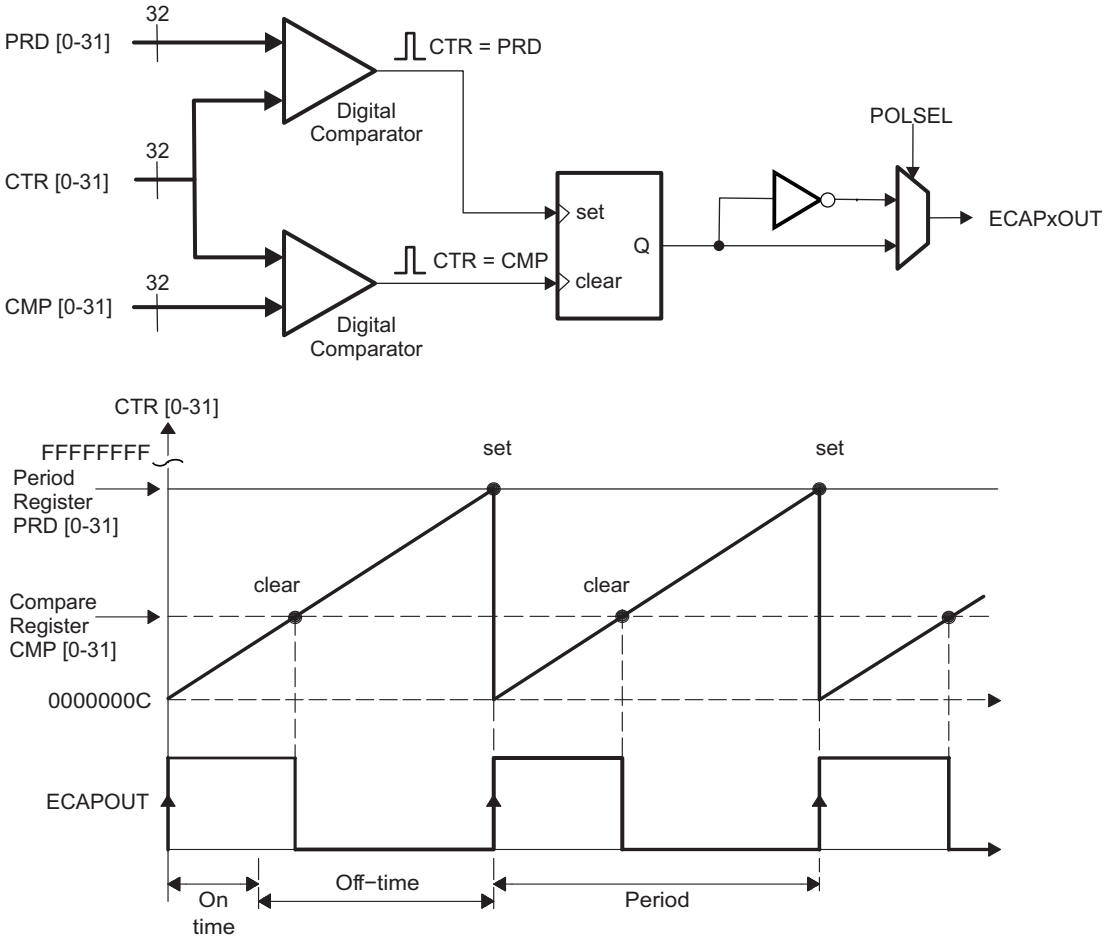
Figure 5-2. Capture and APWM Modes of Operation



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.

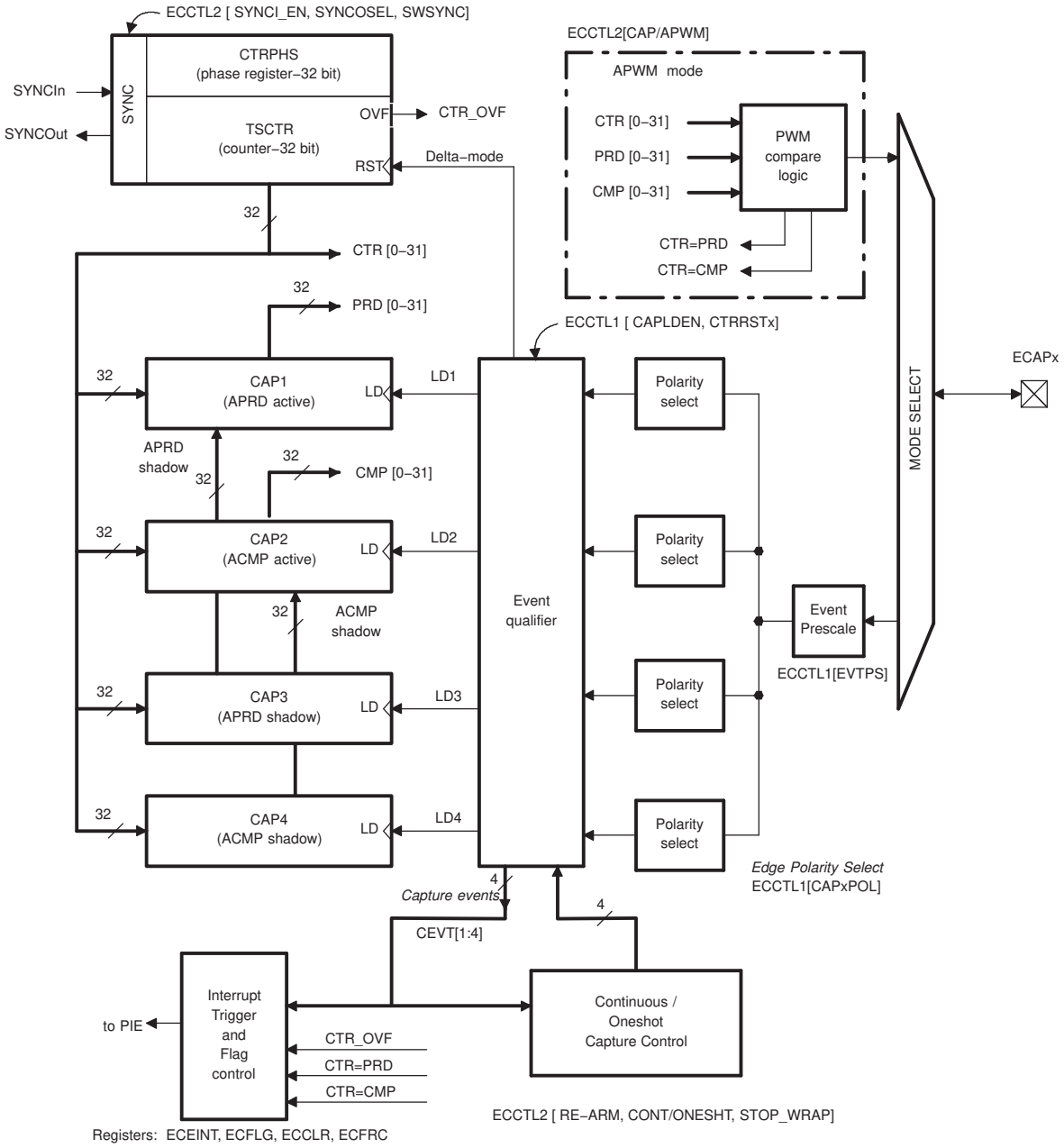
Figure 5-3. Counter Compare and PRD Effects on the eCAP Output in APWM Mode



5.5 Capture Mode Description

Figure 5-4 shows the various components that implement the capture function.

Figure 5-4. eCAP Block Diagram

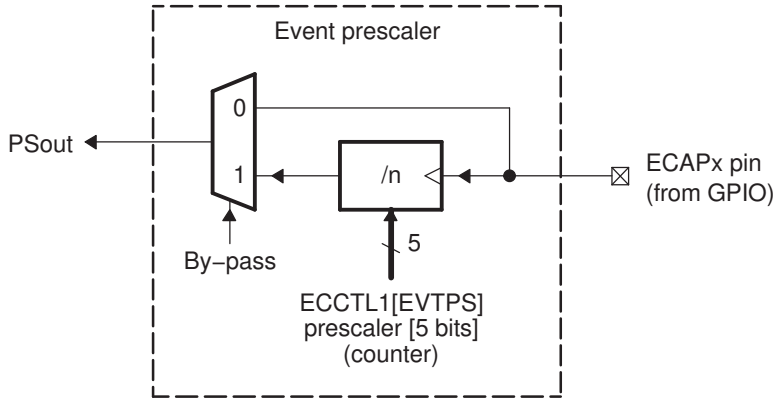


5.5.1 Event Prescaler

- An input capture signal (pulse train) can be prescaled by N = 2-62 (in multiples of 2) or can bypass the prescaler.

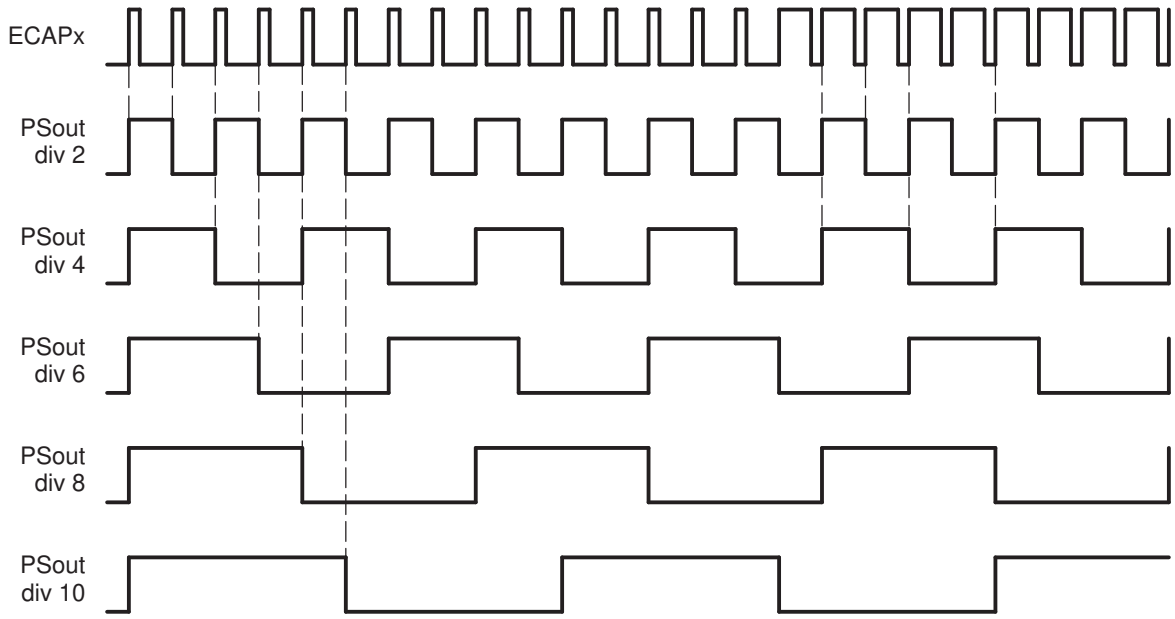
  This is useful when very high frequency signals are used as inputs. Figure 5-5 shows a functional diagram and Figure 5-6 shows the operation of the prescale function.

Figure 5-5. Event Prescale Control



A When a prescale value of 1 is chosen (ECCTL1[13:9] = 0,0,0,0,0), the input capture signal bypasses the prescale logic completely.

Figure 5-6. Prescale Function Waveforms



5.5.2 Edge Polarity Select and Qualifier

Functionality and features include:

- Four independent edge polarity (rising edge/falling edge) selection muxes are used, one for each capture event.
- Each edge (up to 4) is event qualified by the Modulo4 sequencer.
- The edge event is gated to its respective CAPx register by the Mod4 counter. The CAPx register is

loaded on the falling edge.

5.5.3 Continuous/One-Shot Control

Operation of eCAP in Continuous/One-Shot mode:

- The Mod4 (2-bit) counter is incremented via edge qualified events (CEVT1-CEVT4).
- The Mod4 counter continues counting (0->1->2->3->0) and wraps around unless stopped.
- 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.

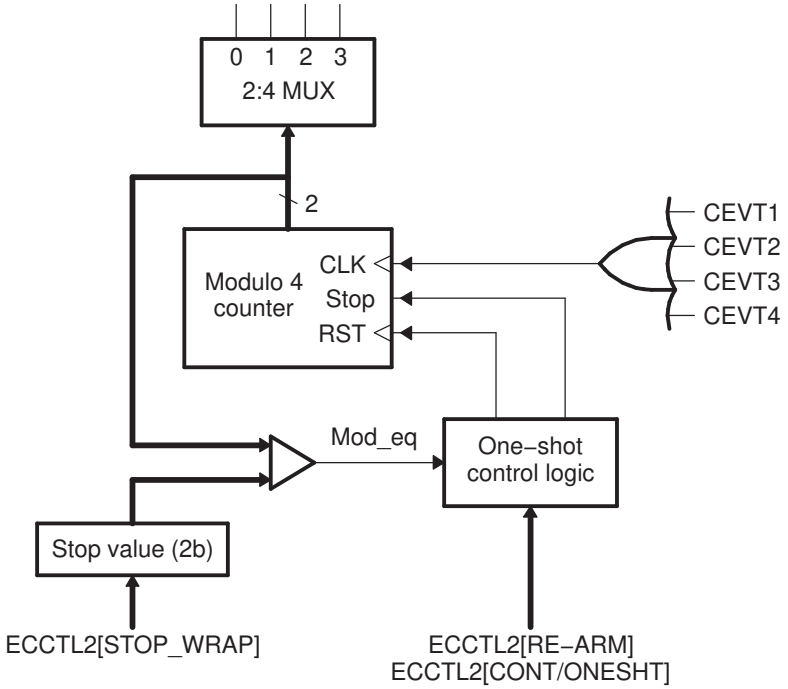
The continuous/one-shot block controls the start, stop and reset (zero) functions of the Mod4 counter, via a mono-shot type of action that can be triggered by the stop-value comparator and re-armed via software control.

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 (time stamps).

Re-arming prepares the eCAP module for another capture sequence. Also, re-arming clears (to zero) the Mod4 counter and permits loading of CAP1-4 registers again, providing the CAPLDEN bit is set.

In continuous mode, the Mod4 counter continues to run (0->1->2->3->0, the one-shot action is ignored, and capture values continue to be written to CAP1-4 in a circular buffer sequence.

Figure 5-7. Details of the Continuous/One-shot Block



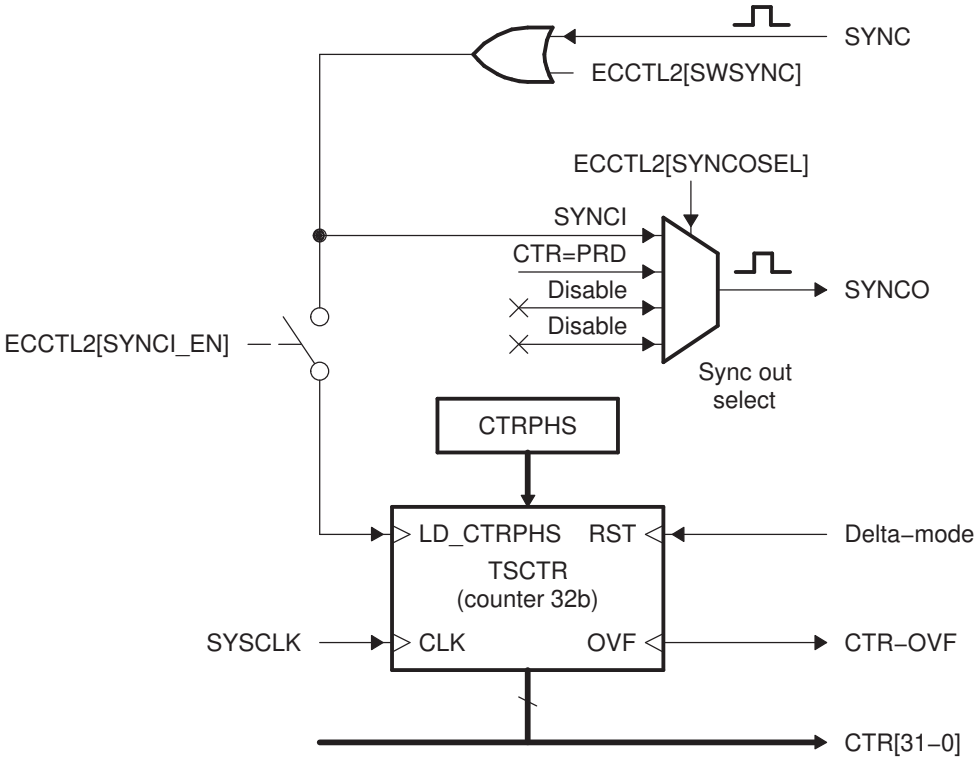
5.5.4 32-Bit Counter and Phase Control

This counter provides the time-base for event captures, and is clocked via the system clock.

A phase register is provided to achieve synchronization with other counters, via a hardware and software forced sync. This is useful in APWM mode when a phase offset between modules is needed.

On any of the four event loads, an option to reset the 32-bit counter is given. This is useful for time difference capture. The 32-bit counter value is captured first, then it is reset to 0 by any of the LD1-LD4 signals.

Figure 5-8. Details of the Counter and Synchronization Block



5.5.5 CAP1-CAP4 Registers

These 32-bit registers are fed by the 32-bit counter timer bus, CTR[0-31] and are loaded (capture a time-stamp) when their respective LD inputs are strobed.

Control bit CAPLDEN can inhibit loading of the capture registers. During one-shot operation, this bit is cleared (loading is inhibited) automatically when a stop condition occurs, StopValue = Mod4.

CAP1 and CAP2 registers become the active period and compare registers, respectively, in APWM mode.

CAP3 and CAP4 registers become the respective shadow registers (APRD and ACMP) for CAP1 and CAP2 during APWM operation.

5.5.6 Interrupt Control

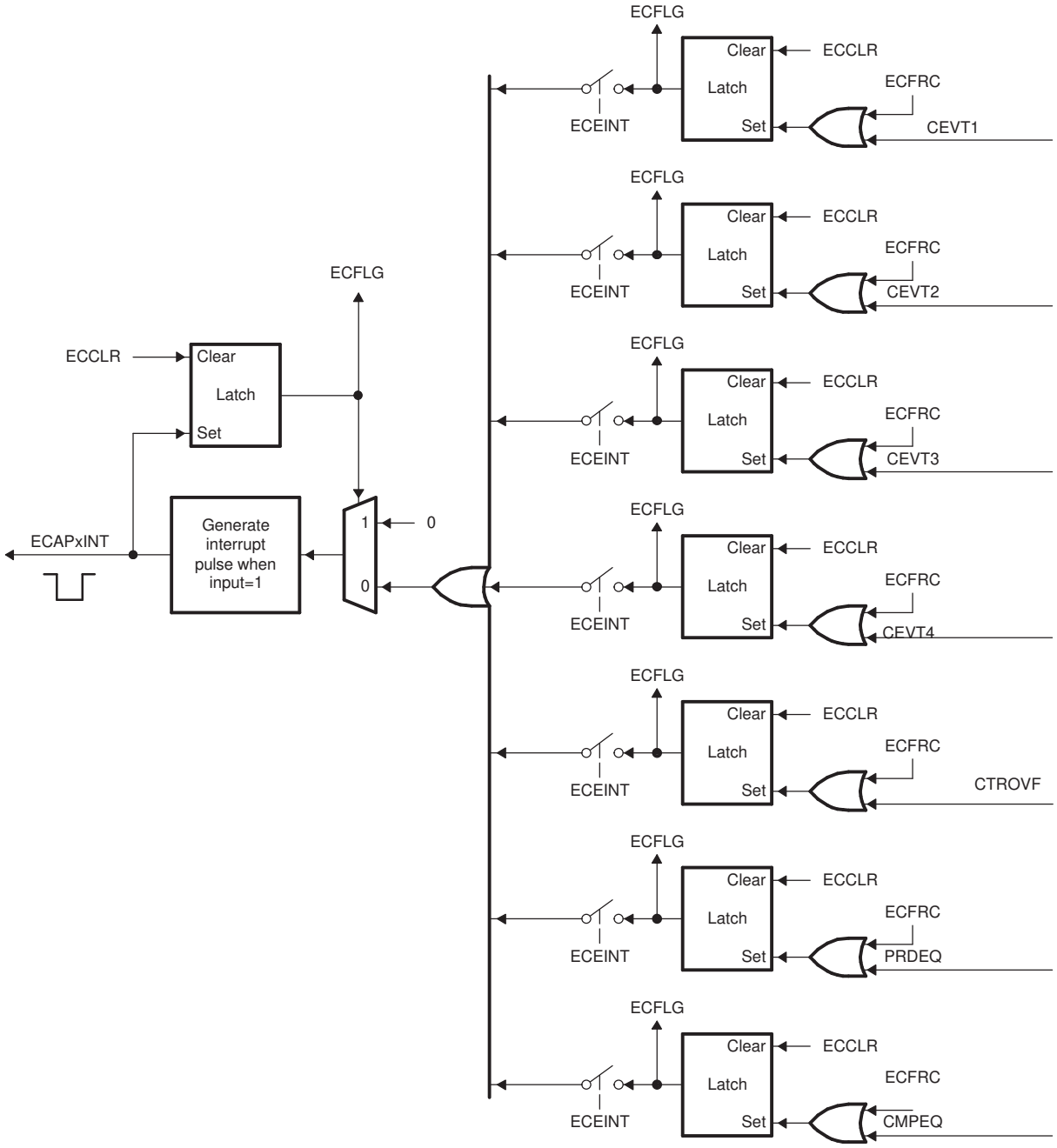
Operation and features of eCAP Interrupt Control include:

- 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 (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 5-9. Interrupts in eCAP Module



5.5.7 Shadow Load and Lockout Control

In capture mode, this logic inhibits (locks out) any shadow loading of CAP1 or CAP2 from APRD and ACMP registers, respectively.

In APWM mode, shadow loading is active and two choices are permitted:

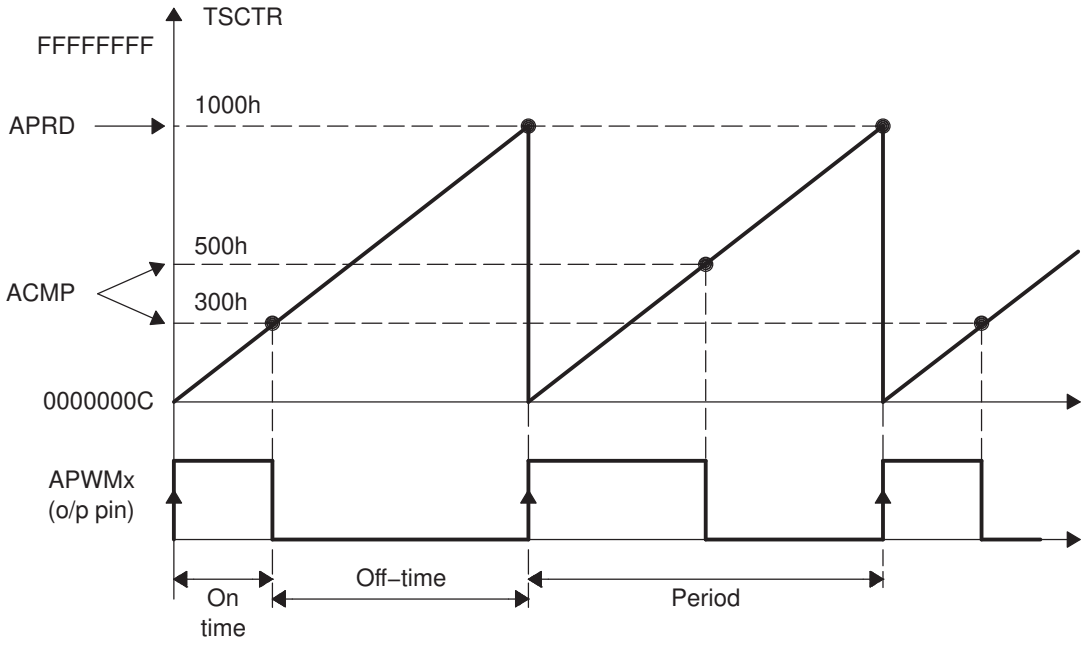
- Immediate - APRD or ACMP are transferred to CAP1 or CAP2 immediately upon writing a new value.
- On period equal, CTR[31:0] = PRD[31:0].

5.5.8 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, during run-time, you only need to use the shadow registers.

Figure 5-10. PWM Waveform Details Of APWM Mode Operation



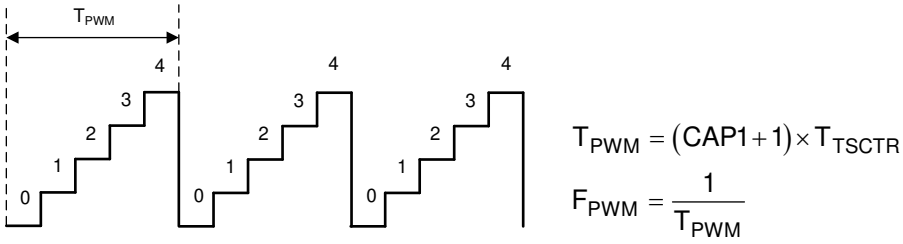
The behavior of APWM active high mode (APWMPOL == 0) is as follows:

```
CMP = 0x00000000, output low for duration of period (0% duty)
CMP = 0x00000001, output high 1 cycle
CMP = 0x00000002, output high 2 cycles
CMP = PERIOD, output high except for 1 cycle (<100% duty)
CMP = PERIOD+1, output high for complete period (100% duty)
CMP > PERIOD+1, output high for complete period
```

The behavior of APWM active low mode (APWMPOL == 1) is as follows:

```
CMP = 0x00000000, output high for duration of period (0% duty)
CMP = 0x00000001, output low 1 cycle
CMP = 0x00000002, output low 2 cycles
CMP = PERIOD, output low except for 1 cycle (<100% duty)
CMP = PERIOD+1, output low for complete period (100% duty)
CMP > PERIOD+1, output low for complete period
```

Figure 5-11. Time-Base Frequency and Period Calculation



$$T_{PWM} = (CAP1 + 1) \times T_{TSCTR}$$

$$F_{PWM} = \frac{1}{T_{PWM}}$$

5.6 Application of the eCAP Module

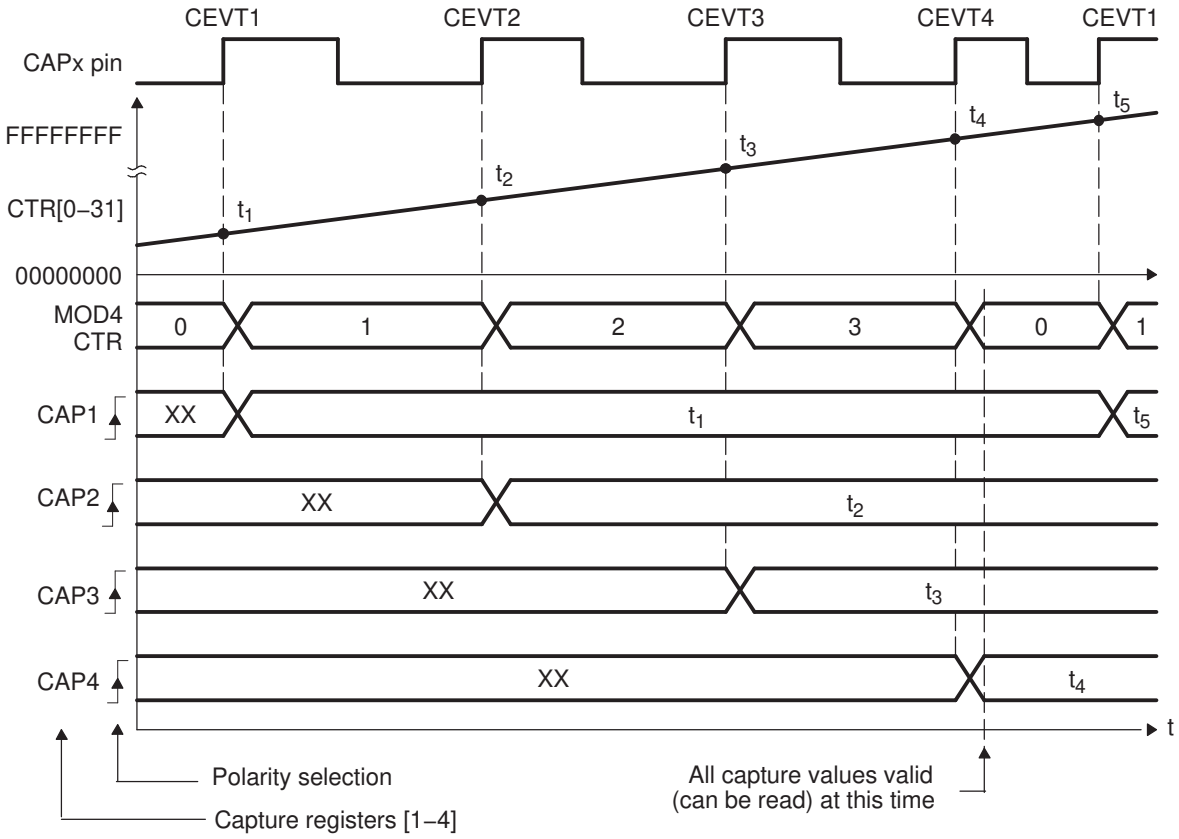
The following sections will provide applications examples to show how to operate the eCAP module.

5.6.1 Example 1 - Absolute Time-Stamp Operation Rising Edge Trigger

Figure 5-12 shows an example of continuous capture operation (Mod4 counter wraps around). In this figure, TSCTR counts-up without resetting and capture events are qualified on the rising edge only, this gives period (and frequency) information.

On an event, the TSCTR contents (time-stamp) is first captured, then Mod4 counter is incremented to the next state. When the TSCTR reaches FFFFFFFF (maximum value), it wraps around to 00000000 (not shown in Figure 5-12), if this occurs, the CTROVF (counter overflow) flag is set, and an interrupt (if enabled) occurs, CTROVF (counter overflow) Flag is set, and an Interrupt (if enabled) occurs. Captured Time-stamps are valid at the point indicated by the diagram (after the 4th event), hence event CEVT4 can conveniently be used to trigger an interrupt and the CPU can read data from the CAPx registers.

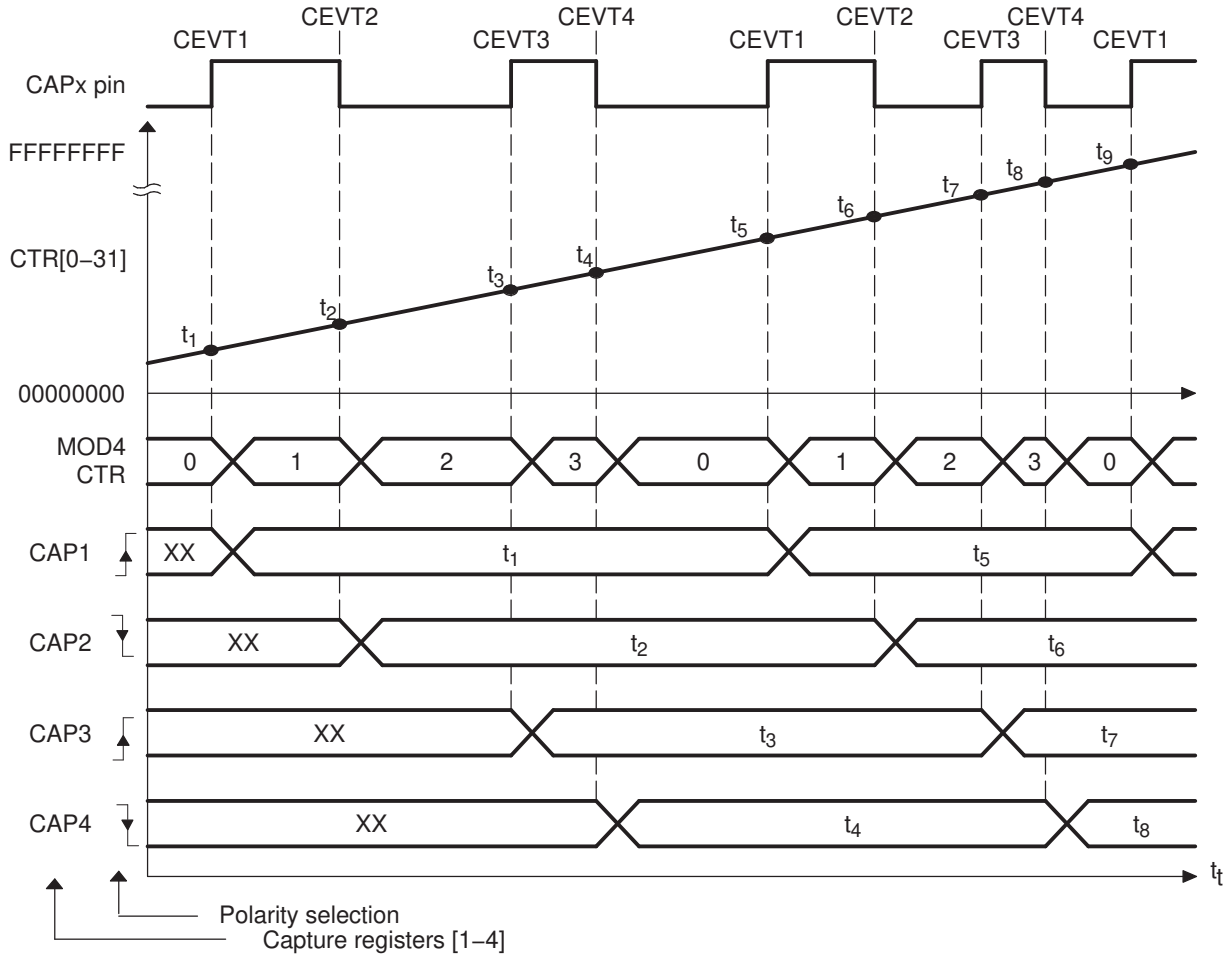
Figure 5-12. Capture Sequence for Absolute Time-stamp and Rising Edge Detect



5.6.2 Example 2 - Absolute Time-Stamp Operation Rising and Falling Edge Trigger

In Figure 5-13 the eCAP operating mode is almost the same as in the previous section except capture events are qualified as either rising or falling edge, this now gives both period and duty cycle information, that is: Period1 = $t_3 - t_1$, Period2 = $t_5 - t_3$, …and so on. Duty Cycle1 (on-time %) = ($t_2 - t_1$) / Period1 x 100%, etc. Duty Cycle1 (off-time %) = ($t_3 - t_2$) / Period1 x 100%, and so on.

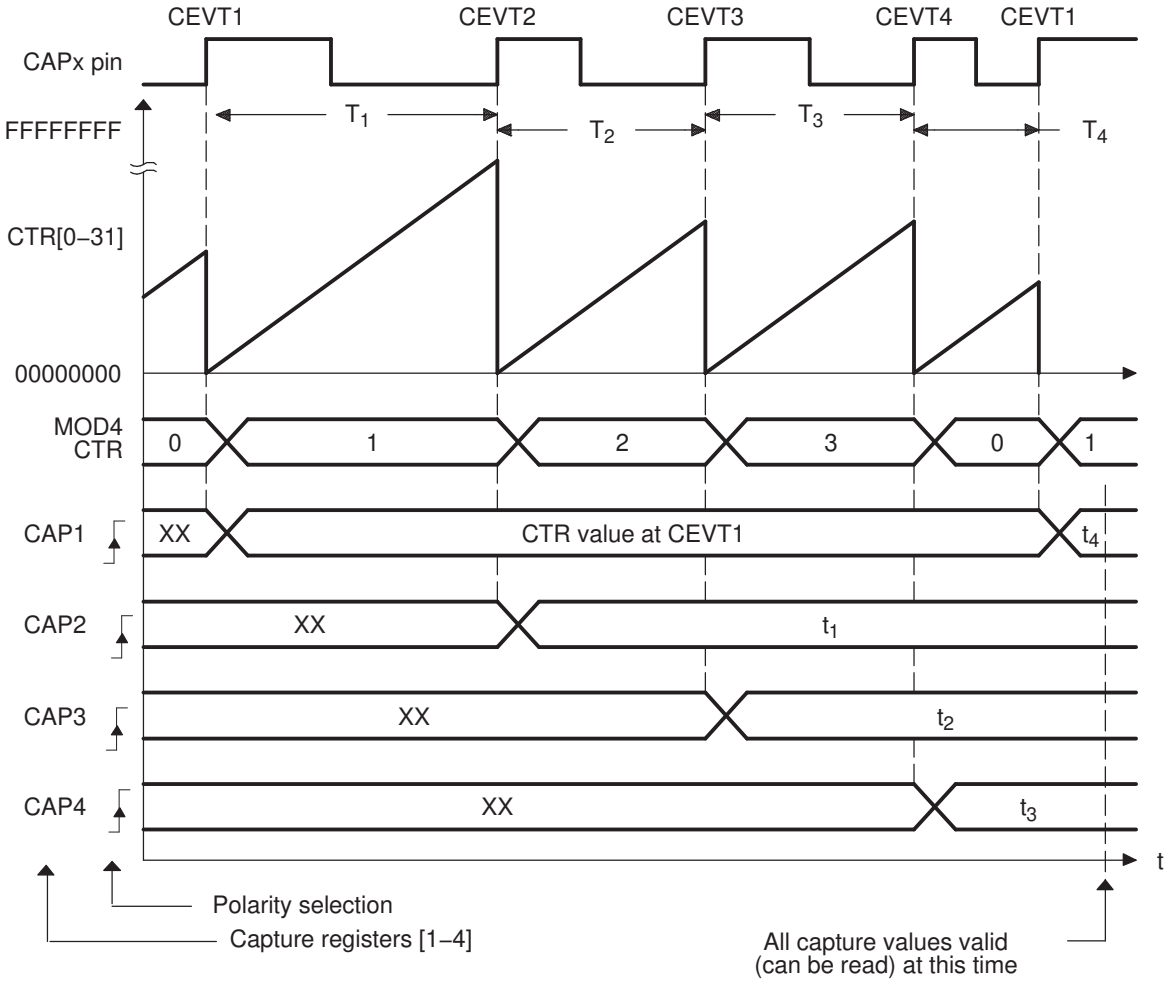
Figure 5-13. Capture Sequence for Absolute Time-stamp With Rising and Falling Edge Detect



5.6.3 Example 3 - Time Difference (Delta) Operation Rising Edge Trigger

This example Figure 5-14 shows how the eCAP module can be used to collect Delta timing data from pulse train waveforms. Here Continuous Capture mode (TSCTR counts-up without resetting, and Mod4 counter wraps around) is used. In Delta-time mode, TSCTR is Reset back to Zero on every valid event. Here Capture events are qualified as Rising edge only. On an event, TSCTR contents (Time-Stamp) is captured first, and then TSCTR is reset to Zero. The Mod4 counter then increments to the next state. If TSCTR reaches FFFFFFFF (Max value), before the next event, it wraps around to 00000000 and continues, a 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, that is, 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.

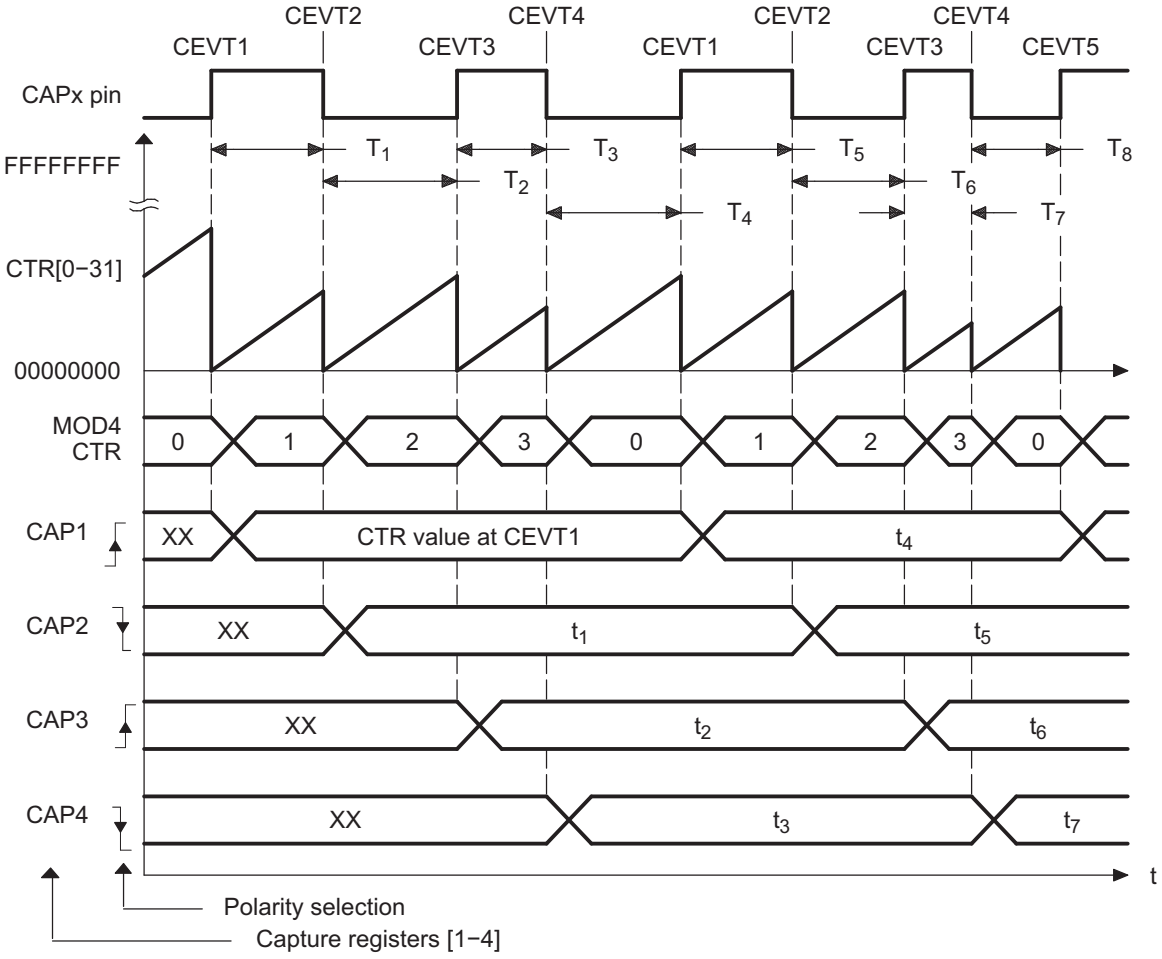
Figure 5-14. Capture Sequence for Delta Mode Time-stamp and Rising Edge Detect



5.6.4 Example 4 - Time Difference (Delta) Operation Rising and Falling Edge Trigger

In Figure 5-15 the eCAP operating mode is almost the same as in previous section except Capture events are qualified as either Rising or Falling edge, this now gives both Period and Duty cycle information, that is: Period1 = T_1+T_2, Period2 = T_3+T_4, …and so on, Duty Cycle1 (on-time %) = T_1 / Period1 x 100%, Duty Cycle1 (off-time %) = T_2 / Period1 x 100%, and so on.

Figure 5-15. Capture Sequence for Delta Mode Time-stamp With Rising and Falling Edge Detect



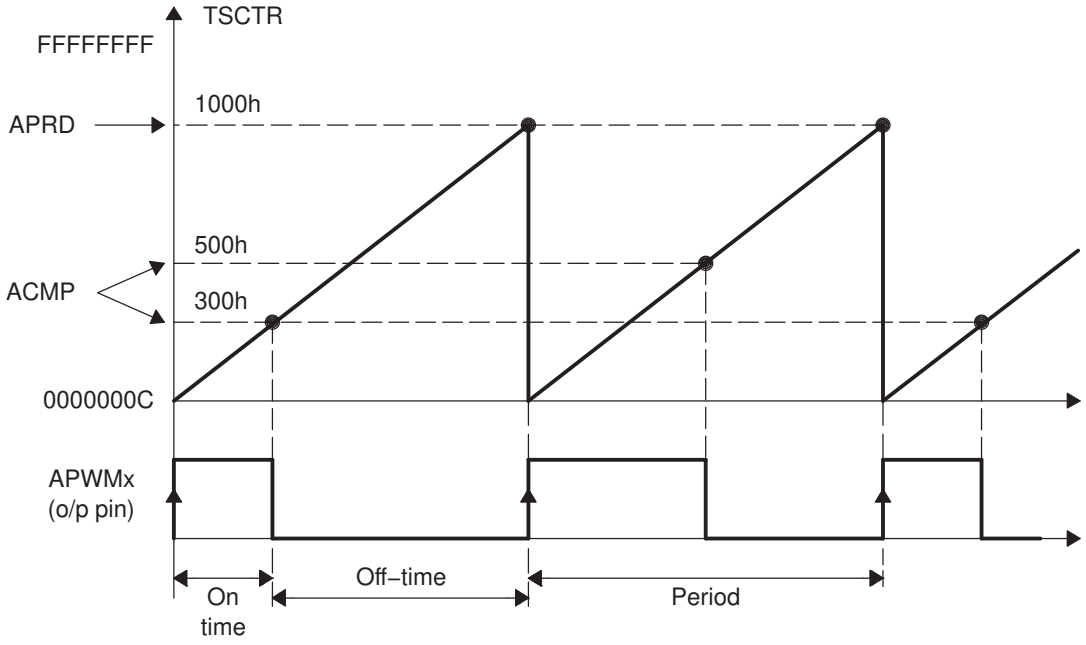
During initialization, you must write to the active registers for both period and compare. This action will automatically copy the init values into the shadow values. For subsequent compare updates during run-time, the shadow registers must be used.

5.7 Application of the APWM Mode

In this example, the eCAP module is configured to operate as a PWM generator. Here, a very simple single-channel PWM waveform is generated from the APWMx output pin. The PWM polarity is active high, which means that the compare value (CAP2 reg is now a compare register) represents the on-time (high level) of the period. Alternatively, if the APWMPOL bit is configured for active low, then the compare value represents the off-time.

5.7.1 Example 1 - Simple PWM Generation (Independent Channel/s)

Figure 5-16. PWM Waveform Details of APWM Mode Operation

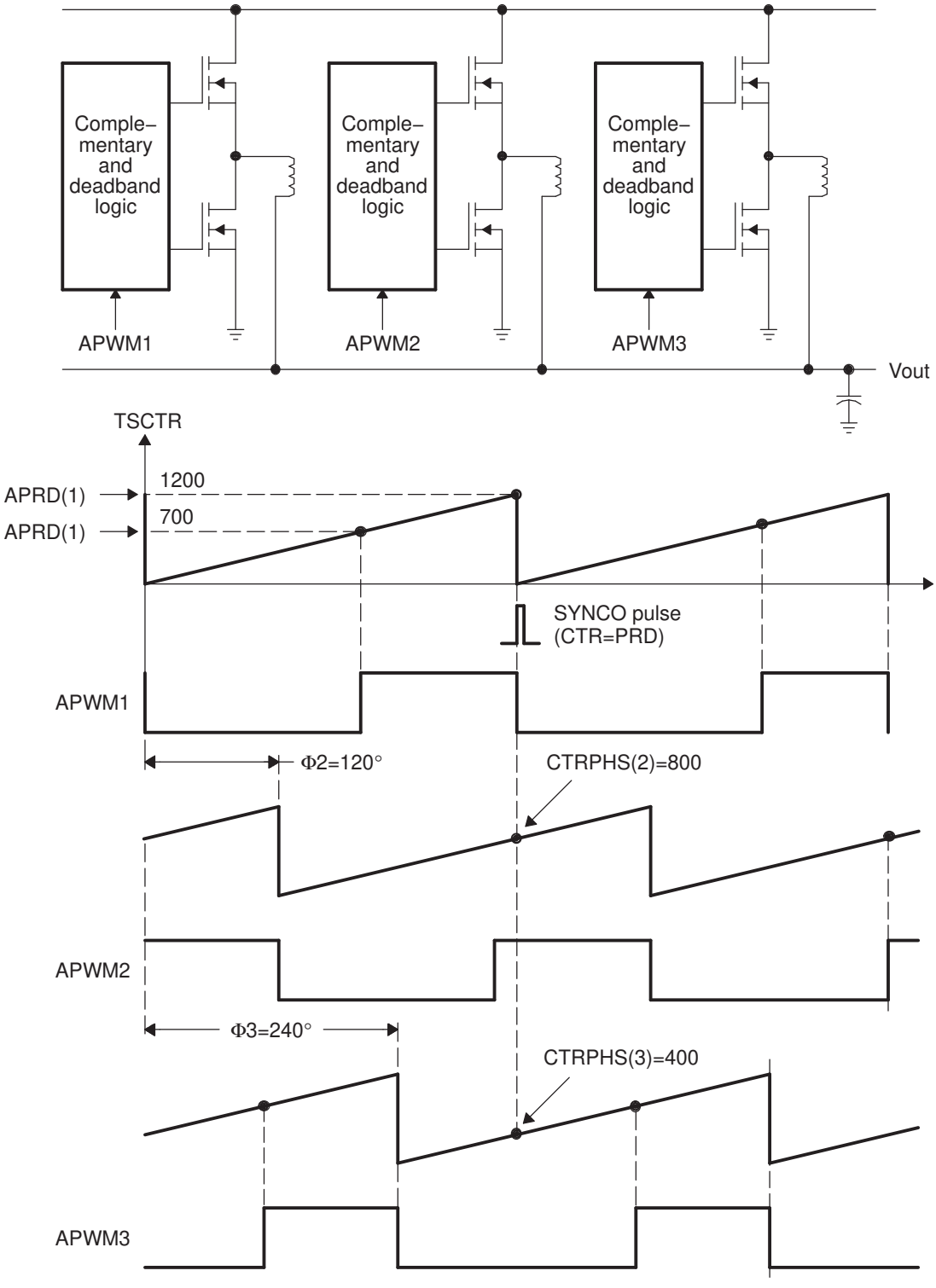


NOTE: Values are in hexadecimal (“h”) notation.

5.7.2 Example 2 - Multi-channel PWM Generation With Phase Control

In this example the Phase control feature of the APWM mode is used to control a 3 phase Interleaved DC/DC converter topology. This topology requires each phase to be off-set by 120° from each other. Hence if “Leg” 1 (controlled by APWM1) is the reference Leg (or phase), i.e. 0°, then Leg 2 need 120° off-set and Leg 3 needs 240° off-set. The waveforms in Figure 5-17 show the timing relationship between each of the phases (Legs). Note eCAP1 module is the Master and issues a sync out pulse to the slaves (modules 2, 3) whenever TSCTR = Period value.

Figure 5-17. Multi-phase (channel) Interleaved PWM Example Using 3 eCAP Modules



Example 5-1. Code Snippet for APWM Mode

```
// Code snippet for APWM mode Example 2

// Initialization Time
//======================
// ECAP module 1 config
    ECap1Regs.ECCTL2.bit.CAP_APWM = EC_APWM_MODE;
    ECap1Regs.CAP1 = 1200; // Set period value
    ECap1Regs.CTRPHS = 0; // make eCAP1 reference phase = zero
    ECap1Regs.ECCTL2.bit.APWMPOL = EC_ACTV_HI;
    ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE; // No sync in for Master
    ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_CTR_PRD; // eCAP1 is Master
    ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN; // Allow TSCTR to run

// ECAP module 2 config
    ECap2Regs.CAP1 = 1200; // Set period value
    ECap2Regs.CTRPHS = 800; // Phase offset = 1200-400 = 120 deg
    ECap2Regs.ECCTL2.bit.CAP_APWM = EC_APWM_MODE;
    ECap2Regs.ECCTL2.bit.APWMPOL = EC_ACTV_HI;
    ECap2Regs.ECCTL2.bit.SYNCI_EN = EC_ENABLE; // slaved off master
    ECap2Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCI; // sync "flow-through"
    ECap2Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN; // Allow TSCTR to run

// ECAP module 3 config
    ECap3Regs.CAP1 = 1200; // Set period value
    ECap3Regs.CTRPHS = 400; // Phase offset = 1200-800 = 240 deg
    ECap3Regs.ECCTL2.bit.CAP_APWM = EC_APWM_MODE;
    ECap3Regs.ECCTL2.bit.APWMPOL = EC_ACTV_HI;
    ECap3Regs.ECCTL2.bit.SYNCI_EN = EC_ENABLE; // slaved off master
    ECap3Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS; // "break the chain"
    ECap3Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN; // Allow TSCTR to run

// Run Time (Note: Example execution of one run-time instant)
//===========================================================
// All phases are set to the same duty cycle
    ECap1Regs.CAP2 = 700; // Set Duty cycle i.e. compare value = 700
    ECap2Regs.CAP2 = 700; // Set Duty cycle i.e. compare value = 700
    ECap3Regs.CAP2 = 700; // Set Duty cycle i.e. compare value = 700
```

5.8 eCAP Registers

This section describes the Enhanced Capture Registers.

5.8.1 eCAP Base Addresses

Table 5-1. ECAP Base Address Table

Bit Field Name		Base Address
Instance	Structure	
ECap1Regs	ECAP_REGS	0x0000_6A00
ECap2Regs	ECAP_REGS	0x0000_6A20
ECap3Regs	ECAP_REGS	0x0000_6A40
ECap4Regs	ECAP_REGS	0x0000_6A60
ECap5Regs	ECAP_REGS	0x0000_6A80
ECap6Regs	ECAP_REGS	0x0000_6AA0

5.8.2 ECAP_REGS Registers

Table 5-2 lists the ECAP_REGS registers. All register offset addresses not listed in Table 5-2 should be considered as reserved locations and the register contents should not be modified.

Table 5-2. ECAP_REGS Registers

Offset	Acronym	Register Name	Write Protection	Section
0h	TSCTR	Time-Stamp Counter		Go
2h	CTRPHS	Counter Phase Offset Value Register		Go
4h	CAP1	Capture 1 Register		Go
6h	CAP2	Capture 2 Register		Go
8h	CAP3	Capture 3 Register		Go
Ah	CAP4	Capture 4 Register		Go
14h	ECCTL1	Capture Control Register 1		Go
15h	ECCTL2	Capture Control Register 2		Go
16h	ECEINT	Capture Interrupt Enable Register		Go
17h	ECFLG	Capture Interrupt Flag Register		Go
18h	ECCLR	Capture Interrupt Clear Register		Go
19h	ECFRC	Capture Interrupt Force Register		Go

Complex bit access types are encoded to fit into small table cells. Table 5-3 shows the codes that are used for access types in this section.

Table 5-3. ECAP_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R -0	Read Returns 0s
Write Type		
W	W	Write
W1C	W 1C	Write 1 to clear
W1S	W 1S	Write 1 to set
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

5.8.2.1 TSCTR Register (Offset = 0h) [reset = 0h]

TSCTR is shown in Figure 5-18 and described in Table 5-4.

Return to the Summary Table.

Time-Stamp Counter

Figure 5-18. TSCTR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSCTR																															
R/W-0h																															

Table 5-4. TSCTR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	TSCTR	R/W	0h	Active 32-bit counter register that is used as the capture time-base Reset type: SYSRSn

5.8.2.2 CTRPHS Register (Offset = 2h) [reset = 0h]

CTRPHS is shown in Figure 5-19 and described in Table 5-5.

Return to the Summary Table.

Counter Phase Offset Value Register

Figure 5-19. CTRPHS Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTRPHS																															
R/W-0h																															

Table 5-5. CTRPHS Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	CTRPHS	R/W	0h	Counter phase value register that can be programmed for phase lag/lead. This register CTRPHS is loaded into TSCTR upon either a SYNCI event or S/W force via a control bit. Used to achieve phase control synchronization with respect to other eCAP and EPWM timebases. Reset type: SYSRSn

5.8.2.3 CAP1 Register (Offset = 4h) [reset = 0h]

CAP1 is shown in Figure 5-20 and described in Table 5-6.

Return to the Summary Table.

Capture 1 Register

Figure 5-20. CAP1 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP1																															
R/W-0h																															

Table 5-6. CAP1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	CAP1	R/W	0h	This register can be loaded (written) by: - Time-Stamp counter value (TSCTR) during a capture event - Software - may be useful for test purposes or initialization - ARPD shadow register (CAP3) when used in APWM mode Reset type: SYSRSn

5.8.2.4 CAP2 Register (Offset = 6h) [reset = 0h]

CAP2 is shown in Figure 5-21 and described in Table 5-7.

Return to the Summary Table.

Capture 2 Register

Figure 5-21. CAP2 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP2																															
R/W-0h																															

Table 5-7. CAP2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	CAP2	R/W	0h	This register can be loaded (written) by: - Time-Stamp (counter value) during a capture event - Software - may be useful for test purposes - ACMP shadow register (CAP4) when used in APWM mode Reset type: SYSRSn

5.8.2.5 CAP3 Register (Offset = 8h) [reset = 0h]

CAP3 is shown in Figure 5-22 and described in Table 5-8.

Return to the Summary Table.

Capture 3 Register

Figure 5-22. CAP3 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP3																															
R/W-0h																															

Table 5-8. CAP3 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	CAP3	R/W	0h	In CMP mode, this is a time-stamp capture register. In APWM mode, this is the period shadow (APRD) register. You can update the PWM period value through this register. CAP3 (APRD) shadows CAP1 in this mode. Reset type: SYSRSn

5.8.2.6 CAP4 Register (Offset = Ah) [reset = 0h]

CAP4 is shown in Figure 5-23 and described in Table 5-9.

Return to the Summary Table.

Capture 4 Register

Figure 5-23. CAP4 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP4																															
R/W-0h																															

Table 5-9. CAP4 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	CAP4	R/W	0h	In CMP mode, this is a time-stamp capture register. In APWM mode, this is the compare shadow (ACMP) register. You can update the PWM compare value via this register. CAP4 (ACMP) shadows CAP2 in this mode. Reset type: SYSRSn

5.8.2.7 ECCTL1 Register (Offset = 14h) [reset = 0h]

ECCTL1 is shown in Figure 5-24 and described in Table 5-10.

Return to the Summary Table.

Capture Control Register 1

Figure 5-24. ECCTL1 Register

15	14	13	12	11	10	9	8
FREE_SOFT		PRESCALE					CAPLDEN
R/W-0h		R/W-0h					R/W-0h
7	6	5	4	3	2	1	0
CTRRST4	CAP4POL	CTRRST3	CAP3POL	CTRRST2	CAP2POL	CTRRST1	CAP1POL
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 5-10. ECCTL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	FREE_SOFT	R/W	0h	Emulation Control Reset type: SYSRSn 0h (R/W) = TSCTR counter stops immediately on emulation suspend 1h (R/W) = TSCTR counter runs until = 0 2h (R/W) = TSCTR counter is unaffected by emulation suspend (Run Free) 3h (R/W) = TSCTR counter is unaffected by emulation suspend (Run Free)
13-9	PRESCALE	R/W	0h	Event Filter prescale select Reset type: SYSRSn 0h (R/W) = Divide by 1 (i.e,. no prescale, by-pass the prescaler) 1h (R/W) = Divide by 2 2h (R/W) = Divide by 4 3h (R/W) = Divide by 6 4h (R/W) = Divide by 8 5h (R/W) = Divide by 10 1Eh (R/W) = Divide by 60 1Fh (R/W) = Divide by 62
8	CAPLDEN	R/W	0h	Enable Loading of CAP1-4 registers on a capture event. Note that this bit does not disable CEVTn events from being generated. Reset type: SYSRSn 0h (R/W) = Disable CAP1-4 register loads at capture event time. 1h (R/W) = Enable CAP1-4 register loads at capture event time.
7	CTRRST4	R/W	0h	Counter Reset on Capture Event 4 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 4 (absolute time stamp operation) 1h (R/W) = Reset counter after Capture Event 4 time-stamp has been captured (used in difference mode operation)
6	CAP4POL	R/W	0h	Capture Event 4 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 4 triggered on a rising edge (RE) 1h (R/W) = Capture Event 4 triggered on a falling edge (FE)

Table 5-10. ECCTL1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
5	CTRRST3	R/W	0h	Counter Reset on Capture Event 3 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 3 (absolute time stamp) 1h (R/W) = Reset counter after Event 3 time-stamp has been captured (used in difference mode operation)
4	CAP3POL	R/W	0h	Capture Event 3 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 3 triggered on a rising edge (RE) 1h (R/W) = Capture Event 3 triggered on a falling edge (FE)
3	CTRRST2	R/W	0h	Counter Reset on Capture Event 2 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 2 (absolute time stamp) 1h (R/W) = Reset counter after Event 2 time-stamp has been captured (used in difference mode operation)
2	CAP2POL	R/W	0h	Capture Event 2 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 2 triggered on a rising edge (RE) 1h (R/W) = Capture Event 2 triggered on a falling edge (FE)
1	CTRRST1	R/W	0h	Counter Reset on Capture Event 1 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 1 (absolute time stamp) 1h (R/W) = Reset counter after Event 1 time-stamp has been captured (used in difference mode operation)
0	CAP1POL	R/W	0h	Capture Event 1 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 1 triggered on a rising edge (RE) 1h (R/W) = Capture Event 1 triggered on a falling edge (FE)

5.8.2.8 ECCTL2 Register (Offset = 15h) [reset = 6h]

ECCTL2 is shown in Figure 5-25 and described in Table 5-11.

Return to the Summary Table.

Capture Control Register 2

Figure 5-25. ECCTL2 Register

15	14	13	12	11	10	9	8
RESERVED					APWMPOL	CAP_APWM	SWSYNC
R-0h					R/W-0h	R/W-0h	R-0/W1S-0h
7	6	5	4	3	2	1	0
SYNCO_SEL		SYNCI_EN	TSCTRSTOP	REARM	STOP_WRAP		CONT_ONESHT
R/W-0h		R/W-0h	R/W-0h	R-0/W1S-0h	R/W-3h		R/W-0h

Table 5-11. ECCTL2 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-11	RESERVED	R	0h	Reserved
10	APWMPOL	R/W	0h	APWM output polarity select. This is applicable only in APWM operating mode. Reset type: SYSRSn 0h (R/W) = Output is active high (Compare value defines high time) 1h (R/W) = Output is active low (Compare value defines low time)
9	CAP_APWM	R/W	0h	CAP/APWM operating mode select Reset type: SYSRSn 0h (R/W) = ECAP module operates in capture mode. This mode forces the following configuration: - Inhibits TSCTR resets via CTR = PRD event - Inhibits shadow loads on CAP1 and 2 registers - Permits user to enable CAP1-4 register load - CAPx/APWMx pin operates as a capture input 1h (R/W) = ECAP module operates in APWM mode. This mode forces the following configuration: - Resets TSCTR on CTR = PRD event (period boundary - Permits shadow loading on CAP1 and 2 registers - Disables loading of time-stamps into CAP1-4 registers - CAPx/APWMx pin operates as a APWM output
8	SWSYNC	R-0/W1S	0h	Software-forced Counter (TSCTR) Synchronizer. This provides the user a method to generate a synchronization pulse through software. In APWM mode, the synchronization pulse can also be sourced from the CTR = PRD event. Reset type: SYSRSn 0h (R/W) = Writing a zero has no effect. Reading always returns a zero 1h (R/W) = Writing a one forces a TSCTR shadow load of current ECAP module and any ECAP modules down-stream providing the SYNCO_SEL bits are 0,0. After writing a 1, this bit returns to a zero. Note: Selection CTR = PRD is meaningful only in APWM mode however, you can choose it in CAP mode if you find doing so useful.
7-6	SYNCO_SEL	R/W	0h	Sync-Out Select Reset type: SYSRSn 0h (R/W) = Select sync-in event to be the sync-out signal (pass through) 1h (R/W) = Select CTR = PRD event to be the sync-out signal 2h (R/W) = Disable sync out signal 3h (R/W) = Disable sync out signal

Table 5-11. ECCTL2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
5	SYNCI_EN	R/W	0h	Counter (TSCTR) Sync-In select mode Reset type: SYSRSn 0h (R/W) = Disable sync-in option 1h (R/W) = Enable counter (TSCTR) to be loaded from CTRPHS register upon either a SYNCI signal or a S/W force event.
4	TSCTRSTOP	R/W	0h	Time Stamp (TSCTR) Counter Stop (freeze) Control Reset type: SYSRSn 0h (R/W) = TSCTR stopped 1h (R/W) = TSCTR free-running
3	REARM	R-0/W1S	0h	Re-Arming Control. Note: The re-arm function is valid in one shot or continuous mode. Reset type: SYSRSn 0h (R/W) = Has no effect (reading always returns a 0) 1h (R/W) = Arms the one-shot sequence as follows: 1) Resets the Mod4 counter to zero 2) Unfreezes the Mod4 counter 3) Enables capture register loads
2-1	STOP_WRAP	R/W	3h	Stop value for one-shot mode. This is the number (between 1-4) of captures allowed to occur before the CAP(1-4) registers are frozen, that is, capture sequence is stopped. Wrap value for continuous mode. This is the number (between 1-4) of the capture register in which the circular buffer wraps around and starts again. Notes: STOP_WRAP is compared to Mod4 counter and, when equal, 2 actions occur: - Mod4 counter is stopped (frozen) - Capture register loads are inhibited In one-shot mode, further interrupt events are blocked until re-armed. Reset type: SYSRSn 0h (R/W) = Stop after Capture Event 1 in one-shot mode Wrap after Capture Event 1 in continuous mode. 1h (R/W) = Stop after Capture Event 2 in one-shot mode Wrap after Capture Event 2 in continuous mode. 2h (R/W) = Stop after Capture Event 3 in one-shot mode Wrap after Capture Event 3 in continuous mode. 3h (R/W) = Stop after Capture Event 4 in one-shot mode Wrap after Capture Event 4 in continuous mode.
0	CONT_ONESHT	R/W	0h	Continuous or one-shot mode control (applicable only in capture mode) Reset type: SYSRSn 0h (R/W) = Operate in continuous mode 1h (R/W) = Operate in one-Shot mode

5.8.2.9 ECEINT Register (Offset = 16h) [reset = 0h]

ECEINT is shown in Figure 5-26 and described in Table 5-12.

Return to the Summary Table.

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 5-26. ECEINT Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_EQ_CMP	CTR_EQ_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	RESERVED
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h

Table 5-12. ECEINT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_EQ_CMP	R/W	0h	Counter Equal Compare Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Compare Equal as an Interrupt source 1h (R/W) = Enable Compare Equal as an Interrupt source
6	CTR_EQ_PRD	R/W	0h	Counter Equal Period Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Period Equal as an Interrupt source 1h (R/W) = Enable Period Equal as an Interrupt source
5	CTROVF	R/W	0h	Counter Overflow Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disabled counter Overflow as an Interrupt source 1h (R/W) = Enable counter Overflow as an Interrupt source
4	CEVT4	R/W	0h	Capture Event 4 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 4 as an Interrupt source 1h (R/W) = Capture Event 4 Interrupt Enable
3	CEVT3	R/W	0h	Capture Event 3 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 3 as an Interrupt source 1h (R/W) = Enable Capture Event 3 as an Interrupt source
2	CEVT2	R/W	0h	Capture Event 2 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 2 as an Interrupt source 1h (R/W) = Enable Capture Event 2 as an Interrupt source

Table 5-12. ECEINT Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
1	CEVT1	R/W	0h	Capture Event 1 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 1 as an Interrupt source 1h (R/W) = Enable Capture Event 1 as an Interrupt source
0	RESERVED	R	0h	Reserved

5.8.2.10 ECFLG Register (Offset = 17h) [reset = 0h]

ECFLG is shown in Figure 5-27 and described in Table 5-13.

Return to the Summary Table.

Capture Interrupt Flag Register

Figure 5-27. ECFLG Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	INT
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

Table 5-13. ECFLG Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_CMP	R	0h	Compare Equal Compare Status Flag. This flag is active only in APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) reached the compare register value (ACMP)
6	CTR_PRD	R	0h	Counter Equal Period Status Flag. This flag is only active in APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) reached the period register value (APRD) and was reset.
5	CTROVF	R	0h	Counter Overflow Status Flag. This flag is active in CAP and APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) has made the transition from FFFFFFFF " 00000000
4	CEVT4	R	0h	Capture Event 4 Status Flag This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the fourth event occurred at ECAPx pin
3	CEVT3	R	0h	Capture Event 3 Status Flag. This flag is active only in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the third event occurred at ECAPx pin.
2	CEVT2	R	0h	Capture Event 2 Status Flag. This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the second event occurred at ECAPx pin.
1	CEVT1	R	0h	Capture Event 1 Status Flag. This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the first event occurred at ECAPx pin.

Table 5-13. ECFLG Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
0	INT	R	0h	Global Interrupt Status Flag Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates that an interrupt was generated.

5.8.2.11 ECCLR Register (Offset = 18h) [reset = 0h]

ECCLR is shown in Figure 5-28 and described in Table 5-14.

Return to the Summary Table.

Capture Interrupt Clear Register

Figure 5-28. ECCLR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	INT
R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h

Table 5-14. ECCLR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_CMP	R-0/W1C	0h	Counter Equal Compare Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CTR=CMP flag.
6	CTR_PRD	R-0/W1C	0h	Counter Equal Period Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CTR=PRD flag.
5	CTROVF	R-0/W1C	0h	Counter Overflow Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CTROVF flag.
4	CEVT4	R-0/W1C	0h	Capture Event 4 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT4 flag.
3	CEVT3	R-0/W1C	0h	Capture Event 3 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT3 flag.
2	CEVT2	R-0/W1C	0h	Capture Event 2 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT2 flag.
1	CEVT1	R-0/W1C	0h	Capture Event 1 Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the CEVT1 flag.
0	INT	R-0/W1C	0h	ECAP Global Interrupt Status Clear Reset type: SYSRSn 0h (R/W) = Writing a 0 has no effect. Always reads back a 0 1h (R/W) = Writing a 1 clears the INT flag and enable further interrupts to be generated if any of the event flags are set to 1

5.8.2.12 ECFRC Register (Offset = 19h) [reset = 0h]

ECFRC is shown in Figure 5-29 and described in Table 5-15.

Return to the Summary Table.

Capture Interrupt Force Register

Figure 5-29. ECFRC Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	RESERVED
R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0h

Table 5-15. ECFRC Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_CMP	R-0/W1S	0h	Force Counter Equal Compare Interrupt. This event is only active in APWM mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CTR=CMP flag.
6	CTR_PRD	R-0/W1S	0h	Force Counter Equal Period Interrupt. This event is only active in APWM mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CTR=PRD flag.
5	CTROVF	R-0/W1S	0h	Force Counter Overflow. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 to this bit sets the CTROVF flag.
4	CEVT4	R-0/W1S	0h	Force Capture Event 4. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CEVT4 flag.
3	CEVT3	R-0/W1S	0h	Force Capture Event 3. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CEVT3 flag.
2	CEVT2	R-0/W1S	0h	Force Capture Event 2. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Writing a 1 sets the CEVT2 flag.
1	CEVT1	R-0/W1S	0h	Force Capture Event 1. This event is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = No effect. Always reads back a 0. 1h (R/W) = Sets the CEVT1 flag.
0	RESERVED	R	0h	Reserved

Enhanced Quadrature Encoder Pulse (eQEP)

The enhanced Quadrature Encoder Pulse (eQEP) module described here is a Type-0 eQEP. See the *C2000 Real-Time Control Peripheral Reference Guide* 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.

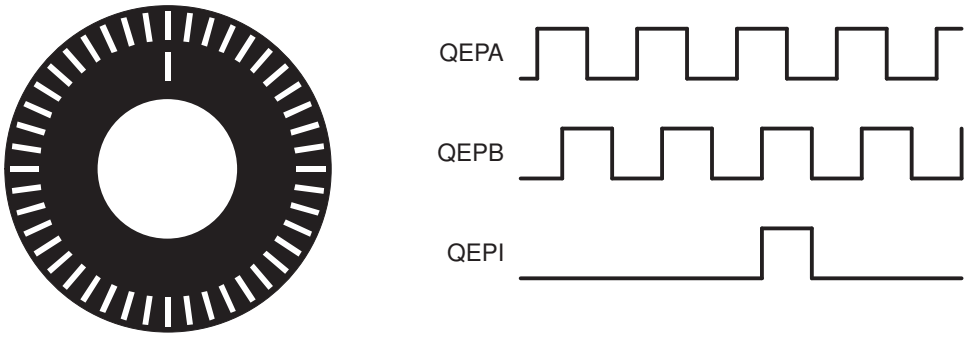
The enhanced quadrature encoder pulse (eQEP) module is used for direct interface with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine for use in a high-performance motion and position-control system.

Topic **Page**

6.1 Introduction

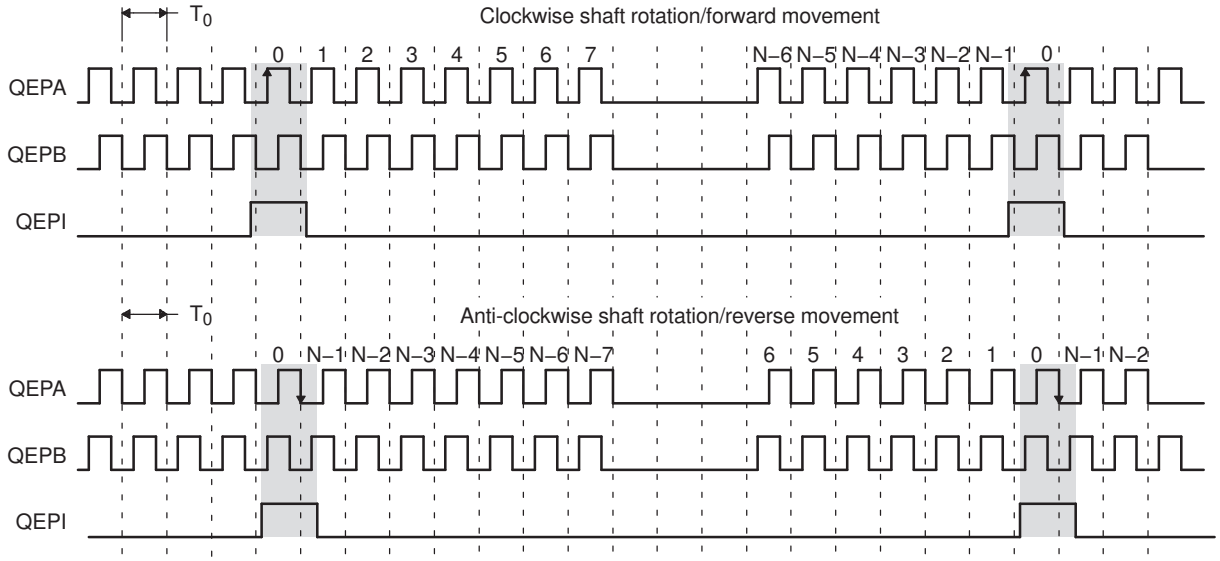
An incremental encoder disk is patterned with a track of slots along its periphery, as shown in Figure 6-1. These slots create an alternating pattern of dark and light lines. The disk count is defined as the number of dark and light line pairs that occur per revolution (lines per revolution). As a rule, 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 6-1. 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 detected 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 6-2.

Figure 6-2. QEP Encoder Output Signal for Forward/Reverse Movement

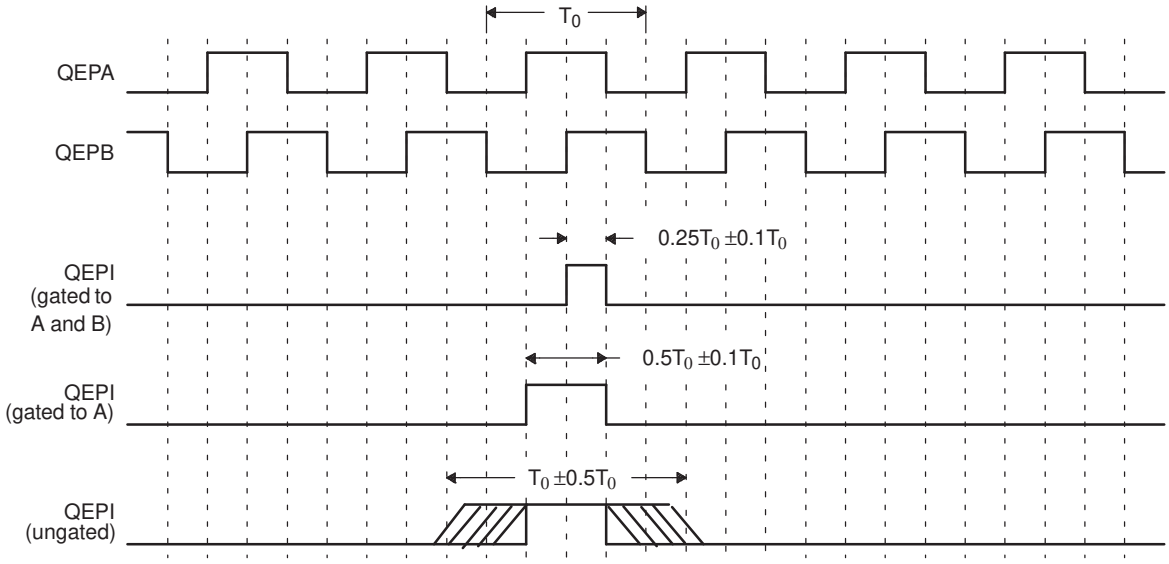


Legend: N = lines per revolution

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 6-3. 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 6-3. Index Pulse Example



Some typical applications of shaft encoders include robotics and 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:

$$v(k) \approx \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \quad (1)$$

$$v(k) \approx \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \quad (2)$$

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 a 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, for example 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.

6.2 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins.

For proper operation of the eQEP module, input GPIO pins must be configured via the GPxQSELn registers for synchronous input mode (with or without qualification). The asynchronous mode should not be used for eQEP input pins. The internal pullups can be configured in the GPyPUD register.

See the *GPIO* chapter for more details on GPIO mux and settings.

6.3 Description

This section provides the eQEP inputs, memory map, and functional description.

6.3.1 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* chapter 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. This 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.

- *QEPI: 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*

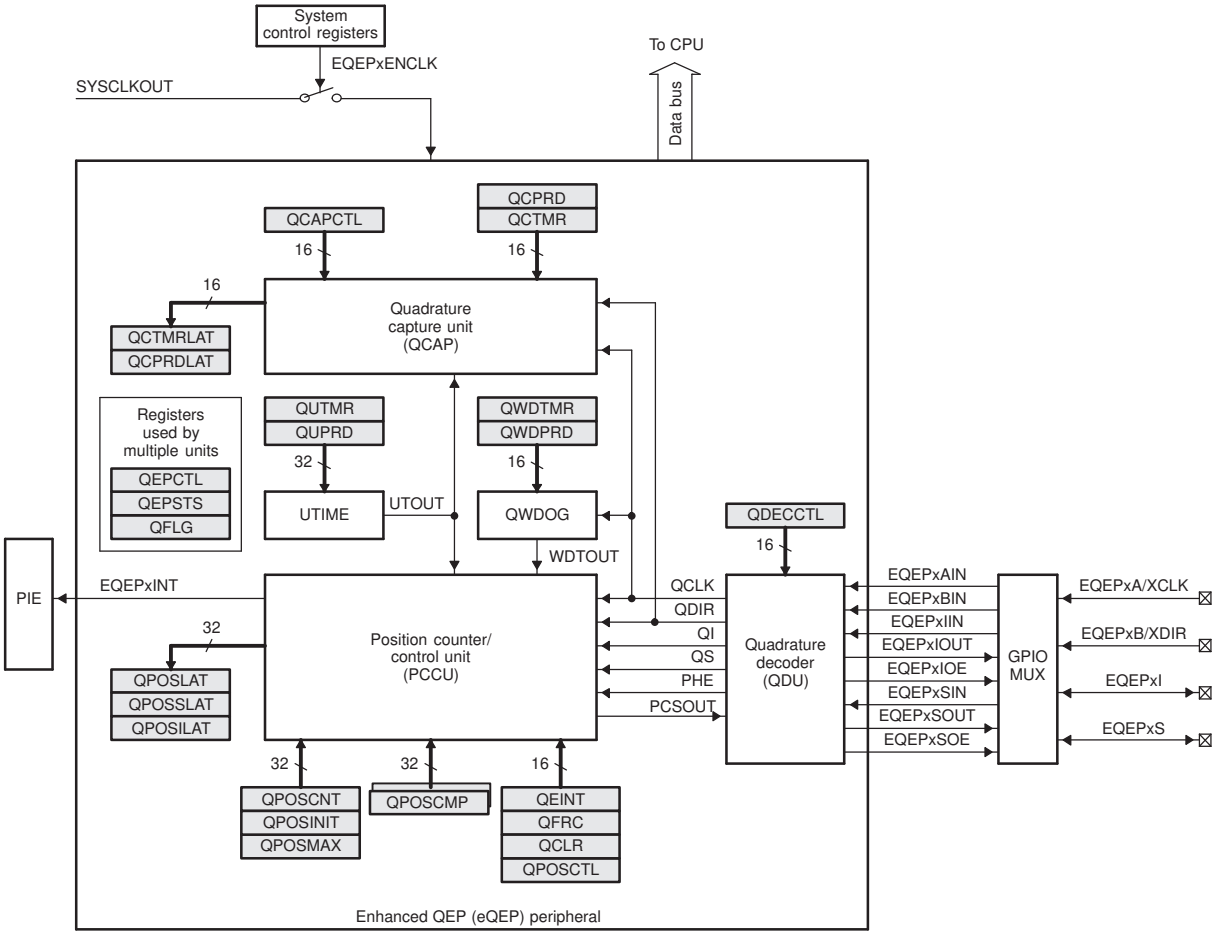
  This general-purpose strobe signal can initialize or latch the position counter on the occurrence of a desired event on the strobe pin. This signal is typically connected to a sensor or limit switch to notify that the motor has reached a defined position.

6.3.2 Functional Description

The eQEP peripheral contains the following major functional units (as shown in Figure 6-4):

- 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 6-4. Functional Block Diagram of the eQEP Peripheral



6.3.3 eQEP Memory Map

Table 6-1 lists the registers with their memory locations, sizes, and reset values.

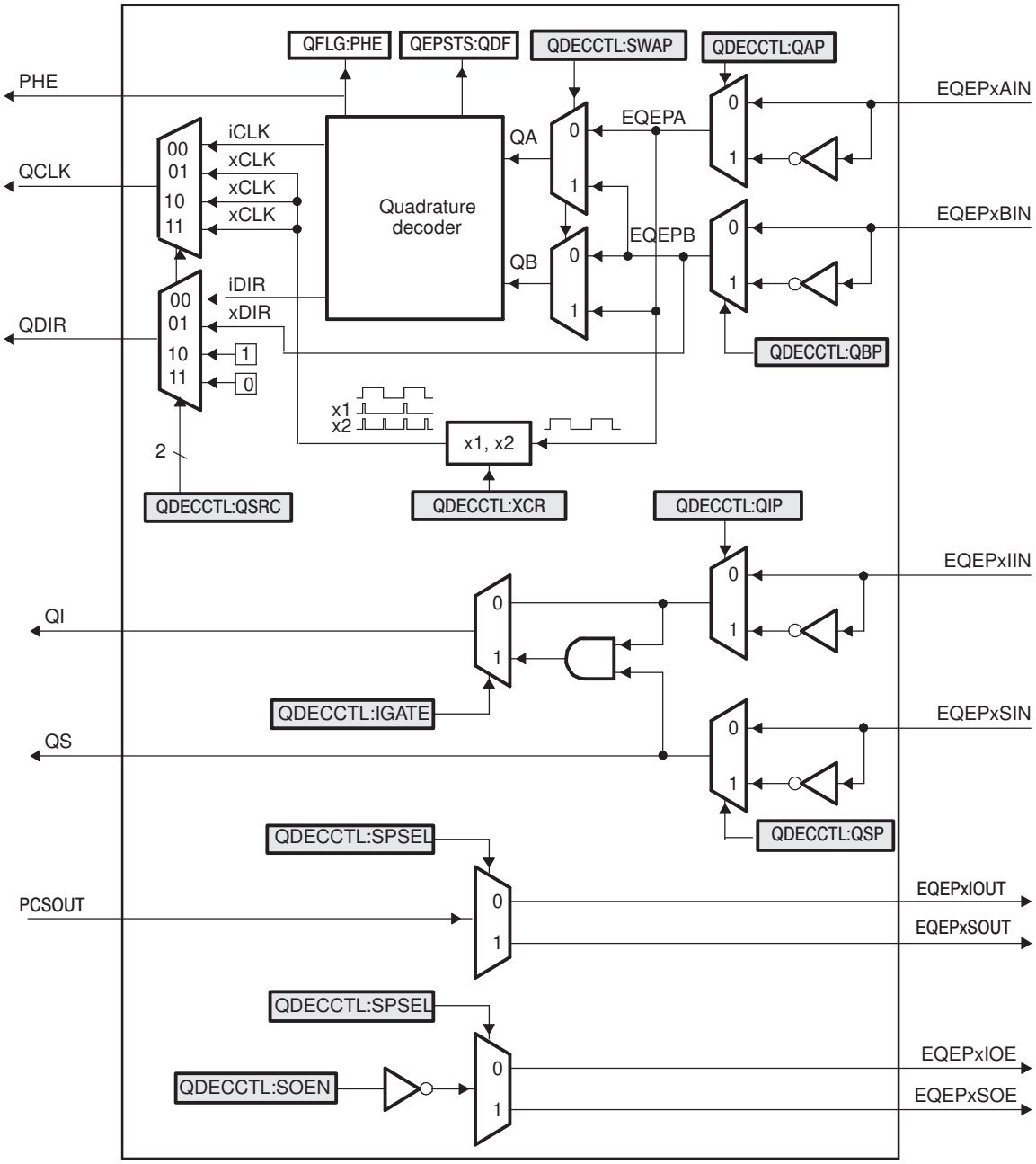
Table 6-1. EQEP Memory Map

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
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		

6.4 Quadrature Decoder Unit (QDU)

Figure 6-5 shows a functional block diagram of the QDU.

Figure 6-5. Functional Block Diagram of Decoder Unit



6.4.1 Position Counter Input Modes

Clock and direction input to the 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

6.4.1.1 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 the QEPSTS[QDF] bit. Table 6-2 and Figure 6-6 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 6-7 shows the direction decoding and clock generation from the eQEP input signals.

Table 6-2. 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 6-6. Quadrature Decoder State Machine

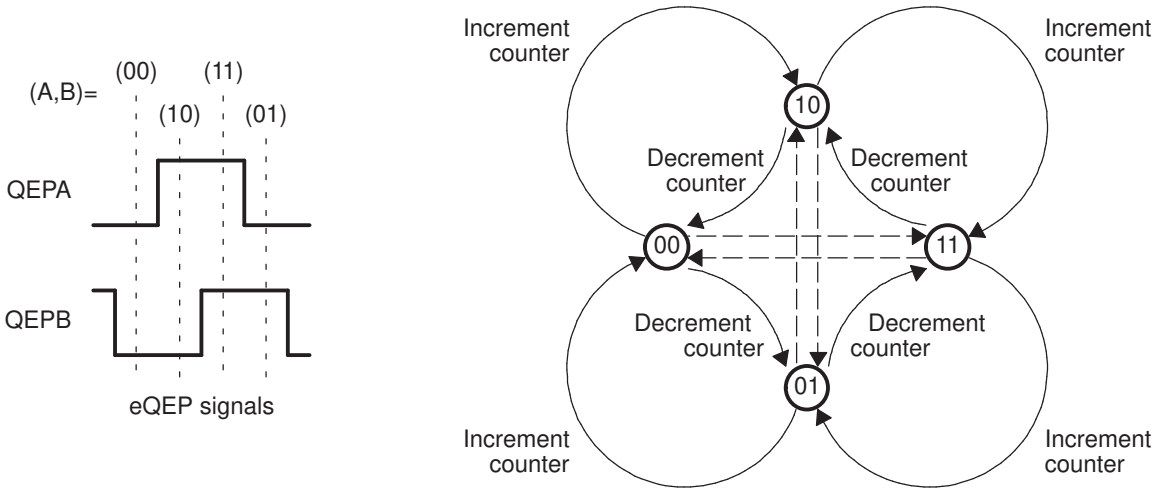
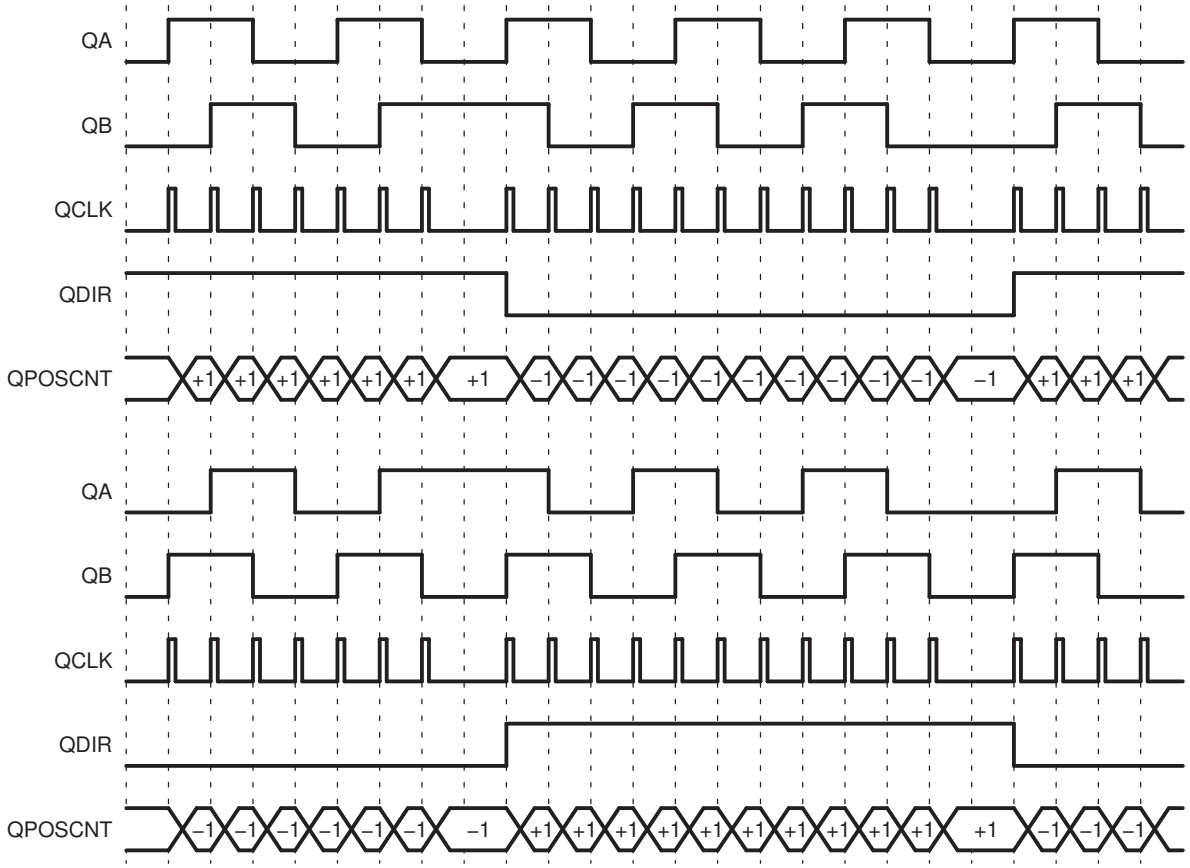


Figure 6-7. Quadrature-clock and Direction Decoding



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 and the QPOSCNT value can be incorrect and offset by multiples of 1 or 3. That is, when edge transition is detected simultaneously on the QEPA and QEPB signals to optionally generate interrupts. State transitions marked by dashed lines in Figure 6-6 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 6-7 .

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.

6.4.1.2 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 the 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.

6.4.1.3 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 the QDECCTL[XCR] bit enables clock generation to the position counter on both edges of the QEPA input, thereby increasing the measurement resolution by a factor of 2x. In up-count mode, it is recommended that the application not configure QEPB as a GPIO mux option, or ensure that a signal edge is not generated on the QEPB input.

6.4.1.4 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 the QDECCTL[XCR] bit enables clock generation to the position counter on both edges of a QEPA input, thereby increasing the measurement resolution by a factor of 2x. In down-count mode, it is recommended that the application not configure QEPB as a GPIO mux option, or ensure that a signal edge is not generated on the QEPB input.

6.4.2 eQEP Input Polarity Selection

Each eQEP input can be inverted using QDECCTL[8:5] control bits. As an example, setting the QDECCTL[QIP] bit will invert the index input.

6.4.3 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.

6.5 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.

6.5.1 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 the 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 the position counter provides a rotor angle with respect to the index pulse position.

The 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, the position counter is reset to 0 on overflow and to the QPOSMAX register value on underflow. Overflow occurs when the position counter counts up after the QPOSMAX value. Underflow occurs when the position counter counts down after "0". The Interrupt flag is set to indicate overflow/underflow in QFLG register.

6.5.1.1 Position Counter Reset on Index Event (QEPCTL[PCRM]=00)

If the index event occurs during the 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.

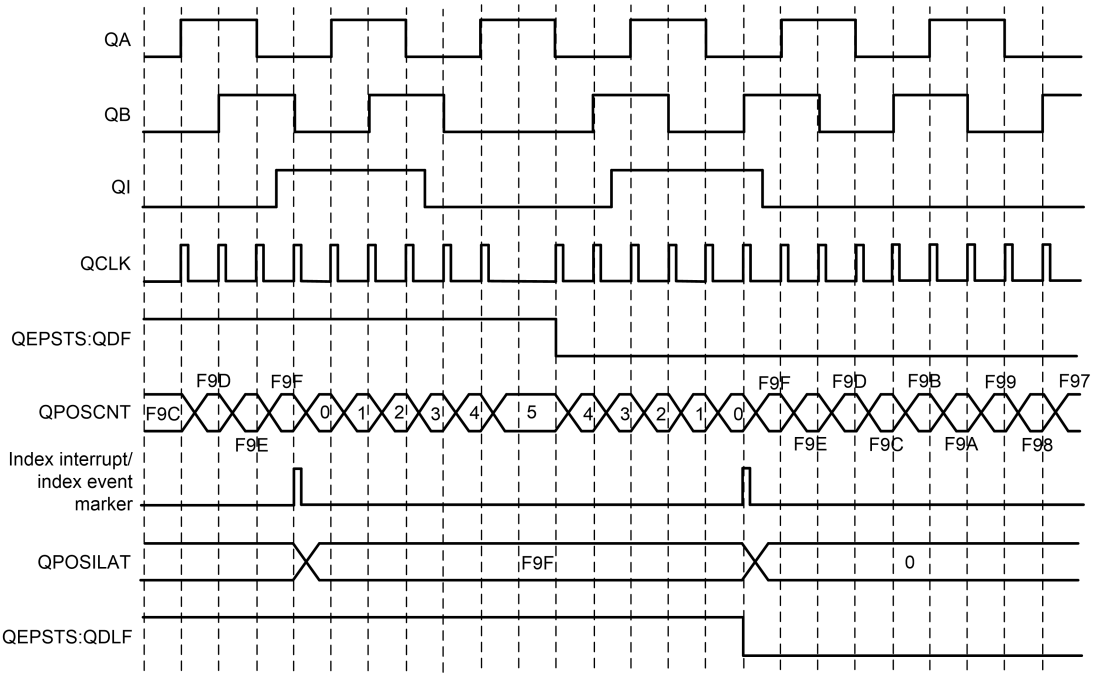
First index marker is defined as the quadrature edge following the first index edge. The eQEP peripheral records the occurrence of the first index marker (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in QEPSTS registers, it also remembers the quadrature edge on the first 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 6-8.

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] must be configured to '00' or '11' when pcrm=0 and the position counter error flag/interrupt flag are generated only in index event reset mode. The position counter value is latched into the IPOSLAT register on every index marker.

Figure 6-8. Position Counter Reset by Index Pulse for 1000 Line Encoder (QPOSMAX = 3999 or 0xF9F)



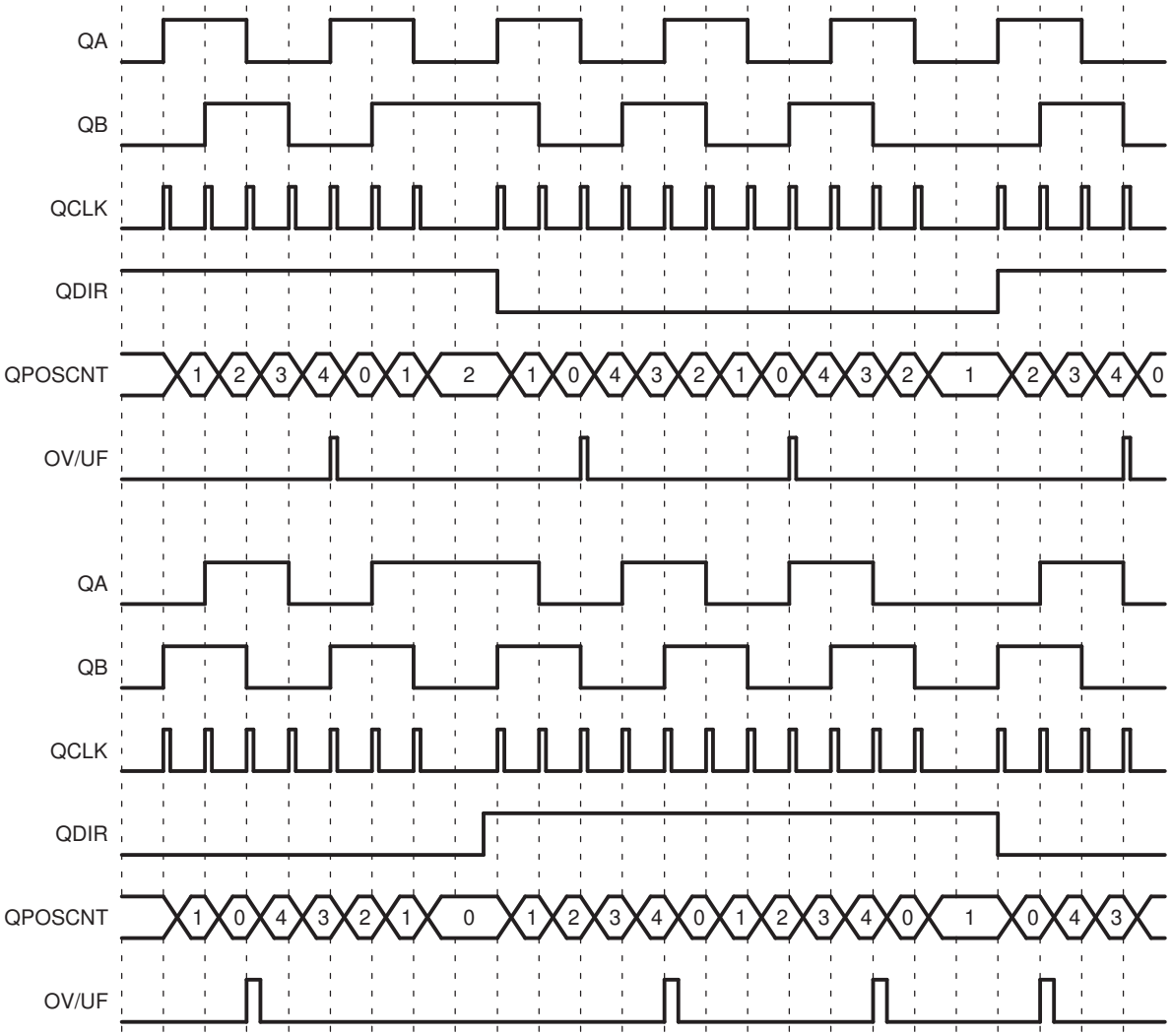
NOTE: In case of boundary condition where time period between Index Event and previous QCLK edge is less than SYSCLK period, then QPOSCNT gets reset to zero or QPOSMAX in the same SYSCLK cycle and does not wait for next QCLK edge to occur.

6.5.1.2 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 6-9 shows the position-counter reset operation in this mode.

The first index marker fields (QEPSTS[FIDF] and QEPSTS[FIMF]) are not applicable in this mode.

Figure 6-9. Position Counter Underflow/Overflow (QPOSMAX = 4)



6.5.1.3 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 6.5.1.2.

The first index marker fields (QEPSTS[FIDF] and QEPSTS[FIMF]) are not applicable in this mode.

6.5.1.4 Position Counter Reset on Unit Time out Event (QEPCTL[PCRM] = 11)

In this mode, QPOSCNT is set to 0 or QPOMAX, depending on the direction mode selected by QDECCTL[QSRC] bits on a unit time event. This is useful for frequency measurement.

6.5.2 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.

6.5.2.1 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. The index event latch configuration bits (QEPCTZ[IEL]) are ignored when QEPCTL[PCRM] = 00.

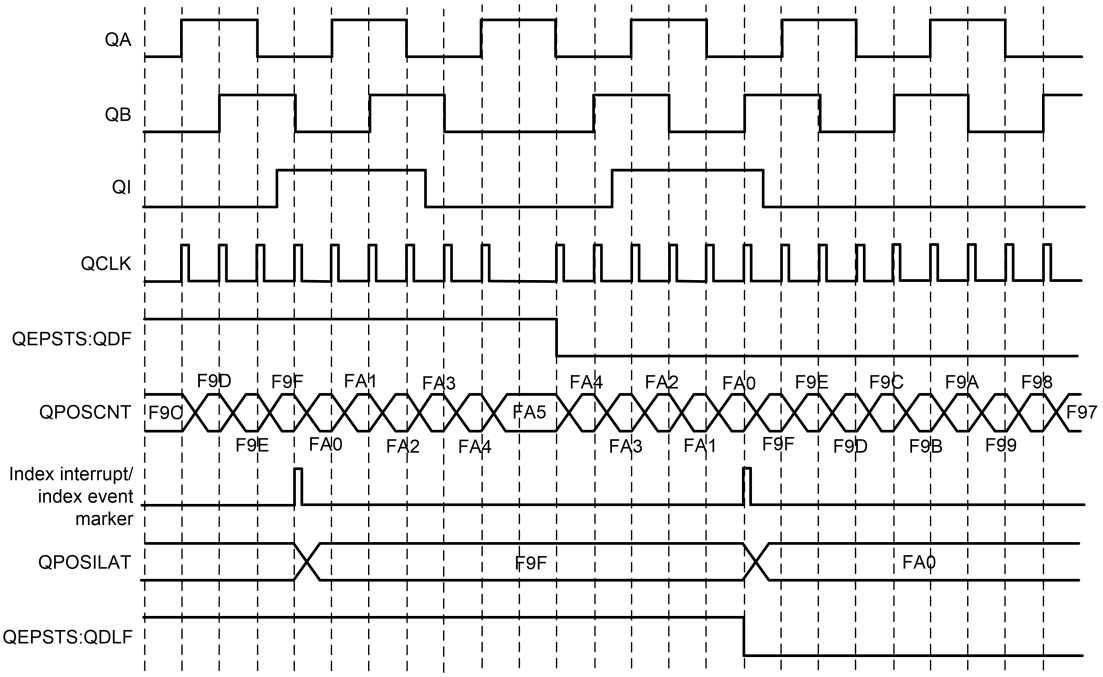
Latch on Rising Edge (QEPCTL[IEL]=01)— The position-counter value (QPOSCNT) is latched to the QPOSILAT register on every rising edge of an index input.

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 6-10 shows the position counter latch using an index event marker.

Figure 6-10. Software Index Marker for 1000-line Encoder (QEPCTL[IEL] = 1)



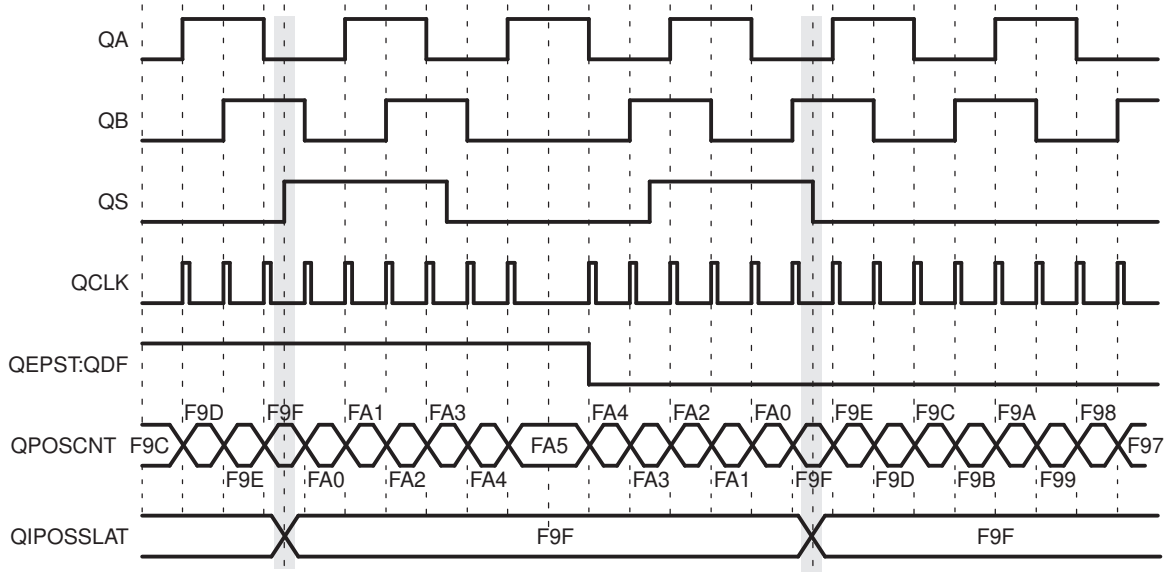
6.5.2.2 Strobe Event Latch

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 6-11.

The strobe event latch interrupt flag (QFLG[SEL) is set when the position counter is latched to the QPOSSLAT register.

Figure 6-11. Strobe Event Latch (QEPCTL[SEL] = 1)



6.5.3 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.

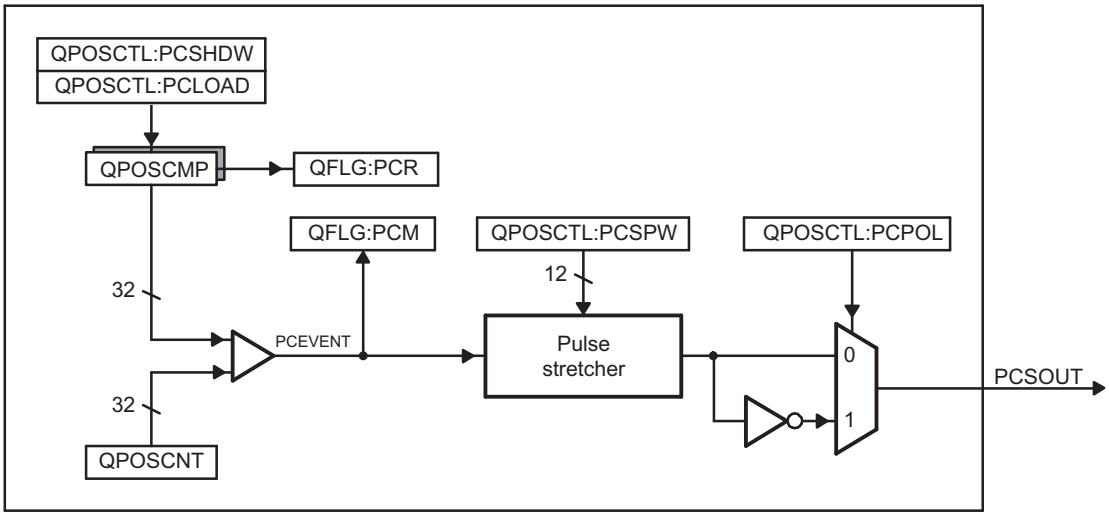
If QEPCTL[SEL] bits are 11, then the position counter is initialized with a value in the QPOSINIT register on the rising edge of strobe input for forward direction and on the falling edge of strobe input for reverse direction.

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.

6.5.4 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 6-12 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 6-12. 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.

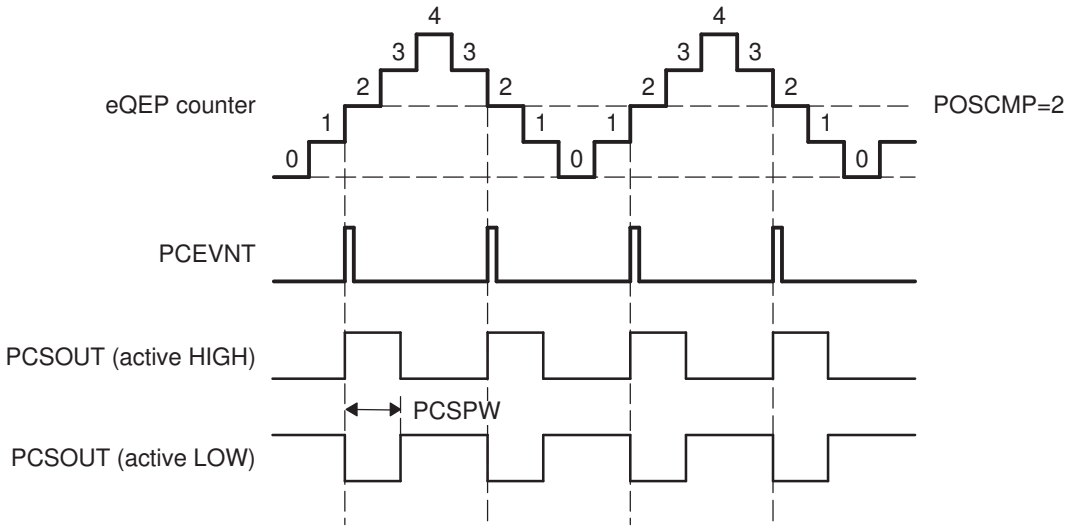
- Load on compare match
- Load on position-counter zero event

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 6-13).

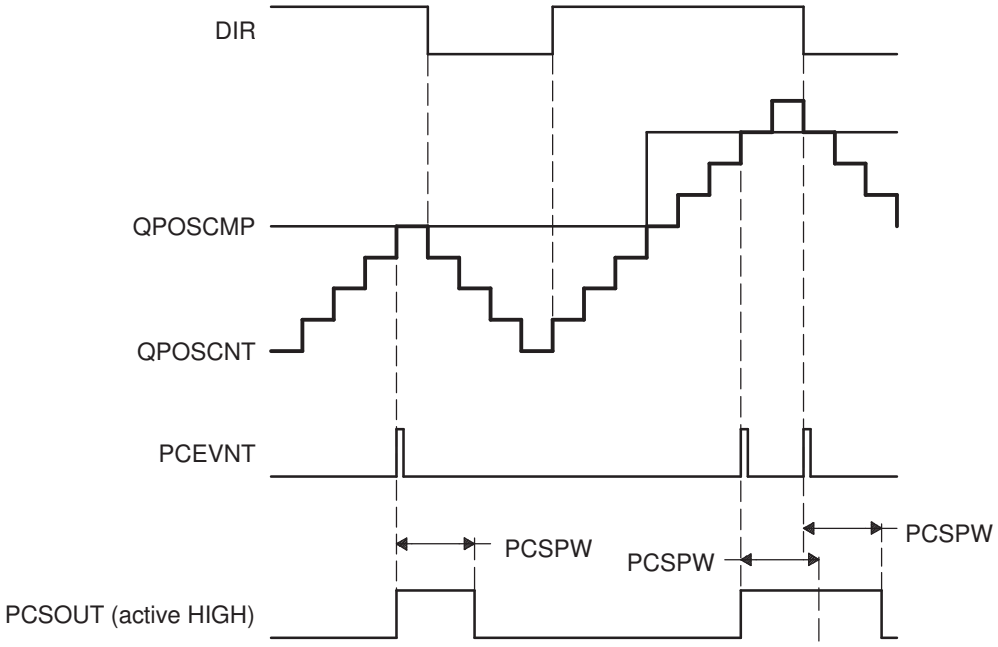
See the register section for the layout of the eQEP Position-Compare Control Register (QPOSCTL) and description of the QPOSCTL bit fields.

Figure 6-13. 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 6-14.

Figure 6-14. eQEP Position-compare Sync Output Pulse Stretcher



6.6 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 6-15. 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 6-16)
- 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]).

The Capture Timer (QCTMR) and Capture Period register (QCPRD) can be configured to latch on following events.

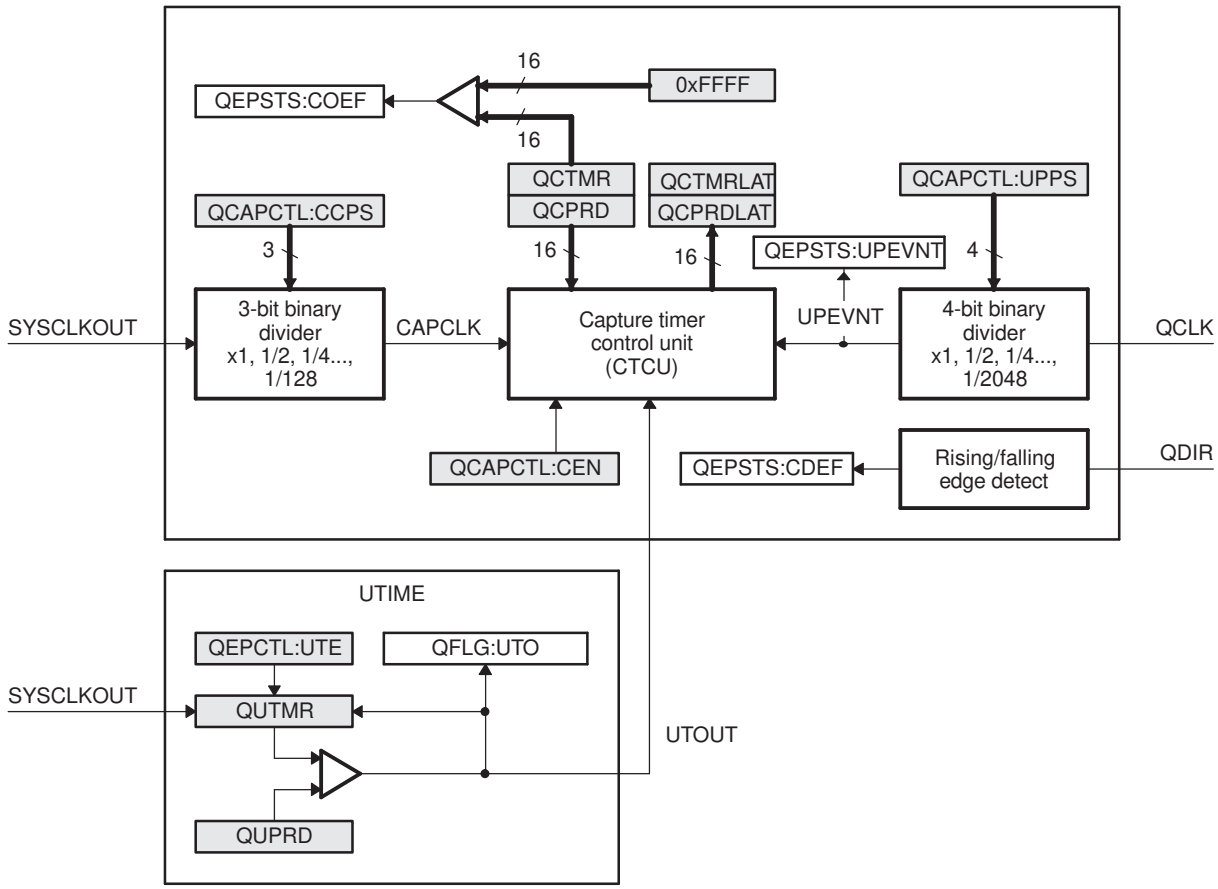
- CPU read of QPOSCNT register
- Unit time-out event

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.

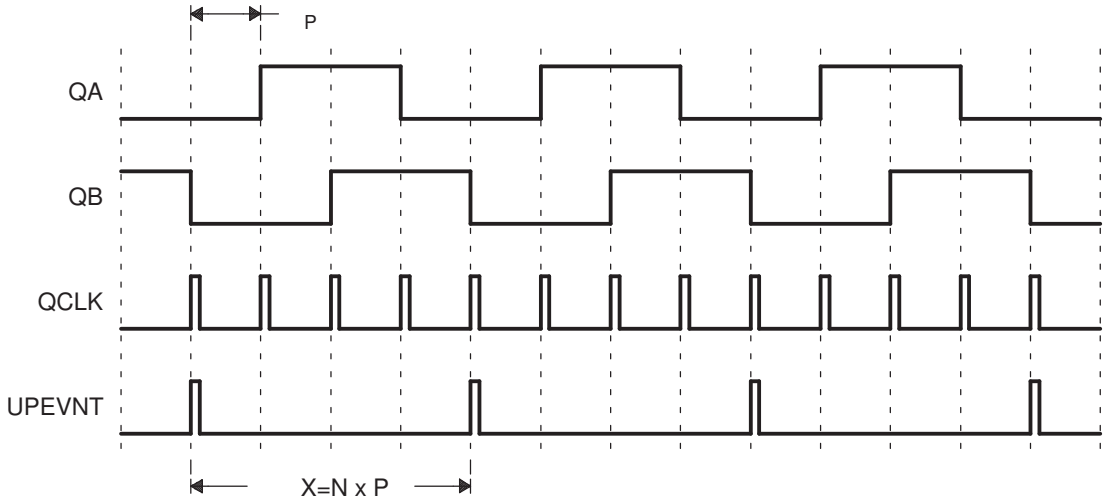
Figure 6-17 shows the capture unit operation along with the position counter.

Figure 6-15. eQEP Edge Capture Unit



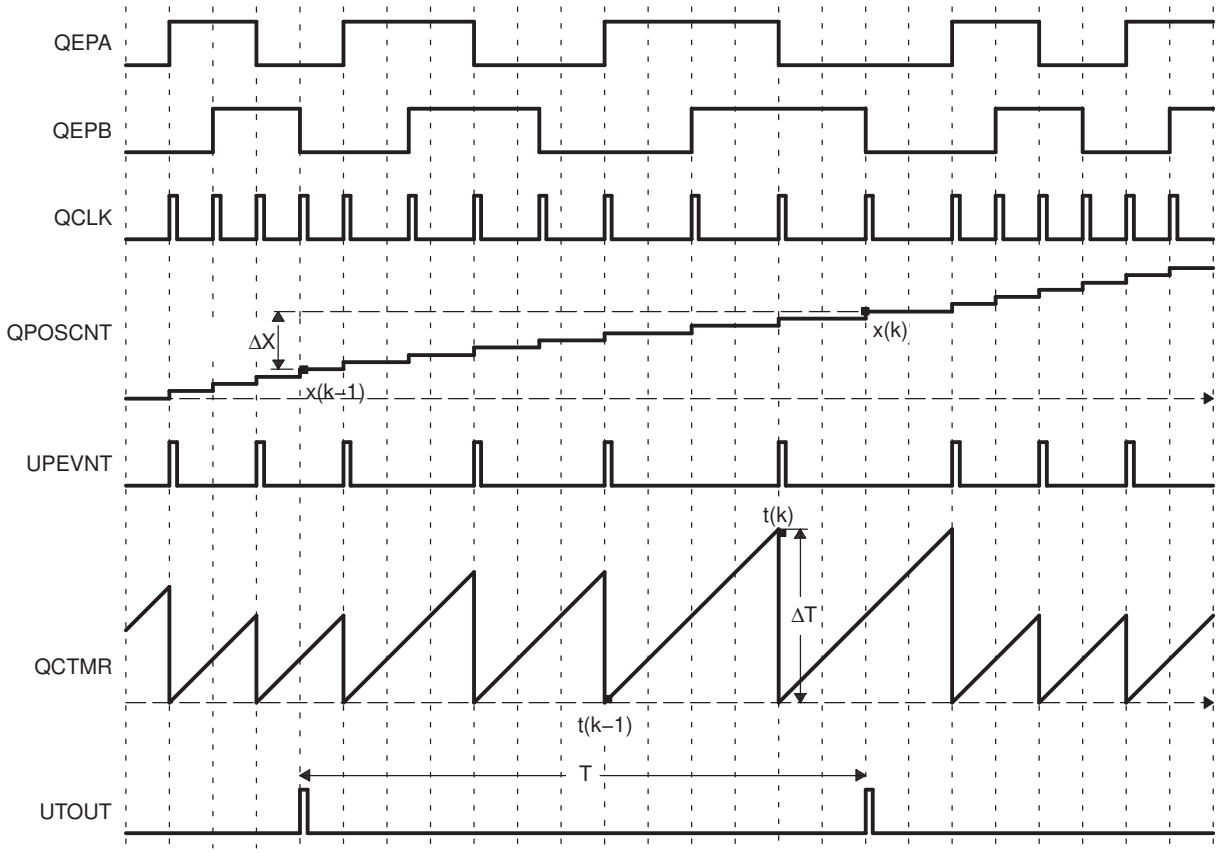
NOTE: The QCAPCTL[UPPS] prescaler should not be modified dynamically (such as switching the unit event prescaler from QCLK/4 to QCLK/8). Doing so may result in undefined behavior. The QCAPCTL[CPPS] prescaler can be modified dynamically (such as switching CAPCLK prescaling mode from SYSCLK/4 to SYSCLK/8) only after the capture unit is disabled.

Figure 6-16. Unit Position Event for Low Speed Measurement (QCAPCTL[UPPS] = 0010)



A N - Number of quadrature periods selected using QCAPCTL[UPPS] bits

Figure 6-17. 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"

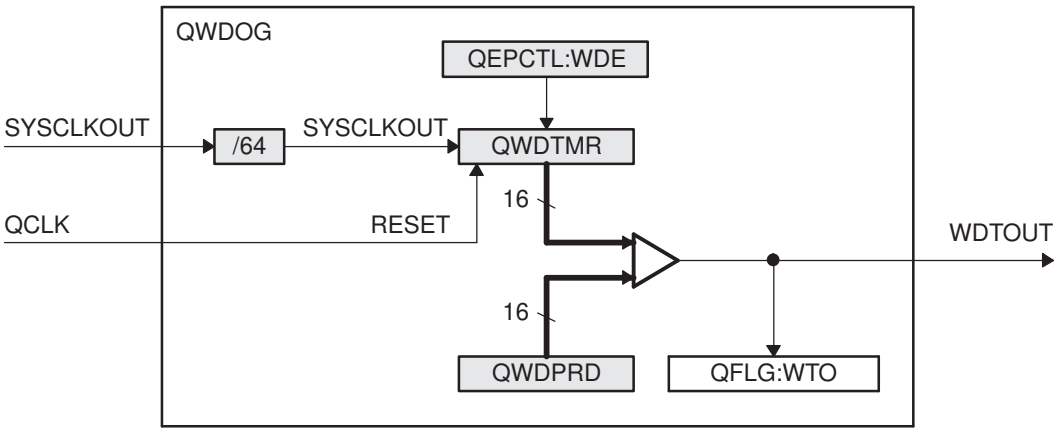
Unit time (T) and unit period(X) are configured using the QUPRD and QCAPCTL[UPPS] registers. Incremental position output and incremental time output is available in the QPOSLAT and QCPRDLAT registers.

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)

6.7 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 6-18. eQEP Watchdog Timer

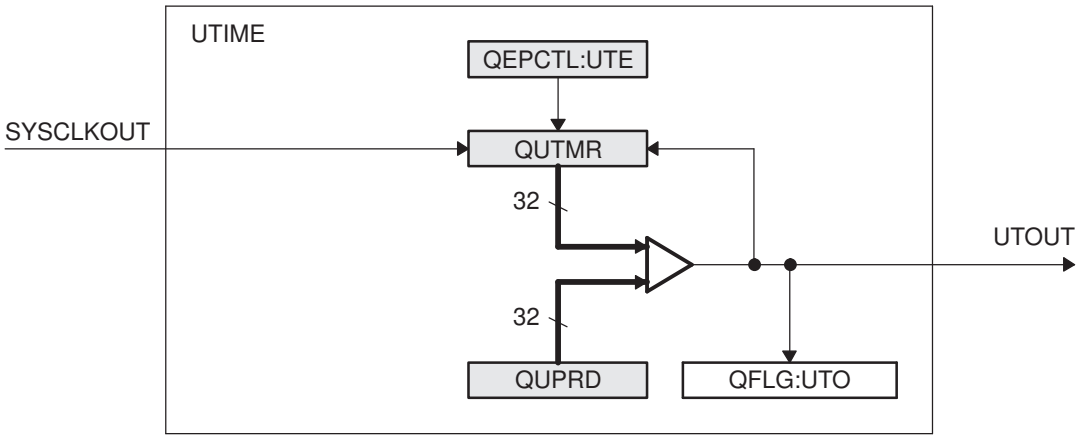


6.8 Unit Timer Base

The eQEP peripheral includes a 32-bit timer (QUTMR) that is clocked by SYSCLKOUT to generate periodic interrupts for velocity calculations. Whenever the unit timer (QUTMR) matches the unit period register (QUPRD), it resets the unit timer (QUTMR) and also generates the unit time out interrupt flag (QFLG[UTO]). The unit timer gets reset whenever timer value equals to configured period value.

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 6.6.

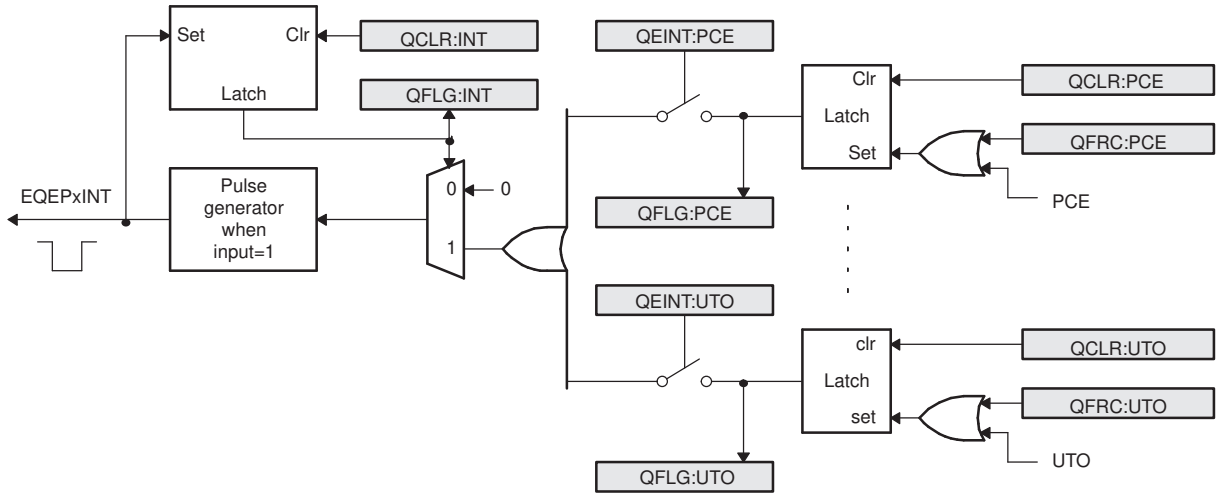
Figure 6-19. eQEP Unit Time Base



6.9 eQEP Interrupt Structure

Figure 6-20 shows how the interrupt mechanism works in the EQEP module.

Figure 6-20. 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 to PIE when:

a. Interrupt is enabled for eQEP event inside QEINT register

b. Interrupt flag for eQEP event inside QFLG register is set, and

c. Global interrupt status flag bit QFLG[INT] had been cleared for previously generated interrupt event. 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.If either flags inside the QFLG register are not cleared, further interrupt event will not generate interrupt to PIE. You can force an interrupt event by way of the interrupt force register (QFRC), which is useful for test purposes.

6.10 eQEP Registers

This section describes the Enhanced Quadrature Encoder Pulse Registers.

6.10.1 eQEP Base Addresses

Table 6-3. EQEP Base Address Table

Bit Field Name		Base Address
Instance	Structure	
EQep1Regs	EQEP_REGS	0x0000_6B00
EQep2Regs	EQEP_REGS	0x0000_6B40

6.10.2 EQEP_REGS Registers

Table 6-4 lists the EQEP_REGS registers. All register offset addresses not listed in Table 6-4 should be considered as reserved locations and the register contents should not be modified.

Table 6-4. EQEP_REGS Registers

Offset	Acronym	Register Name	Write Protection	Section
0h	QPOSCNT	Position Counter		Go
2h	QPOSINIT	Position Counter Init		Go
4h	QPOSMAX	Maximum Position Count		Go
6h	QPOSCMP	Position Compare		Go
8h	QPOSILAT	Index Position Latch		Go
Ah	QPOSSLAT	Strobe Position Latch		Go
Ch	QPOSLAT	Position Latch		Go
Eh	QUTMR	QEP Unit Timer		Go
10h	QUPRD	QEP Unit Period		Go
12h	QWDTMR	QEP Watchdog Timer		Go
13h	QWDPRD	QEP Watchdog Period		Go
14h	QDECCTL	Quadrature Decoder Control		Go
15h	QEPCTL	QEP Control		Go
16h	QCAPCTL	Qaudrature Capture Control		Go
17h	QPOSCTL	Position Compare Control		Go
18h	QEINT	QEP Interrupt Control		Go
19h	QFLG	QEP Interrupt Flag		Go
1Ah	QCLR	QEP Interrupt Clear		Go
1Bh	QFRC	QEP Interrupt Force		Go
1Ch	QEPSTS	QEP Status		Go
1Dh	QCTMR	QEP Capture Timer		Go
1Eh	QCPRD	QEP Capture Period		Go
1Fh	QCTMRLAT	QEP Capture Latch		Go
20h	QCPRDLAT	QEP Capture Period Latch		Go

Complex bit access types are encoded to fit into small table cells. Table 6-5 shows the codes that are used for access types in this section.

Table 6-5. EQEP_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R -0	Read Returns 0s
Write Type		
W	W	Write
W1S	W 1S	Write 1 to set
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		

Table 6-5. EQEP_REGS Access Type Codes (continued)

Access Type	Code	Description
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

6.10.2.1 QPOSCNT Register (Offset = 0h) [reset = 0h]

QPOSCNT is shown in Figure 6-21 and described in Table 6-6.

Return to the Summary Table.

Position Counter

Figure 6-21. QPOSCNT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QPOSCNT																															
R/W-0h																															

Table 6-6. QPOSCNT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QPOSCNT	R/W	0h	Position Counter 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. This Register acts as a Read ONLY register while counter is counting up/down. Note: It is recommended to only write to the position counter register (QPOSCNT) during initialization, i.e. when the eQEP position counter is disabled (QPEN bit of QEPCTL is zero). Once the position counter is enabled (QPEN bit is one), writing to the eQEP position counter register (QPOSCNT) may cause unexpected results. Reset type: SYSRSn

6.10.2.2 QPOSINIT Register (Offset = 2h) [reset = 0h]

QPOSINIT is shown in Figure 6-22 and described in Table 6-7.

Return to the Summary Table.

Position Counter Init

Figure 6-22. QPOSINIT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QPOSINIT																															
R/W-0h																															

Table 6-7. QPOSINIT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QPOSINIT	R/W	0h	Position Counter Init 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. Writes to this register should always be full 32-bit writes. Reset type: SYSRSn

6.10.2.3 QPOSMAX Register (Offset = 4h) [reset = 0h]

QPOSMAX is shown in Figure 6-23 and described in Table 6-8.

Return to the Summary Table.

Maximum Position Count

Figure 6-23. QPOSMAX Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QPOSMAX																															
R/W-0h																															

Table 6-8. QPOSMAX Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QPOSMAX	R/W	0h	Maximum Position Count This register contains the maximum position counter value. Writes to this register should always be full 32-bit writes. Reset type: SYSRSn

6.10.2.4 QPOSCMP Register (Offset = 6h) [reset = 0h]

QPOSCMP is shown in Figure 6-24 and described in Table 6-9.

Return to the Summary Table.

Position Compare

Figure 6-24. QPOSCMP Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QPOSCMP																															
R/W-0h																															

Table 6-9. QPOSCMP Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QPOSCMP	R/W	0h	Position Compare The position-compare value in this register is compared with the position counter (QPOSCNT) to generate sync output and/or interrupt on compare match. Reset type: SYSRSn

6.10.2.5 QPOSILAT Register (Offset = 8h) [reset = 0h]

QPOSILAT is shown in Figure 6-25 and described in Table 6-10.

Return to the Summary Table.

Index Position Latch

Figure 6-25. QPOSILAT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QPOSILAT																															
R-0h																															

Table 6-10. QPOSILAT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QPOSILAT	R	0h	Index Position Latch The position-counter value is latched into this register on an index event as defined by the QEPCTL[IEL] bits. Reset type: SYSRSn

6.10.2.6 QPOSSLAT Register (Offset = Ah) [reset = 0h]

QPOSSLAT is shown in Figure 6-26 and described in Table 6-11.

Return to the Summary Table.

Strobe Position Latch

Figure 6-26. QPOSSLAT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QPOSSLAT																															
R-0h																															

Table 6-11. QPOSSLAT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QPOSSLAT	R	0h	Strobe Position Latch The position-counter value is latched into this register on a strobe event as defined by the QEPCTL[SEL] bits. Reset type: SYSRSn

6.10.2.7 QPOSLAT Register (Offset = Ch) [reset = 0h]

QPOSLAT is shown in Figure 6-27 and described in Table 6-12.

Return to the Summary Table.

Position Latch

Figure 6-27. QPOSLAT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QPOSLAT																															
R-0h																															

Table 6-12. QPOSLAT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QPOSLAT	R	0h	Position Latch The position-counter value is latched into this register on a unit time out event. Reset type: SYSRSn

6.10.2.8 QUTMR Register (Offset = Eh) [reset = 0h]

QUTMR is shown in Figure 6-28 and described in Table 6-13.

Return to the Summary Table.

QEP Unit Timer

Figure 6-28. QUTMR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QUTMR																															
R/W-0h																															

Table 6-13. QUTMR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QUTMR	R/W	0h	QEP Unit Timer This register acts as time base for unit time event generation. When this timer value matches the unit time period value a unit time event is generated. Reset type: SYSRSn

6.10.2.9 QUPRD Register (Offset = 10h) [reset = 0h]

QUPRD is shown in Figure 6-29 and described in Table 6-14.

Return to the Summary Table.

QEP Unit Period

Figure 6-29. QUPRD Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QUPRD																															
R/W-0h																															

Table 6-14. QUPRD Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	QUPRD	R/W	0h	QEP Unit Period This register contains the period count for the unit timer to generate periodic unit time events. These events latch the eQEP position information at periodic intervals and optionally generate an interrupt. Writes to this register should always be full 32-bit writes. Reset type: SYSRSn

6.10.2.10 QWDTMR Register (Offset = 12h) [reset = 0h]

QWDTMR is shown in Figure 6-30 and described in Table 6-15.

Return to the Summary Table.

QEP Watchdog Timer

Figure 6-30. QWDTMR Register

15	14	13	12	11	10	9	8
QWDTMR							
R/W-0h							
7	6	5	4	3	2	1	0
QWDTMR							
R/W-0h							

Table 6-15. QWDTMR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	QWDTMR	R/W	0h	QEP Watchdog Timer This register acts as time base for the watchdog to detect motor stalls. When this timer value matches with the watchdog's period value a watchdog timeout interrupt is generated. This register is reset upon edge transition in quadrature-clock indicating the motion. Reset type: SYSRSn

6.10.2.11 QWDPRD Register (Offset = 13h) [reset = 0h]

QWDPRD is shown in Figure 6-31 and described in Table 6-16.

Return to the Summary Table.

QEP Watchdog Period

Figure 6-31. QWDPRD Register

15	14	13	12	11	10	9	8
QWDPRD							
R/W-0h							
7	6	5	4	3	2	1	0
QWDPRD							
R/W-0h							

Table 6-16. QWDPRD Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	QWDPRD	R/W	0h	QEP Watchdog Period 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. Reset type: SYSRSn

6.10.2.12 QDECCTL Register (Offset = 14h) [reset = 0h]

QDECCTL is shown in Figure 6-32 and described in Table 6-17.

Return to the Summary Table.

Quadrature Decoder Control

Figure 6-32. QDECCTL Register

15	14	13	12	11	10	9	8
QSRC		SOEN	SPSEL	XCR	SWAP	IGATE	QAP
R/W-0h		R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
QBP	QIP	QSP	RESERVED				
R/W-0h	R/W-0h	R/W-0h	R-0h				

Table 6-17. QDECCTL Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	QSRC	R/W	0h	Position-counter source selection Reset type: SYSRSn 0h (R/W) = Quadrature count mode (QCLK = iCLK, QDIR = iDIR) 1h (R/W) = Direction-count mode (QCLK = xCLK, QDIR = xDIR) 2h (R/W) = UP count mode for frequency measurement (QCLK = xCLK, QDIR = 1) 3h (R/W) = DOWN count mode for frequency measurement (QCLK = xCLK, QDIR = 0)
13	SOEN	R/W	0h	Sync output-enable Reset type: SYSRSn 0h (R/W) = Disable position-compare sync output 1h (R/W) = Enable position-compare sync output
12	SPSEL	R/W	0h	Sync output pin selection Reset type: SYSRSn 0h (R/W) = Index pin is used for sync output 1h (R/W) = Strobe pin is used for sync output
11	XCR	R/W	0h	External Clock Rate Reset type: SYSRSn 0h (R/W) = 2x resolution: Count the rising/falling edge 1h (R/W) = 1x resolution: Count the rising edge only
10	SWAP	R/W	0h	CLK/DIR Signal Source for Position Counter Reset type: SYSRSn 0h (R/W) = Quadrature-clock inputs are not swapped 1h (R/W) = Quadrature-clock inputs are swapped
9	IGATE	R/W	0h	Index pulse gating option Reset type: SYSRSn 0h (R/W) = Disable gating of Index pulse 1h (R/W) = Gate the index pin with strobe
8	QAP	R/W	0h	QEPA input polarity Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Negates QEPA input
7	QBP	R/W	0h	QEPB input polarity Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Negates QEPB input

Table 6-17. QDECCTL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
6	QIP	R/W	0h	QEPI input polarity Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Negates QEPI input
5	QSP	R/W	0h	QEPS input polarity Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Negates QEPS input
4-0	RESERVED	R	0h	Reserved

6.10.2.13 QEPCTL Register (Offset = 15h) [reset = 0h]

QEPCTL is shown in Figure 6-33 and described in Table 6-18.

Return to the Summary Table.

QEP Control

Figure 6-33. QEPCTL Register

15	14	13	12	11	10	9	8
FREE_SOFT		PCRM		SEI		IEI	
R/W-0h		R/W-0h		R/W-0h		R/W-0h	
7	6	5	4	3	2	1	0
SWI	SEL	IEL		QPEN	QCLM	UTE	WDE
R/W-0h	R/W-0h	R/W-0h		R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 6-18. QEPCTL Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	FREE_SOFT	R/W	0h	Emulation mode Reset type: SYSRSn 0h (R/W) = QPOSCNT behavior Position counter stops immediately on emulation suspend 0h (R/W) = QWDTMR behavior Watchdog counter stops immediately 0h (R/W) = QUTMR behavior Unit timer stops immediately 0h (R/W) = QCTMR behavior Capture Timer stops immediately 1h (R/W) = QPOSCNT behavior Position counter continues to count until the rollover 1h (R/W) = QWDTMR behavior Watchdog counter counts until WD period match roll over 1h (R/W) = QUTMR behavior Unit timer counts until period rollover 1h (R/W) = QCTMR behavior Capture Timer counts until next unit period event 2h (R/W) = QPOSCNT behavior Position counter is unaffected by emulation suspend 2h (R/W) = QWDTMR behavior Watchdog counter is unaffected by emulation suspend 2h (R/W) = QUTMR behavior Unit timer is unaffected by emulation suspend 2h (R/W) = QCTMR behavior Capture Timer is unaffected by emulation suspend 3h (R/W) = Same as FREE_SOFT_2
13-12	PCRM	R/W	0h	Postion counter reset Reset type: SYSRSn 0h (R/W) = Position counter reset on an index event 1h (R/W) = Position counter reset on the maximum position 2h (R/W) = Position counter reset on the first index event 3h (R/W) = Position counter reset on a unit time event
11-10	SEI	R/W	0h	Strobe event initialization of position counter Reset type: SYSRSn 0h (R/W) = Does nothing (action disabled) 1h (R/W) = Does nothing (action disabled) 2h (R/W) = Initializes the position counter on rising edge of the QEPS signal 3h (R/W) = 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

Table 6-18. QEPCTL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
9-8	IEI	R/W	0h	Index event init of position count Reset type: SYSRSn 0h (R/W) = Do nothing (action disabled) 1h (R/W) = Do nothing (action disabled) 2h (R/W) = Initializes the position counter on the rising edge of the QEPI signal (QPOSCNT = QPOSINIT) 3h (R/W) = Initializes the position counter on the falling edge of QEPI signal (QPOSCNT = QPOSINIT)
7	SWI	R/W	0h	Software init position counter Reset type: SYSRSn 0h (R/W) = Do nothing (action disabled) 1h (R/W) = Initialize position counter (QPOSCNT=QPOSINIT). This bit is not cleared automatically
6	SEL	R/W	0h	Strobe event latch of position counter Reset type: SYSRSn 0h (R/W) = 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 1h (R/W) = 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	R/W	0h	Index event latch of position counter (software index marker) Reset type: SYSRSn 0h (R/W) = Reserved 1h (R/W) = Latches position counter on rising edge of the index signal 2h (R/W) = Latches position counter on falling edge of the index signal 3h (R/W) = 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.
3	QPEN	R/W	0h	Quadrature position counter enable/software reset Reset type: SYSRSn 0h (R/W) = Reset the eQEP peripheral internal operating flags/read-only registers. Control/configuration registers are not disturbed by a software reset. When QPEN is disabled, some flags in the QFLG register do not get reset or cleared and show the actual state of that flag. 1h (R/W) = eQEP position counter is enabled
2	QCLM	R/W	0h	QEP capture latch mode Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = 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	R/W	0h	QEP unit timer enable Reset type: SYSRSn 0h (R/W) = Disable eQEP unit timer 1h (R/W) = Enable unit timer

Table 6-18. QEPCTL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
0	WDE	R/W	0h	QEP watchdog enable Reset type: SYSRSn 0h (R/W) = Disable the eQEP watchdog timer 1h (R/W) = Enable the eQEP watchdog timer

6.10.2.14 QCAPCTL Register (Offset = 16h) [reset = 0h]

QCAPCTL is shown in Figure 6-34 and described in Table 6-19.

Return to the Summary Table.

Qaudrature Capture Control

Figure 6-34. QCAPCTL Register

15	14	13	12	11	10	9	8
CEN	RESERVED						
R/W-0h	R-0h						
7	6	5	4	3	2	1	0
RESERVED	CCPS			UPPS			
R-0h	R/W-0h			R/W-0h			

Table 6-19. QCAPCTL Register Field Descriptions

Bit	Field	Type	Reset	Description
15	CEN	R/W	0h	Enable eQEP capture Reset type: SYSRSn 0h (R/W) = eQEP capture unit is disabled 1h (R/W) = eQEP capture unit is enabled
14-7	RESERVED	R	0h	Reserved
6-4	CCPS	R/W	0h	eQEP capture timer clock prescaler Reset type: SYSRSn 0h (R/W) = CAPCLK = SYSCLKOUT/1 1h (R/W) = CAPCLK = SYSCLKOUT/2 2h (R/W) = CAPCLK = SYSCLKOUT/4 3h (R/W) = CAPCLK = SYSCLKOUT/8 4h (R/W) = CAPCLK = SYSCLKOUT/16 5h (R/W) = CAPCLK = SYSCLKOUT/32 6h (R/W) = CAPCLK = SYSCLKOUT/64 7h (R/W) = CAPCLK = SYSCLKOUT/128
3-0	UPPS	R/W	0h	Unit position event prescaler Reset type: SYSRSn 0h (R/W) = UPEVNT = QCLK/1 1h (R/W) = UPEVNT = QCLK/2 2h (R/W) = UPEVNT = QCLK/4 3h (R/W) = UPEVNT = QCLK/8 4h (R/W) = UPEVNT = QCLK/16 5h (R/W) = UPEVNT = QCLK/32 6h (R/W) = UPEVNT = QCLK/64 7h (R/W) = UPEVNT = QCLK/128 8h (R/W) = UPEVNT = QCLK/256 9h (R/W) = UPEVNT = QCLK/512 Ah (R/W) = UPEVNT = QCLK/1024 Bh (R/W) = UPEVNT = QCLK/2048 Ch (R/W) = Reserved Dh (R/W) = Reserved Eh (R/W) = Reserved Fh (R/W) = Reserved

6.10.2.15 QPOSCTL Register (Offset = 17h) [reset = 0h]

QPOSCTL is shown in Figure 6-35 and described in Table 6-20.

Return to the Summary Table.

Position Compare Control

Figure 6-35. QPOSCTL Register

15	14	13	12	11	10	9	8
PCSHDW	PCLOAD	PCPOL	PCE	PCSPW			
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h			
7	6	5	4	3	2	1	0
PCSPW							
R/W-0h							

Table 6-20. QPOSCTL Register Field Descriptions

Bit	Field	Type	Reset	Description
15	PCSHDW	R/W	0h	Position compare of shadow enable Reset type: SYSRSn 0h (R/W) = Shadow disabled, load Immediate 1h (R/W) = Shadow enabled
14	PCLOAD	R/W	0h	Position compare of shadow load Reset type: SYSRSn 0h (R/W) = Load on QPOSCNT = 0 1h (R/W) = Load when QPOSCNT = QPOSCMP
13	PCPOL	R/W	0h	Polarity of sync output Reset type: SYSRSn 0h (R/W) = Active HIGH pulse output 1h (R/W) = Active LOW pulse output
12	PCE	R/W	0h	Position compare enable/disable Reset type: SYSRSn 0h (R/W) = Disable position compare unit 1h (R/W) = Enable position compare unit
11-0	PCSPW	R/W	0h	Select-position-compare sync output pulse width Reset type: SYSRSn 0h (R/W) = 1 * 4 * SYSCLKOUT cycles 1h (R/W) = 2 * 4 * SYSCLKOUT cycles FFFh (R/W) = 4096 * 4 * SYSCLKOUT cycles

6.10.2.16 QEINT Register (Offset = 18h) [reset = 0h]

QEINT is shown in Figure 6-36 and described in Table 6-21.

Return to the Summary Table.

QEP Interrupt Control

Figure 6-36. QEINT Register

15	14	13	12	11	10	9	8
RESERVED				UTO	IEL	SEL	PCM
R-0h				R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	QPE	PCE	RESERVED
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h

Table 6-21. QEINT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reserved
11	UTO	R/W	0h	Unit time out interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
10	IEL	R/W	0h	Index event latch interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
9	SEL	R/W	0h	Strobe event latch interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
8	PCM	R/W	0h	Position-compare match interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
7	PCR	R/W	0h	Position-compare ready interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
6	PCO	R/W	0h	Position counter overflow interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
5	PCU	R/W	0h	Position counter underflow interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
4	WTO	R/W	0h	Watchdog time out interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled

Table 6-21. QEINT Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3	QDC	R/W	0h	Quadrature direction change interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
2	QPE	R/W	0h	Quadrature phase error interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
1	PCE	R/W	0h	Position counter error interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt is disabled 1h (R/W) = Interrupt is enabled
0	RESERVED	R	0h	Reserved

6.10.2.17 QFLG Register (Offset = 19h) [reset = 0h]

QFLG is shown in Figure 6-37 and described in Table 6-22.

Return to the Summary Table.

QEP Interrupt Flag

Figure 6-37. QFLG Register

15	14	13	12	11	10	9	8
RESERVED				UTO	IEL	SEL	PCM
R-0h				R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	PHE	PCE	INT
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

Table 6-22. QFLG Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reserved
11	UTO	R	0h	Unit time out interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
10	IEL	R	0h	Index event latch interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
9	SEL	R	0h	Strobe event latch interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
8	PCM	R	0h	eQEP compare match event interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
7	PCR	R	0h	Position-compare ready interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
6	PCO	R	0h	Position counter overflow interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
5	PCU	R	0h	Position counter underflow interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
4	WTO	R	0h	Watchdog timeout interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated

Table 6-22. QFLG Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3	QDC	R	0h	Quadrature direction change interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
2	PHE	R	0h	Quadrature phase error interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
1	PCE	R	0h	Position counter error interrupt flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated
0	INT	R	0h	Global interrupt status flag Reset type: SYSRSn 0h (R/W) = No interrupt generated 1h (R/W) = Interrupt was generated

6.10.2.18 QCLR Register (Offset = 1Ah) [reset = 0h]

QCLR is shown in Figure 6-38 and described in Table 6-23.

Return to the Summary Table.

QEP Interrupt Clear

Figure 6-38. QCLR Register

15	14	13	12	11	10	9	8
RESERVED				UTO	IEL	SEL	PCM
R-0h				R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h
7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	PHE	PCE	INT
R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h

Table 6-23. QCLR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reserved
11	UTO	R-0/W1S	0h	Clear unit time out interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
10	IEL	R-0/W1S	0h	Clear index event latch interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
9	SEL	R-0/W1S	0h	Clear strobe event latch interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
8	PCM	R-0/W1S	0h	Clear eQEP compare match event interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
7	PCR	R-0/W1S	0h	Clear position-compare ready interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
6	PCO	R-0/W1S	0h	Clear position counter overflow interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
5	PCU	R-0/W1S	0h	Clear position counter underflow interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
4	WTO	R-0/W1S	0h	Clear watchdog timeout interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag

Table 6-23. QCLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3	QDC	R-0/W1S	0h	Clear quadrature direction change interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
2	PHE	R-0/W1S	0h	Clear quadrature phase error interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
1	PCE	R-0/W1S	0h	Clear position counter error interrupt flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag
0	INT	R-0/W1S	0h	Global interrupt clear flag Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Clears the interrupt flag

6.10.2.19 QFRC Register (Offset = 1Bh) [reset = 0h]

QFRC is shown in Figure 6-39 and described in Table 6-24.

Return to the Summary Table.

QEP Interrupt Force

Figure 6-39. QFRC Register

15	14	13	12	11	10	9	8
RESERVED				UTO	IEL	SEL	PCM
R-0h				R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
PCR	PCO	PCU	WTO	QDC	PHE	PCE	RESERVED
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h

Table 6-24. QFRC Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reserved
11	UTO	R/W	0h	Force unit time out interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
10	IEL	R/W	0h	Force index event latch interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
9	SEL	R/W	0h	Force strobe event latch interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
8	PCM	R/W	0h	Force position-compare match interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
7	PCR	R/W	0h	Force position-compare ready interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
6	PCO	R/W	0h	Force position counter overflow interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
5	PCU	R/W	0h	Force position counter underflow interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
4	WTO	R/W	0h	Force watchdog time out interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt

Table 6-24. QFRC Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3	QDC	R/W	0h	Force quadrature direction change interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
2	PHE	R/W	0h	Force quadrature phase error interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
1	PCE	R/W	0h	Force position counter error interrupt Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Force the interrupt
0	RESERVED	R	0h	Reserved

6.10.2.20 QEPSTS Register (Offset = 1Ch) [reset = 0h]

QEPSTS is shown in Figure 6-40 and described in Table 6-25.

Return to the Summary Table.

QEP Status

Figure 6-40. QEPSTS Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
UPEVNT	FIDF	QDF	QDLF	COEF	CDEF	FIMF	PCEF
R/W-0h	R-0h	R-0h	R-0h	R/W-0h	R/W-0h	R/W-0h	R-0h

Table 6-25. QEPSTS Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	UPEVNT	R/W	0h	Unit position event flag Reset type: SYSRSn 0h (R/W) = No unit position event detected 1h (R/W) = Unit position event detected. Write 1 to clear
6	FIDF	R	0h	Direction on the first index marker Status of the direction is latched on the first index event marker. Reset type: SYSRSn 0h (R/W) = Counter-clockwise rotation (or reverse movement) on the first index event 1h (R/W) = Clockwise rotation (or forward movement) on the first index event
5	QDF	R	0h	Quadrature direction flag Reset type: SYSRSn 0h (R/W) = Counter-clockwise rotation (or reverse movement) 1h (R/W) = Clockwise rotation (or forward movement)
4	QDLF	R	0h	eQEP direction latch flag Reset type: SYSRSn 0h (R/W) = Counter-clockwise rotation (or reverse movement) on index event marker 1h (R/W) = Clockwise rotation (or forward movement) on index event marker
3	COEF	R/W	0h	Capture overflow error flag Reset type: SYSRSn 0h (R/W) = Overflow has not occurred. 1h (R/W) = Overflow occurred in eQEP Capture timer (QEPCTMR).
2	CDEF	R/W	0h	Capture direction error flag Reset type: SYSRSn 0h (R/W) = Capture direction error has not occurred. 1h (R/W) = Direction change occurred between the capture position event.
1	FIMF	R/W	0h	First index marker flag Note: Once this flag has been set, if the flag is cleared the flag will not be set again until the module is reset by a peripheral or system reset. Reset type: SYSRSn 0h (R/W) = First index pulse has not occurred. 1h (R/W) = Set by first occurrence of index pulse.

Table 6-25. QEPSTS Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
0	PCEF	R	0h	Position counter error flag. This bit is not sticky and it is updated for every index event. Reset type: SYSRSn 0h (R/W) = No error occurred during the last index transition 1h (R/W) = Position counter error

6.10.2.21 QCTMR Register (Offset = 1Dh) [reset = 0h]

QCTMR is shown in Figure 6-41 and described in Table 6-26.

Return to the Summary Table.

QEP Capture Timer

Figure 6-41. QCTMR Register

15	14	13	12	11	10	9	8
QCTMR							
R/W-0h							
7	6	5	4	3	2	1	0
QCTMR							
R/W-0h							

Table 6-26. QCTMR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	QCTMR	R/W	0h	This register provides time base for edge capture unit. Reset type: SYSRSn

6.10.2.22 QCPRD Register (Offset = 1Eh) [reset = 0h]

QCPRD is shown in Figure 6-42 and described in Table 6-27.

Return to the Summary Table.

QEP Capture Period

Figure 6-42. QCPRD Register

15	14	13	12	11	10	9	8
QCPRD							
R/W-0h							
7	6	5	4	3	2	1	0
QCPRD							
R/W-0h							

Table 6-27. QCPRD Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	QCPRD	R/W	0h	This register holds the period count value between the last successive eQEP position events Reset type: SYSRSn

6.10.2.23 QCTMRLAT Register (Offset = 1Fh) [reset = 0h]

QCTMRLAT is shown in Figure 6-43 and described in Table 6-28.

Return to the Summary Table.

QEP Capture Latch

Figure 6-43. QCTMRLAT Register

15	14	13	12	11	10	9	8
QCTMRLAT							
R-0h							
7	6	5	4	3	2	1	0
QCTMRLAT							
R-0h							

Table 6-28. QCTMRLAT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	QCTMRLAT	R	0h	The eQEP capture timer value can be latched into this register on two events viz., unit timeout event, reading the eQEP position counter. Reset type: SYSRSn

6.10.2.24 QCPRDLAT Register (Offset = 20h) [reset = 0h]

QCPRDLAT is shown in Figure 6-44 and described in Table 6-29.

Return to the Summary Table.

QEP Capture Period Latch

Figure 6-44. QCPRDLAT Register

15	14	13	12	11	10	9	8
QCPRDLAT							
R-0h							
7	6	5	4	3	2	1	0
QCPRDLAT							
R-0h							

Table 6-29. QCPRDLAT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	QCPRDLAT	R	0h	eQEP capture period value can be latched into this register on two events viz., unit timeout event, reading the eQEP position counter. Reset type: SYSRSn

Analog-to-Digital Converter (ADC)

This ADC is applicable for the ADC found on the TMS320x2833x family of processors. This includes all Flash-based, ROM-based and RAM-based devices within the 2833x family.

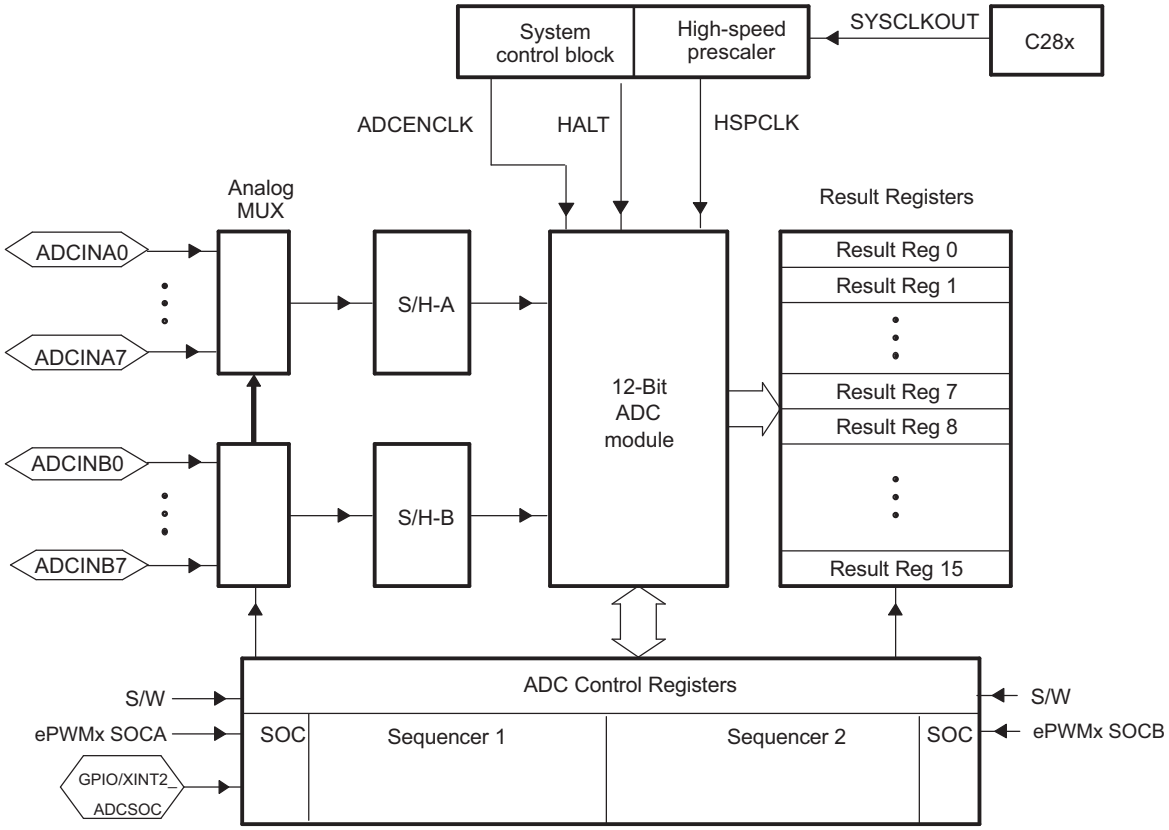
The TMS320x2833x ADC module is a 12-bit pipelined analog-to-digital converter (ADC). The analog circuits of this converter, referred to as the core in this document, include the front-end analog multiplexers (MUXs), sample-and-hold (S/H) circuits, the conversion core, voltage regulators, and other analog supporting circuits. Digital circuits, referred to as the wrapper in this document, include programmable conversion sequencer, result registers, interface to analog circuits, interface to device peripheral bus, and interface to other on-chip modules.

Topic **Page**

7.1 Features and Implementation

The ADC module has 16 channels, configurable as two independent 8-channel modules. The two independent 8-channel modules can be cascaded to form a 16-channel module. Although there are multiple input channels and two sequencers, there is only one converter in the ADC module. Figure 7-1 shows the block diagram of the ADC module.

Figure 7-1. Block Diagram of the ADC Module



The two 8-channel modules can autosequence a series of conversions; each module has the choice of selecting any one of the respective eight channels available through an analog MUX. In the cascaded mode, the autosequencer functions as a single 16-channel sequencer. On each sequencer, once the conversion is completed, the selected channel value is stored in its respective ADCRESULT register. Autosequencing allows the system to convert the same channel multiple times, allowing the user to perform oversampling algorithms. This oversampling gives increased resolution over traditional single-sampled conversion results.

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
- Analog input: 0 V to 3 V
- Fast conversion time runs at 25 MHz, ADC clock, or 12.5 MSPS
- 16-channel, multiplexed inputs
- Autosequencing capability provides up to 16 "autoconversions" in a single session. Each conversion can be programmed to select any 1 of 16 input channels.
- Sequencer can be operated as two independent 8-state sequencers or as one large 16-state sequencer (that is, two cascaded 8-state sequencers).
- Sixteen result registers (individually addressable) to store conversion values
  - The digital value of the input analog voltage is derived by:

$$\text{Digital Value} = 0, \quad \text{when input} \le 0\ \text{V}$$

$$\text{Digital Value} = 4096 \times \frac{\text{Input Analog Voltage} - \text{ADCLO}}{3}, \quad \text{when } 0\ \text{V} < \text{input} < 3\ \text{V}$$

$$\text{Digital Value} = 4095, \quad \text{when input} \ge 3\ \text{V}$$

NOTE: All fractional values are truncated.

- Multiple triggers as sources for the start-of-conversion (SOC) sequence
  - S/W - software immediate start
  - ePWM 1-6
  - GPIO XINT2
- Flexible interrupt control allows interrupt request on every end-of-sequence (EOS) or every other EOS
- Sequencer can operate in "start/stop" mode, allowing multiple "time-sequenced triggers" to synchronize conversions.
- ePWM triggers can operate independently in dual-sequencer mode.
- Sample-and-hold (S/H) acquisition time window has separate prescale control.

NOTE: To obtain the specified accuracy of the ADC, proper board layout is very critical. To the best extent possible, traces leading to the ADCINxx pins should not run in close proximity to the digital signal paths. This is to minimize switching noise on the digital lines from getting coupled to the ADC inputs. Furthermore, proper isolation techniques must be used to isolate the ADC module power pins from the digital supply. The TMDSCNCD28335 controlCARD can be used as a reference for the above constraints.

7.2 ADC Circuit

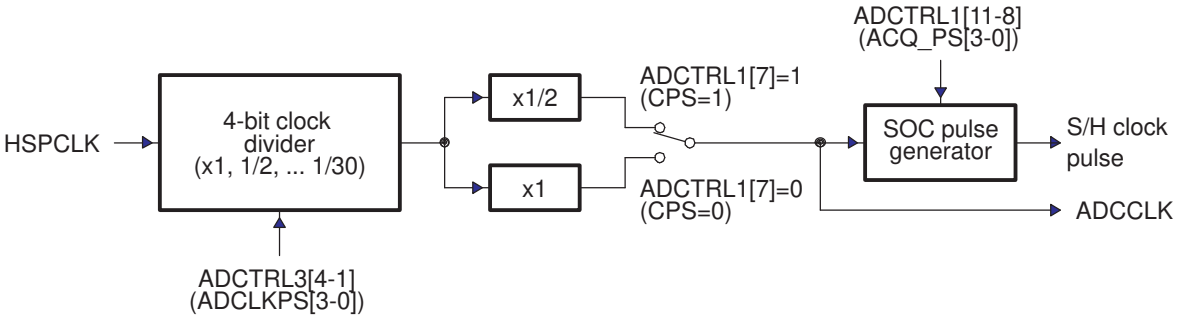
The following sub-sections describe the physical implementation of the ADC circuit. Topics that are covered include:

- Clocking
- Sample and Hold Circuitry
- Reference Selection
- Power Modes and Sequencing
- Calibration and Offset Correction

7.2.1 ADC Clocking and Sample Rate Calculations

The peripheral clock HSPCLK is divided down by the ADCCLKPS[3:0] bits of the ADCTRL3 register. An extra divide-by-two is provided via the CPS bit of the ADCTRL1 register. In addition, the ADC can be tailored to accommodate variations in source impedances by widening the sampling/acquisition period. This is controlled by the ACQ_PS[3:0] bits in the ADCTRL1 register. These bits do not affect the conversion portion of the S/H and conversion process, but do extend the length of time in which the sampling portion takes by extending the start of the conversion pulse. See Figure 7-2.

Figure 7-2. ADC Core Clock and Sample-and-Hold (S/H) Clock



NOTE: See register bit definition for clock divider ratio and S/H pulse control. S/H pulse width determines the size of acquisition window (the time period for which sampling switch is closed).

The ADC module has several prescaler stages to generate any desired ADC operating clock speed. Figure 7-3 defines the clock selection stages that feed the ADC module. Table 7-1 gives two example settings and shows both the effective sustained sampling rate and the sample and hold window time for those settings in both sequential sampling mode (SMODE_SEL = 0) and simultaneous sampling mode (SMODE_SEL = 1).

Figure 7-3. Clock Chain to the ADC

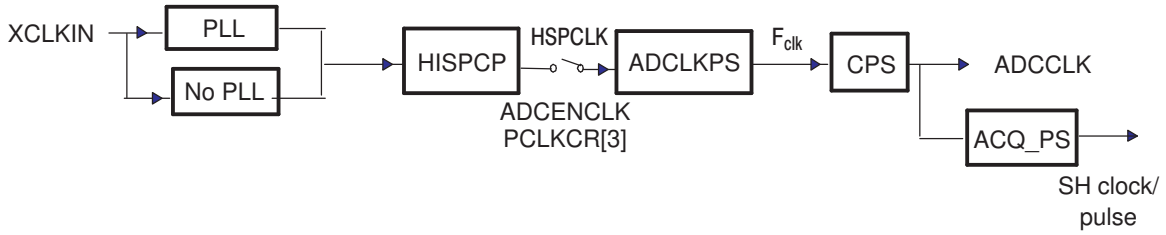


Table 7-1. Clock Chain to the ADC

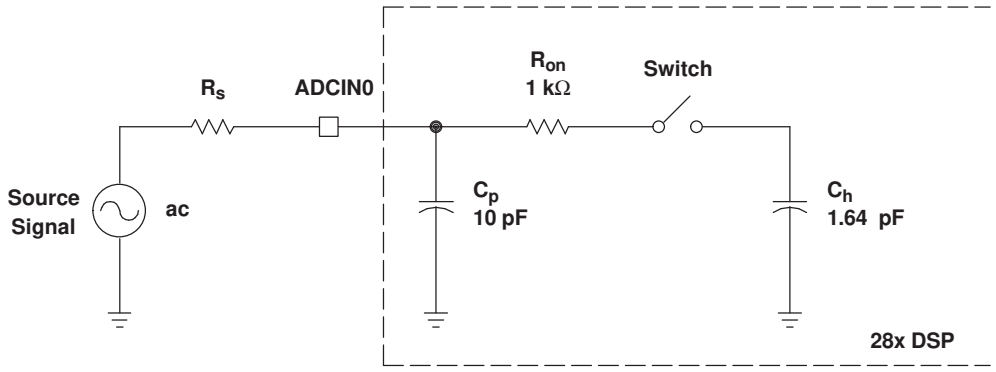
SYSCLK OUT	HISPCLK	ADCTRL3 [4-1]	ADCTRL1[7]	ADCCLK	ADCTRL1 [11-8]	Maximum Sustained Conversion Rate	
	HSPCP = 3	ADCLKPS = 0	CPS = 0		ACQ_PS = 0	SMODE_SEL = 0	SMODE_SEL = 1 (1)
150 MHz	150 MHz/ 2 × 3 = **25 MHz**	**25 MHz**	**25 MHz**	**25 MHz**	(1 / 25 MHz) × (ACQ_PS + 1) **Acquisition time = 40 ns**	1 / (Acq time + 1 / ADCCLK) 1 / (40ns + 40ns) = **12.5 MSPS**	1 / (Acq time + 2 × (1 / ADCCLK)) 1/(40ns + 2(40ns)) = **8.33 MSPS**
	HSPCP = 2	ADCLKPS = 2	CPS = 1		ACQ_PS = 15		
100 MHz	100 MHz/ 2 × 2 = **25 MHz**	25/2 × 2 = **6.25 MHz**	6.25 MHz/ 2 × 1 = **3.125 MHz**	**3.125 MHz**	(1 / 3.125 MHz) × (ACQ_PS + 1) **Acquisition time = 5.12 µs**	1 / (Acq time + 1 / ADCCLK) 1 / (5.12µs + 320ns) = **184 kSPS**	1 / (Acq time + 2 × (1 / ADCCLK)) 1/(5.12µs + 2(320ns)) = **174 kSPS**

(1) Simultaneous mode generates 2 results for every ADC start of conversion.

7.2.2 ADC Sample and Hold Circuit and Modeling

As shown in Figure 7-4 , 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 7-4. ADCINx Input Model



Typical Values of the Input Circuit Components:

Switch Resistance (R_{on}):	**1 kΩ**
Sampling Capacitor (C_h):	**1.64 pF**
Parasitic Capacitance (C_p):	**10 pF**
Source Resistance (R_s):	**50 Ω**

NOTE: The ADC does not precondition the Ch capacitor voltage before conversions, therefore the following behaviors apply:

1. There is no predetermined ADC conversion value when the ADCIN pin is not connected to a Source Signal
2. Residual charge will remain on Ch between ADC conversions
3. Sequential conversions may suffer from cross-talk if the ACQPS window is too short for Ch to settle

For correct operation, the input signal to the ADC must be allowed adequate time to charge the sample and hold capacitor, C_h. Typically, the S+H duration is chosen such that C_h will be charged to within ½ LSB or ¼ LSB of the final value, depending on the tolerable settling error.

The S+H time required to satisfy the settling error is largely influenced by the bandwidth of the source signal. Therefore, the following recommendations for approximating the S+H duration will be simplified into two practical scenarios of either high bandwidth or low bandwidth signals. A high bandwidth source signal will be characterized as being able to meet the settling error and real-time requirements of the system using a supported ACQPS setting. A low bandwidth source signal is one that requires a longer S+H duration than is acceptable.

7.2.2.1 ACQPS Approximation for High Bandwidth Signals

Signals that must be sampled frequently with minimal phase delay (such as feedback sensors used in control-loop calculations) are high bandwidth signals. These signal paths require a small R_sC_s time constant as seen by the ADCINx pin. An external signal buffer (such as an op-amp) may be used to boost the sampling bandwidth; such buffers should ideally have a bandwidth that is high enough to fully charge C_h within the selected ACPQS S+H window.

7.2.2.1.1 ACQPS Approximation Equations for High Bandwidth Signals

An approximation of the required settling time can be determined using an RC settling model. The time constant (τ) for the model is given by the equation:

$$\tau = (R_s + R_{on})(C_h) + (R_s)(C_s + C_p)$$

And the number of time constants needed is given by the equation:

$$k = \ln\left(\frac{2^n}{settling\ error}\right) - \ln\left(\frac{C_s + C_p}{C_h}\right)$$

So the total S+H time (t_{S+H})should be set to at least:

$$t_{s+h} = k \cdot \tau$$

Finally, t_{S+H} is used to determine the minimum value to program into the ACQPS field of the ADCSOCxCTL registers:

$$ACQPS = (t_{s+h} \cdot f_{ADCCLK}) - 1$$

Where the following parameters are provided by the ADC input model:

- n = ADC resolution (in bits)
- R_{on} = ADC sampling switch resistance
- C_h = ADC sampling capacitor
- C_p = ADC parasitic pin capacitance for the channel

And the following parameters are dependent on the application design:

- settling error = tolerable settling error (in LSBs)
- R_s = ADC driving circuit source impedance
- C_s = ADC driving circuit source capacitance on ADC input pin
- f_{ADCCLK} = ADC clock frequency

7.2.2.1.2 ACQPS Approximation Example for High Bandwidth Signals

For example, assuming the following parameters:

- n = 12-bits
- settling error =¼ LSB
- R_{on} = 1000Ω
- C_h = 1.6pF
- C_p = 10pF
- R_s = 56Ω
- C_s = 2.2nF
- f_{ADCCLK} = 30MHz

The time constant would be calculated as:

$$\tau = (56\Omega + 3400\Omega)(1.6pF) + (56\Omega)(2200pF + 5pF) = 5.5ns + 123.5ns = 129ns \quad (5)$$

And the number of required time constants would be:

$$k = \ln\left(\frac{2^{12}\ LSB}{0.25\ LSB}\right) - \ln\left(\frac{2200pF + 5pF}{1.6pF}\right) = 9.7 - 7.2 = 2.5$$

So the S+H time should be set to at least:

$$t_{s+h} = 2.5 \cdot 129ns = 322.5ns$$

Finally, the minimum ACQPS value is calculated and rounded up to the nearest supported value:

$$ACQPS = (322.5ns \cdot 30MHz) - 1 = 8.7 \rightarrow 9$$

While this gives a rough estimate of the required acquisition window, a better method would be to setup a circuit with the ADC input model, a model of the source impedance/capacitance, and any board parasitics in SPICE (or similar software) and simulate to verify that the sampling capacitor settles to the desired accuracy.

7.2.2.2 ACQPS Approximation for Low Bandwidth Signals

Signals that are sampled infrequently and are tolerant of low-pass filtering (such as ambient temperature sensors) can be treated as low bandwidth signals. A large C_s that is sized to be much larger than C_h will allow the ADC to sample the R_sC_s filtered signal quickly at the expense of increased phase delay. C_h will receive the bulk of its charge from C_s during the S+H window, and C_s will recover its charge through R_s between ADC samples.

7.2.2.2.1 ACQPS Approximation Equations for Low Bandwidth Signals

The desired settling accuracy will determine the value of C_s:

$$C_s = C_h\left(\frac{2^n}{settling\ error}\right)$$

The desired recovery time and acceptable charge error will determine the value of R_s:

$$R_s = \frac{t_{recovery}}{n_\tau \cdot C_s}$$

With this configuration, C_s acts as the effective source signal, which simplifies the equations for calculating (k) and (τ) as follows:

$$\tau = R_{on} \cdot C_h$$

$$k = \ln\left(\frac{2^n}{settling\ error}\right)$$

So the total S+H time (t_{S+H})should be set to at least:

$$t_{s+h} = k \cdot \tau \quad (6)$$

Finally, t_{S+H} is used to determine the minimum value to program into the ACQPS field of the ADCSOCxCTL registers:

$$ACQPS = (t_{s+h} \cdot f_{ADCCLK}) - 1 \quad (7)$$

Where the following parameters are provided by the ADC input model:

- n = ADC resolution (in bits)
- R_{on} = ADC sampling switch resistance
- C_h = ADC sampling capacitor
- C_p = ADC parasitic pin capacitance for the channel

And the following parameters are dependent on the application design:

- settling error = tolerable settling error (in LSBs)
- $t_{recovery}$ = amount of time between ADC samples
- n_τ = number of RC time constants that comprise $t_{recovery}$
- R_s = ADC driving circuit source impedance
- C_s = ADC driving circuit source capacitance on ADC input pin
- f_{ADCCLK} = ADC clock frequency

The selection of n_τ will determine how much the voltage on C_s is able to recover between samples. An insufficient amount of recovery time will introduce a droop error by way of an undercharged C_s. Table 7-2 shows the relationship between n_τ and the estimated droop error.

Table 7-2. Estimated Droop Error from n_τ Value

n_τ	Droop Error (% of Settling Error)	Droop Error for C_s sized to ¼ LSB Settling Error (LSB)	Total Error (Droop Error + ¼ LSB Settling Error)
0.25	352%	0.88 LSB	1.13 LSB
0.50	154%	0.39 LSB	0.64 LSB
0.75	90%	0.23 LSB	0.48 LSB
1.00	58%	0.15 LSB	0.40 LSB
2.00	16%	0.04 LSB	0.29 LSB
3.00	5%	0.01 LSB	0.26 LSB
4.00	2%	0.01 LSB	0.26 LSB
5.00	1%	0.00 LSB	0.25 LSB

7.2.2.2.2 ACQPS Approximation Example for Low Bandwidth Signals

For example, assuming the following parameters:

- n = 12-bits
- settling error =¼ LSB
- $t_{recovery}$ = 1ms
- n_τ = 2
- R_{on} = 3400Ω
- C_h = 1.6pF
- C_p = 5pF
- f_{ADCCLK} = 30MHz

The minimum source capacitance would be calculated and rounded up to a common value:

$$C_s = 1.6pF\left(\frac{2^{12}\ LSB}{0.25\ LSB}\right) = 26.2nF \rightarrow 33nF$$

Then the maximum source resistance would be calculated and rounded down to a common value:

$$R_s = \frac{1ms}{2 \cdot 33nF} = 15.2k\Omega \rightarrow 12\text{k}\Omega$$

Now the time constant (τ) and multiple (k) can be calculated as:

$$\tau = 3400\Omega \cdot 1.6pF = 5.4ns$$

$$k = \ln\left(\frac{2^{12}\ LSB}{0.25\ LSB}\right) = 9.7$$

So the S+H time should be set to at least:

$$t_{s+h} = 9.7 \cdot 5.4ns = 52.4ns$$

Finally, the minimum ACQPS value is calculated and rounded up to the nearest supported value:

$$ACQPS = (52.4ns \cdot 30MHz) - 1 = 0.6 \rightarrow 6$$

While this gives a rough estimate of the required acquisition window, a better method would be to setup a circuit with the ADC input model, a model of the source impedance/capacitance, and any board parasitics in SPICE (or similar software) and simulate to verify that the sampling capacitor settles to the desired accuracy.

7.2.3 Reference Selection

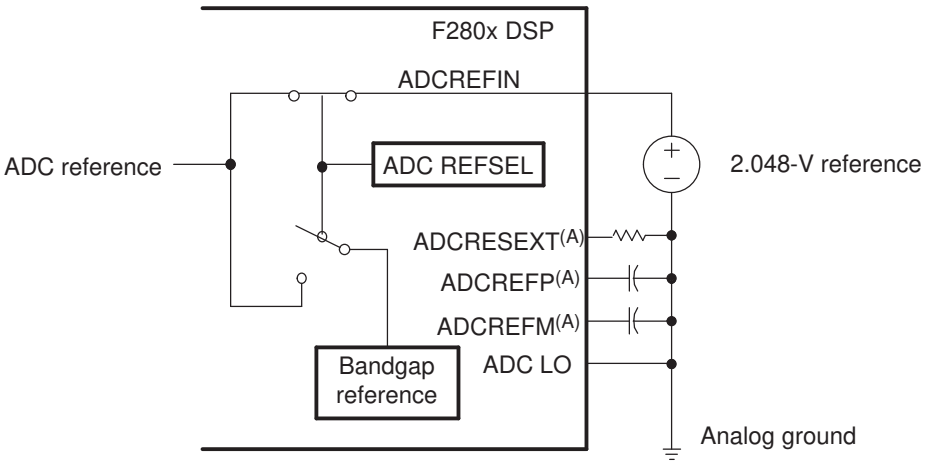
By default, an internally generated bandgap voltage reference is selected to supply the ADC logic.

Based on customer application requirements, the ADC logic may be supplied by an external voltage reference. The ADC will accept 2.048 V, 1.5 V, or 1.024 V on the ADCREFIN pin. The value of the ADCREFSEL register determines the reference source selected.

If the internal reference option is chosen, the ADCREFIN pin can be left connected to the selected source, left floating, or grounded. Regardless of which option is chosen, the external circuit for the ADCRESEXT, ADCREFP, and ADCREFM pins is the same.

The external reference voltage of 2.048 V was chosen to match industry standard reference components. These components are available in various temperature ratings. A recommended Texas Instruments part is REF3020AIDBZ.

Figure 7-5. External Bias for 2.048-V External Reference



NOTE: For component values, see the TMS320F2833x, TMS320F2823x Digital Signal Controllers (DSCs) Data Manual.

7.2.4 Power-up Sequence and Power Modes

7.2.4.1 Power-up Sequencing

The ADC resets to the ADC off state. When powering up the ADC, use the following sequence:

1. If external reference is desired, enable this mode using bits 15-14 in the ADCREFSEL Register. This mode must be enabled before band gap is powered.
2. Power up the reference, bandgap, and analog circuits together by setting bits 7-5 (ADCBGRFDN[1:0], ADCPWDN) in the ADCTRL3 register.
3. Before performing the first conversion, a delay of 5 ms is required.

When powering down the ADC, all three bits can be cleared simultaneously. The ADC power level must be controlled via software and they are independent of the state of the device power modes.

Sometimes it is desirable to power down the ADC while leaving the band-gap and reference powered by clearing the ADCPWDN bit only. When the ADC is re-powered, a delay of 1 ms is required after this bit is set before performing any conversions.

NOTE: The 2833x ADC requires a 5-ms delay after all of the circuits are powered up. This differs from the 281x ADC.

7.2.4.2 Power Modes

The ADC supports three separate power sources each controlled by independent bits in the ADCTRL3 register. These three bits combine to make up three power levels: ADC power up, ADC power down, and ADC off.

Table 7-3. Power Options

Power Level	ADCBGRFDN1	ADCBGRFDN0	ADCPWDN
ADC power-up	1	1	1
ADC power-down	1	1	0
ADC off	0	0	0
Reserved	1	0	X
Reserved	0	1	X

7.2.5 Calibration and Offset Correction

7.2.5.1 ADC Calibration

The ADC_cal() routine is programmed into TI reserved OTP memory by the factory. The boot ROM automatically calls the ADC_cal() routine to initialize the ADCREFSEL and ADCOFFTRIM registers 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 ADCREFSEL and ADCOFFTRIM must be initialized by the application. For working examples, see the ADC initialization in (C2000Ware).

The next two sections describe different methods for calling the ADC_Cal() routine.

NOTE: FAILURE TO INITIALIZE THESE REGISTERS WILL CAUSE THE ADC TO FUNCTION OUT OF SPECIFICATION.

Because TI reserved OTP memory is secure, the ADC_Cal() routine must be called from secure memory or called from non-secure memory after the Code Security Module is unlocked. If the system is reset or the ADC module is reset using Bit 14 (RESET) from the ADC Control Register 1, the routine must be repeated.

7.2.5.1.1 ADC_Cal Assembly Routine Method

The following three steps describe how to call the ADC_cal routine from an application:

Step 1. Add the ADC_cal assembly function to your project. The source is included with (C2000Ware). The following code shows the contents of the ADC_cal function. The values 0xAAAA and 0xBBBB are place holders. The actual values programmed by TI are device specific.

```
;---------------------------------------------
; This is the ADC cal routine. This routine is programmed
; into reserved memory by the factory. 0xAAAA and 0xBBBB
; are place holders. The actual values programmed by TI
; are device specific.
; The ADC clock must be enabled before calling
; this function.
;---------------------------------------------
   .def _ADC_cal
   .asg "0x711C", ADCREFSEL_LOC
   .sect ".adc_cal"
   _ADC_cal
      MOVW DP, #ADCREFSEL_LOC >> 6
      MOV @28, #0xAAAA
      MOV @29, #0xBBBB
      LRETR
Step 2. Add the .adc_cal section to your linker command file using the following:

   MEMORY
   {
   PAGE 0 :
      ...
      ADC_CAL : origin = 0x380080, length = 0x000009
   ...
   }
   SECTIONS
   {
     ...
     .adc_cal : load = ADC_CAL, PAGE = 0, TYPE = NOLOAD
   ...
   }
Step 3. Call the ADC_cal function before using the ADC. The ADC clocks must be enabled before
making this call.
   extern void ADC_cal(void);

   ...
   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
   ADC_cal();
   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0;
   EDIS;
```

7.2.5.1.2 Pointer to-Function Method

Because the ADC_CAL() routine is already programmed to TI reserved OTP memory by the factory, it can be called via a pointer to the function by using the following steps:

Step 1. Define ADC_Cal as a pointer to the programmed function in the OTP memory.

```
 #define ADC_Cal (void (*) (void)) 0x380080
```

Step 2. Call the ADC_Cal() function.

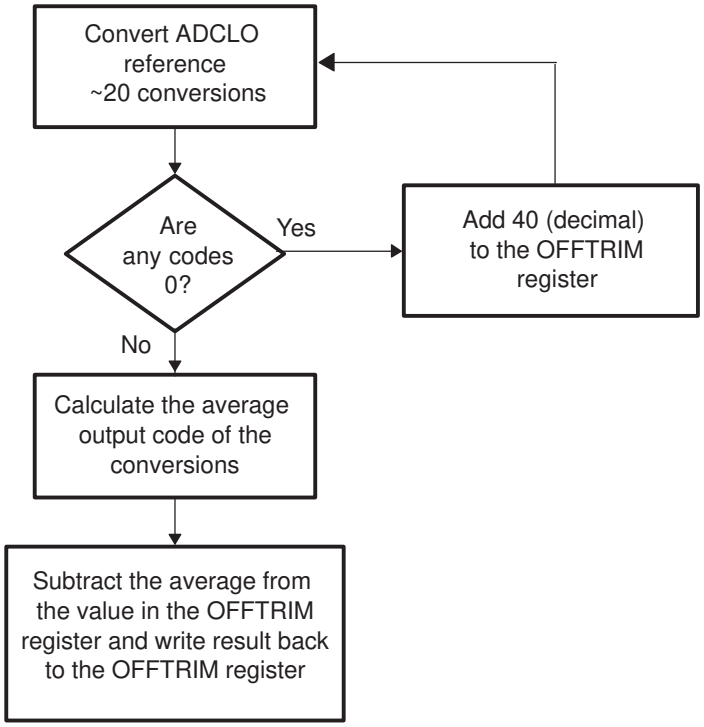
```
 ... EALLOW; SysCtrlRegs.PCLKCR0.bit.ADCENCLK=1; (*ADC_Cal) ();
SysCTRLRegs.PCLKCR0.bit.ADCENCLK=0; EDIS;
```

7.2.5.2 Offset Error Correction

The 2833x ADC supports offset correction via a 9-bit field in the ADC Offset Trim Register(ADCOFFTRIM). The value contained in this register will be added/subtracted before the results are available in the ADC result registers. This operation is contained in the ADC module, so timing for results will not be affected. Furthermore, since the operation is handled inside the ADC, the full dynamic range of the ADC will be maintained for any trim value.

The ADCOFFTRIM register is pre-loaded by the ADC_cal routine in the boot ROM. To further reduce the offset error in the target application, connect the signal ADCLO to one of the ADC channels and convert that channel, modifying the value in the ADCOFFTRIM, until a centered zero code is observed. See Figure 7-6 for a flow diagram.

Figure 7-6. Flow Chart of Offset Error Correction Process



Example 7-1. Negative Offset

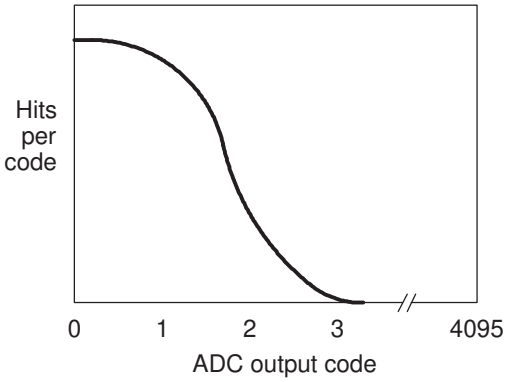
At startup, most of the reference conversions yield a zero result. After writing the value 0x28 (40 decimal) into the OFFTRIM register, all of the reference conversions give a positive result and average out to 0x19 (25 decimal). The final value written to the OFFTRIM register should be 0x0F (15 decimal).

Example 7-2. Positive Offset

At startup, all of the reference conversions yield a positive result with an average of 0x14 (20 decimal). The final value written to the OFFTRIM register should be 0x1EC (-20 decimal).

After the offset error correction process is completed, a half bell curved distribution similar to Figure 7-7 should be seen when multiple ADCLO samples are converted. The other half of the bell curve is hidden due to the fact that the converter bottoms out at a code of zero.

Figure 7-7. Ideal Code Distribution of Sampled 0-V Reference



7.3 ADC Interface

The following sub-sections describe the physical implementation of the ADC circuit. Topics that are covered include:

- Trigger Sources
- ADC Sequencer State Machine
- ADC Interrupts
- ADC to DMA Interface

7.3.1 Input Trigger Description

Each sequencer has a set of trigger inputs that can be enabled/disabled. See Table 7-4 for the valid input triggers for SEQ1, SEQ2, and cascaded SEQ.

Table 7-4. Input Triggers

SEQ1 (sequencer 1)	SEQ2 (sequencer 2)	Cascaded SEQ
Software trigger (software SOC)	Software trigger (software SOC)	Software trigger (software SOC)
ePWMx SOCA	ePWMx SOCB	ePWMx SOCA
XINT2_ADCSOC		ePWMx SOCB
		XINT2_ADCSOC

NOTE:

- An SOC trigger can initiate an autoconversion sequence whenever a sequencer is in an idle state. An idle state is either CONV00 prior to receiving a trigger, or any state which the sequencer lands on at the completion of a conversion sequence, i.e., when SEQ_CNTR has reached a count of zero.
- If an SOC trigger occurs while a current conversion sequence is underway, it sets the SOC_SEQn bit (which would have been cleared on the commencement of a previous conversion sequence) in the ADCTRL2 register. If yet another SOC trigger occurs, it is lost (i.e., when the SOC_SEQn bit is already set (SOC pending), subsequent triggers will be ignored).
- Once triggered, the sequencer cannot be stopped/halted in mid sequence. The program must either wait until an end-of-sequence (EOS) or initiate a sequencer reset, which brings the sequencer immediately back to the idle start state (CONV00 for SEQ1 and cascaded cases; CONV08 for SEQ2).
- When SEQ1/2 are used in cascaded mode, triggers going to SEQ2 are ignored, while SEQ1 triggers are active. Cascaded mode can be viewed as SEQ1 with 16 states instead of eight.

7.3.2 Autoconversion Sequencer Principle of Operation

The ADC sequencer consists of two independent 8-state sequencers (SEQ1 and SEQ2) that can also be cascaded together to form one 16-state sequencer (SEQ). The word "state" represents the number of autoconversions that can be performed with the sequencer. Block diagrams of the single (16-state, cascaded) and dual (two 8-state, separated) sequencer modes are shown in Figure 7-8 and Figure 7-9, respectively.

In both cases, the ADC has the ability to autosequence a series of conversions. This means that each time the ADC receives a start-of-conversion request, it can perform multiple conversions automatically. For every conversion, any one of the available 16 input channels can be selected through the analog MUX. After conversion, the digital value of the selected channel is stored in the appropriate result register (ADCRESULTn). (The first result is stored in ADCRESULT0, the second result in ADCRESULT1, and so on). It is also possible to sample the same channel multiple times, allowing the user to perform "over-sampling", which gives increased resolution over traditional single-sampled conversion results.

NOTE: In the sequential sampling dual-sequencer mode, a pending SOC request from either sequencer is taken up as soon as the sequence initiated by the currently active sequencer is finished. For example, assume that the A/D converter is busy catering to SEQ2 when an SOC request from SEQ1 occurs. The A/D converter will start SEQ1 immediately after completing the request in progress on SEQ2. If SOC requests are pending from both SEQ1 and SEQ2, the SOC for SEQ1 has priority. For example, assume that the A/D converter is busy catering to SEQ1. During that process, SOC requests from both SEQ1 and SEQ2 are made. When SEQ1 completes its active sequence, the SOC request for SEQ1 will be taken up immediately. The SOC request for SEQ2 will remain pending.

The ADC can also operate in simultaneous sampling mode or sequential sampling mode. For each conversion (or pair of conversions in simultaneous sampling mode), the current CONVxx bit field defines the pin (or pair of pins) to be sampled and converted. In sequential sampling mode, all four bits of CONVxx define the input pin. The MSB defines which sample-and-hold buffer the input pin is associated with, and the three LSBs define the offset. For example, if CONVxx contains the value 0101b, ADCINA5 is the selected input pin. If it contains the value 1011b, ADCINB3 is the selected input pin. In simultaneous sampling mode, the MSB of the CONVxx register is discarded. Each sample and hold buffer samples the associated pin given by the offset provided in the three LSBs of the CONVxx register. For instance, if the CONVxx register contains the value 0110b, ADCINA6 is sampled by S/H-A and ADCINB6 is sampled by S/H-B. If the value is 1001b, ADCINA1 is sampled by S/H-A and ADCINB1 is sampled by S/H-B. The voltage in S/H-A is converted first, followed by the S/H-B voltage. The result of the S/H-A conversion is placed in the current ADCRESULTn register (ADCRESULT0 for SEQ1, assuming the sequencer has been reset). The result of the S/H-B conversion is placed in the next ADCRESULTn register (ADCRESULT1 for SEQ1, assuming the sequencer has been reset). The result register pointer is then increased by two (to point to ADCRESULT2 for SEQ1, assuming the sequencer had originally been reset).

Figure 7-8. Block Diagram of Autosequenced ADC in Cascaded Mode

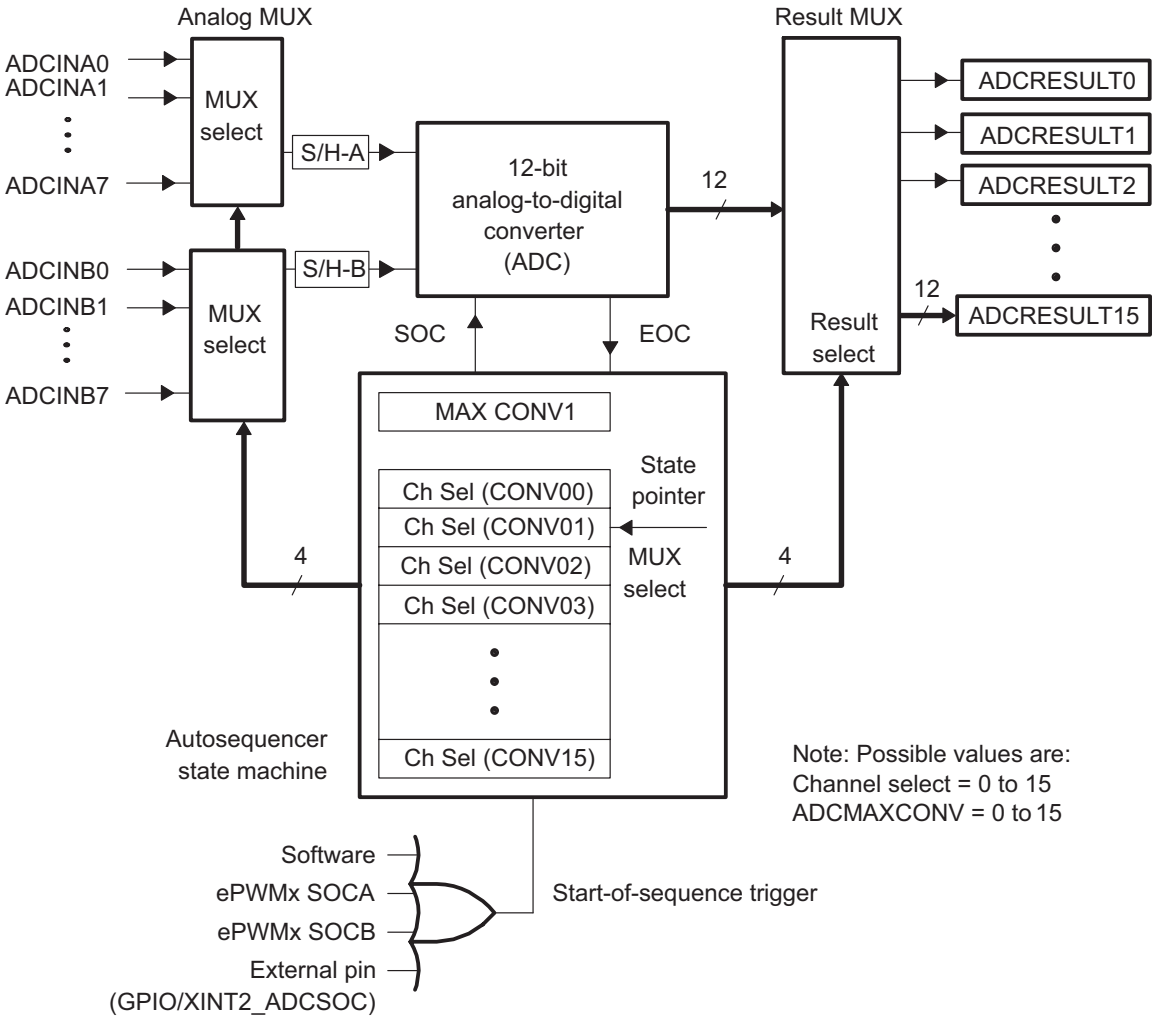
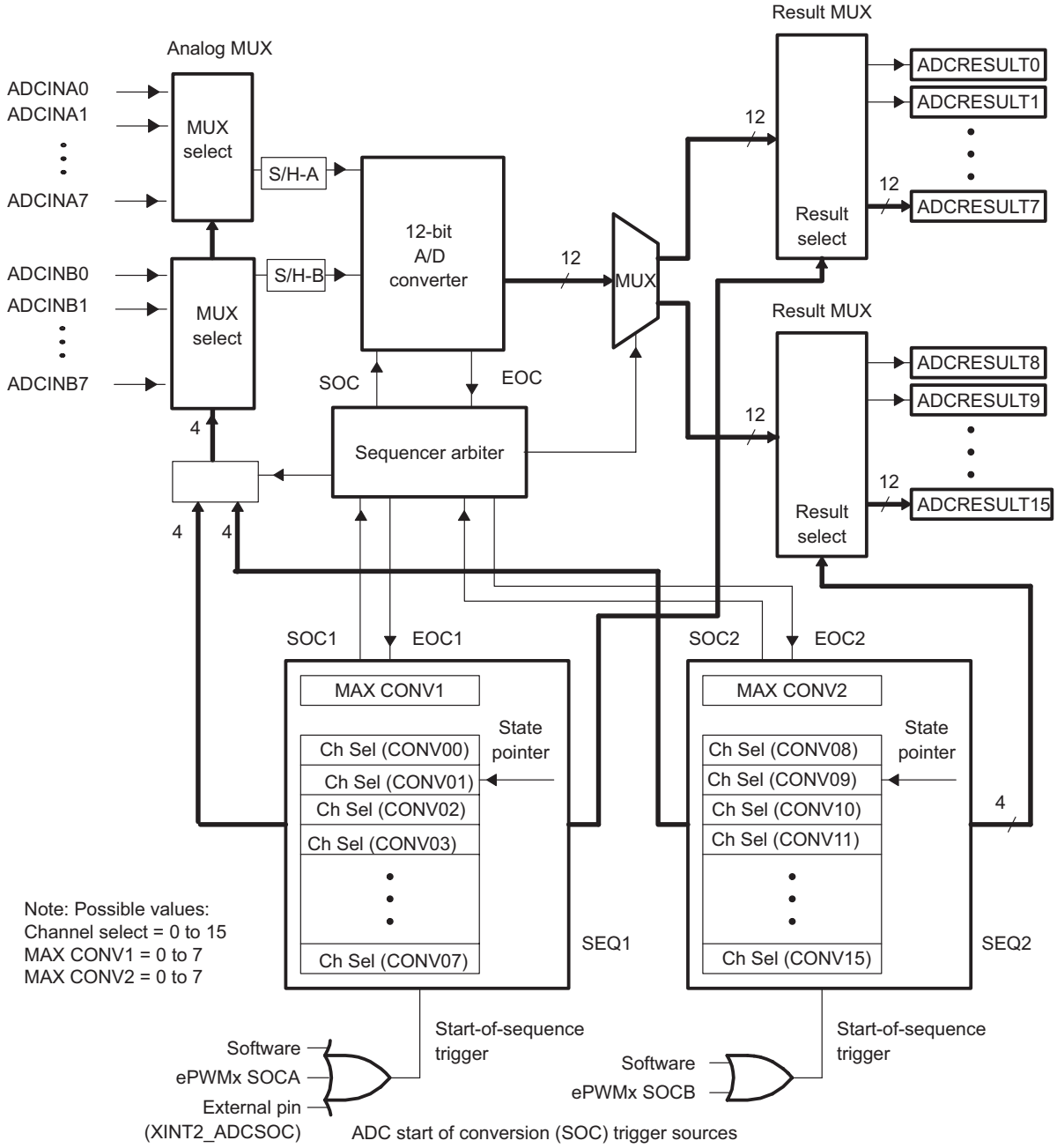


Figure 7-9. Block Diagram of Autosequenced ADC With Dual Sequencers



NOTE: There is only one ADC in the DSP. This converter is shared by the two sequencers in dual-sequencer mode.

The sequencer operation for both 8-state and 16-state modes is almost identical; the few differences are highlighted in Table 7-5.

Table 7-5. Comparison of Single and Cascaded Operating Modes

Feature	Single 8-state sequencer #1 (SEQ1)	Single 8-state sequencer #2 (SEQ2)	Cascaded 16-state sequencer (SEQ)
Start-of-conversion (SOC) triggers	ePWMx SOCA, software, external pin	ePWMx SOCB, software	ePWMx SOCA, ePWMx SOCB, software, external pin
Maximum number of autoconversions (that is, sequence length)	8	8	16
Autostop at end-of-sequence (EOS)	Yes	Yes	Yes
Arbitration priority	High	Low	Not applicable
ADC conversion result register locations	0 to 7	8 to 15	0 to 15
ADCCHSELSEQn bit field assignment	CONV00 to CONV07	CONV08 to CONV15	CONV00 to CONV15

For convenience, the sequencer states will be subsequently referred to as:

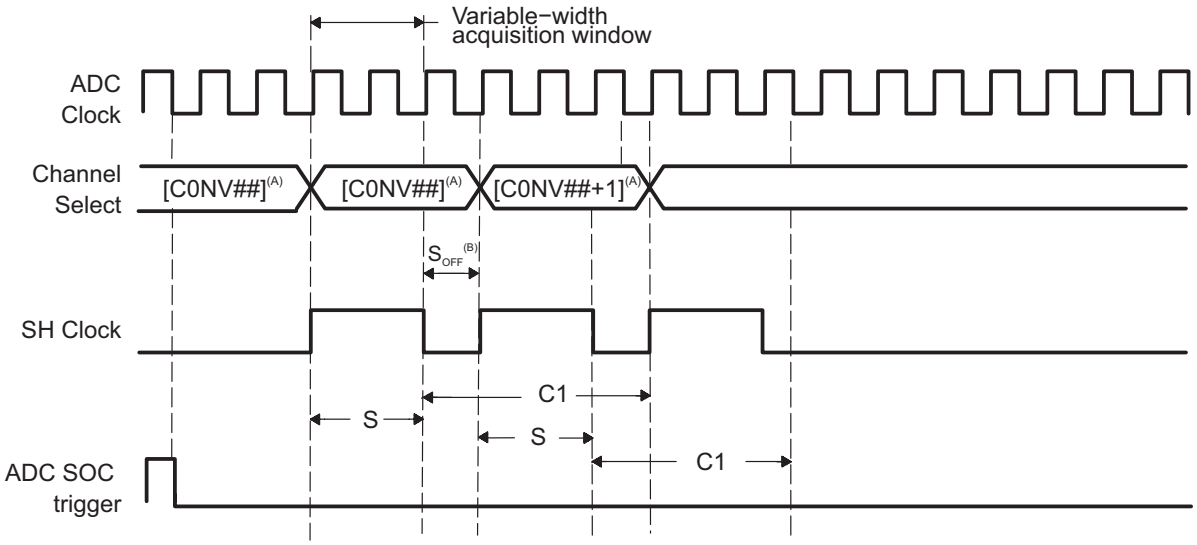
- For SEQ1: CONV00 to CONV07
- For SEQ2: CONV08 to CONV15
- For Cascaded SEQ: CONV00 to CONV15

The analog input channel selected for each sequenced conversion is defined by CONVxx bit fields in the ADC input channel select sequencing control registers (ADCCHSELSEQn). CONVxx is a 4-bit field that specifies any one of the 16 channels for conversion. Since a maximum of 16 conversions in a sequence is possible when using the sequencers in cascaded mode, 16 such 4-bit fields (CONV00 - CONV15) are available and are spread across four 16-bit registers (ADCCHSELSEQ1 - ADCCHSELSEQ4). The CONVxx bits can have any value from 0 to 15. The analog channels can be chosen in any desired order and the same channel may be selected multiple times.

7.3.2.1 Sequential Sampling Mode

Figure 7-10 shows the timing of sequential sampling mode. In this example, the ACQ_PS bits are set to 0001b.

Figure 7-10. Sequential Sampling Mode (SMODE = 0)



A ADC channel address contained in [CONV##] 4-bit register.

B S_{OFF} in sequential sampling mode is 1 ADC clock to allow for S/H circuit availability until the sampled voltage is passed to the converter.

Legend:

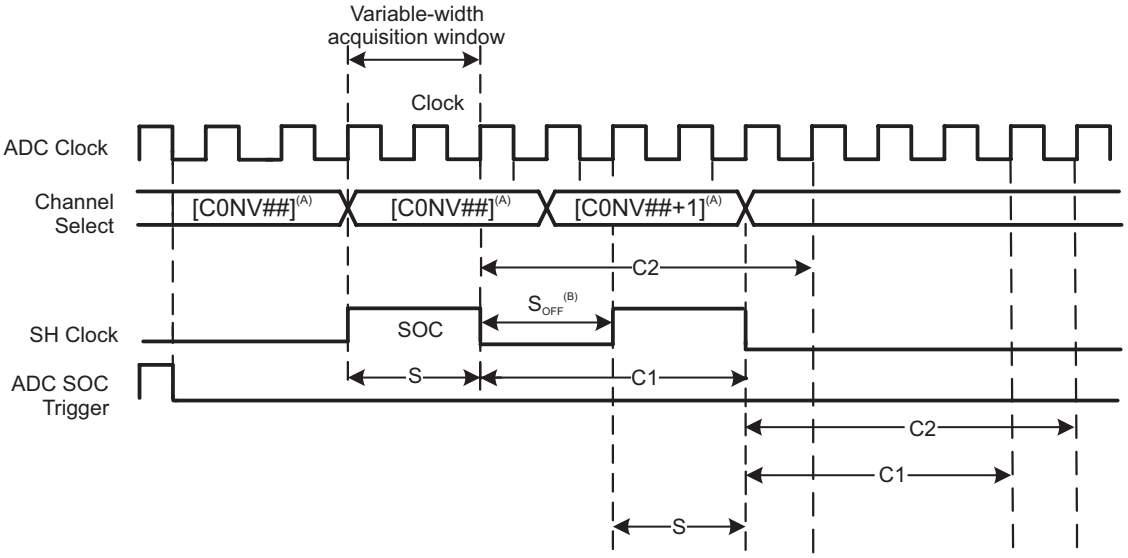
C1	Time from end of acquisition window until result register update
S	Acquisition window
S_{OFF}	Acquisition HOLD OFF

7.3.2.2 Simultaneous Sampling Mode

The ADC has the ability to sample two ADCINxx inputs simultaneously, provided that one input is from the range ADCINA0 - ADCINA7 and the other input is from the range ADCINB0 - ADCINB7. Furthermore, the two inputs must have the same sample-and-hold offset (i.e., ADCINA4 and ADCINB4, but not ADCINA7 and ADCINB6). To put the ADC into simultaneous sampling mode, the SMODE_SEL bit in the ADCTRL3 register must be set.

Figure 7-11 describes the timing of simultaneous sampling mode. In this example, the ACQ_PS bits are set to 0001b.

Figure 7-11. Simultaneous Sampling Mode (SMODE = 1)



A ADC channel address contained in [CONV##] 4-bit register; [CONV00].

B S_{OFF} in simultaneous sampling mode is 2 ADC clocks to allow for S/H circuit availability until the B channel sampled voltage is passed to the converter.

Legend:

C1	Time from end of acquisition window until Ax channel result in result register
C2	Time from end of acquisition window until Bx channel result in result register
S	Acquisition window
S_{OFF}	Acquisition HOLD OFF

Example 7-3. Simultaneous Sampling Dual Sequencer Mode Example

Example initialization:

```
AdcRegs.ADCTRL3.bit.SMODE_SEL = 0x1; // Setup simultaneous sampling mode
AdcRegs.ADCMAXCONV.all = 0x0033; // 4 double conv's each sequencer (8 total)
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; // Setup conv from channels ADCINA0 and ADCINB0
AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; // Setup conv from channels ADCINA1 and ADCINB1
AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2; // Setup conv from channels ADCINA2 and ADCINB2
AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3; // Setup conv from channels ADCINA3 and ADCINB3
AdcRegs.ADCCHSELSEQ3.bit.CONV08 = 0x4; // Setup conv from channels ADCINA4 and ADCINB4
AdcRegs.ADCCHSELSEQ3.bit.CONV09 = 0x5; // Setup conv from channels ADCINA5 and ADCINB5
AdcRegs.ADCCHSELSEQ3.bit.CONV10 = 0x6; // Setup conv from channels ADCINA6 and ADCINB6
AdcRegs.ADCCHSELSEQ3.bit.CONV11 = 0x7; // Setup conv from channels ADCINA7 and ADCINB7
```

If SEQ1 and SEQ2 were both executed, the results would go to the following RESULT registers:

```
ADCINA0 -> ADCRESULT0
ADCINB0 -> ADCRESULT1
ADCINA1 -> ADCRESULT2
ADCINB1 -> ADCRESULT3
ADCINA2 -> ADCRESULT4
ADCINB2 -> ADCRESULT5
ADCINA3 -> ADCRESULT6
ADCINB3 -> ADCRESULT7
ADCINA4 -> ADCRESULT8
ADCINB4 -> ADCRESULT9
ADCINA5 -> ADCRESULT10
ADCINB5 -> ADCRESULT11
ADCINA6 -> ADCRESULT12
ADCINB6 -> ADCRESULT13
ADCINA7 -> ADCRESULT14
ADCINB7 -> ADCRESULT15
```

Example 7-4. Simultaneous Sampling Cascaded Sequencer Mode Example

```
AdcRegs.ADCTRL3.bit.SMODE_SEL = 0x1; // Setup simultaneous sampling mode
AdcRegs.ADCTRL1.bit.SEQ_CASC = 0x1; // Setup cascaded sequencer mode
AdcRegs.ADCMAXCONV.all = 0x0007; // 8 double conv's (16 total)
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; // Setup conv from channels ADCINA0 and ADCINB0
AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; // Setup conv from channels ADCINA1 and ADCINB1
AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2; // Setup conv from channels ADCINA2 and ADCINB2
AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3; // Setup conv from channels ADCINA3 and ADCINB3
AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x4; // Setup conv from channels ADCINA4 and ADCINB4
AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0x5; // Setup conv from channels ADCINA5 and ADCINB5
AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x6; // Setup conv from channels ADCINA6 and ADCINB6
AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 0x7; // Setup conv from channels ADCINA7 and ADCINB7
```

If the cascaded SEQ was executed, the results would go to the following ADCRESULT registers:

```
ADCINA0 -> ADCRESULT0
ADCINB0 -> ADCRESULT1
ADCINA1 -> ADCRESULT2
ADCINB1 -> ADCRESULT3
ADCINA2 -> ADCRESULT4
ADCINB2 -> ADCRESULT5
ADCINA3 -> ADCRESULT6
ADCINB3 -> ADCRESULT7
ADCINA4 -> ADCRESULT8
ADCINB4 -> ADCRESULT9
ADCINA5 -> ADCRESULT10
ADCINB5 -> ADCRESULT11
ADCINA6 -> ADCRESULT12
ADCINB6 -> ADCRESULT13
ADCINA7 -> ADCRESULT14
ADCINB7 -> ADCRESULT15
```

7.3.3 ADC Sequencer State Machine

The following description applies to the 8-state sequencers (SEQ1 or SEQ2). In this mode, SEQ1/SEQ2 can autosequence up to eight conversions of any channel in a single sequencing session (16 when sequencers are cascaded together). Figure 7-12 shows the flow diagram. The result of each conversion is stored in one of the eight result registers (ADCRESULT0 - ADCRESULT7 for SEQ1 and ADCRESULT8 - ADCRESULT15 for SEQ2). These registers are filled from the lowest address to the highest address.

The number of conversions in a sequence is controlled by MAX_CONVn (a 3-bit or 4-bit field in the ADCMAXCONV register), which is automatically loaded into the sequencing counter status bits (SEQ_CNTR[3:0]) in the autosequence status register (ADCASEQSR) at the start of an autosequenced conversion session. The MAX_CONVn field can have a value ranging from zero to seven (zero to fifteen when sequencers are cascaded together). SEQ_CNTR bits count down from their loaded value as the sequencer starts from state CONV00 and continues sequentially (CONV01, CONV02, and so on) until SEQ_CNTR has reached zero. The number of conversions completed during an autosequencing session is equal to (MAX_CONVn + 1).

Example 7-5. Conversion in Dual-Sequencer Mode Using SEQ1

Suppose seven conversions are desired from SEQ1 (i.e., inputs ADCINA2 and ADCINA3 twice, then ADCINA6, ADCINA7, and ADCINB4 must be converted as part of the autosequenced session), then MAX_CONV1 should be set to 6 and the ADCCHSELSEQn registers should be set to the values shown in Table 7-6.

Conversion begins once the start-of-conversion (SOC) trigger is received by the sequencer. The SOC trigger also loads the SEQ_CNTR bits. Those channels that are specified in the ADCCHSELSEQn registers are taken up for conversion, in the predetermined sequence. The SEQ_CNTR bits are decremented by one automatically after every conversion. Once SEQ_CNTR reaches zero, two things can happen, depending on the status of the continuous run bit (CONT_RUN) in the ADCTRL1 register. See Figure 7-12 for an illustration of the flow.

- If CONT_RUN is set, the conversion sequence starts all over again automatically (i.e., SEQ_CNTR gets reloaded with the original value in MAX_CONV1 and SEQ1 state is set to CONV00 [See Section 7.3.3.2 for more options]). In this case, to avoid overwriting the data, you must be sure that the result registers are read before the next conversion sequence begins. The arbitration logic designed into the ADC ensures that the result registers are not corrupted should a contention arise (ADC module trying to write into the result registers while you try to read from them at the same time).
- If CONT_RUN is not set, the sequencer stays in the last state (CONV06, in this example) and SEQ_CNTR continues to hold a value of zero. To repeat the sequence on the next SOC, the sequencer must be reset using the RST_SEQn bit prior to the next SOC.

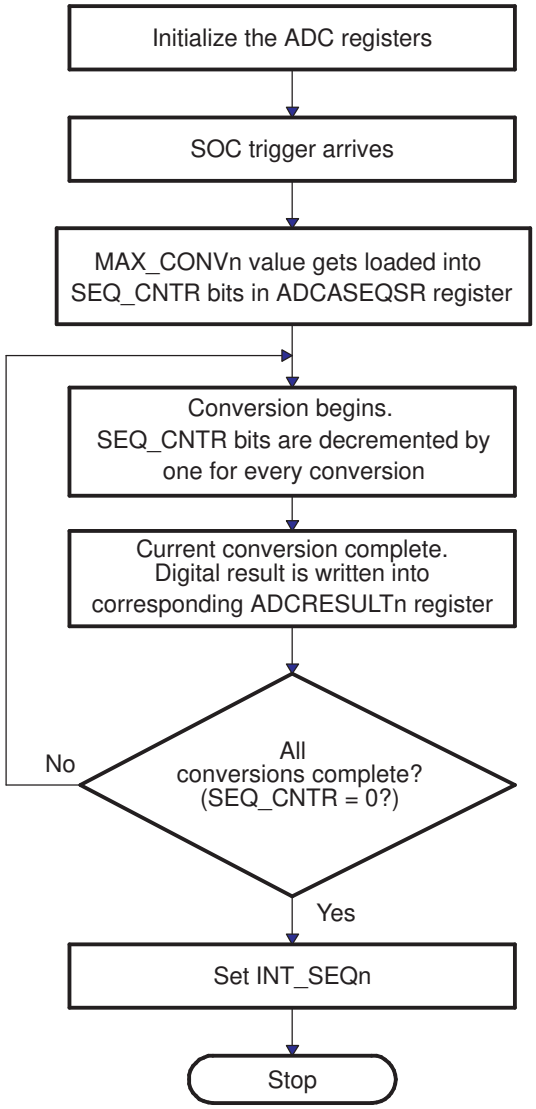
If the interrupt flag is set every time SEQ_CNTR reaches zero (INT_ENA_SEQn = 1 and INT_MOD_SEQn = 0), you can (if needed) manually reset the sequencer (using the RST_SEQn bit in the ADCTRL2 register) in the interrupt service routine (ISR). This causes the SEQn state to be reset to its original value (CONV00 for SEQ1 and CONV08 for SEQ2). This feature is useful in the Start/Stop operation of the sequencer. Example 7-5 also applies to SEQ2 and the cascaded 16-state sequencer (SEQ) with differences outlined in Table 7-5.

Table 7-6. Values for ADCCHSELSEQn Registers (MAX_CONV1 Set to 6)

	Bits 15-12[1]	Bits 11-8[1]	Bits 7-4[1]	Bits 3-0[1]	
7103h	3	2	3	2	ADCCHSELSEQ1
7104h	x	12	7	6	ADCCHSELSEQ2
7105h	x	x	x	x	ADCCHSELSEQ3
7106h	x	x	x	x	ADCCHSELSEQ4

(1) Values are in decimal, and x = don't care

Figure 7-12. Flow Chart for Uninterrupted Autosequenced Mode



NOTE: The flow chart corresponds to CONT_RUN bit = 0 and INT_MOD_SEQn bit = 0.

7.3.3.1 Sequencer Start/Stop Mode (Sequencer Start/Stop Operation With Multiple Time-Sequenced Triggers)

In addition to the uninterrupted autosequenced mode, any sequencer (SEQ1, SEQ2, or SEQ) can be operated in a Stop/Start mode which is synchronized to multiple start-of-conversion (SOC) triggers, separated in time. This mode is similar to Example 7-5, but the sequencer is allowed to be retriggered without being reset to the initial state CONV00, once it has completed its first sequence (i.e., the sequencer is not reset in the interrupt service routine). Therefore, when one conversion sequence ends, the sequencer stays in the current conversion state. The continuous run bit (CONT_RUN) in the ADCTRL1 register must be set to zero (that is, disabled) for this mode.

Example 7-6. Sequencer Start/Stop Operation

Requirement: To start three autoconversions (e.g., I_1,I_2,I_3) off trigger 1 (underflow) and three autoconversions (for example, V_1,V_2,V_3) off trigger 2 (period). Triggers 1 and 2 are separated in time by 25 µs and are provided by an ePWM. See Figure 7-13. Only SEQ1 is used in this case.

NOTE: Triggers 1 and 2 may be an SOC signal from ePWM, external pin, or software. The same trigger source may occur twice to satisfy the dual-trigger requirement of this example. Care must be taken such that multiple ePWM triggers are not lost due sequences already in progress. See Section 7.3.1.

Here MAX_CONV1 is set to 2 and the ADC Input Channel Select Sequencing Control Registers (ADCCHSELSEQn) are set as shown in Table 7-7.

Once reset and initialized, SEQ1 waits for a trigger. With the first trigger, three conversions with channel-select values of: CONV00 (I_1), CONV01 (I_2), and CONV02 (I_3) are performed. SEQ1 then waits at current state for another trigger. Twenty-five microseconds later when the second trigger arrives, another three conversions occur, with channel-select values of CONV03 (V_1), CONV04 (V_2), and CONV05 (V_3).

The value of MAX_CONV1 is automatically loaded into SEQ_CNTR for both trigger cases. If a different number of conversions are required at the second trigger point, you must (at some appropriate time before the second trigger) change the value of MAX_CONV1 through software, otherwise, the current (originally loaded) value will be reused. This can be done by an ISR that changes the value of MAX_CONV1 at the appropriate time. The interrupt operation modes are described in Section 7.3.4.

At the end of the second autoconversion session, the ADC result registers will have the values shown in Table 7-8.

At this point, SEQ1 keeps "waiting" at the current state for another trigger. Now, the user can reset SEQ1 (by software) to state CONV00 and repeat the same trigger1, 2 sessions.

Figure 7-13. Example of ePWM Triggers to Start the Sequencer

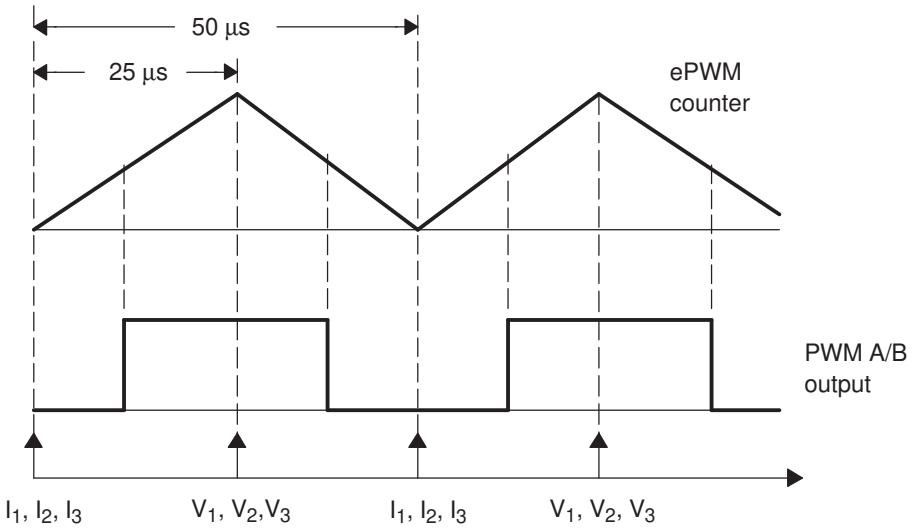


Table 7-7. Values for ADCCHSELSEQn (MAX_CONV1 set to 2)

	Bits 15-12	Bits 11-8	Bits 7-4	Bits 3-0	
7103h	V_1	I_3	I_2	I_1	ADCCHSELSEQ1
7104h	x	x	V_3	V_2	ADCCHSELSEQ2
7105h	x	x	x	x	ADCCHSELSEQ3
7106h	x	x	x	x	ADCCHSELSEQ4

Table 7-8. Values After Second Autoconversion Session

Buffer Register	ADC Conversion Result Buffer
ADCRESULT0	I_1
ADCRESULT1	I_2
ADCRESULT2	I_3
ADCRESULT3	V_1
ADCRESULT4	V_2
ADCRESULT5	V_3
ADCRESULT6	x
ADCRESULT7	x
ADCRESULT8	x
ADCRESULT9	x
ADCRESULT10	x
ADCRESULT11	x
ADCRESULT12	x
ADCRESULT13	x
ADCRESULT14	x
ADCRESULT15	x

7.3.3.2 Sequencer Override Feature

In normal operation, sequencers SEQ1, SEQ2 or cascaded SEQ1 help to convert selected ADC channels and store them in the respective ADCRESULTn registers, sequentially. The sequence naturally wraps around at the end of the MAX_CONVn setting. With the sequencer override feature, the natural wraparound of the sequencers can be controlled in software. The sequencer override feature is controlled by bit 5 of the ADC Control Register 1 (ADCCTRL1).

For example, assume the SEQ_OVRD bit is 0 and the ADC is in cascaded-sequencer, continuous-conversion mode with MAX_CONV1 set to 7. Normally, the sequencer would increment sequentially and update up to ADCRESULT7 register with ADC conversions and wraps around to 0. At the end of the ADCRESULT7 register update, the relevant interrupt flag would be set.

With the SEQ_OVRD bit set to 1, the sequencer updates seven result registers and does *not* wrap around to 0. Instead, the sequencer will increment sequentially and update the ADCRESULT8 register onwards until the ADCRESULT15 register is reached. After updating ADCRESULT15 register, the natural wrap around to 0 will occur. This feature treats the result registers (0-15) like a FIFO for sequential data capture from the ADC. This feature is very helpful to capture ADC data when ADC conversions are done at the maximum data rate.

Recommendations and caution on sequencer override feature:

- After reset, SEQ_OVRD bit will be 0; therefore the sequencer override feature remains disabled.
- When SEQ _OVRD bit is set for all nonzero values of MAX_CONVn, the related interrupt flag bit will be set for every MAX_CONVn count of result register update.
- For example, if ADCMAXCONV is set to 3, then the interrupt flag for the selected sequencer will be set

every four result register updates. The wraparound always occurs at the end of the sequencer (i.e., after ADCRESULT15 register update in cascaded sequencer mode).

- This will be functional in conversions using SEQ1, SEQ2, and cascaded sequencers using SEQ1.
- It is recommended that this feature not be enabled/controlled dynamically within the program. Always enable this feature during the ADC module initialization.
- In continuous-conversion mode with sequencer changes, the ADC channel address uses the preset values in CONVxx registers. If continuous conversions of the same channel are needed then all the CONVxx registers should have the same channel address.
- In continuous-conversion mode, if a sequencer reset is needed: set CONT_RUN bit to 0, wait 2 cycles in the ADC Clock domain, then reset the sequencer. CONT_RUN can then be set back to 1.
- For example, to get 16 contiguous samples for the ADCINA0 channel using the sequencer override feature, all 16 CONVxx registers should be set to 0x0000.

7.3.4 Interrupt Operation During Sequenced Conversions

There are two ADC interrupt signals that correspond to each sequencer, SEQ1 INT and SEQ2 INT. Each interrupt can be selectively enabled/disabled by its corresponding INT_ENA_SEQ# bit in the ADCTRL2 register. The status(active = 1) of each interrupt is presented in the ADCST register as INT_SEQ2 and INT_SEQ1 bits respectively.

Once an interrupt is latched in either INT_SEQ1 or INT_SEQ2, it must be cleared in order for another interrupt to be generated to the C28x CPU/PIE module. The interrupt clear bit is also located in ADCST as INT_SEQ1_CLR and INT_SEQ2_CLR. Finally, cascaded sequencer mode only uses SEQ1 INT, as this mode treats SEQ1 as the lone 16-state sequencer for the ADC.

The sequencer can generate interrupts under two operating modes. These modes are determined by the Interrupt-Mode-Enable control bits in ADCTRL2.

A variation of Example 7-6 can be used to show how interrupt mode 1 and mode 2 are useful under different operating conditions.

Case 1: Number of samples in the first and second sequences are not equal

- Mode 1 Interrupt operation (that is, Interrupt request occurs at every EOS)
  1. Sequencer is initialized with MAX_CONVn = 1 for converting I_1 and I_2
  2. At ISR "a", MAX_CONVn is changed to 2 (by software) for converting V_1, V_2, and V_3
  3. At ISR "b", the following events take place :
     a. MAX_CONVn is changed to 1 again for converting I_1 and I_2.
     b. Values I_1, I_2, V_1, V_2, and V_3 are read from ADC result registers.
     c. The sequencer is reset.
  4. Steps 2 and 3 are repeated. Note that the interrupt flag is set every time SEQ_CNTR reaches zero and both interrupts are recognized.

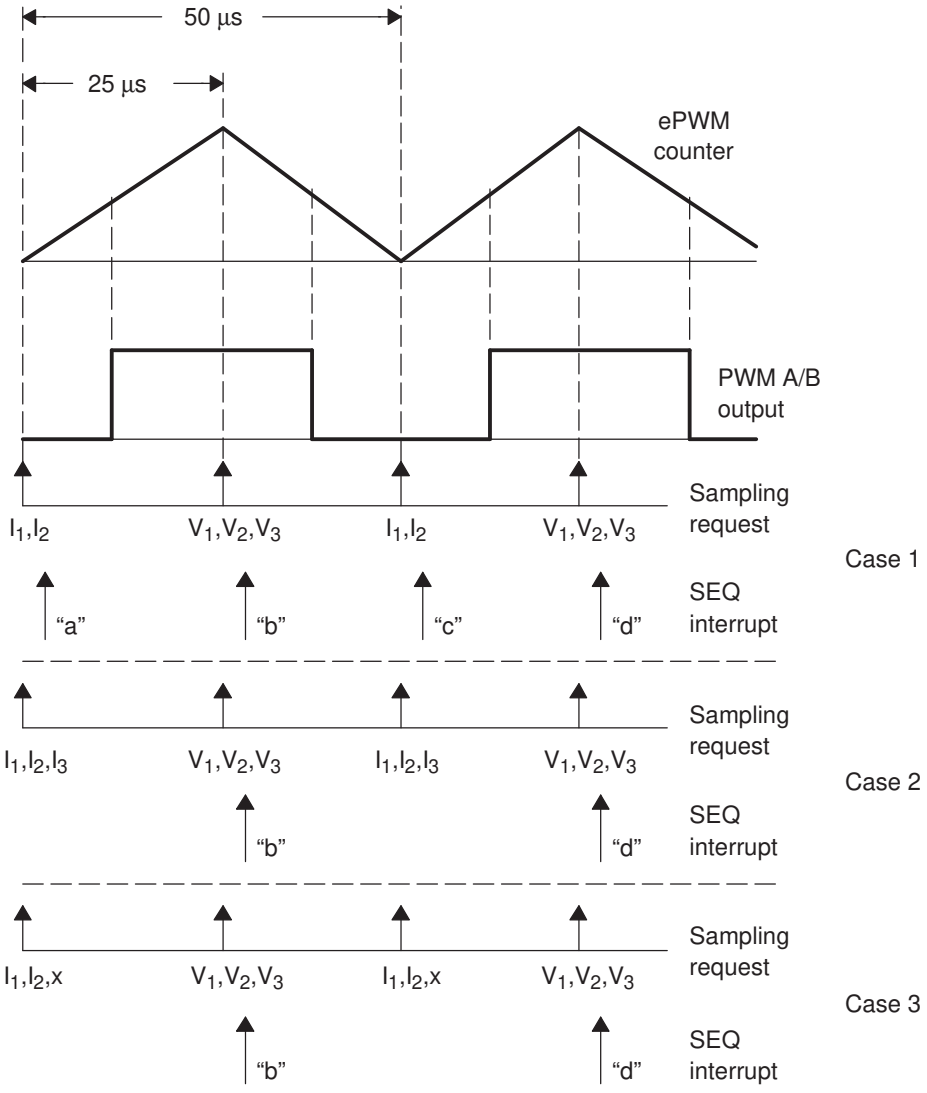
Case 2: Number of samples in the first and second sequences are equal

- Mode 2 Interrupt operation (that is, Interrupt request occurs at every other EOS)
  1. Sequencer is initialized with MAX_CONVn = 2 for converting I_1, I_2, and I_3 (or V_1, V_2, and V_3).
  2. At ISR "b" and "d", the following events take place :
     a. Values I_1, I_2, I_3,V_1, V_2, and V_3 are read from ADC result registers.
     b. The sequencer is reset.
  3. Step 2 is repeated.

Case 3: Number of samples in the first and second sequences are equal (with dummy read)

- Mode 2 Interrupt operation (that is, Interrupt request occurs at every other EOS)
  1. Sequencer is initialized with MAX_CONVn = 2 for I_1, I_2, and x(dummy sample).
  2. At ISR "b" and "d", the following events take place :
     a. Values I_1, I_2, x,V_1, V_2, and V_3 are read from ADC result registers.
     b. The sequencer is reset.
  3. Step 2 is repeated. Note that the third I-sample (x) is a dummy sample, and is not really required. However, to minimize ISR overhead and CPU intervention, advantage is taken of the "every other" Interrupt request feature of Mode 2.

Figure 7-14. Interrupt Operation During Sequenced Conversions



7.3.5 ADC to DMA Interface

The ADC result registers located in peripheral frame 0 (0x0B00 – 0x0B0F) are accessible by the DMA unit on the F2833x. These registers can also be accessed by the CPU at the same time as the DMA without bus contention. The result registers in peripheral frame 2 (0x7108 – 0x710F) are not accessible by the DMA.

There is a sync signal provided automatically by the ADC to the DMA for a sequencer 1 conversion when both SEQ_OVRD and CONT_RUN bits are set. The sync pulse will be generated by the ADC after the first MAXCONV limit is reached for each pass through the sequencer. When the sequencer 1 is in this configuration it is possible that the DMA could become misaligned to the currently populated result registers, depending on the loading of the other DMA channels. If a misalignment occurs, the DMA can use the sync signal to detect and flag a sync error event.

For more information on how the sync signal is used locally in the DMA, please see the TMS320x2833x, 2823x Direct Memory Access (DMA) Module Reference Guide.

7.4 ADC Registers

Table 7-9 lists the memory-mapped registers for the ADC. All register offset addresses not listed in Table 7-9 should be considered as reserved locations and the register contents should not be modified.

The starting address is 7100h.

Table 7-9. ADC Registers

Offset	Offset[1]	Acronym	Register Name	Section
0h		ADCTRL1	ADC Control Register 1	Section 7.4.1
1h		ADCTRL2	ADC Control Register 2	Section 7.4.2
2h		ADCMAXCONV	ADC Maximum Conversion Channels Register	Section 7.4.3
3h		ADCCHSELSEQ1	ADC Channel Select Sequencing Control Register 1	Section 7.4.4
4h		ADCCHSELSEQ2	ADC Channel Select Sequencing Control Register 2	Section 7.4.5
5h		ADCCHSELSEQ3	ADC Channel Select Sequencing Control Register 3	Section 7.4.6
6h		ADCCHSELSEQ4	ADC Channel Select Sequencing Control Register 4	Section 7.4.7
7h		ADCASEQSR	ADC Auto-Sequence Status Register	Section 7.4.8
8h to 17h	0h to 9h	ADCRESULT_0 to ADCRESULT_15	ADC Conversion Result Register N	Section 7.4.9
18h		ADCTRL3	ADC Control Register 3	Section 7.4.10
19h		ADCST	ADC Status Register	Section 7.4.11
1Ch		ADCREFSEL	ADC Reference Select Register	Section 7.4.12
1Dh		ADCOFFTRIM	ADC Offset Trim Register	Section 7.4.13

(1) The ADC result registers are dual mapped in the device. Locations in Peripheral Frame 2 (0x7108-0x7117) are 2 wait states and left justified. Locations in Peripheral Frame 0 space (0x0B00-0x0B0F) are 0 wait states and right justified. During high speed/continuous conversion use of the ADC, use the 0 wait state locations to avoid missing ADC conversions.

7.4.1 ADCTRL1 Register (Offset = 0h) [reset = 0h]

ADCTRL1 is shown in Figure 7-15 and described in Table 7-10.

Figure 7-15. ADCTRL1 Register

15	14	13	12	11	10	9	8
RESERVED	RESET	SUSMOD[1:0]		ACQ_PS[3:0]			
R-0h	R/W-0h	R/W-0h		R/W-0h			
7	6	5	4	3	2	1	0
CPS	CONT_RUN	SEQ_OVRD	SEQ_CASC	RESERVED			
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h			

Table 7-10. ADCTRL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reads return a zero. Writes have no effect.
14	RESET	R/W	0h	ADC module software reset. This bit causes a master reset on the entire ADC module. All register bits and sequencer 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 ADC clock cycles after the instruction that resets the ADC. **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 ADC clock domain cycles, you can then write the appropriate values to the ADCCTRL1 register bits. The example below assumes 150-MHz DSP Clock and 25-MHz ADCCLK. Assembly code: MOV ADCTRL1, #00xxxxxxxxxxxxxxb Configures ADCTRL1 to user-desired value. RPT #10|| NOP; Provides the required delay between writes to ADCTRL1 MOV ADCTRL1, #01xxxxxxxxxxxxxxb Resets the ADC (RESET = 1) Note that the second MOV is not required if the default configuration is sufficient. 0h = No effect 1h = Resets entire ADC module (bit is then set back to 0 by ADC logic)
13-12	SUSMOD[1:0]	R/W	0h	Emulation-suspend mode. These bits determine what occurs when an emulation-suspend occurs (due to the debugger hitting a breakpoint, for example). 00b = Mode 0. Emulation suspend is ignored. 01b = Mode 1. Sequencer and other wrapper logic stops after current **sequence** is complete, final result is latched, and state machine is updated. 10b = Mode 2. Sequencer and other wrapper logic stops after current **conversion** is complete, result is latched, and state machine is updated. 11b = Mode 3. Sequencer and other wrapper logic stops immediately on emulation suspend.
11-8	ACQ_PS[3:0]	R/W	0h	Acquisition window size. This bit field controls the width of SOC pulse, which, in turn, determines for what time duration the sampling switch is closed. The width of SOC pulse is ADCTRL1[11:8] + 1 times the ADCLK period.
7	CPS	R/W	0h	Core clock prescaler. The prescaler is applied to divided device peripheral clock, HSPCLK. **Note:** F_{clk} = Prescaled HSPCLK (ADCCLKPS[3:0]) 0h = ADCCLK = F_{clk}/1 1h = ADCCLK = F_{clk}/2

Table 7-10. ADCTRL1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
6	CONT_RUN	R/W	0h	Continuous run. This bit determines whether the sequencer operates in continuous conversion mode or start-stop mode. This bit can be written while a current conversion sequence is active. This bit will take effect at the end of the current conversion sequence i.e., software can set/clear this bit until EOS has occurred, for valid action to be taken. In the continuous conversion mode, there is no need to reset the sequencer however, the sequencer must be reset in the start-stop mode to put the converter in state CONV00. 0h = Start-stop mode. Sequencer stops after reaching EOS. On the next SOC, the sequencer starts from the state where it ended unless a sequencer reset is performed. 1h = Continuous conversion mode. After reaching EOS, the behavior of the sequencer depends on the state of the SEQ_OVRD bit. If this bit is cleared, the sequencer starts over again from its reset state (CONV00 for SEQ1 and cascaded, CONV08 for SEQ2). If SEQ_OVRD is set, the sequencer starts again from its current position, without resetting.
5	SEQ_OVRD	R/W	0h	Sequencer override. Provides additional sequencer flexibility in continuous run mode by overriding the wrapping around at the end of conversions set by MAX_CONVn. 0h = Disabled - Allows the sequencer to wrap around at the end of conversions set by MAX_CONVn. 1h = Enabled - Overrides the sequencer from wrapping around at the end of conversions set by MAX_CONVn. Wraparound occurs only at the end of the sequencer.
4	SEQ_CASC	R/W	0h	Cascaded sequencer operation. This bit determines whether SEQ1 and SEQ2 operate as two 8-state sequencers or as a single 16-state sequencer (SEQ). 0h = Dual-sequencer mode. SEQ1 and SEQ2 operate as two 8-state sequencers. 1h = Cascaded mode. SEQ1 and SEQ2 operate as a single 16-state sequencer (SEQ).
3-0	RESERVED	R	0h	Reads return zero. Writes have no effect.

7.4.2 ADCTRL2 Register (Offset = 1h) [reset = 0h]

ADCTRL2 is shown in Figure 7-16 and described in Table 7-11.

Figure 7-16. ADCTRL2 Register

15	14	13	12	11	10	9	8
ePWM_SOCB_SEQ	RST_SEQ1	SOC_SEQ1	RESERVED	INT_ENA_SEQ1	INT_MOD_SEQ1	RESERVED	ePWM_SOCA_SEQ1
R/W-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R-0h	R/W-0h
7	6	5	4	3	2	1	0
EXT_SOC_SEQ1	RST_SEQ2	SOC_SEQ2	RESERVED	INT_ENA_SEQ2	INT_MOD_SEQ2	RESERVED	ePWM_SOCB_SEQ2
R/W-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R-0h	R/W-0h

Table 7-11. ADCTRL2 Register Field Descriptions

Bit	Field	Type	Reset	Description
15	ePWM_SOCB_SEQ	R/W	0h	ePWM SOCB enable for cascaded sequencer *(Note: This bit is active only in cascaded mode.)* 0h = No action 1h = Setting this bit allows the cascaded sequencer to be started by an ePWM SOCB signal. The ePWM modules can be programmed to start a conversion on various events. See the *TMS320x28xx , 28xxx Enhanced Pulse Width Modulation Module Reference Guide* (literature number SPRU791) for more information on the ePWM modules.
14	RST_SEQ1	R/W	0h	Reset sequencer1 Writing a 1 to this bit resets SEQ1 or the cascaded sequencer immediately to an initial "pretriggered" state, i.e., waiting for a trigger at CONV00. A currently active conversion sequence will be aborted. 0h = No action 1h = Immediately reset sequencer to state CONV00

Table 7-11. ADCTRL2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
13	SOC_SEQ1	R/W	0h	Start-of-conversion (SOC) trigger for Sequencer 1 (SEQ1) or the cascaded sequencer. This bit can be set by the following triggers: • S/W - Software writing a 1 to this bit • ePWM SOCA • ePWM SOCB (only in cascaded mode) • EXT - External pin (i.e., GPIO Port A pin (GPIO31-0) configured as XINT2 in the GPIOxINT2SEL register. See the *TMS320x2833x System Control and Interrupts Reference Guide* (literature number SPRUFB0) for details on how to configure a GPIO pin as XINT2. When a trigger occurs, there are three possibilities: **Case 1:** SEQ1 idle and SOC bit clear SEQ1 starts immediately (under arbiter control). This bit is set and cleared, allowing for any "pending" trigger requests. **Case 2:** SEQ1 busy and SOC bit clear Bit is set signifying a trigger request is pending. When SEQ1 finally starts after completing current conversion, this bit is cleared. **Case 3:** SEQ1 busy and SOC bit set Any trigger occurring in this case is ignored (lost). 0h = Clears a pending SOC trigger. Note: If the sequencer has already started, this bit is automatically cleared, and hence, writing a zero has no effect; i.e., an already started sequencer cannot be stopped by clearing this bit. 1h = Software trigger - Start SEQ1 from currently stopped position (i.e., Idle mode) **Note:** The RST_SEQ1 (ADCTRL2.14) and the SOC_SEQ1 (ADCTRL2.13) bits should not be set in the same instruction. This resets the sequencer, but does not start the sequence. The correct sequence of operation is to set the RST_SEQ1 bit first, and the SOC_SEQ1 bit in the following instruction. This makes certain that the sequencer is reset and a new sequence started. This sequence applies to the RST_SEQ2 (ADCTRL2.6) and SOC_SEQ2 (ADCTRL2.5) bits also.
12	RESERVED	R	0h	Reads return a zero. Writes have no effect.
11	INT_ENA_SEQ1	R/W	0h	SEQ1 interrupt enable. This bit enables the interrupt request to CPU by INT SEQ1. 0h = Interrupt request by INT_SEQ1 is disabled. 1h = Interrupt request by INT_SEQ1 is enabled.
10	INT_MOD_SEQ1	R/W	0h	SEQ1 interrupt mode. This bit selects SEQ1 interrupt mode. It affects the setting of INT SEQ1 at the end of the SEQ1 conversion sequence. 0h = INT_SEQ1 is set at the end of every SEQ1 sequence. 1h = INT_SEQ1 is set at the end of every other SEQ1 sequence.
9	RESERVED	R	0h	Reads return a zero. Writes have no effect.
8	ePWM_SOCA_SEQ1	R/W	0h	ePWM SOCA enable bit for SEQ1 0h = SEQ1 cannot be started by ePWMx SOCA trigger. 1h = Allows SEQ1/SEQ to be started by ePWMx SOCA trigger. The ePWMs can be programmed to start a conversion on various events.
7	EXT_SOC_SEQ1	R/W	0h	External signal start-of-conversion bit for SEQ1 0h = No action 1h = Setting this bit enables an ADC autoconversion sequence to be started by a signal from a GPIO Port A pin (GPIO31-0) configured as XINT2 in the GPIOXINT2SEL register. See the *TMS320x2833x System Control and Interrupts Reference Guide* (literature number SPRUFB0)
6	RST_SEQ2	R/W	0h	Reset SEQ2 0h = No action 1h = Immediately resets SEQ2 to an initial "pretriggered" state, i.e., waiting for a trigger at CONV08. A currently active conversion sequence will be aborted.

Table 7-11. ADCTRL2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
5	SOC_SEQ2	R/W	0h	Start of conversion trigger for sequencer 2 (SEQ2). *(Only applicable in dual-sequencer mode; ignored in cascaded mode.)* This bit can be set by the following triggers: • S/W - Software writing of 1 to this bit • ePWM SOCB When a trigger occurs, there are three possibilities: **Case 1:** SEQ2 idle and SOC bit clear SEQ2 starts immediately (under arbiter control) and the bit is cleared, allowing for any pending trigger requests. **Case 2:** SEQ2 busy and SOC bit clear Bit is set signifying a trigger request is pending. When SEQ2 finally starts after completing current conversion, this bit will be cleared. **Case 3:** SEQ2 busy and SOC bit set Any trigger occurring in this case will be ignored (lost). will be ignored (lost). 0h = Clears a Pending SOC trigger **Note:** If the sequencer has already started, this bit is automatically cleared, and writing a zero has no effect; i.e., an already started sequencer cannot be stopped by clearing this bit. 1h = Starts SEQ2 from currently stopped position (i.e., Idle mode)
4	RESERVED	R	0h	Reads return a zero. Writes have no effect.
3	INT_ENA_SEQ2	R/W	0h	SEQ2 interrupt enable. This bit enables or disables an interrupt request to the CPU by INT SEQ2. 0h = Interrupt request by INT_SEQ2 is disabled. 1h = Interrupt request by INT_SEQ2 is enabled.
2	INT_MOD_SEQ2	R/W	0h	SEQ2 interrupt mode. This bit selects SEQ2 interrupt mode. It affects the setting of INT SEQ2 at the end of the SEQ2 conversion sequence. 0h = INT_SEQ2 is set at the end of every SEQ2 sequence. 1h = INT_SEQ2 is set at the end of every other SEQ2 sequence.
1	RESERVED	R	0h	Reads return a zero. Writes have no effect.
0	ePWM_SOCB_SEQ2	R/W	0h	ePWM SOCB enable bit for SEQ2. 0h = SEQ2 cannot be started by ePWMx SOCB trigger. 1h = Allows SEQ2 to be started by ePWMx SOCB trigger. The ePWMs can be programmed to start a conversion on various events.

7.4.3 ADCMAXCONV Register (Offset = 2h) [reset = 0h]

ADCMAXCONV is shown in Figure 7-17 and described in Table 7-12.

Figure 7-17. ADCMAXCONV Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	MAX_CONV2			MAX_CONV1			
R-0h	R/W-0h			R/W-0h			

Table 7-12. ADCMAXCONV Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reads return a zero. Writes have no effect.
6-4	MAX_CONV2	R/W	0h	MAX_CONV2 bit field defines the maximum number of conversions executed in an autoconversion session for sequencer 2 (SEQ2)
3-0	MAX_CONV1	R/W	0h	MAX_CONV1 bit field defines the maximum number of conversions executed in an autoconversion session for sequencer 1 (SEQ1). The bit fields and their operation vary according to the sequencer modes (dual/cascaded). For SEQ1 operation, bits MAX_CONV1[2:0] are used. For SEQ operation, bits MAX_CONV1[3:0] are used. An autoconversion session always starts with the initial state and continues sequentially until the end state if allowed. The result buffer is filled in a sequential order. Any number of conversions between 1 and (MAX_CONVn +1) can be programmed for a session.

ADCMAXCONV Register Bit Programming

If only five conversions are required, then MAX_CONVn is set to four.

Case 1: Dual mode SEQ1 and cascaded mode Sequencer goes from CONV00 to CONV04, and the five conversion results are stored in the registers Result 00 to Result 04 of the Conversion Result Buffer.

Case 2: Dual mode SEQ2 Sequencer goes from CONV08 to CONV12, and the five conversion results are stored in the registers Result 08 to Result 12 of the Conversion Result Buffer.

MAX_CONV1 Value >7 for Dual-Sequencer Mode

If a value for MAX_CONV1, which is greater than 7, is chosen for the dual-sequencer mode (i.e., two separate 8-state sequencers), then SEQ_CNTR will continue counting past seven, causing the sequencer to wrap around to CONV00 and continue counting.

Table 7-13. Bit Selections for MAX_CONV1 for Various Number of Conversions

ADCMAXCONV[3-0]	Number of Conversions
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12

Table 7-13. Bit Selections for MAX_CONV1 for Various Number of Conversions (continued)

ADCMAXCONV[3-0]	Number of Conversions
1100	13
1101	14
1110	15
1111	16

7.4.4 ADCCHSELSEQ1 Register (Offset = 3h) [reset = 0h]

ADCCHSELSEQ1 is shown in Figure 7-18 and described in Table 7-14.

Figure 7-18. ADCCHSELSEQ1 Register

15	14	13	12	11	10	9	8
CONV03				CONV02			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
CONV01				CONV00			
R/W-0h				R/W-0h			

Table 7-14. ADCCHSELSEQ1 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	CONV03	R/W	0h	
11-8	CONV02	R/W	0h	
7-4	CONV01	R/W	0h	
3-0	CONV00	R/W	0h	

7.4.5 ADCCHSELSEQ2 Register (Offset = 4h) [reset = 0h]

ADCCHSELSEQ2 is shown in Figure 7-19 and described in Table 7-15.

Figure 7-19. ADCCHSELSEQ2 Register

15	14	13	12	11	10	9	8
CONV07				CONV06			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
CONV05				CONV04			
R/W-0h				R/W-0h			

Table 7-15. ADCCHSELSEQ2 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	CONV07	R/W	0h	
11-8	CONV06	R/W	0h	
7-4	CONV05	R/W	0h	
3-0	CONV04	R/W	0h	

7.4.6 ADCCHSELSEQ3 Register (Offset = 5h) [reset = 0h]

ADCCHSELSEQ3 is shown in Figure 7-20 and described in Table 7-16.

Figure 7-20. ADCCHSELSEQ3 Register

15	14	13	12	11	10	9	8
CONV11				CONV10			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
CONV09				CONV08			
R/W-0h				R/W-0h			

Table 7-16. ADCCHSELSEQ3 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	CONV11	R/W	0h	
11-8	CONV10	R/W	0h	
7-4	CONV09	R/W	0h	
3-0	CONV08	R/W	0h	

Each of the 4-bit fields, CONVxx, selects one of the 16 MUXed analog input ADC channels for an autosequenced conversion.

Table 7-17. CONVnn Bit Values and the ADC Input Channels Selected

CONVnn Value	ADC Input Channel Selected
0000	ADCINA0
0001	ADCINA1
0010	ADCINA2
0011	ADCINA3
0100	ADCINA4
0101	ADCINA5
0110	ADCINA6
0111	ADCINA7
1000	ADCINB0
1001	ADCINB1
1010	ADCINB2
1011	ADCINB3
1100	ADCINB4
1101	ADCINB5
1110	ADCINB6
1111	ADCINB7

7.4.7 ADCCHSELSEQ4 Register (Offset = 6h) [reset = 0h]

ADCCHSELSEQ4 is shown in Figure 7-21 and described in Table 7-18.

Figure 7-21. ADCCHSELSEQ4 Register

15	14	13	12	11	10	9	8
CONV15				CONV14			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
CONV13				CONV12			
R/W-0h				R/W-0h			

Table 7-18. ADCCHSELSEQ4 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	CONV15	R/W	0h	
11-8	CONV14	R/W	0h	
7-4	CONV13	R/W	0h	
3-0	CONV12	R/W	0h	

7.4.8 ADCASEQSR Register (Offset = 7h) [reset = 0h]

ADCASEQSR is shown in Figure 7-22 and described in Table 7-19.

Figure 7-22. ADCASEQSR Register

15	14	13	12	11	10	9	8
RESERVED				SEQ_CNTR[3:0]			
R-0h				R-0h			
7	6	5	4	3	2	1	0
RESERVED	SEQ2_STATE			SEQ1_STATE			
R-0h	R-0h			R-0h			

Table 7-19. ADCASEQSR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reads return a zero. Writes have no effect.
11-8	SEQ_CNTR[3:0]	R	0h	Sequencing counter status bits. The SEQ_CNTRn 4-bit status field is used by SEQ1, SEQ2, and the cascaded sequencer. SEQ2 is irrelevant in cascaded mode. The Sequencer Counter bit field, SEQ_CNTR[3:0], is initialized to the value in MAX_CONV at the start of a conversion sequence. After each conversion (or a pair of conversions in simultaneous sampling mode) in an auto conversion sequence, the Sequencer Counter decreases by 1.The SEQ_CNTR bits can be read at any time during the countdown process to check status of the sequencer. This value, together with the SEQ1 and SEQ2 busy bits, uniquely identifies the progress or state of the active sequencer at any point in time. See Table 7-20.
7	RESERVED	R	0h	Reads return a zero. Writes have no effect.
6-4	SEQ2_STATE	R	0h	SEQ2_STATE and SEQ1_STATE bit fields are the pointers of SEQ2 and SEQ1, respectively.
3-0	SEQ1_STATE	R	0h	SEQ2_STATE and SEQ1_STATE bit fields are the pointers of SEQ2 and SEQ1, respectively.

Table 7-20. State of Active Sequencer

SEQ_CNTR (read only)	Number of conversions remaining
0000	1 or 0, depending on the busy bit
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

7.4.9 ADCRESULT_0 to ADCRESULT_15 Register (Offset = 8h to 17h) [reset = 0h]

ADCRESULT_0 to ADCRESULT_15 is shown in Figure 7-23 and described in Table 7-21.

In the cascaded sequencer mode, registers ADCRESULT8 through ADCRESULT15 holds the results of the ninth through sixteenth conversions. The ADCRESULTn registers are left justified when read from Peripheral Frame 2 (0x7108-0x7117) with two wait states and right justified when read from the mirror registers in Peripheral Frame 0 (0x0B00-0x0B0F) with zero wait states.

Figure 7-23. ADCRESULT_0 to ADCRESULT_15 Register

15	14	13	12	11	10	9	8
D11	D10	D9	D8	D7	D6	D5	D4
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
D3	D2	D1	D0	RESERVED			
R-0h	R-0h	R-0h	R-0h	R-0h			

Figure 7-24. ADCRESULT_0 to ADCRESULT_15 Register (Addresses 0x0B00-0x0B0F)

15	14	13	12	11	10	9	8
RESERVED				D11	D10	D9	D8
R-0h				R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

Table 7-21. ADCRESULT_0 to ADCRESULT_15 Register Field Descriptions

Bit	Field	Type	Reset	Description
15	D11	R	0h	
14	D10	R	0h	
13	D9	R	0h	
12	D8	R	0h	
11	D7	R	0h	
10	D6	R	0h	
9	D5	R	0h	
8	D4	R	0h	
7	D3	R	0h	
6	D2	R	0h	
5	D1	R	0h	
4	D0	R	0h	
3-0	RESERVED	R	0h	

7.4.10 ADCTRL3 Register (Offset = 18h) [reset = 0h]

ADCTRL3 is shown in Figure 7-25 and described in Table 7-22.

Figure 7-25. ADCTRL3 Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
ADCBGRFDN[1:0]		ADCPWDN	ADCCLKPS[3:0]				SMODE_SEL
R/W-0h		R/W-0h	R/W-0h				R/W-0h

Table 7-22. ADCTRL3 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reads return a zero. Writes have no effect.
7-6	ADCBGRFDN[1:0]	R/W	0h	ADC bandgap and reference power down. These bits control the power up and power down of the bandgap and reference circuitry inside the analog core. See Section 7.2.4for power-up sequence requirements. 00b = The bandgap and reference circuitry is powered down. 11b = The bandgap and reference circuitry is powered up.
5	ADCPWDN	R/W	0h	ADC power down. This bit controls the power up and power down of all the analog circuitry inside the analog core except the bandgap and reference circuitry. See Section 7.2.4for power-up sequence requirements. 0h = All analog circuitry inside the core except the bandgap and reference circuitry is powered down. 1h = The analog circuitry inside the core is powered up.
4-1	ADCCLKPS[3:0]	R/W	0h	Core clock divider. 28x peripheral clock, HSPCLK, is divided by 2*ADCCLKPS[3-0], except when ADCCLKPS[3-0] is 0000, in which case HSPCLK is directly passed on. The divided clock is further divided by ADCTRL1[7]+1 to generate the core clock, ADCLK. **ADCCLKPS [3:0] Core Clock Divider ADCLK** 0000b = HSPCLK/(ADCTRL1[7] + 1) 0001b = HSPCLK/[2*(ADCTRL1[7] + 1)] 0010b = HSPCLK/[4*(ADCTRL1[7] + 1)] 0011b = HSPCLK/[6*(ADCTRL1[7] + 1)] 0100b = HSPCLK/[8*(ADCTRL1[7] + 1)] 0101b = HSPCLK/[10*(ADCTRL1[7] + 1)] 0110b = HSPCLK/[12*(ADCTRL1[7] + 1)] 0111b = HSPCLK/[14*(ADCTRL1[7] + 1)] 1000b = HSPCLK/[16*(ADCTRL1[7] + 1)] 1001b = HSPCLK/[18*(ADCTRL1[7] + 1)] 1010b = HSPCLK/[20*(ADCTRL1[7] + 1)] 1011b = HSPCLK/[22*(ADCTRL1[7] + 1)] 1100b = HSPCLK/[24*(ADCTRL1[7] + 1)] 1101b = HSPCLK/[26*(ADCTRL1[7] + 1)] 1110b = HSPCLK/[28*(ADCTRL1[7] + 1)] 1111b = HSPCLK/[30*(ADCTRL1[7] + 1)]
0	SMODE_SEL	R/W	0h	Sampling mode select. This bit selects either sequential or simultaneous sampling mode. 0h = Sequential sampling mode is selected. 1h = Simultaneous sampling mode is selected.

7.4.11 ADCST Register (Offset = 19h) [reset = 0h]

ADCST is shown in Figure 7-26 and described in Table 7-23.

This register is a dedicated status and flag register. The bits in this register are either read-only status or flag bits, or read-return-zero condition clearing bits.

Figure 7-26. ADCST Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
EOS_BUF2	EOS_BUF1	INT_SEQ2_CLR	INT_SEQ1_CLR	SEQ2_BSY	SEQ1_BSY	INT_SEQ2	INT_SEQ1
R-0h	R-0h	R/W-0h	R/W-0h	R-0h	R-0h	R-0h	R-0h

Table 7-23. ADCST Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reads return a zero. Writes have no effect.
7	EOS_BUF2	R	0h	End of sequence buffer bit for SEQ2. This bit is not used and remains as zero in interrupt mode 0, i.e. when ADCTRL2[2]=0. In interrupt mode 1, i.e. when ADCTRL2[2]=1, it toggles on every end of sequence of SEQ2. This bit is cleared on device reset and is not affected by sequencer reset or clearing of the corresponding interrupt flag.
6	EOS_BUF1	R	0h	End of sequence buffer bit for SEQ1. This bit is not used and remains as zero in interrupt mode 0, i.e. when ADCTRL2[10]=0. In interrupt mode 1, i.e. when ADCTRL2[10]=1, it toggles on every end of sequence of SEQ1. This bit is cleared on device reset and is not affected by sequencer reset or clearing of the corresponding interrupt flag.
5	INT_SEQ2_CLR	R/W	0h	Interrupt clear bit. Read of this bit always returns 0. The clear action is a one-shot event following a write of 1 to this bit. 0h = Writing a zero to this bit has no effect. 1h = Writing a 1 to this bit clears the SEQ2 interrupt flag bit, INT_SEQ2. This bit does not affect the EOS_BUF2 bit.
4	INT_SEQ1_CLR	R/W	0h	Interrupt clear bit. Read of this bit always returns 0. The clear action is a one-shot event following a write of 1 to this bit. 0h = Writing a zero to this bit has no effect. 1h = Writing a 1 to this bit clears the SEQ1 interrupt flag bit, INT_SEQ1. This bit does not affect the EOS_BUF1 bit.
3	SEQ2_BSY	R	0h	SEQ2 busy status bit. 0h = SEQ2 is in idle, waiting for trigger. 1h = SEQ2 is in progress. Write to this bit has no effect.
2	SEQ1_BSY	R	0h	SEQ1 busy status bit. Write to this bit has no effect. 0h = SEQ1 is in idle, waiting for trigger. 1h = SEQ1 is in progress.
1	INT_SEQ2	R	0h	SEQ2 interrupt flag bit. Write to this bit has no effect. In interrupt mode 0, i.e. when ADCTRL2[2]=0, this bit is set on every end of sequence of Seq 2. In interrupt mode 1, i.e., when ADCTRL2[2]=1, this bit is set on an end of sequence of Seq 2 if EOS_BUF2 is set. 0h = No SEQ2 interrupt event. 1h = SEQ2 interrupt event occurred.
0	INT_SEQ1	R	0h	SEQ1 interrupt flag bit. Write to this bit has no effect. In interrupt mode 0, i.e. when ADCTRL2[10]=0, this bit is set on every end of sequence of Seq 1. In interrupt mode 1, i.e., when ADCTRL2[10]=1, this bit is set on an end of sequence of Seq 1 if EOS_BUF1 is set. 0h = No SEQ1 interrupt event. 1h = SEQ1 interrupt event occurred.

7.4.12 ADCREFSEL Register (Offset = 1Ch) [reset = 0h]

ADCOFFTRIM is shown in Figure 7-27 and described in Table 7-24.

Figure 7-27. ADCREFSEL Register

15	14	13	12	11	10	9	8
REF_SEL[1:0]		RESERVED					
R/W-0h		R-0h					
7	6	5	4	3	2	1	0
RESERVED							
R-0h							

Table 7-24. ADCREFSEL Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	REF_SEL[1:0]	R/W	0h	Reference select bits for ADC voltage generation circuit options are listed below: 00b = Internal reference selected (default) 01b = External reference, 2.048 V on ADCREFIN 10b = External reference, 1.500 V on ADCREFIN 11b = External reference, 1.024 V on ADCREFIN
13-0	Reserved	R/W	0h	Reads return a zero. Writes have no effect.

7.4.13 ADCOFFTRIM Register (Offset = 1Dh) [reset = 0h]

ADCOFFTRIM is shown in Figure 7-28 and described in Table 7-25.

Figure 7-28. ADCOFFTRIM Register

15	14	13	12	11	10	9	8
RESERVED							OFFSET_TRIM [8:0]
R-0h							R/W-0h
7	6	5	4	3	2	1	0
OFFSET_TRIM[8:0]							
R/W-0h							

Table 7-25. ADCOFFTRIM Register Field Descriptions

Bit	Field	Type	Reset	Description
15-9	RESERVED	R	0h	Reads return a zero. Writes have no effect.
8-0	OFFSET_TRIM[8:0]	R/W	0h	Offset trim value in LSBs, two's complement format - 256/255 range

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**

8.1 Introduction

The strength of a digital signal controller (DSC) 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 sequencer 1 and sequencer 2
  - Multichannel Buffered Serial Port A and B (McBSP-A, McBSP-B) transmit and receive
  - XINT1-7 and XINT13
  - CPU Timers
  - Software
  - ePWM1-6 ADCSOCA and ADSOCB signals
- Data sources/destinations:
  - L4-L7 16K x 16 SARAM
  - All XINTF zones
  - ADC memory bus mapped result registers
  - McBSP-A and McBSP-B transmit and receive buffers
  - ePWM1-6 / HRPWM1-6 Peripheral Frame 3 mapped registers
- Word Size: 16-bit or 32-bit (McBSPs limited to 16-bit)
- Throughput: 4 cycles/word (5 cycles/word for McBSP reads)

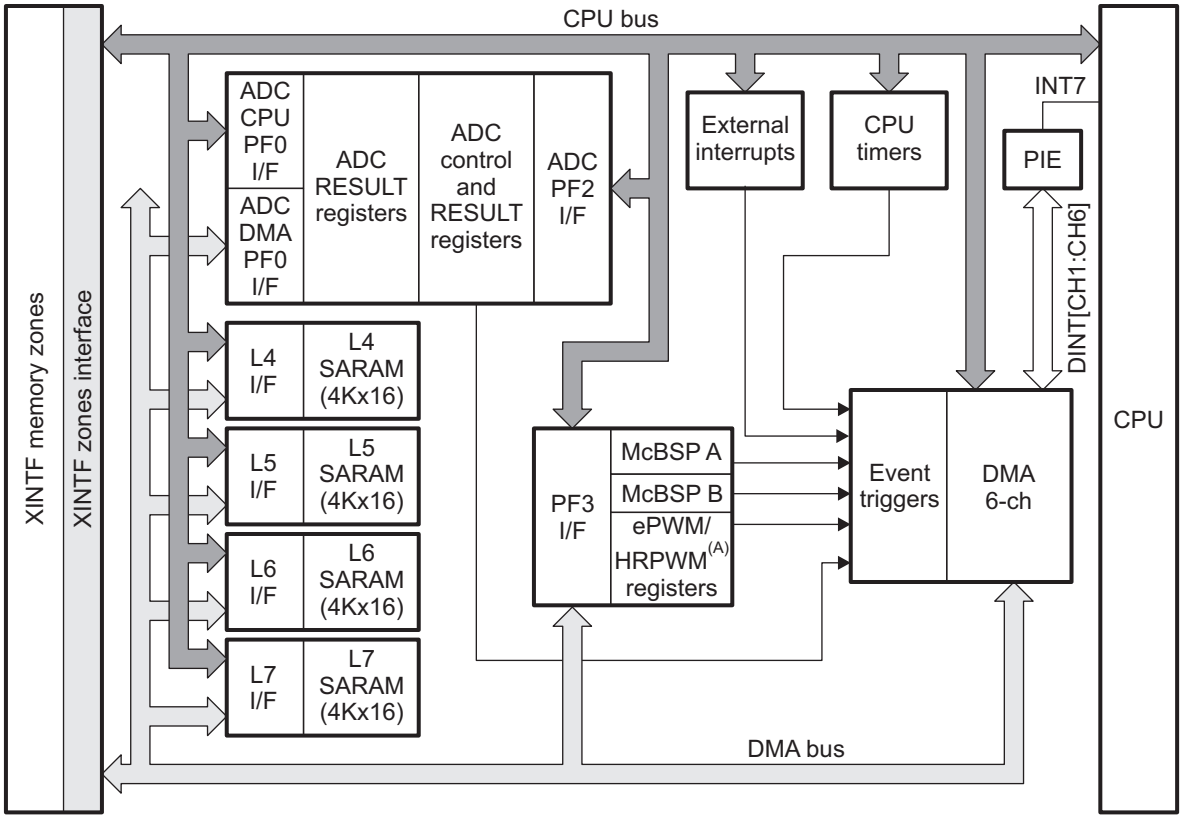
NOTE: The ePWM/HRPWM are not present on all devices and/or revisions. See the *TMS320x28xx, 28xxx device Peripheral Reference Guide* (SPRU566) for specifics.

8.2 Architecture

8.2.1 Block Diagram

Figure 8-1 shows a device level block diagram of the DMA.

Figure 8-1. DMA Block Diagram



A The ePWM/HRPWM registers must be remapped to PF3 (through bit 0 of the MAPCNF register) before they can be accessed by the DMA. The ePWM/HRPWM connection to DMA is not present in silicon revision 0.

8.2.2 Peripheral Interrupt Event Trigger Sources

The peripheral interrupt event trigger can be independently configured as one of eighteen different sources for each of the six DMA channels. Included in these sources are 8 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 8-2 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 8-2. Peripheral Interrupt Trigger Input Diagram

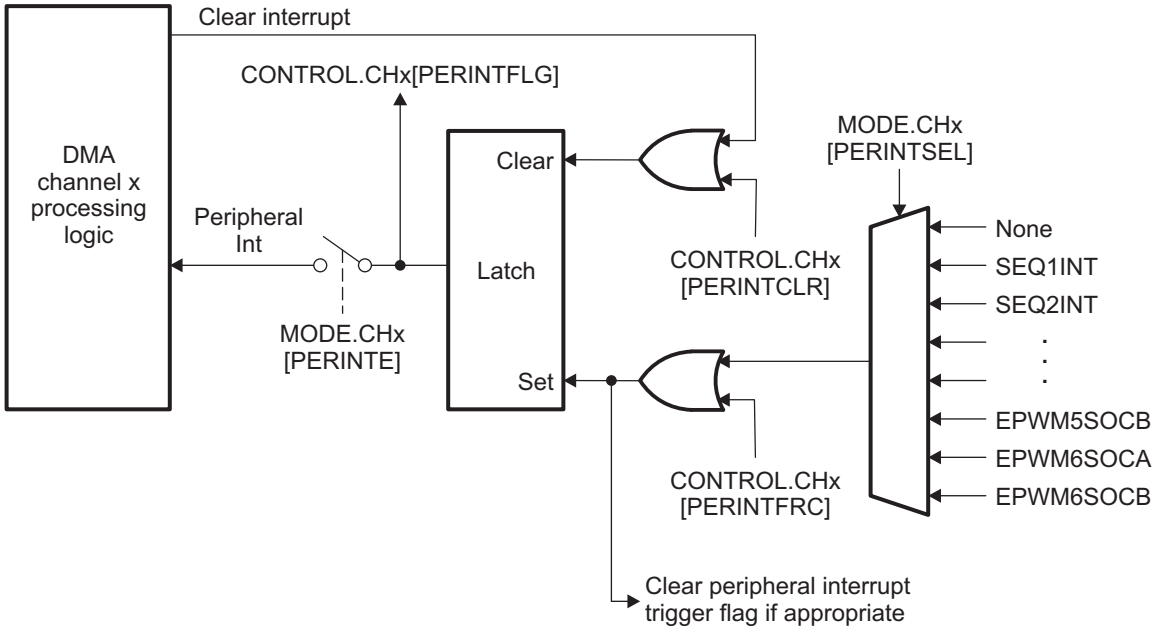


Table 8-1 shows the interrupt trigger source options that are available for each channel.

Table 8-1. Peripheral Interrupt Trigger Source Options

Peripheral	Interrupt Trigger Source
CPU	DMA Software bit (CHx.CONTROL.PERINTFRC) only
ADC	Sequencer 1 Interrupt
	Sequencer 2 Interrupt
External Interrupts	External Interrupt 1
	External Interrupt 2
	External Interrupt 3
	External Interrupt 4
	External Interrupt 5
	External Interrupt 6
	External Interrupt 7
	External Interrupt 13
CPU Timers	Timer 0 Overflow
	Timer 1 Overflow
	Timer 2 Overflow
McBSP-A	McBSP-A Transmit Buffer Empty
	McBSP-A Receive Buffer Full
McBSP-B	McBSP-B Transmit Buffer Empty
	McBSP-B Receive Buffer Full
ePWM1[(1)]	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM2[(1)]	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM3[(1)]	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM4[(1)]	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM5[(1)]	ADC Start of Conversion A
	ADC Start of Conversion B
ePWM6[(1)]	ADC Start of Conversion A
	ADC Start of Conversion B

(1) The ePWM1-6 are not present on all devices and/or revisions. See the *TMS320x28xx, 28xxx device Peripheral Reference Guide* (SPRU566) for specifics.

8.2.3 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 8.4. The following resources are connected to the DMA bus:

- XINTF Zones 0, 6, and 7
- L0-L7 SARAM
- L4 SARAM
- L5 SARAM
- L6 SARAM
- L7 SARAM

- ADC Memory Mapped Result Registers
- McBSP-A and McBSP-B Data Receive Registers (DRR2/DRR1) and Data Transmit Registers (DXR2/DXR1)
- ePWM1-6/HRPWM1-6 Register when mapped to Peripheral Frame 3

8.3 Pipeline Timing and Throughput

The DMA consists of a 4-stage pipeline as shown in Figure 8-3. The one exception to this is when a DMA channel is configured to have one of the McBSPs 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 8-4.

Figure 8-3. 4-Stage Pipeline DMA Transfer

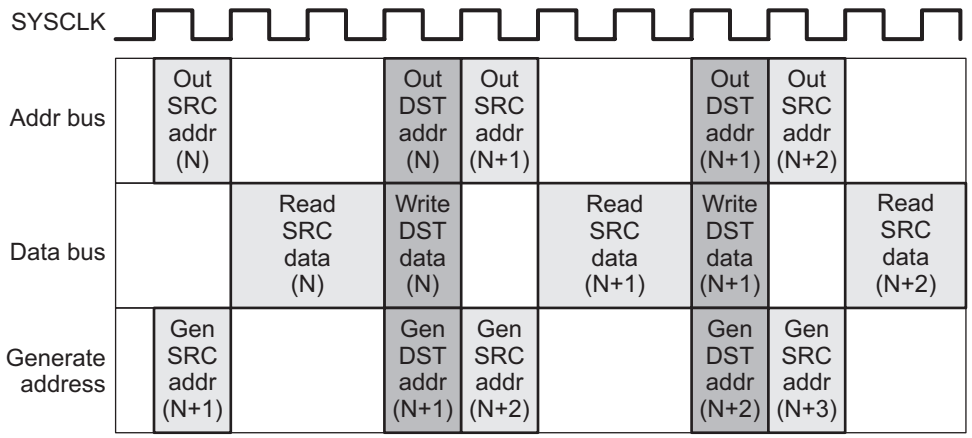
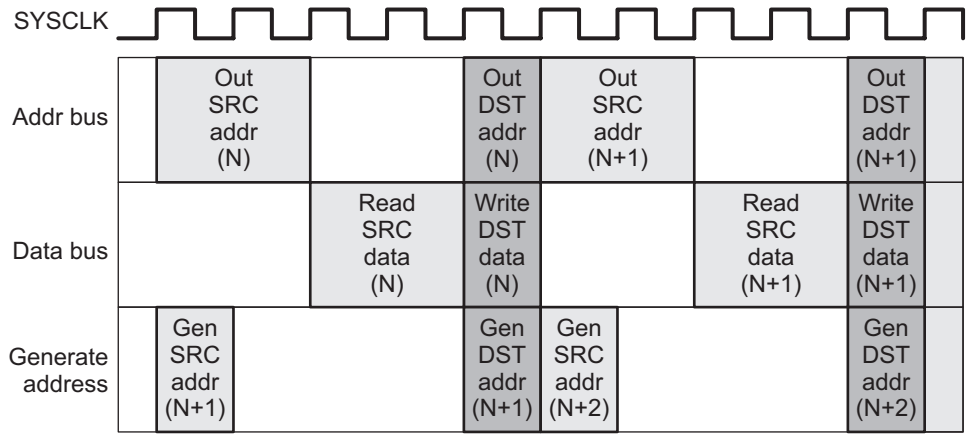


Figure 8-4. 4-Stage Pipeline With One Read Stall (McBSP as source)



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 8.4)

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

8.4 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 that internally contain conflicts are:

- XINTF Memory Zones 0, 6, and 7
- L4 RAM
- L5 RAM
- L6 RAM
- L7 RAM
- Peripheral Frame 3 (McBSP-A, McBSP-B)

NOTE: The ePWM/HRPWM are not present on all devices and/or revisions. See the *TMS320x28xx, 28xxx device Peripheral Reference Guide* (SPRU566) for specifics.

8.4.1 For the External Memory Interface (XINTF) Zones

- If the CPU and the DMA attempt an access to any of the XINTF zones on the same cycle, the DMA is serviced first, followed by all the pending CPU accesses (in the proper priority order for CPU accesses: write → read → fetch).
- If CPU accesses to an XINTF zone are pending or being processed by the XINTF and a DMA access to an XINTF zone is attempted, the DMA access is stalled until all CPU pending accesses are completed. For example, if a CPU write and read access is pending and a fetch is in progress, first the fetch is completed, then the CPU write is performed, then the CPU read is performed, and then the DMA access is performed.
- There is a 1 cycle stall if simultaneous write accesses by the CPU and the DMA are attempted.

If the DMA or CPU is used to write to the XINTF zones, then the write buffer of the XINTF can help to avoid CPU or DMA stalls. If the CPU or DMA are performing reads from XINTF, then significant stalls can occur. The only concern here is if the DMA is stalled and the DMA misses other higher priority DMA events such as servicing the ADC which can generate data at a high rate. In such situations, the DMA should not be used to transfer data on XINTF, if the stalls are too long that there is potential to miss other DMA events.

The DMA does not support abort mechanisms for DMA reads from XINTF. If the DMA is performing an access to one of the XINTF zones and the DMA access is stalled (XREADY not responding) then the CPU can issue a HARDRESET that would abort the access. HARDRESET behaves like a System Reset on the DMA. Likewise, a HARDRESET needs to be applied to the XINTF hence releasing the peripheral from the struck ready condition. Any data that is write buffered or pending on the XINTF or DMA will be lost.

8.4.2 For All Other Peripherals/Memories

- 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.

8.5 Channel Priority

Two priority schemes exist when determining channel priority: Round-robin mode and Channel 1 high-priority mode.

8.5.1 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.

8.5.2 Channel 1 High Priority Mode

In this mode, if a CH1 trigger occurs, the current word transfer or the current + 1 word transfer (depends on which phase of the current DMA transfer the new CH1 trigger occurred) 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.

NOTE: High-priority mode and ONESHOT mode may not be used at the same time on channel 1. Other channels may use ONESHOT mode when channel 1 is in high-priority mode.

8.6 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.

High-priority mode and ONESHOT mode may not be used at the same time on channel 1. Other channels may use ONESHOT mode when channel 1 is in high-priority mode.

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.

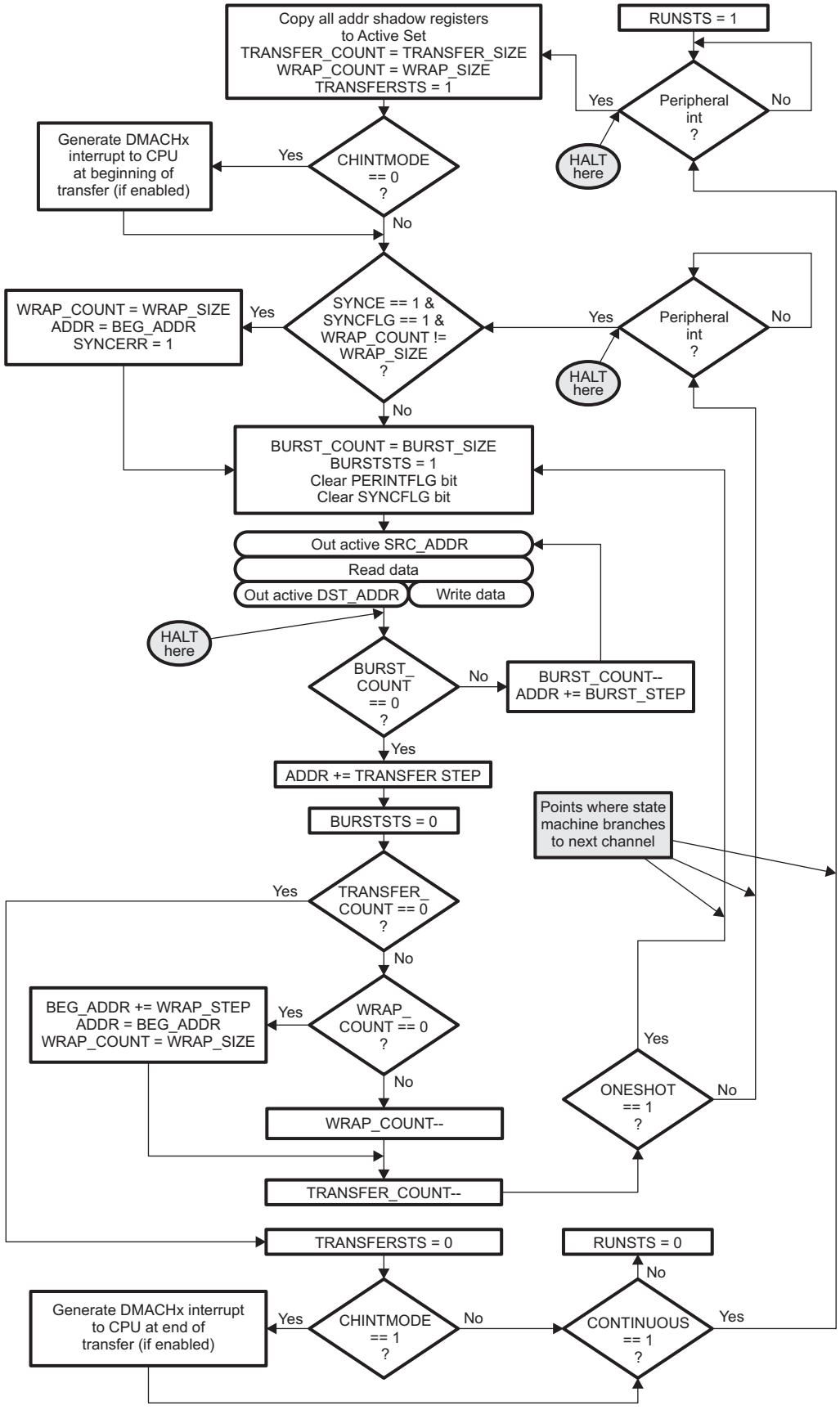
The channel must be re-enabled by setting the RUN bit in the CONTROL register before another transfer can be started on that channel. If the continuous mode is enabled the RUNSTS bit is not cleared at the end of the transfer.

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 8-5.

Figure 8-5. DMA State Diagram



The following items are in reference to Figure 8-5.

- 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.

8.7 ADC Sync Feature

The DMA provides a hardware method of synchronizing to the ADC Sequencer 1 interrupt (SEQ1INT) when it is running in continuous conversion mode with the sequencer override function enabled. In this specific mode, the ADC will be continuously converting a sequence of ADC channels without resetting the sequencer pointer at the end of each sequence. Since the DMA does not know which ADC RESULT register the sequencer pointer is pointing to when it receives a trigger, there is a potential for the DMA and the ADC to be out of step. For this reason, when the ADC is configured in this mode, it provides a synchronization signal to the DMA each time an event trigger is generated for a sequence starting at the RESULT0 register. The DMA expects this signal to line up with a wrap procedure or the beginning of a transfer. If it does not, a re-sync procedure occurs:

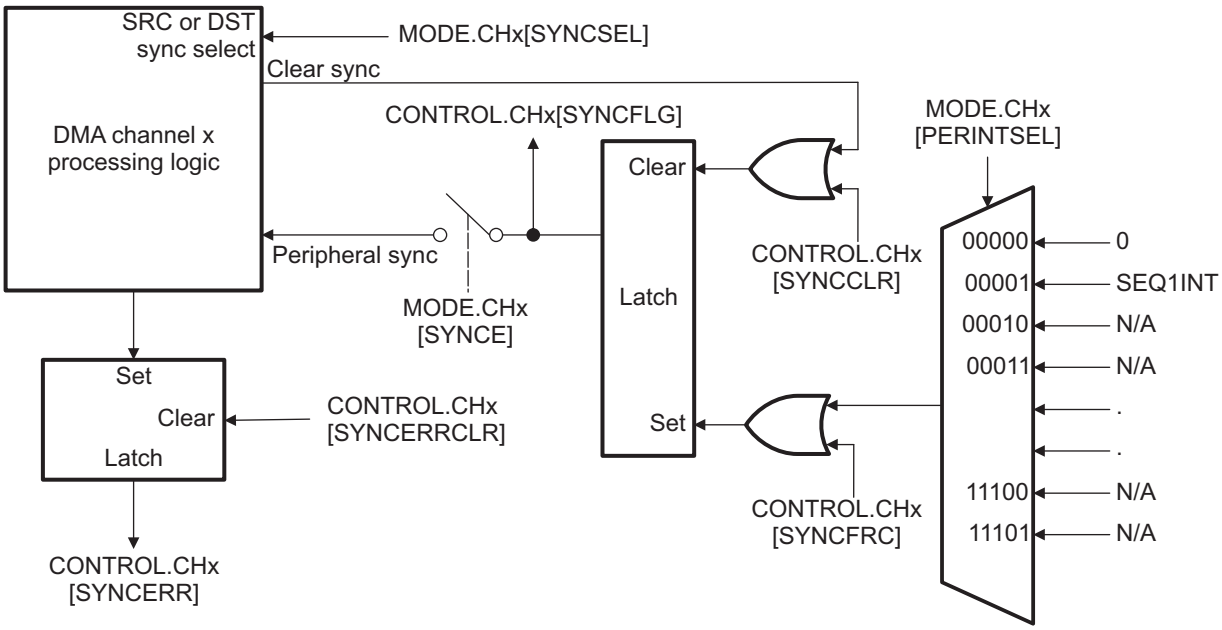
1. Reload the WRAP_COUNT register with WRAP_SIZE
2. Load the ADDR.active register with BEG_ADDR.active
3. Set the SYNCERR bit in the CONTROL register

This allows the use of multiple buffers to store the data and for the DMA and the ADC to resynchronize if necessary. For example, assume four ADC channels are configured to be converted at a time on sequencer 1. Since the maximum length of sequencer 1 is eight elements, the sequencer will reset itself, and generate a sync signal every other sequence. Assume the software expects that the first four results will be placed by the DMA into Buffer A, and the second four into Buffer B. If DMA activity becomes overburdened and an event trigger is lost, the DMA and the ADC will become unsynchronized. At this point the DMA will set the SYNCERR bit in the CONTROL register and perform the above re-sync procedure, bringing the DMA and the ADC back into synchronization.

Alternatively, the sync feature can be configured to work on the source address pointer via the SYNCSEL bit in the MODE register.

As shown in Figure 8-6, the synchronization source is chosen by the PERINTSEL bit field in the MODE register. If the SYNC feature is enabled for the selected source and channel, the transfer for that channel will not commence until reception of the first SYNC after the RUN bit is set. All peripheral interrupt triggers are ignored up to the first SYNC event.

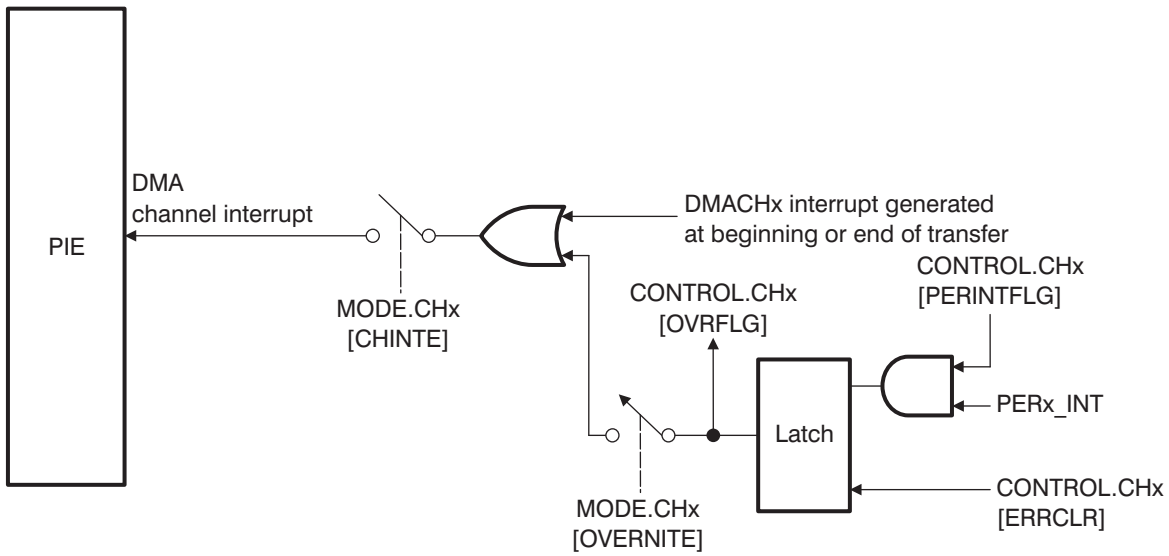
Figure 8-6. ADC Sync Input Diagram



8.8 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 8-7. If the overrun interrupt is enabled then the channel interrupt will be generated to the PIE module.

Figure 8-7. Overrun Detection Logic



8.9 Register Descriptions

Table 8-2 lists the memory-mapped registers for the DMA_REGISTER_SUMMARY. All register offset addresses not listed in Table 8-2 should be considered as reserved locations and the register contents should not be modified.

Table 8-2. DMA Register Summary[1]

Offset	Acronym	Register Name	Section
1000h	DMACTRL	DMA Control Register	Section 8.9.1
1001h	DEBUGCTRL	Debug Control Register	Section 8.9.2
1002h	REVISION	Peripheral Revision Register	Section 8.9.3
1004h	PRIORITYCTRL1	Priority Control Register 1	Section 8.9.4
1006h	PRIORITYSTAT	Priority Status Register	Section 8.9.5
1020h	MODE_0	Mode Register	Section 8.9.6
1021h	CONTROL_0	Control Register	Section 8.9.7
1022h	BURST_SIZE_0	Burst Size Register	Section 8.9.8
1023h	BURST_COUNT_0	Burst Count Register	Section 8.9.9
1024h	SRC_BURST_STEP_0	Source Burst Step Size Register	Section 8.9.10
1025h	DST_BURST_STEP_0	Destination Burst Step Size Register	Section 8.9.11
1026h	TRANSFER_SIZE_0	Transfer Size Register	Section 8.9.12
1027h	TRANSFER_COUNT_0	Transfer Count Register	Section 8.9.13
1028h	SRC_TRANSFER_STEP_0	Source Transfer Step Size Register	Section 8.9.14
1029h	DST_TRANSFER_STEP_0	Destination Transfer Step Size Register	Section 8.9.15

(1) All DMA register writes are EALLOW protected.

Table 8-2. DMA Register Summary[1] (continued)

Offset	Acronym	Register Name	Section
102Ah	SRC_WRAP_SIZE_0	Source Wrap Size Register	Section 8.9.16
102Bh	SRC_WRAP_COUNT_0	Source Wrap Count Register	Section 8.9.17
102Ch	SRC_WRAP_STEP_0	Source Wrap Step Size Register	Section 8.9.18
102Dh	DST_WRAP_SIZE_0	Destination Wrap Size Register	Section 8.9.19
102Eh	DST_WRAP_COUNT_0	Destination Wrap Count Register	Section 8.9.20
102Fh	DST_WRAP_STEP_0	Destination Wrap Step Size Register	Section 8.9.21
1030h	SRC_BEG_ADDR_SHADOW_0	Shadow Source Begin and Current Address Pointer Registers	Section 8.9.22
1032h	SRC_ADDR_SHADOW_0	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.23
1034h	SRC_BEG_ADDR_0	Active Source Begin and Current Address Pointer Registers	Section 8.9.24
1036h	SRC_ADDR_0	Active Destination Begin and Current Address Pointer Registers	Section 8.9.25
1038h	DST_BEG_ADDR_SHADOW_0	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.26
103Ah	DST_ADDR_SHADOW_0	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.27
103Ch	DST_BEG_ADDR_0	Active Destination Begin and Current Address Pointer Registers	Section 8.9.28
103Eh	DST_ADDR_0	Active Destination Begin and Current Address Pointer Registers	Section 8.9.29
1103h	MODE_1	Mode Register	Section 8.9.6
1104h	CONTROL_1	Control Register	Section 8.9.7
1105h	BURST_SIZE_1	Burst Size Register	Section 8.9.8
1106h	BURST_COUNT_1	Burst Count Register	Section 8.9.9
1107h	SRC_BURST_STEP_1	Source Burst Step Size Register	Section 8.9.10
1108h	DST_BURST_STEP_1	Destination Burst Step Size Register	Section 8.9.11
1109h	TRANSFER_SIZE_1	Transfer Size Register	Section 8.9.12
110Ah	TRANSFER_COUNT_1	Transfer Count Register	Section 8.9.13
110Bh	SRC_TRANSFER_STEP_1	Source Transfer Step Size Register	Section 8.9.14
110Ch	DST_TRANSFER_STEP_1	Destination Transfer Step Size Register	Section 8.9.15
110Dh	SRC_WRAP_SIZE_1	Source Wrap Size Register	Section 8.9.16
110Eh	SRC_WRAP_COUNT_1	Source Wrap Count Register	Section 8.9.17
110Fh	SRC_WRAP_STEP_1	Source Wrap Step Size Register	Section 8.9.18
1110h	DST_WRAP_SIZE_1	Destination Wrap Size Register	Section 8.9.19
1111h	DST_WRAP_COUNT_1	Destination Wrap Count Register	Section 8.9.20
1112h	DST_WRAP_STEP_1	Destination Wrap Step Size Register	Section 8.9.21
1113h	SRC_BEG_ADDR_SHADOW_1	Shadow Source Begin and Current Address Pointer Registers	Section 8.9.22
1115h	SRC_ADDR_SHADOW_1	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.23
1117h	SRC_BEG_ADDR_1	Active Source Begin and Current Address Pointer Registers	Section 8.9.24
1119h	SRC_ADDR_1	Active Destination Begin and Current Address Pointer Registers	Section 8.9.25
111Bh	DST_BEG_ADDR_SHADOW_1	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.26
111Dh	DST_ADDR_SHADOW_1	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.27
111Fh	DST_BEG_ADDR_1	Active Destination Begin and Current Address Pointer Registers	Section 8.9.28

Table 8-2. DMA Register Summary[1] (continued)

Offset	Acronym	Register Name	Section
1121h	DST_ADDR_1	Active Destination Begin and Current Address Pointer Registers	Section 8.9.29
11E6h	MODE_2	Mode Register	Section 8.9.6
11E7h	CONTROL_2	Control Register	Section 8.9.7
11E8h	BURST_SIZE_2	Burst Size Register	Section 8.9.8
11E9h	BURST_COUNT_2	Burst Count Register	Section 8.9.9
11EAh	SRC_BURST_STEP_2	Source Burst Step Size Register	Section 8.9.10
11EBh	DST_BURST_STEP_2	Destination Burst Step Size Register	Section 8.9.11
11ECh	TRANSFER_SIZE_2	Transfer Size Register	Section 8.9.12
11EDh	TRANSFER_COUNT_2	Transfer Count Register	Section 8.9.13
11EEh	SRC_TRANSFER_STEP_2	Source Transfer Step Size Register	Section 8.9.14
11EFh	DST_TRANSFER_STEP_2	Destination Transfer Step Size Register	Section 8.9.15
11F0h	SRC_WRAP_SIZE_2	Source Wrap Size Register	Section 8.9.16
11F1h	SRC_WRAP_COUNT_2	Source Wrap Count Register	Section 8.9.17
11F2h	SRC_WRAP_STEP_2	Source Wrap Step Size Register	Section 8.9.18
11F3h	DST_WRAP_SIZE_2	Destination Wrap Size Register	Section 8.9.19
11F4h	DST_WRAP_COUNT_2	Destination Wrap Count Register	Section 8.9.20
11F5h	DST_WRAP_STEP_2	Destination Wrap Step Size Register	Section 8.9.21
11F6h	SRC_BEG_ADDR_SHADOW_2	Shadow Source Begin and Current Address Pointer Registers	Section 8.9.22
11F8h	SRC_ADDR_SHADOW_2	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.23
11FAh	SRC_BEG_ADDR_2	Active Source Begin and Current Address Pointer Registers	Section 8.9.24
11FCh	SRC_ADDR_2	Active Destination Begin and Current Address Pointer Registers	Section 8.9.25
11FEh	DST_BEG_ADDR_SHADOW_2	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.26
1200h	DST_ADDR_SHADOW_2	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.27
1202h	DST_BEG_ADDR_2	Active Destination Begin and Current Address Pointer Registers	Section 8.9.28
1204h	DST_ADDR_2	Active Destination Begin and Current Address Pointer Registers	Section 8.9.29
12C9h	MODE_3	Mode Register	Section 8.9.6
12CAh	CONTROL_3	Control Register	Section 8.9.7
12CBh	BURST_SIZE_3	Burst Size Register	Section 8.9.8
12CCh	BURST_COUNT_3	Burst Count Register	Section 8.9.9
12CDh	SRC_BURST_STEP_3	Source Burst Step Size Register	Section 8.9.10
12CEh	DST_BURST_STEP_3	Destination Burst Step Size Register	Section 8.9.11
12CFh	TRANSFER_SIZE_3	Transfer Size Register	Section 8.9.12
12D0h	TRANSFER_COUNT_3	Transfer Count Register	Section 8.9.13
12D1h	SRC_TRANSFER_STEP_3	Source Transfer Step Size Register	Section 8.9.14
12D2h	DST_TRANSFER_STEP_3	Destination Transfer Step Size Register	Section 8.9.15
12D3h	SRC_WRAP_SIZE_3	Source Wrap Size Register	Section 8.9.16
12D4h	SRC_WRAP_COUNT_3	Source Wrap Count Register	Section 8.9.17
12D5h	SRC_WRAP_STEP_3	Source Wrap Step Size Register	Section 8.9.18
12D6h	DST_WRAP_SIZE_3	Destination Wrap Size Register	Section 8.9.19
12D7h	DST_WRAP_COUNT_3	Destination Wrap Count Register	Section 8.9.20
12D8h	DST_WRAP_STEP_3	Destination Wrap Step Size Register	Section 8.9.21

Table 8-2. DMA Register Summary[1] (continued)

Offset	Acronym	Register Name	Section
12D9h	SRC_BEG_ADDR_SHADOW_3	Shadow Source Begin and Current Address Pointer Registers	Section 8.9.22
12DBh	SRC_ADDR_SHADOW_3	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.23
12DDh	SRC_BEG_ADDR_3	Active Source Begin and Current Address Pointer Registers	Section 8.9.24
12DFh	SRC_ADDR_3	Active Destination Begin and Current Address Pointer Registers	Section 8.9.25
12E1h	DST_BEG_ADDR_SHADOW_3	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.26
12E3h	DST_ADDR_SHADOW_3	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.27
12E5h	DST_BEG_ADDR_3	Active Destination Begin and Current Address Pointer Registers	Section 8.9.28
12E7h	DST_ADDR_3	Active Destination Begin and Current Address Pointer Registers	Section 8.9.29
13ACh	MODE_4	Mode Register	Section 8.9.6
13ADh	CONTROL_4	Control Register	Section 8.9.7
13AEh	BURST_SIZE_4	Burst Size Register	Section 8.9.8
13AFh	BURST_COUNT_4	Burst Count Register	Section 8.9.9
13B0h	SRC_BURST_STEP_4	Source Burst Step Size Register	Section 8.9.10
13B1h	DST_BURST_STEP_4	Destination Burst Step Size Register	Section 8.9.11
13B2h	TRANSFER_SIZE_4	Transfer Size Register	Section 8.9.12
13B3h	TRANSFER_COUNT_4	Transfer Count Register	Section 8.9.13
13B4h	SRC_TRANSFER_STEP_4	Source Transfer Step Size Register	Section 8.9.14
13B5h	DST_TRANSFER_STEP_4	Destination Transfer Step Size Register	Section 8.9.15
13B6h	SRC_WRAP_SIZE_4	Source Wrap Size Register	Section 8.9.16
13B7h	SRC_WRAP_COUNT_4	Source Wrap Count Register	Section 8.9.17
13B8h	SRC_WRAP_STEP_4	Source Wrap Step Size Register	Section 8.9.18
13B9h	DST_WRAP_SIZE_4	Destination Wrap Size Register	Section 8.9.19
13BAh	DST_WRAP_COUNT_4	Destination Wrap Count Register	Section 8.9.20
13BBh	DST_WRAP_STEP_4	Destination Wrap Step Size Register	Section 8.9.21
13BCh	SRC_BEG_ADDR_SHADOW_4	Shadow Source Begin and Current Address Pointer Registers	Section 8.9.22
13BEh	SRC_ADDR_SHADOW_4	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.23
13C0h	SRC_BEG_ADDR_4	Active Source Begin and Current Address Pointer Registers	Section 8.9.24
13C2h	SRC_ADDR_4	Active Destination Begin and Current Address Pointer Registers	Section 8.9.25
13C4h	DST_BEG_ADDR_SHADOW_4	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.26
13C6h	DST_ADDR_SHADOW_4	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.27
13C8h	DST_BEG_ADDR_4	Active Destination Begin and Current Address Pointer Registers	Section 8.9.28
13CAh	DST_ADDR_4	Active Destination Begin and Current Address Pointer Registers	Section 8.9.29
148Fh	MODE_5	Mode Register	Section 8.9.6
1490h	CONTROL_5	Control Register	Section 8.9.7
1491h	BURST_SIZE_5	Burst Size Register	Section 8.9.8
1492h	BURST_COUNT_5	Burst Count Register	Section 8.9.9

Table 8-2. DMA Register Summary[1] (continued)

Offset	Acronym	Register Name	Section
1493h	SRC_BURST_STEP_5	Source Burst Step Size Register	Section 8.9.10
1494h	DST_BURST_STEP_5	Destination Burst Step Size Register	Section 8.9.11
1495h	TRANSFER_SIZE_5	Transfer Size Register	Section 8.9.12
1496h	TRANSFER_COUNT_5	Transfer Count Register	Section 8.9.13
1497h	SRC_TRANSFER_STEP_5	Source Transfer Step Size Register	Section 8.9.14
1498h	DST_TRANSFER_STEP_5	Destination Transfer Step Size Register	Section 8.9.15
1499h	SRC_WRAP_SIZE_5	Source Wrap Size Register	Section 8.9.16
149Ah	SRC_WRAP_COUNT_5	Source Wrap Count Register	Section 8.9.17
149Bh	SRC_WRAP_STEP_5	Source Wrap Step Size Register	Section 8.9.18
149Ch	DST_WRAP_SIZE_5	Destination Wrap Size Register	Section 8.9.19
149Dh	DST_WRAP_COUNT_5	Destination Wrap Count Register	Section 8.9.20
149Eh	DST_WRAP_STEP_5	Destination Wrap Step Size Register	Section 8.9.21
149Fh	SRC_BEG_ADDR_SHADOW_5	Shadow Source Begin and Current Address Pointer Registers	Section 8.9.22
14A1h	SRC_ADDR_SHADOW_5	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.23
14A3h	SRC_BEG_ADDR_5	Active Source Begin and Current Address Pointer Registers	Section 8.9.24
14A5h	SRC_ADDR_5	Active Destination Begin and Current Address Pointer Registers	Section 8.9.25
14A7h	DST_BEG_ADDR_SHADOW_5	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.26
14A9h	DST_ADDR_SHADOW_5	Shadow Destination Begin and Current Address Pointer Registers	Section 8.9.27
14ABh	DST_BEG_ADDR_5	Active Destination Begin and Current Address Pointer Registers	Section 8.9.28
14ADh	DST_ADDR_5	Active Destination Begin and Current Address Pointer Registers	Section 8.9.29

8.9.1 DMACTRL Register (Offset = 1000h) [reset = 0h]

DMACTRL is shown in Figure 8-8 and described in Table 8-3.

Figure 8-8. DMACTRL Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED						PRIORITYRESET	HARDRESET
R-0h						RS-0h	RS-0h

Table 8-3. DMACTRL Register Field Descriptions

Bit	Field	Type	Reset	Description
15-2	RESERVED	R	0h	Reserved
1	PRIORITYRESET	RS	0h	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).
0	HARDRESET	RS	0h	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.

8.9.2 DEBUGCTRL Register (Offset = 1001h) [reset = 0h]

DEBUGCTRL is shown in Figure 8-9 and described in Table 8-4.

Figure 8-9. DEBUGCTRL Register

15	14	13	12	11	10	9	8
FREE	RESERVED						
R/W-0h	R-0h						
7	6	5	4	3	2	1	0
RESERVED							
R-0h							

Table 8-4. DEBUGCTRL Register Field Descriptions

Bit	Field	Type	Reset	Description
15	FREE	R/W	0h	Emulation Control Bit: This bit specifies the action when an emulation halt event occurs. 0h = 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 8-5 for possible halt states. 1h = DMA is unaffected by emulation suspend (run free)
14-0	RESERVED	R	0h	Reserved

8.9.3 REVISION Register (Offset = 1002h) [reset = 0h]

REVISION is shown in Figure 8-10 and described in Table 8-5.

Figure 8-10. REVISION Register

15	14	13	12	11	10	9	8
TYPE							
R-0h							
7	6	5	4	3	2	1	0
REV							
R-0h							

Table 8-5. REVISION Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	TYPE	R	0h	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). 0000h = This document describes a Type0 DMA.
7-0	REV	R	0h	DMA Silicon Revision Bits: These bits specify the DMA revision and are changed if any bug fixes are performed. 0000h = First release

8.9.4 PRIORITYCTRL1 Register (Offset = 1004h) [reset = 0h]

PRIORITYCTRL1 is shown in Figure 8-11 and described in Table 8-6.

Figure 8-11. PRIORITYCTRL1 Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							CH1PRIORITY
R-0h							R/W-0h

Table 8-6. PRIORITYCTRL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-1	RESERVED	R	0h	Reserved
0	CH1PRIORITY	R/W	0h	DMA Ch1 Priority: This bit selects whether channel 1 has higher priority or not: Channel priority can only be changed when all channels are disabled. A priority reset should be performed before restarting channels after changing priority. 0h = Same priority as all other channels 1h = Highest priority channel

8.9.5 PRIORITYSTAT Register (Offset = 1006h) [reset = 0h]

PRIORITYSTAT is shown in Figure 8-12 and described in Table 8-7.

Figure 8-12. PRIORITYSTAT Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	ACTIVESTS_SHADOW			RESERVED	ACTIVESTS		
R-0h	R-0h			R-0h	R-0h		

Table 8-7. PRIORITYSTAT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6-4	ACTIVESTS_SHADOW	R	0h	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: 0h = No channel pending 1h = CH 1 2h = CH 2 3h = CH 3 4h = CH 4 5h = CH 5 6h = CH 6
3	RESERVED	R	0h	Reserved
2-0	ACTIVESTS	R	0h	Active Channel Status Bits: These bits indicate which channel is currently active or performing a transfer: 000b = no channel active 1h = CH 1 2h = CH 2 3h = CH 3 4h = CH 4 5h = CH 5 6h = CH 6

8.9.6 MODE Register (Offset = 1020h + [i * E3h]) [reset = 0h]

MODE is shown in Figure 8-13 and described in Table 8-8.

Figure 8-13. MODE Register

15	14	13	12	11	10	9	8
CHINTE	DATASIZE	SYNCSEL	SYNCE	CONTINUOUS	ONESHOT	CHINTMODE	PERINTE
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
OVRINTE	RESERVED		PERINTSEL				
R/W-0h	R-0h		R/W-0h				

Table 8-8. MODE Register Field Descriptions[1][2][3]

Bit	Field	Type	Reset	Description
15	CHINTE	R/W	0h	Channel Interrupt Enable Bit: This bit enables/disables the respective DMA channel interrupt to the CPU (via the PIE). 0h = Interrupt disabled 1h = Interrupt enabled
14	DATASIZE	R/W	0h	Data Size Mode Bit: This bit selects if the DMA channel transfers 16-bits or 32-bits of data at a time. 0h = 16-bit data transfer size 1h = 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 8.6 for details.
13	SYNCSEL	R/W	0h	Sync Mode Select Bit: This bit selects if the SRC or DST wrap counters are controlled by the sync function (if enabled by SYNCE bit). 0h = SRC wrap counter controlled 1h = DST wrap counter controlled
12	SYNCE	R/W	0h	Sync Enable Bit: If this bit is set to 1, the ADCSYNC signal is recognized if selected by the PERINTSEL bit field. This sync signal is used to synchronize ADC interrupt event triggers to the DMA wrap counter. If this bit is 0 then the ADCSYNC event is ignored.
11	CONTINUOUS	R/W	0h	Continuous Mode Bit: 0h = DMA stops and clears the RUNSTS bit to 0. 1h = DMA re-initializes when TRANSFER_COUNT is zero and waits for the next interrupt event trigger.
10	ONESHOT	R/W	0h	One Shot Mode Bit: 0h = only one burst transfer is performed per event trigger. 1h = subsequent burst transfers occur without additional event triggers after the first event trigger.

(1) You should write to the MODE register only when the RUNSTS bit is 0 (DMA channel stopped or halted). Typically though, the values should only be configured when the channel is stopped.
(2) The ADCSYNC only works when the sequencer override bit is set in the ADC sequencer control registers.
(3) The overflow interrupt is ORed together with the DMACH interrupt as shown in Figure 8-7.

Table 8-8. MODE Register Field Descriptions[1][2][3] (continued)

Bit	Field	Type	Reset	Description
9	CHINTMODE	R/W	0h	Channel Interrupt Generation Mode Bit: This bit specifies when the respective DMA channel interrupt should be generated to the CPU (via the PIE). 0h = Generate interrupt at beginning of new transfer 1h = Generate interrupt at end of transfer.
8	PERINTE	R/W	0h	Peripheral Interrupt Trigger Enable Bit: This bit enables/disables the selected peripheral interrupt trigger to the DMA. 0h = Interrupt trigger disabled. Neither the selected peripheral nor software can start a DMA burst. 1h = Interrupt trigger enabled.
7	OVRINTE	R/W	0h	Overflow Interrupt Enable: This bit when set to 1 enables the DMA to generate an interrupt when an overflow event is detected. 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. 0h = Overflow interrupt disabled 1h = Overflow interrupt enabled
6-5	RESERVED	R	0h	Reserved
4-0	PERINTSEL	R/W	0h	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. These bits also select whether the ADCSYNC signal connects to the channel. See Table 8-9.

Table 8-9. PREINTSEL Values

Value	Interrupt	Sync	Peripheral
0	None	None	No Peripheral Connection
1	SEQ1INT	ADCSYNC	ADC
2	SEQ2INT	None	
3	XINT1	None	External Interrupts
4	XINT2	None	
5	XINT3	None	
6	XINT4	None	
7	XINT5	None	
8	XINT6	None	
9	XINT7	None	
10	XINT13	None	
11	TINT0	None	CPU Timers
12	TINT1	None	
13	TINT2	None	
14	MXEVTA	None	McBSP-A
15	MREVTA	None	
16	MXEVTB	None	McBSP-B
17	MREVTB	None	
18	ePWM1SOCA	None	ePWM1
19	ePWM1SOCB	None	
20	ePWM2SOCA	None	ePWM2
21	ePWM2SOCB	None	

Table 8-9. PREINTSEL Values (continued)

Value	Interrupt	Sync	Peripheral
22	ePWM3SOCA	None	ePWM3
23	ePWM3SOCB	None	
24	ePWM4SOCA	None	ePWM4
25	ePWM4SOCB	None	
26	ePWM5SOCA	None	ePWM5
27	ePWM5SOCB	None	
28	ePWM6SOCA	None	ePWM6
29	ePWM6SOCB	None	
31:30	Reserved		No peripheral connection

8.9.7 CONTROL Register (Offset = 1021h + [i * E3h]) [reset = 0h]

CONTROL is shown in Figure 8-14 and described in Table 8-10.

Figure 8-14. CONTROL Register

15	14	13	12	11	10	9	8
RESERVED	OVRFLG	RUNSTS	BURSTSTS	TRANSFERSTS	SYNCERR	SYNCFLG	PERINTFLG
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
ERRCLR	SYNCCLR	SYNCFRC	PERINTCLR	PERINTFRC	SOFTRESET	HALT	RUN
RS-0h	RS-0h	RS-0h	RS-0h	RS-0h	RS-0h	RS-0h	RS-0h

Table 8-10. CONTROL Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reserved
14	OVRFLG	R	0h	Overflow Flag Bit: This bit indicates if a peripheral interrupt event trigger is received from the selected peripheral and the PERINTFLG is already set. 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. 0h = No overflow event 1h = Overflow event
13	RUNSTS	R	0h	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. 0h = Channel is disabled. 1h = Channel is enabled.
12	BURSTSTS	R	0h	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. 0h = No burst activity 1h = The DMA is currently servicing or suspending a burst transfer from this channel.
11	TRANSFERSTS	R	0h	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. 0h = No transfer activity 1h = The channel is currently in the middle of a transfer regardless of whether a burst of data is actively being transferred or not.
10	SYNCERR	R	0h	Sync ERR Bit: This bit indicates if an ADCSYNC event error has occurred. This bit is set to 1 when the ADCSYNC event occurs and the selected SRC or DST_WRAP_COUNT is not zero: You should read the SYNCERR flag when servicing the respective DMA channel interrupt to determine if a sync error did occur. 0h = No sync error event 1h = Sync error event

Table 8-10. CONTROL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
9	SYNCFLG	R	0h	Sync Flag Bit: This bit indicates if an ADCSYNC event has occurred. This flag is automatically cleared when the first burst transfer begins. The SYNCFRC bit can be used to set the state of this bit to 1. The SYNCCLR bit can be used to clear the state of this bit to 0. 0h = No sync event 1h = Sync event
8	PERINTFLG	R	0h	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. 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. 0h = No interrupt event trigger 1h = Interrupt event trigger
7	ERRCLR	RS	0h	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	SYNCCLR	RS	0h	Sync Clear Bit: Writing a 1 to this bit will clear a latched sync event and clear the SYNCFLG bit. This bit would normally be used when initializing the DMA for the first time. If an ADCSYNC event occurs at the same time as writing to this bit, the ADC has priority and the SYNCFLG bit is set.
5	SYNCFRC	RS	0h	Sync Force Bit: Writing a 1 to this bit latches a sync event and sets the SYNCFLG bit. This bit can be used like a software sync for the wrap counter.
4	PERINTCLR	RS	0h	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	RS	0h	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	RS	0h	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	RS	0h	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 8-5 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.

Table 8-10. CONTROL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
0	RUN	RS	0h	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 8-5 for the various positions the state machine can be at when HALTED.

8.9.8 BURST_SIZE Register (Offset = 1022h + [i * E3h]) [reset = 0h]

BURST_SIZE is shown in Figure 8-15 and described in Table 8-11.

Figure 8-15. BURST_SIZE Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			BURSTSIZE				
R-0h			R/W-0h				

Table 8-11. BURST_SIZE Register Field Descriptions

Bit	Field	Type	Reset	Description
15-5	RESERVED	R	0h	Reserved
4-0	BURSTSIZE	R/W	0h	These bits specify the burst transfer size: 0h = Transfer 1 word in a burst 1h = Transfer 2 words in a burst ... 31h = Transfer 32 words in a burst

8.9.9 BURST_COUNT Register (Offset = 1023h + [i * E3h]) [reset = 0h]

BURST_COUNT is shown in Figure 8-16 and described in Table 8-12.

Figure 8-16. BURST_COUNT Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			BURSTCOUNT				
R-0h			R/W-0h				

Table 8-12. BURST_COUNT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-5	RESERVED	R	0h	Reserved
4-0	BURSTCOUNT	R/W	0h	These bits indicate the current burst counter value: 0h = 0 word left in a burst 1h = 1 word left in a burst 2h = 2 words left in a burst ... 31h = 31 words left in a burst The above values represent the state of the counter at the HALT conditions.

8.9.10 SRC_BURST_STEP Register (Offset = 1024h + [i * E3h]) [reset = 0h]

SRC_BURST_STEP is shown in Figure 8-17 and described in Table 8-13.

Figure 8-17. SRC_BURST_STEP Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRCBURSTSTEP															
R/W-0h															

Table 8-13. SRC_BURST_STEP Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	SRCBURSTSTEP	R/W	0h	These bits specify the source address post-increment/decrement step size while processing a burst of data: Only values from -4096 to 4095 are valid. 0000h = No address change 0001h = Add 1 to address 0002h = Add 2 to address 0FFFh = Add 4095 to address F000h = Sub 4096 from address FFFEh = Sub 2 from address FFFFh = Sub 1 from address

8.9.11 DST_BURST_STEP Register (Offset = 1025h + [i * E3h]) [reset = 0h]

DST_BURST_STEP is shown in Figure 8-18 and described in Table 8-14.

Figure 8-18. DST_BURST_STEP Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DSTBURSTSTEP															
R/W-0h															

Table 8-14. DST_BURST_STEP Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	DSTBURSTSTEP	R/W	0h	These bits specify the destination address post-increment/decrement step size while processing a burst of data: Only values from -4096 to 4095 are valid. 0000h = No address change 0001h = Add 1 to address 0002h = Add 2 to address 0FFFh = Add 4095 to address F000h = Sub 4096 from address FFFEh = Sub 2 from address FFFFh = Sub 1 from address

8.9.12 TRANSFER_SIZE Register (Offset = 1026h + [i * E3h]) [reset = 0h]

TRANSFER_SIZE is shown in Figure 8-19 and described in Table 8-15.

Figure 8-19. TRANSFER_SIZE Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRANSFERSIZE															
R/W-0h															

Table 8-15. TRANSFER_SIZE Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	TRANSFERSIZE	R/W	0h	These bits specify the number of bursts to transfer: 0000h = Transfer 1 burst 0001h = Transfer 2 bursts 0002h = Transfer 3 bursts ... FFFFh = Transfer 65536 bursts

8.9.13 TRANSFER_COUNT Register (Offset = 1027h + [i * E3h]) [reset = 0h]

TRANSFER_COUNT is shown in Figure 8-20 and described in Table 8-16.

Figure 8-20. TRANSFER_COUNT Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRANSFERCOUNT															
R/W-0h															

Table 8-16. TRANSFER_COUNT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	TRANSFERCOUNT	R/W	0h	These bits specify the current transfer counter value: 0000h = 0 bursts left to transfer 0001h = 1 burst left to transfer 0002h = 2 bursts left to transfer ... FFFFh = 65535 bursts left to transfer The above values represent the state of the counter at the HALT conditions.

8.9.14 SRC_TRANSFER_STEP Register (Offset = 1028h + [i * E3h]) [reset = 0h]

SRC_TRANSFER_STEP is shown in Figure 8-21 and described in Table 8-17.

Figure 8-21. SRC_TRANSFER_STEP Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRCTRANSFERSTEP															
R/W-0h															

Table 8-17. SRC_TRANSFER_STEP Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	SRCTRANSFERSTEP	R/W	0h	These bits specify the source address pointer post-increment/decrement step size after processing a burst of data: Only values from -4096 to 4095 are valid. 0000h = No address change 0001h = Add 1 to address 0002h = Add 2 to address F000h = Sub 4096 from address FFFEh = Sub 2 from address FFFFh = Sub 1 from address

8.9.15 DST_TRANSFER_STEP Register (Offset = 1029h + [i * E3h]) [reset = 0h]

DST_TRANSFER_STEP is shown in Figure 8-22 and described in Table 8-18.

Figure 8-22. DST_TRANSFER_STEP Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DSTTRANSFERSTEP															
R/W-0h															

Table 8-18. DST_TRANSFER_STEP Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	DSTTRANSFERSTEP	R/W	0h	These bits specify the destination address pointer post-increment/decrement step size after processing a burst of data: Only values from -4096 to 4095 are valid. 0000h = No address change 0001h = Add 1 to address 0002h = Add 2 to address 0FFFh = Add 4095 to address F000h = Sub 4096 from address FFFEh = Sub 2 from address FFFFh = Sub 1 from address

8.9.16 SRC_WRAP_SIZE Register (Offset = 102Ah + [i * E3h]) [reset = 0h]

SRC_WRAP_SIZE is shown in Figure 8-23 and described in Table 8-19.

Figure 8-23. SRC_WRAP_SIZE Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRAPSIZE															
R/W-0h															

Table 8-19. SRC_WRAP_SIZE Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	WRAPSIZE	R/W	0h	These bits specify the number of bursts to transfer before wrapping back to begin address pointer: ... To *disable* the wrap function, set the WRAPSIZE bit field to a number larger than the TRANSFERSIZE bit field. 0000h = Wrap after 1 burst 0001h = Wrap after 2 bursts 0002h = Wrap after 3 bursts FFFFh = Wrap after 65536 bursts

8.9.17 SRC_WRAP_COUNT Register (Offset = 102Bh + [i * E3h]) [reset = 0h]

SRC_WRAP_COUNT is shown in Figure 8-24 and described in Table 8-20.

Figure 8-24. SRC_WRAP_COUNT Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRAPCOUNT															
R/W-0h															

Table 8-20. SRC_WRAP_COUNT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	WRAPCOUNT	R/W	0h	These bits indicate the current wrap counter value: 0000h = Wrap complete 0001h = 1 burst left 0002h = 2 burst left .... FFFFh = 65535 burst left The above values represent the state of the counter at the HALT conditions.

8.9.18 SRC_WRAP_STEP Register (Offset = 102Ch + [i * E3h]) [reset = 0h]

SRC_WRAP_STEP is shown in Figure 8-25 and described in Table 8-21.

Figure 8-25. SRC_WRAP_STEP Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRAPSTEP															
R/W-0h															

Table 8-21. SRC_WRAP_STEP Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	WRAPSTEP	R/W	0h	These bits specify the source begin address pointer post-increment/decrement step size after wrap counter expires: ... Only values from -4096 to 4095 are valid. 0000h = No address change 0001h = Add 1 to address 0002h = Add 2 to address 0FFFh = Add 4095 to address F000h = Sub 4096 from address FFFEh = Sub 2 from address FFFFh = Sub 1 from address

8.9.19 DST_WRAP_SIZE Register (Offset = 102Dh + [i * E3h]) [reset = 0h]

DST_WRAP_SIZE is shown in Figure 8-26 and described in Table 8-22.

Figure 8-26. DST_WRAP_SIZE Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRAPSIZE															
R/W-0h															

Table 8-22. DST_WRAP_SIZE Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	WRAPSIZE	R/W	0h	These bits specify the number of bursts to transfer before wrapping back to begin address pointer: ... To *disable* the wrap function, set the WRAPSIZE bit field to a number larger than the TRANSFERSIZE bit field. 0000h = Wrap after 1 burst 0001h = Wrap after 2 bursts 0002h = Wrap after 3 bursts FFFFh = Wrap after 65536 bursts

8.9.20 DST_WRAP_COUNT Register (Offset = 102Eh + [i * E3h]) [reset = 0h]

DST_WRAP_COUNT is shown in Figure 8-27 and described in Table 8-23.

Figure 8-27. DST_WRAP_COUNT Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRAPCOUNT															
R/W-0h															

Table 8-23. DST_WRAP_COUNT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	WRAPCOUNT	R/W	0h	These bits indicate the current wrap counter value: 0000h = Wrap complete 0001h = 1 burst left 0002h = 2 burst left ... FFFFh = 65535 burst left The above values represent the state of the counter at the HALT conditions.

8.9.21 DST_WRAP_STEP Register (Offset = 102Fh + [i * E3h]) [reset = 0h]

DST_WRAP_STEP is shown in Figure 8-28 and described in Table 8-24.

Figure 8-28. DST_WRAP_STEP Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRAPSTEP															
R/W-0h															

Table 8-24. DST_WRAP_STEP Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	WRAPSTEP	R/W	0h	These bits specify the source begin address pointer post-increment/decrement step size after wrap counter expires: ... ... Only values from -4096 to 4095 are valid. 0000h = No address change 0001h = Add 1 to address 0002h = Add 2 to address 0FFFh = Add 4095 to address F000h = Sub 4096 from address FFFEh = Sub 2 from address FFFFh = Sub 1 from address

8.9.22 SRC_BEG_ADDR_SHADOW Register (Offset = 1030h + [i * E3h]) [reset = 0h]

SRC_BEG_ADDR_SHADOW is shown in Figure 8-29 and described in Table 8-25.

Figure 8-29. SRC_BEG_ADDR_SHADOW Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										BEGADDR																					
R-0h										R/W-0h																					

Table 8-25. SRC_BEG_ADDR_SHADOW Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	BEGADDR	R/W	0h	22-bit address value

8.9.23 SRC_ADDR_SHADOW Register (Offset = 1032h + [i * E3h]) [reset = 0h]

SRC_ADDR_SHADOW is shown in Figure 8-30 and described in Table 8-26.

Figure 8-30. SRC_ADDR_SHADOW Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										ADDR																					
R-0h										R/W-0h																					

Table 8-26. SRC_ADDR_SHADOW Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	ADDR	R/W	0h	22-bit address value

8.9.24 SRC_BEG_ADDR Register (Offset = 1034h + [i * E3h]) [reset = 0h]

SRC_BEG_ADDR is shown in Figure 8-31 and described in Table 8-27.

Figure 8-31. SRC_BEG_ADDR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										BEGADDR																					
R-0h										R-0h																					

Table 8-27. SRC_BEG_ADDR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	BEGADDR	R	0h	22-bit address value

8.9.25 SRC_ADDR Register (Offset = 1036h + [i * E3h]) [reset = 0h]

SRC_ADDR is shown in Figure 8-32 and described in Table 8-28.

Figure 8-32. SRC_ADDR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										ADDR																					
R-0h										R-0h																					

Table 8-28. SRC_ADDR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	ADDR	R	0h	22-bit address value

8.9.26 DST_BEG_ADDR_SHADOW Register (Offset = 1038h + [i * E3h]) [reset = 0h]

DST_BEG_ADDR_SHADOW is shown in Figure 8-33 and described in Table 8-29.

Figure 8-33. DST_BEG_ADDR_SHADOW Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										BEGADDR																					
R-0h										R/W-0h																					

Table 8-29. DST_BEG_ADDR_SHADOW Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	BEGADDR	R/W	0h	22-bit address value

8.9.27 DST_ADDR_SHADOW Register (Offset = 103Ah + [i * E3h]) [reset = 0h]

DST_ADDR_SHADOW is shown in Figure 8-34 and described in Table 8-30.

Figure 8-34. DST_ADDR_SHADOW Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										ADDR																					
R-0h										R/W-0h																					

Table 8-30. DST_ADDR_SHADOW Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	ADDR	R/W	0h	22-bit address value

8.9.28 DST_BEG_ADDR Register (Offset = 103Ch + [i * E3h]) [reset = 0h]

DST_BEG_ADDR is shown in Figure 8-35 and described in Table 8-31.

Figure 8-35. DST_BEG_ADDR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										BEGADDR																					
R-0h										R-0h																					

Table 8-31. DST_BEG_ADDR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	BEGADDR	R	0h	22-bit address value

8.9.29 DST_ADDR Register (Offset = 103Eh + [i * E3h]) [reset = 0h]

DST_ADDR is shown in Figure 8-36 and described in Table 8-32.

Figure 8-36. DST_ADDR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										ADDR																					
R-0h										R-0h																					

Table 8-32. DST_ADDR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-22	RESERVED	R	0h	Reserved
21-0	ADDR	R	0h	22-bit address value

Serial Peripheral Interface (SPI)

This chapter describes the serial peripheral interface (SPI) which is a high-speed synchronous serial input and output (I/O) port that allows a serial bit stream of programmed length (one to 16 bits) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI is normally used for communications between the MCU controller and external peripherals or another controller. Typical applications include external I/O or peripheral expansion via devices such as shift registers, display drivers, and analog-to-digital converters (ADCs). Multi-device communications are supported by the master or slave operation of the SPI. The port supports a 16-level, receive and transmit FIFO for reducing CPU servicing overhead.

Topic **Page**

9.1 Introduction

9.1.1 Features

The SPI module features include:

- SPISOMI: SPI slave-output/master-input pin
- SPISIMO: SPI slave-input/master-output pin
- $\overline{\text{SPISTE}}$: SPI slave transmit-enable pin
- SPICLK: SPI serial-clock pin

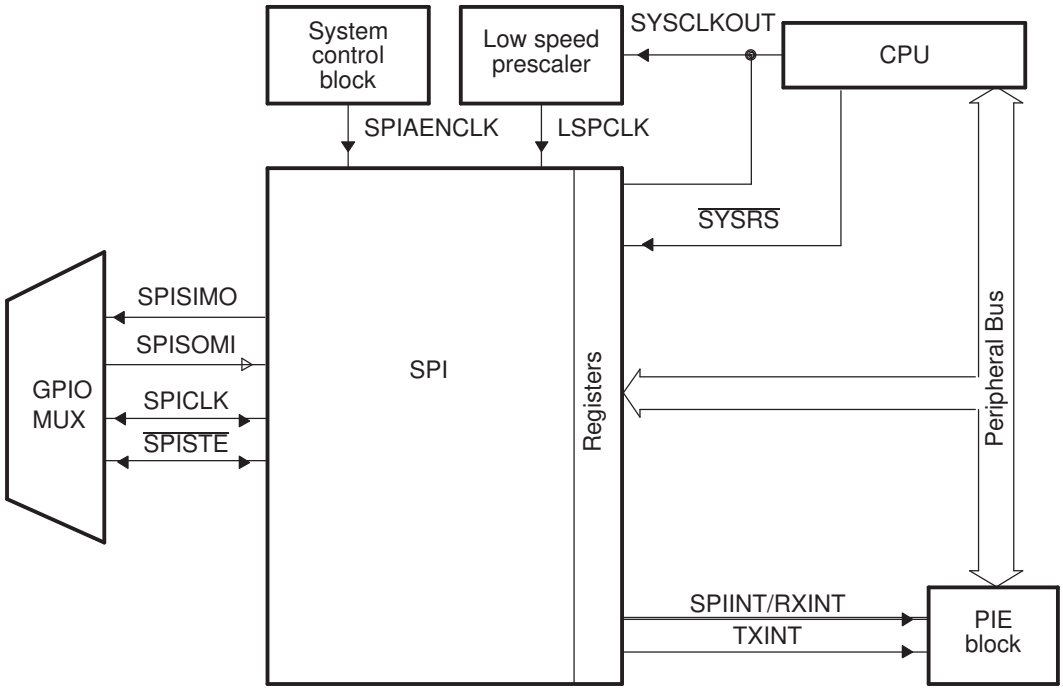
NOTE: All four pins can be used as GPIO if the SPI module is not used.

- Two operational modes: Master and Slave
- Baud rate: 125 different programmable rates. The maximum baud rate that can be employed is limited by the maximum speed of the I/O buffers used on the SPI pins. See the device-specific data manual 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 rising edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.
- Simultaneous receive and transmit operation (transmit function can be disabled in software)
- Transmitter and receiver operations are accomplished through either interrupt- driven or polled algorithm
- 16-level transmit/receive FIFO
- Delayed transmit control

9.1.2 Block Diagram

Figure 9-1 shows the SPI CPU interfaces.

Figure 9-1. SPI CPU Interface



9.2 System-Level Integration

This section describes the various functionality that is applicable to the device integration. These features require configuration of other modules in the device that are not within the scope of this chapter.

9.2.1 SPI Module Signals

Table 9-1 classifies and provides a summary of the SPI module signals.

Table 9-1. SPI Module Signal Summary

Signal Name	Description
External Signals	
SPICLK	SPI clock
SPISIMO	SPI slave in, master out
SPISOMI	SPI slave out, master in
$\overline{\text{SPISTE}}$	SPI slave transmit enable
Control	
SPI Clock Rate	LSPCLK
Interrupt Signals	
SPIINT/SPIRXINT	Transmit interrupt/ Receive Interrupt in non FIFO mode (referred to as SPIINT)
	Receive interrupt in FIFO mode
SPITXINT	Transmit interrupt in FIFO mode

Special Considerations

The $\overline{\text{SPISTE}}$ signal provides the ability to gate any spurious clock and data pulses when the SPI is in slave mode. An active $\overline{\text{SPISTE}}$ will not allow the slave to receive data. This prevents the SPI slave from losing synchronization with the master. It is this reason that TI does not recommend that the $\overline{\text{SPISTE}}$ always be tied to the active state.

If the SPI slave does ever lose synchronization with the master, toggling SPISWRESET will reset internal bit counter as well as the various status flags in the module. By resetting the bit counter, the SPI will interpret the next clock transition as the first bit of a new transmission. The register bit fields which are reset by SPISWRESET can be found in Section 9.5

Configuring a GPIO to emulate $\overline{\text{SPISTE}}$

In many systems, a SPI master may be connected to multiple SPI slaves using multiple instances of $\overline{\text{SPISTE}}$. Though this SPI module does not natively support multiple $\overline{\text{SPISTE}}$ signals, it is possible to emulate this behavior in software using GPIOs. In this configuration, the SPI must be configured as the master. Rather than using the GPIO Mux to select $\overline{\text{SPISTE}}$, the application would configure pins to be GPIO outputs, one GPIO per SPI slave. Before transmitting any data, the application would drive the desired GPIO to the active state. Immediately after the transmission has been completed, the GPIO chip select would be driven to the inactive state. This process can be repeated for many slaves which share the SPICLK, SPISIMO, and SPISOMI lines.

9.2.2 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPyPUD register.

See the *GPIO* chapter for more details on GPIO mux and settings.

9.2.3 SPI Interrupts

This section includes information on the available interrupts present in the SPI module.

The SPI module contains two interrupt lines: SPIINT/SPIRXINT and SPITXINT. When the SPI is operating in non-FIFO mode, all available interrupts are routed together to generate the single SPIINT interrupt. When FIFO mode is used, both SPIRXINT and SPITXINT can be generated.

SPIINT/SPIRXINT

When the SPI is operating in non-FIFO mode, the interrupt generated is called SPIINT. If FIFO enhancements are enabled, the interrupt is called SPIRXINT. These interrupts share the same interrupt vector in the Peripheral Interrupt Expansion (PIE) block.

In non-FIFO mode, two conditions can trigger an interrupt: a transmission is complete (INT_FLAG), or there is overrun in the receiver (OVERRUN_FLAG). Both of these conditions share the same interrupt vector: SPIINT.

The transmission complete flag (INT_FLAG) indicates that the SPI has completed sending or receiving the last bit and is ready to be serviced. At the same time this bit is set, the received character is placed in the receiver buffer (SPIRXBUF). The INT_FLAG will generate an interrupt on the SPIINT vector if the SPIINTENA bit is set.

The receiver overrun flag (OVERRUN_FLAG) indicates that a transmit or receive operation has completed before the previous character has been read from the buffer. The OVERRUN_FLAG will generate an interrupt on the SPIINT vector if the OVERRUNINTENA bit is set and OVERRUN_FLAG was previously cleared.

In FIFO mode, the SPI can interrupt the CPU upon a match condition between the current receive FIFO status (RXFFST) and the receive FIFO interrupt level (RXFFIL). If RXFFST is greater than or equal to RXFFIL, the receive FIFO interrupt flag (RXFFINT) will be set. SPIRXINT will be triggered in the PIE block if RXFFINT is set and the receive FIFO interrupt is enabled (RXFFIENA = 1).

SPITXINT

The SPITXINT interrupt is not available when the SPI is operating in non-FIFO mode.

In FIFO mode, the SPITXINT behavior is similar to the SPIRXINT. SPITXINT is generated upon a match condition between the current transmit FIFO status (TXFFST) and the transmit FIFO interrupt level (TXFFIL). If TXFFST is less than or equal to TXFFIL, the transmit FIFO interrupt flag (TXFFINT) will be set. SPITXINT will be triggered in the PIE block if TXFFINT is set and the transmit FIFO interrupt is enabled in the SPI module (TXFFIENA = 1).

Figure 9-2 and Table 9-2 show how these control bits influence the SPI interrupt generation.

Figure 9-2. SPI Interrupt Flags and Enable Logic Generation

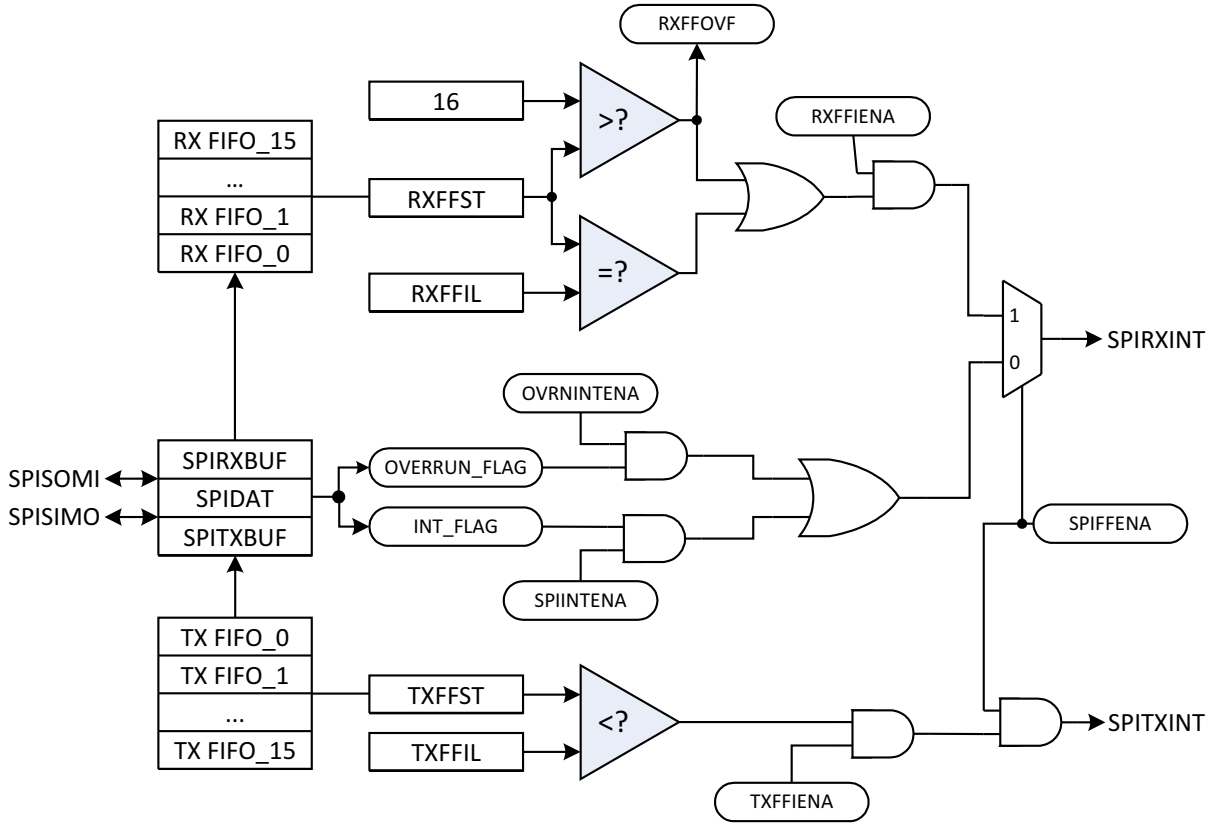


Table 9-2. SPI Interrupt Flag Modes

FIFO Options	SPI Interrupt Source	Interrupt Flags	Interrupt Enables	FIFO Enable (SPIFFENA)	Interrupt Line [1]
SPI without FIFO	Receive overrun	RXOVRN	OVRNINTENA	0	SPIRXINT
	Data receive	SPIINT	SPIINTENA	0	SPIRXINT
	Transmit empty	SPIINT	SPIINTENA	0	SPIRXINT
SPI FIFO mode	FIFO receive	RXFFIL	RXFFIENA	1	SPIRXINT
	Transmit empty	TXFFIL	TXFFIENA	1	SPITXINT

(1) In non-FIFO mode, SPIRXINT is the same name as the SPIINT interrupt in 28x devices.

9.3 SPI Operation

This section describes the various modes of operation of the SPI. Included are explanations of the operational modes, interrupts, data format, clock sources, and initialization. Typical timing diagrams for data transfers are given.

9.3.1 Introduction to Operation

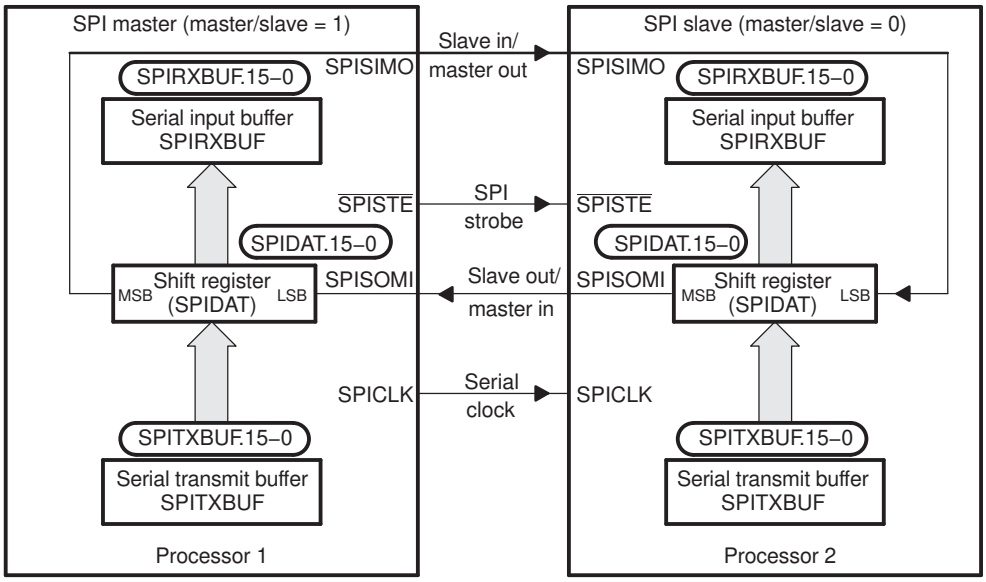
Figure 9-3 shows typical connections of the SPI for communications between two controllers: a master and a slave.

The master transfers data by sending the SPICLK signal. For both the slave and the master, data is shifted out of the shift registers on one edge of the SPICLK and latched into the shift register on the opposite SPICLK clock edge. If the CLK_PHASE bit is high, data is transmitted and received a half-cycle before the SPICLK transition. As a result, both controllers send and receive data simultaneously. The application software determines whether the data is meaningful or dummy data. There are three possible methods for data transmission:

- Master sends data; slave sends dummy data.
- Master sends data; slave sends data.
- Master sends dummy data; slave sends data.

The master can initiate data transfer at any time because it controls the SPICLK signal. The software, however, determines how the master detects when the slave is ready to broadcast data.

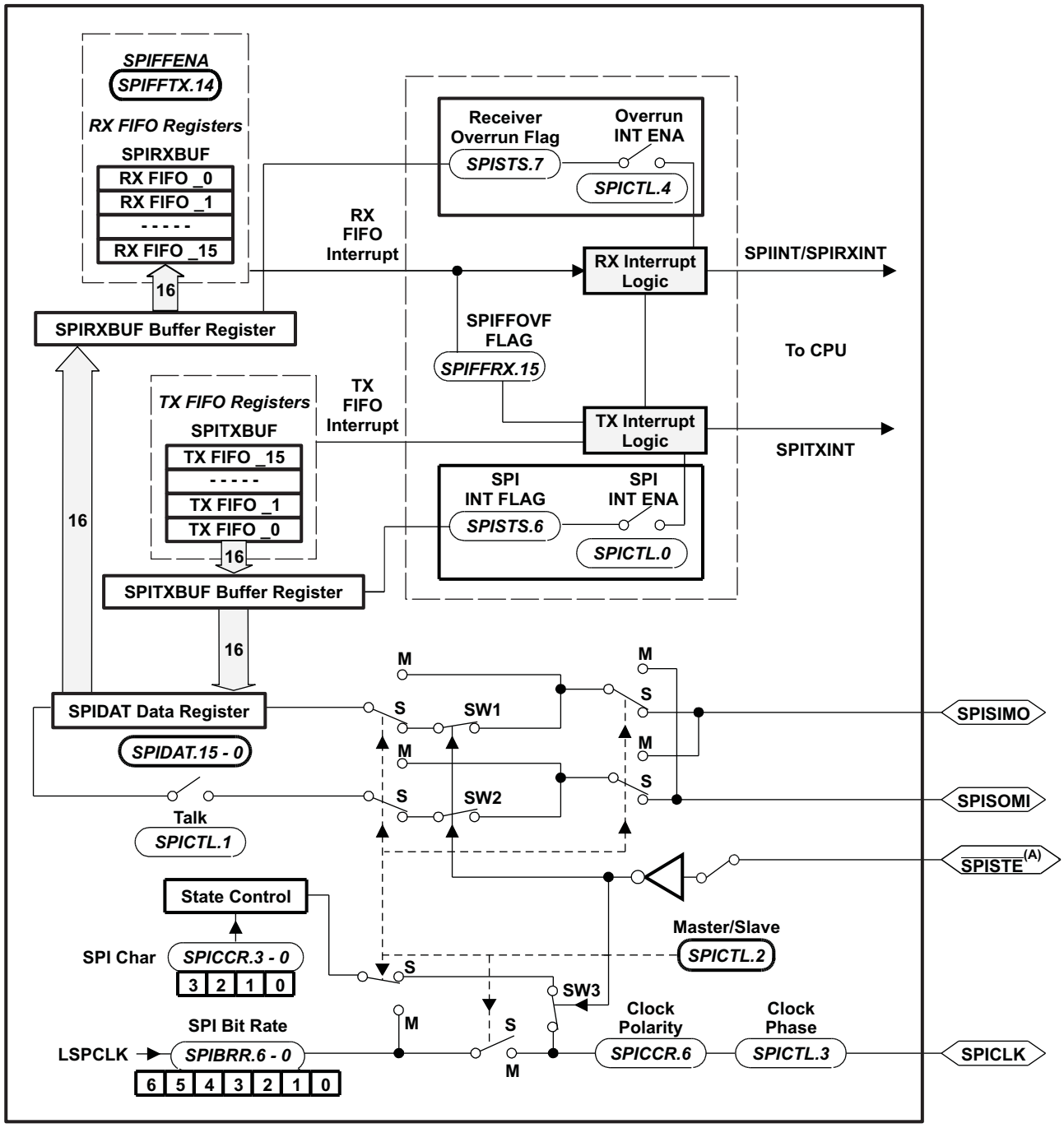
Figure 9-3. SPI Master/Slave Connection



The SPI can operate in master or slave mode. The MASTER_SLAVE bit selects the operating mode and the source of the SPICLK signal.

Figure 9-4 is a block diagram of the SPI module showing all of the basic control blocks available on the SPI module.

Figure 9-4. Serial Peripheral Interface Block Diagram



9.3.2 Master Mode

In master mode (MASTER_SLAVE = 1), the SPI provides the serial clock on the SPICLK pin for the entire serial communications network. Data is output on the SPISIMO pin and latched from the SPISOMI pin.

The SPIBRR register determines both the transmit and receive bit transfer rate for the network. SPIBRR can select 125 different data transfer rates.

Data written to SPIDAT or SPITXBUF initiates data transmission on the SPISIMO pin, MSB (most significant bit) first. Simultaneously, received data is shifted through the SPISOMI pin into the LSB (least significant bit) of SPIDAT. When the selected number of bits has been transmitted, the received data is transferred to the SPIRXBUF (buffered receiver) for the CPU to read. Data is stored right-justified in SPIRXBUF.

When the specified number of data bits has been shifted through SPIDAT, the following events occur:

- SPIDAT contents are transferred to SPIRXBUF.
- INT_FLAG bit is set to 1.
- If there is valid data in the transmit buffer SPITXBUF, as indicated by the Transmit Buffer Full Flag (BUFFULL_FLAG), this data is transferred to SPIDAT and is transmitted; otherwise, SPICLK stops after all bits have been shifted out of SPIDAT.
- If the SPIINTENA bit is set to 1, an interrupt is asserted.

In a typical application, the $\overline{\text{SPISTE}}$ pin serves as a chip-enable pin for a 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.

9.3.3 Slave Mode

In slave mode (MASTER_SLAVE = 0), data shifts out on the SPISOMI pin and in on the SPISIMO pin. The SPICLK pin is used as the input for the serial shift clock, which is supplied from the external network master. The transfer rate is defined by this clock. The SPICLK input frequency should be no greater than the LSPCLK frequency divided by 4.

Data written to SPIDAT or SPITXBUF is transmitted to the network when appropriate edges of the SPICLK signal are received from the network master. 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 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.

9.3.4 Data Format

The four-bit SPICHAR register field specifies the number of bits in the data character (1 to 16). This information directs the state control logic to count the number of bits received or transmitted to determine when a complete character has been processed.

The following statements apply to characters with fewer than 16 bits:

- Data must be left-justified when written to SPIDAT and SPITXBUF.
- Data read back from SPIRXBUF is right-justified.
- SPIRXBUF contains the most recently received character, right-justified, plus any bits that remain from previous transmission(s) that have been shifted to the left (shown in Example 9-1).

Example 9-1. Transmission of Bit From SPIRXBUF

Conditions:

1. Transmission character length = 1 bit (specified in bits SPICHAR)
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.

9.3.5 Baud Rate Selection

The SPI module supports 125 different baud rates and four different clock schemes. Depending on whether the SPI clock is in slave or master mode, the SPICLK pin can receive an external SPI clock signal or provide the SPI clock signal, respectively.

- In the slave mode, the SPI clock is received on the SPICLK pin from the external source, and can be no greater than the LSPCLK frequency divided by 4.
- In the master mode, the SPI clock is generated by the SPI and is output on the SPICLK pin, and can be no greater than the LSPCLK frequency divided by 4.

NOTE: The baud rate should be configured to not exceed the maximum rated GPIO toggle frequency. Refer to the device Data Manual for the maximum GPIO toggle frequency

Example 9-2 shows how to determine the SPI baud rates.

Example 9-2. Baud Rate Determination

For SPIBRR = 3 to 127:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{(\text{SPIBRR} + 1)} \quad (8)$$

For SPIBRR = 0, 1, or 2:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{4} \quad (9)$$

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 9-3 shows how to calculate the baud rate of the SPI module.

Example 9-3. Baud Rate Calculation

$$\text{Maximum SPI Baud Rate} = \frac{\text{LSPCLK}}{4} = \frac{40 \times 10^6}{4} = 10 \times 10^6 \text{ bps} \quad (10)$$

9.3.6 SPI Clocking Schemes

The clock polarity select bit (CLKPOLARITY) and the clock phase select bit (CLK_PHASE) control four different clocking schemes on the SPICLK pin. CLKPOLARITY selects the active edge, either rising or falling, of the clock. CLK_PHASE selects a half-cycle delay of the clock. The four different clocking schemes are as follows:

- Falling Edge Without Delay. The SPI transmits data on the falling edge of the SPICLK and receives data on the rising edge of the SPICLK.
- Falling Edge With Delay. The SPI transmits data one half-cycle ahead of the falling edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
- Rising Edge Without Delay. The SPI transmits data on the rising edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
- Rising Edge With Delay. The SPI transmits data one half-cycle ahead of the rising edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.

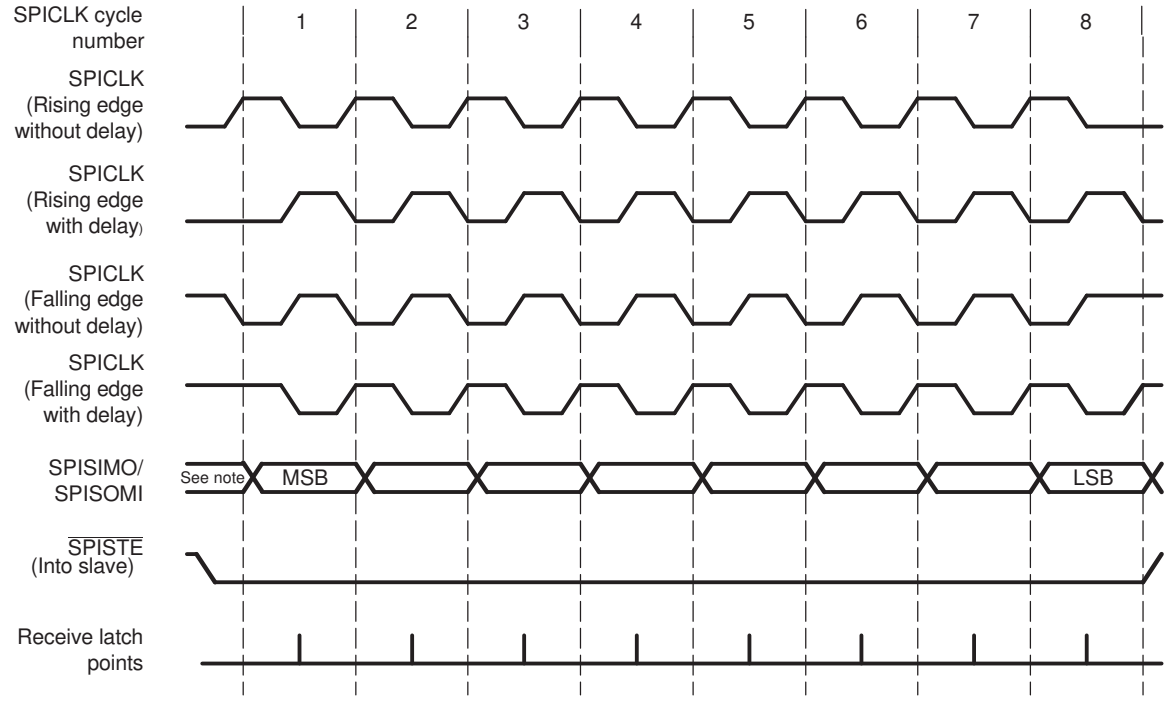
The selection procedure for the SPI clocking scheme is shown in Table 9-3. Examples of these four clocking schemes relative to transmitted and received data are shown in Figure 9-5.

Table 9-3. SPI Clocking Scheme Selection Guide

SPICLK Scheme	CLKPOLARITY	CLK_PHASE [1]
Rising edge without delay	0	0
Rising edge with delay	0	1
Falling edge without delay	1	0
Falling edge with delay	1	1

(1) The description of CLK_PHASE and CLKPOLARITY differs between manufacturers. For proper operation, select the desired waveform to determine the clock phase and clock polarity settings.

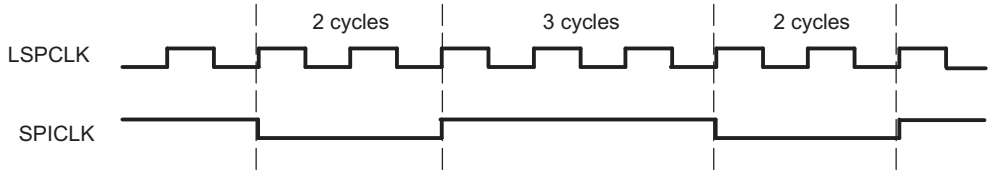
Figure 9-5. SPICLK Signal Options



Note: Previous data bit

SPICLK symmetry is retained only when the result of (SPIBRR+1) is an even value. When (SPIBRR + 1) is an odd value and SPIBRR is greater than 3, SPICLK becomes asymmetrical. The low pulse of SPICLK is one LSPCLK cycle longer than the high pulse when CLKPOLARITY bit is clear (0). When CLKPOLARITY bit is set to 1, the high pulse of the SPICLK is one LSPCLK cycle longer than the low pulse, as shown in Figure 9-6.

Figure 9-6. SPI: SPICLK-LSPCLK Characteristic When (BRR + 1) is Odd, BRR > 3, and CLKPOLARITY = 1



9.3.7 SPI FIFO Description

The following steps explain the FIFO features and help with programming the SPI FIFOs:

1. **Reset.** At reset the SPI powers up in standard SPI mode and the FIFO function is disabled. The FIFO registers SPIFFTX, SPIFFRX and SPIFFCT remain inactive.
2. **Standard SPI.** The standard 28x SPI mode will work with SPIINT/SPIRXINT as the interrupt source.
3. **Mode change.** FIFO mode is enabled by setting the SPIFFENA bit to 1 in the SPIFFTX register. SPIRST can reset the FIFO mode at any stage of its operation.
4. **Active registers.** All the SPI registers and SPI FIFO registers SPIFFTX, SPIFFRX, and SPIFFCT will be active.
5. **Interrupts.** FIFO mode has two interrupts one for the transmit FIFO, SPITXINT and one for the receive FIFO, SPIRXINT. SPIRXINT is the common interrupt for SPI FIFO receive, receive error and receive FIFO overflow conditions. The single SPIINT for both transmit and receive sections of the standard SPI will be disabled and this interrupt will service as SPI receive FIFO interrupt. For more information, refer to Section 9.2.3
6. **Buffers.** Transmit and receive buffers are each supplemented with a 16 word FIFO. The one-word transmit buffer (SPITXBUF) of the standard SPI functions as a transition buffer between the transmit FIFO and shift register. The one-word transmit buffer will be loaded from transmit FIFO only after the last bit of the shift register is shifted out.
7. **Delayed transfer.** The rate at which transmit words in the FIFO are transferred to transmit shift register is programmable. The SPIFFCT register bits (7−0) 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 SPICLK cycles and a maximum of 255 SPICLK cycles. With zero delay, the SPI module can transmit data in continuous mode with the FIFO words shifting out back to back. With the 255 clock delay, the SPI module can transmit data in a maximum delayed mode with the FIFO words shifting out with a delay of 255 SPICLK cycles between each words. The programmable delay facilitates glueless interface to various slow SPI peripherals, such as EEPROMs, ADC, DAC, and so on.
8. **FIFO status bits.** Both transmit and receive FIFOs have status bits TXFFST or RXFFST that define the number of words available in the FIFOs at any time. The transmit FIFO reset bit (TXFIFO) and receive reset bit (RXFIFO) will reset the FIFO pointers to zero when these bits are set to 1. The FIFOs will resume operation from start once these bits are cleared to zero.
9. **Programmable interrupt levels.** Both transmit and receive FIFOs can generate CPU interrupts. The transmit interrupt (SPITXINT) is generated whenever the transmit FIFO status bits (TXFFST) match (less than or equal to) the interrupt trigger level bits (TXFFIL). The receive interrupt (SPIRXINT) is generated whenever the receive FIFO status bits (RXFFST) match (greater than or equal to) the interrupt trigger level RXFFIL. This provides a programmable interrupt trigger for transmit and receive sections of the SPI. The default value for these trigger level bits will be 0x11111 for receive FIFO and 0x00000 for transmit FIFO, respectively.

9.4 Programming Procedure

This section describes the procedure for configuring the SPI for the various modes of operation.

9.4.1 Initialization Upon Reset

A system reset forces the SPI peripheral into the following default configuration:

- Unit is configured as a slave module (MASTER_SLAVE = 0)
- Transmit capability is disabled (TALK = 0)
- Data is latched at the input on the falling edge of the SPICLK signal
- Character length is assumed to be one bit
- SPI interrupts are disabled
- Data in SPIDAT is reset to 0000h

9.4.2 Configuring the SPI

This section describes the procedure in which to configure the SPI module for operation. To prevent unwanted and unforeseen events from occurring during or as a result of initialization changes, clear the SPISWRESET bit before making initialization changes, and then set this bit after initialization is complete. While the SPI is held in reset (SPISWRESET = 0), configuration may be changed in any order. The following list shows the SPI configuration procedure in a logical order. However, the SPI registers can be written with single 16-bit writes, so the order is not required with the exception of SPISWRESET.

To change the SPI configuration:

Step 1. Clear the SPI Software Reset bit (SPISWRESET) to 0 to force the SPI to the reset state.

Step 2. Configure the SPI as desired:

- Select either master or slave mode (MASTER_SLAVE).
- Choose SPICLK polarity and phase (CLKPOLARITY and CLK_PHASE).
- Set the desired baud rate (SPIBRR).
- Set the SPI character length (SPICHAR).
- Clear the SPI Flags (OVERRUN_FLAG, INT_FLAG).
- If using FIFO enhancements:
  - Enable the FIFO enhancements (SPIFFENA).
  - Clear the FIFO Flags (TXFFINTCLR, RXFFOVFCLR, and RXFFINTCLR).
  - Release transmit and receive FIFO resets (TXFIFO and RXFIFORESET).
  - Release SPI FIFO channels from reset (SPIRST).

Step 3. If interrupts are used:

- In non-FIFO mode, enable the receiver overrun and/or SPI interrupts (OVERRUNINTENA and SPIINTENA).
- In FIFO mode, set the transmit and receive interrupt levels (TXFFIL and RXFFIL) then enable the interrupts (TXFFIENA and RXFFIENA).

Step 4. Set SPISWRESET to 1 to release the SPI from the reset state.

NOTE: Do not change the SPI configuration when communication is in progress.

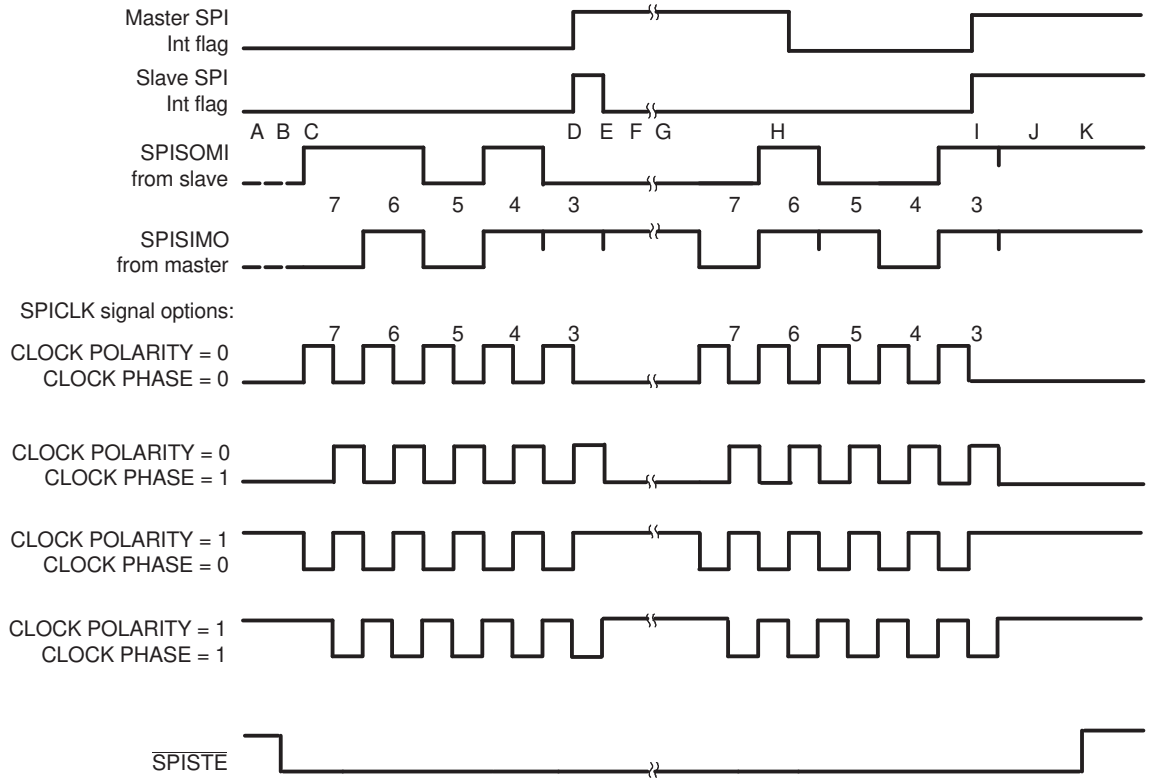
9.4.3 Data Transfer Example

The timing diagram shown in Figure 9-7 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 asymmetrical (Figure 9-6) shares similar characterizations with Figure 9-7 except that the data transfer is one LSPCLK cycle longer per bit during the low pulse (CLKPOLARITY = 0) or during the high pulse (CLKPOLARITY = 1) of the SPICLK.

Figure 9-7 is applicable for 8-bit SPI only and is not for C28x devices that are capable of working with 16-bit data. The figure is shown for illustrative purposes only.

Figure 9-7. 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).

9.5 SPI Registers

This section describes the Serial Peripheral Interface registers. It is important to note that the SPI registers only allow 16-bit accesses.

9.5.1 SPI Base Addresses

Table 9-4. SPI Base Address Table

Bit Field Name		Base Address
Instance	Structure	
SpiaRegs	SPI_REGS	0x0000_7040

9.5.2 SPI_REGS Registers

Table 9-5 lists the SPI_REGS registers. All register offset addresses not listed in Table 9-5 should be considered as reserved locations and the register contents should not be modified.

Table 9-5. SPI_REGS Registers

Offset	Acronym	Register Name	Write Protection	Section
0h	SPICCR	SPI Configuration Control Register		Go
1h	SPICTL	SPI Operation Control Register		Go
2h	SPISTS	SPI Status Register		Go
4h	SPIBRR	SPI Baud Rate Register		Go
6h	SPIRXEMU	SPI Emulation Buffer Register		Go
7h	SPIRXBUF	SPI Serial Input Buffer Register		Go
8h	SPITXBUF	SPI Serial Output Buffer Register		Go
9h	SPIDAT	SPI Serial Data Register		Go
Ah	SPIFFTX	SPI FIFO Transmit Register		Go
Bh	SPIFFRX	SPI FIFO Receive Register		Go
Ch	SPIFFCT	SPI FIFO Control Register		Go
Fh	SPIPRI	SPI Priority Control Register		Go

Complex bit access types are encoded to fit into small table cells. Table 9-6 shows the codes that are used for access types in this section.

Table 9-6. SPI_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
RC	R C	Read to Clear
Write Type		
W	W	Write
W1C	W 1C	Write 1 to clear
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

9.5.2.1 SPICCR Register (Offset = 0h) [reset = 0h]

SPICCR is shown in Figure 9-8 and described in Table 9-7.

Return to the Summary Table.

SPICCR controls the setup of the SPI for operation.

Figure 9-8. SPICCR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
SPISWRESET	CLKPOLARITY	RESERVED	SPILBK	SPICHAR			
R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h			

Table 9-7. SPICCR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	SPISWRESET	R/W	0h	SPI Software Reset When changing configuration, you should clear this bit before the changes and set this bit before resuming operation. Reset type: SYSRSn 0h (R/W) = Initializes the SPI operating flags to the reset condition. Specifically, the RECEIVER OVERRUN Flag bit (SPISTS.7), the SPI INT FLAG bit (SPISTS.6), and the TXBUF FULL Flag bit (SPISTS.5) are cleared. SPISTE will become inactive. SPICLK will be immediately driven to 0 regardless of the clock polarity. The SPI configuration remains unchanged. 1h (R/W) = SPI is ready to transmit or receive the next character. When the SPI SW RESET bit is a 0, a character written to the transmitter will not be shifted out when this bit is set. A new character must be written to the serial data register. SPICLK will be returned to its inactive state one SPICLK cycle after this bit is set.
6	CLKPOLARITY	R/W	0h	Shift Clock Polarity This bit controls the polarity of the SPICLK signal. CLOCK POLARITY and POLARITY CLOCK PHASE (SPICTL.3) control four clocking schemes on the SPICLK pin. Reset type: SYSRSn 0h (R/W) = Data is output on rising edge and input on falling edge. When no SPI data is sent, SPICLK is at low level. The data input and output edges depend on the value of the CLOCK PHASE bit (SPICTL.3) as follows: - 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. 1h (R/W) = Data is output on falling edge and input on rising edge. When no SPI data is sent, SPICLK is at high level. The data input and output edges depend on the value of the CLOCK PHASE bit (SPICTL.3) as follows: - 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	R	0h	Reserved

Table 9-7. SPICCR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	SPILBK	R/W	0h	SPI Loopback Mode Select Loopback mode allows module validation during device testing. This mode is valid only in master mode of the SPI. Reset type: SYSRSn 0h (R/W) = SPI loopback mode disabled. This is the default value after reset. 1h (R/W) = SPI loopback mode enabled, SIMO/SOMI lines are connected internally. Used for module self-tests.
3-0	SPICHAR	R/W	0h	Character Length Control Bits These four bits determine the number of bits to be shifted in or SPI CHAR0 out as a single character during one shift sequence. SPICHAR = Word length - 1 Reset type: SYSRSn 0h (R/W) = 1-bit word 1h (R/W) = 2-bit word 2h (R/W) = 3-bit word 3h (R/W) = 4-bit word 4h (R/W) = 5-bit word 5h (R/W) = 6-bit word 6h (R/W) = 7-bit word 7h (R/W) = 8-bit word 8h (R/W) = 9-bit word 9h (R/W) = 10-bit word Ah (R/W) = 11-bit word Bh (R/W) = 12-bit word Ch (R/W) = 13-bit word Dh (R/W) = 14-bit word Eh (R/W) = 15-bit word Fh (R/W) = 16-bit word

9.5.2.2 SPICTL Register (Offset = 1h) [reset = 0h]

SPICTL is shown in Figure 9-9 and described in Table 9-8.

Return to the Summary Table.

SPICTL controls data transmission, the SPI's ability to generate interrupts, the SPICLK phase, and the operational mode (slave or master).

Figure 9-9. SPICTL Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			OVERRUNINTENA	CLK_PHASE	MASTER_SLAVE	TALK	SPIINTENA
R-0h			R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 9-8. SPICTL Register Field Descriptions

Bit	Field	Type	Reset	Description
15-5	RESERVED	R	0h	Reserved
4	OVERRUNINTENA	R/W	0h	Overrun Interrupt Enable Overrun Interrupt Enable. Setting this bit causes an interrupt to be generated when the RECEIVER OVERRUN Flag bit (SPISTS.7) is set by hardware. Interrupts generated by the RECEIVER OVERRUN Flag bit and the SPI INT FLAG bit (SPISTS.6) share the same interrupt vector. Reset type: SYSRSn 0h (R/W) = Disable RECEIVER OVERRUN interrupts. 1h (R/W) = Enable RECEIVER_OVERRUN interrupts.
3	CLK_PHASE	R/W	0h	SPI Clock Phase Select This bit controls the phase of the SPICLK signal. CLOCK PHASE and CLOCK POLARITY (SPICCR.6) make four different clocking schemes possible (see clocking figures in SPI chapter). When operating with CLOCK PHASE high, the SPI (master or slave) makes the first bit of data available after SPIDAT is written and before the first edge of the SPICLK signal, regardless of which SPI mode is being used. Reset type: SYSRSn 0h (R/W) = Normal SPI clocking scheme, depending on the CLOCK POLARITY bit (SPICCR.6). 1h (R/W) = SPICLK signal delayed by one half-cycle. Polarity determined by the CLOCK POLARITY bit.
2	MASTER_SLAVE	R/W	0h	SPI Network Mode Control This bit determines whether the SPI is a network master or slave. SLAVE During reset initialization, the SPI is automatically configured as a network slave. Reset type: SYSRSn 0h (R/W) = SPI is configured as a slave. 1h (R/W) = SPI is configured as a master.

Table 9-8. SPICTL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
1	TALK	R/W	0h	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. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = Enables transmission For the 4-pin option, ensure to enable the receiver's SPISTEn input pin.
0	SPIINTENA	R/W	0h	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. Reset type: SYSRSn 0h (R/W) = Disables the interrupt. 1h (R/W) = Enables the interrupt.

9.5.2.3 SPISTS Register (Offset = 2h) [reset = 0h]

SPISTS is shown in Figure 9-10 and described in Table 9-9.

Return to the Summary Table.

SPISTS contrains interrupt and status bits.

Figure 9-10. SPISTS Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
OVERRUN_FLAG	INT_FLAG	BUFFULL_FLAG	RESERVED				
W1C-0h	RC-0h	R-0h	R-0h				

Table 9-9. SPISTS Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	OVERRUN_FLAG	W1C	0h	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 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. Reset type: SYSRSn 0h (R/W) = A receive overrun condition has not occurred. 1h (R/W) = The last received character has been overwritten and therefore lost (when the SPIRXBUF was overwritten by the SPI module before the previous character was read by the user application). Writing a '1' will clear this bit. The RECEIVER OVERRUN Flag bit should be cleared during the interrupt service routine because the RECEIVER OVERRUN Flag bit and SPI INT FLAG bit (SPISTS.6) share the same interrupt vector. This will alleviate any possible doubt as to the source of the interrupt when the next byte is received.

Table 9-9. SPISTS Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
6	INT_FLAG	RC	0h	SPI Interrupt Flag SPI INT FLAG is a read-only flag. Hardware sets this bit to indicate that the SPI has completed sending or receiving the last bit and is ready to be serviced. This flag causes an interrupt to be requested if the SPI INT ENA bit (SPICTL.0) is set. The received character is placed in the receiver buffer at the same time this bit is set. This bit is cleared in one of three ways: - Reading SPIRXBUF - Writing a 0 to SPI SW RESET (SPICCR.7) - Resetting the system Note: This bit should not be used if FIFO mode is enabled. The internal process of copying the received word from SPIRXBUF to the Receive FIFO will clear this bit. Use the FIFO status, or FIFO interrupt bits for similar functionality. Reset type: SYSRSn 0h (R/W) = No full words have been received or transmitted. 1h (R/W) = Indicates that the SPI has completed sending or receiving the last bit and is ready to be serviced.
5	BUFFULL_FLAG	R	0h	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. Reset type: SYSRSn 0h (R/W) = Transmit buffer is not full. 1h (R/W) = Transmit buffer is full.
4-0	RESERVED	R	0h	Reserved

9.5.2.4 SPIBRR Register (Offset = 4h) [reset = 0h]

SPIBRR is shown in Figure 9-11 and described in Table 9-10.

Return to the Summary Table.

SPIBRR contains the bits used for baud-rate selection.

Figure 9-11. SPIBRR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	SPI_BIT_RATE						
R-0h	R/W-0h						

Table 9-10. SPIBRR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6-0	SPI_BIT_RATE	R/W	0h	SPI Baud Rate Control These bits determine the bit transfer rate if the SPI is the network SPI BIT RATE 0 master. There are 125 data-transfer rates (each a function of the CPU clock, LSPCLK) that can be selected. One data bit is shifted per SPICLK cycle. (SPICLK is the baud rate clock output on the SPICLK pin.) If the SPI is a network slave, the module receives a clock on the SPICLK pin from the network master. Therefore, these bits have no effect on the SPICLK signal. The frequency of the input clock from the master should not exceed the slave SPI's LSPCLK signal divided by 4. 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: SPI Baud Rate = LSPCLK / (SPIBRR + 1) For SPIBRR = 0, 1, or 2: SPI Baud Rate = LSPCLK / 4 Reset type: SYSRSn 3h (R/W) = SPI Baud Rate = LSPCLK/4 4h (R/W) = SPI Baud Rate = LSPCLK/5 7Eh (R/W) = SPI Baud Rate = LSPCLK/127 7Fh (R/W) = SPI Baud Rate = LSPCLK/128

9.5.2.5 SPIRXEMU Register (Offset = 6h) [reset = 0h]

SPIRXEMU is shown in Figure 9-12 and described in Table 9-11.

Return to the Summary Table.

SPIRXEMU contains the received data. Reading SPIRXEMU does not clear the SPI INT FLAG bit of SPISTS. This is not a real register but a dummy address from which the contents of SPIRXBUF can be read by the emulator without clearing the SPI INT FLAG.

Figure 9-12. SPIRXEMU Register

15	14	13	12	11	10	9	8
ERXBn							
R-0h							
7	6	5	4	3	2	1	0
ERXBn							
R-0h							

Table 9-11. SPIRXEMU Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	ERXBn	R	0h	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. Reset type: SYSRSn

9.5.2.6 SPIRXBUF Register (Offset = 7h) [reset = 0h]

SPIRXBUF is shown in Figure 9-13 and described in Table 9-12.

Return to the Summary Table.

SPIRXBUF contains the received data. Reading SPIRXBUF clears the SPI INT FLAG bit in SPISTS. If FIFO mode is enabled, reading this register will also decrement the RXFFST counter in SPIFFRX.

Figure 9-13. SPIRXBUF Register

15	14	13	12	11	10	9	8
RXBn							
R-0h							
7	6	5	4	3	2	1	0
RXBn							
R-0h							

Table 9-12. SPIRXBUF Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	RXBn	R	0h	Received Data Once SPIDAT has received the complete character, the character is transferred to SPIRXBUF, where it can be read. At the same time, the SPI INT FLAG bit (SPISTS.6) is set. Since data is shifted into the SPI's most significant bit first, it is stored right-justified in this register. Reset type: SYSRSn

9.5.2.7 SPITXBUF Register (Offset = 8h) [reset = 0h]

SPITXBUF is shown in Figure 9-14 and described in Table 9-13.

Return to the Summary Table.

SPITXBUF stores the next character to be tranmitted. Writing to this register sets the TX BUF FULL Flag bit in SPISTS. When the transmission of the current character is complete, the contents of this register are automatically loaded in SPIDAT and the TX BUF FULL Flag is cleared. If no tranmission 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 tranmission is currently active, writing to this register initiates a transmission in the same manner that writing to SPIDAT does.

Figure 9-14. SPITXBUF Register

15	14	13	12	11	10	9	8
TXBn							
R/W-0h							
7	6	5	4	3	2	1	0
TXBn							
R/W-0h							

Table 9-13. SPITXBUF Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	TXBn	R/W	0h	Transmit Data Buffer This is where the next character to be transmitted is stored. When the transmission of the current character has completed, if the TX BUF FULL Flag bit is set, the contents of this register is automatically transferred to SPIDAT, and the TX BUF FULL Flag is cleared. Writes to SPITXBUF must be left-justified. Reset type: SYSRSn

9.5.2.8 SPIDAT Register (Offset = 9h) [reset = 0h]

SPIDAT is shown in Figure 9-15 and described in Table 9-14.

Return to the Summary Table.

SPIDAT is the transmit and receive shift register. Data written to SPIDAT is shifted out (MSB) on subsequent SPICLK cycles. For every bit (MSB) shifted out of the SPI, a bit is shifted into the LSB end of the shift register.

Figure 9-15. SPIDAT Register

15	14	13	12	11	10	9	8
SDATn							
R/W-0h							
7	6	5	4	3	2	1	0
SDATn							
R/W-0h							

Table 9-14. SPIDAT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	SDATn	R/W	0h	Serial Data Shift Register - It provides data to be output on the serial output pin if the TALK bit (SPICTL.1) is set. - When the SPI is operating as a master, a data transfer is initiated. When initiating a transfer, check the CLOCK POLARITY bit (SPICCR.6) described in Section 10.2.1.1 and the CLOCK PHASE bit (SPICTL.3) described in Section 10.2.1.2, for the requirements. In master mode, writing dummy data to SPIDAT initiates a receiver sequence. Since the data is not hardware-justified for characters shorter than sixteen bits, transmit data must be written in left-justified form, and received data read in right-justified form. Reset type: SYSRSn

9.5.2.9 SPIFFTX Register (Offset = Ah) [reset = A000h]

SPIFFTX is shown in Figure 9-16 and described in Table 9-15.

Return to the Summary Table.

SPIFFTX contains both control and status bits related to the output FIFO buffer. This includes FIFO reset control, FIFO interrupt level control, FIFO level status, as well as FIFO interrupt enable and clear bits.

Figure 9-16. SPIFFTX Register

15	14	13	12	11	10	9	8
SPIRST	SPIFFENA	TXFIFO	TXFFST				
R/W-1h	R/W-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL				
R-0h	W-0h	R/W-0h	R/W-0h				

Table 9-15. SPIFFTX Register Field Descriptions

Bit	Field	Type	Reset	Description
15	SPIRST	R/W	1h	SPI Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the SPI transmit and receive channels. The SPI FIFO register configuration bits will be left as is. 1h (R/W) = SPI FIFO can resume transmit or receive. No effect to the SPI registers bits.
14	SPIFFENA	R/W	0h	SPI FIFO Enhancements Enable Reset type: SYSRSn 0h (R/W) = SPI FIFO enhancements are disabled. 1h (R/W) = SPI FIFO enhancements are enabled.
13	TXFIFO	R/W	1h	TX FIFO Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Release transmit FIFO from reset.
12-8	TXFFST	R	0h	Transmit FIFO Status Reset type: SYSRSn 0h (R/W) = Transmit FIFO is empty. 1h (R/W) = Transmit FIFO has 1 word. 2h (R/W) = Transmit FIFO has 2 words. 10h (R/W) = Transmit FIFO has 16 words, which is the maximum. 1Fh (R/W) = Reserved.
7	TXFFINT	R	0h	TX FIFO Interrupt Flag Reset type: SYSRSn 0h (R/W) = TXFIFO interrupt has not occurred, This is a read-only bit. 1h (R/W) = TXFIFO interrupt has occurred, This is a read-only bit.
6	TXFFINTCLR	W	0h	TXFIFO Interrupt Clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFTX[TXFFINT] flag.
5	TXFFIENA	R/W	0h	TX FIFO Interrupt Enable Reset type: SYSRSn 0h (R/W) = TX FIFO interrupt based on TXFFIL match (less than or equal to) will be disabled. 1h (R/W) = TX FIFO interrupt based on TXFFIL match (less than or equal to) will be enabled.

Table 9-15. SPIFFTX Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4-0	TXFFIL	R/W	0h	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). Reset type: SYSRSn 0h (R/W) = A TX FIFO interrupt request is generated when there are no words remaining in the TX buffer. 1h (R/W) = A TX FIFO interrupt request is generated when there is 1 word or no words remaining in the TX buffer. 2h (R/W) = A TX FIFO interrupt request is generated when there is 2 words or fewer remaining in the TX buffer. 10h (R/W) = A TX FIFO interrupt request is generated when there are 16 words or fewer remaining in the TX buffer. 1Fh (R/W) = Reserved.

9.5.2.10 SPIFFRX Register (Offset = Bh) [reset = 201Fh]

SPIFFRX is shown in Figure 9-17 and described in Table 9-16.

Return to the Summary Table.

SPIFFRX contains both control and status bits related to the input FIFO buffer. This includes FIFO reset control, FIFO interrupt level control, FIFO level status, as well as FIFO interrupt enable and clear bits.

Figure 9-17. SPIFFRX Register

15	14	13	12	11	10	9	8
RXFFOVF	RXFFOVFCLR	RXFIFORESET	RXFFST				
R-0h	W-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL				
R-0h	W-0h	R/W-0h	R/W-1Fh				

Table 9-16. SPIFFRX Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RXFFOVF	R	0h	Receive FIFO Overflow Flag Reset type: SYSRSn 0h (R/W) = Receive FIFO has not overflowed. This is a read-only bit. 1h (R/W) = Receive FIFO has overflowed, read-only bit. More than 16 words have been received in to the FIFO, and the first received word is lost.
14	RXFFOVFCLR	W	0h	Receive FIFO Overflow Clear Reset type: SYSRSn 0h (R/W) = Write 0 does not affect RXFFOVF flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFRX[RXFFOVF].
13	RXFIFORESET	R/W	1h	Receive FIFO Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Re-enable receive FIFO operation.
12-8	RXFFST	R	0h	Receive FIFO Status Reset type: SYSRSn 0h (R/W) = Receive FIFO is empty. 1h (R/W) = Receive FIFO has 1 word. 2h (R/W) = Receive FIFO has 2 words. 10h (R/W) = Receive FIFO has 16 words, which is the maximum. 1Fh (R/W) = Reserved.
7	RXFFINT	R	0h	Receive FIFO Interrupt Flag Reset type: SYSRSn 0h (R/W) = RXFIFO interrupt has not occurred. This is a read-only bit. 1h (R/W) = RXFIFO interrupt has occurred. This is a read-only bit.
6	RXFFINTCLR	W	0h	Receive FIFO Interrupt Clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFIFINT flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFRX[RXFFINT] flag

Table 9-16. SPIFFRX Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
5	RXFFIENA	R/W	0h	RX FIFO Interrupt Enable Reset type: SYSRSn 0h (R/W) = RX FIFO interrupt based on RXFFIL match (greater than or equal to) will be disabled. 1h (R/W) = RX FIFO interrupt based on RXFFIL match (greater than or equal to) will be enabled.
4-0	RXFFIL	R/W	1Fh	Receive FIFO Interrupt Level Bits 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. Reset type: SYSRSn 0h (R/W) = A RX FIFO interrupt request is generated when there is 0 or more words in the RX buffer. 1h (R/W) = A RX FIFO interrupt request is generated when there are 1 or more words in the RX buffer. 2h (R/W) = A RX FIFO interrupt request is generated when there are 2 or more words in the RX buffer. 10h (R/W) = A RX FIFO interrupt request is generated when there are 16 words in the RX buffer. 1Fh (R/W) = Reserved.

9.5.2.11 SPIFFCT Register (Offset = Ch) [reset = 0h]

SPIFFCT is shown in Figure 9-18 and described in Table 9-17.

Return to the Summary Table.

SPIFFCT controls the FIFO transmit delay bits.

Figure 9-18. SPIFFCT Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXDLY							
R/W-0h							

Table 9-17. SPIFFCT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	TXDLY	R/W	0h	FIFO Transmit Delay Bits These bits define the delay between every transfer from FIFO transmit buffer to transmit shift register. The delay is defined in number SPI serial clock cycles. The 8-bit register could define a minimum delay of 0 serial clock cycles and a maximum of 255 serial clock cycles. In FIFO mode, the buffer (TXBUF) between the shift register and the FIFO should be filled only after the shift register has completed shifting of the last bit. This is required to pass on the delay between transfers to the data stream. In the FIFO mode TXBUF should not be treated as one additional level of buffer. Reset type: SYSRSn 0h (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF immediately upon completion of transmission of the previous word. 1h (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF1 serial clock cycle after completion of transmission of the previous word. 2h (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF 2 serial clock cycles after completion of transmission of the previous word. FFh (R/W) = The next word in the TX FIFO buffer is transferred to SPITXBUF 255 serial clock cycles after completion of transmission of the previous word.

9.5.2.12 SPIPRI Register (Offset = Fh) [reset = 0h]

SPIPRI is shown in Figure 9-19 and described in Table 9-18.

Return to the Summary Table.

SPIPRI controls auxillary functions for the SPI including emulation control, and 3-wire control.

Figure 9-19. SPIPRI Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	RESERVED	SOFT	FREE	RESERVED			
R-0h		R/W-0h	R/W-0h	R-0h			

Table 9-18. SPIPRI Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	RESERVED	R/W	0h	Reserved
5	SOFT	R/W	0h	Emulation Soft Run This bit only has an effect when the FREE bit is 0. Reset type: SYSRSn 0h (R/W) = Transmission stops midway in the bit stream while TSUSPEND is asserted. Once TSUSPEND is deasserted without a system reset, the remainder of the bits pending in the DATBUF are shifted. Example: If SPIDAT has shifted 3 out of 8 bits, the communication freezes right there. However, if TSUSPEND is later deasserted without resetting the SPI, SPI starts transmitting from where it had stopped (fourth bit in this case) and will transmit 8 bits from that point. 1h (R/W) = If the emulation suspend occurs before the start of a transmission, (that is, before the first SPICLK pulse) then the transmission will not occur. If the emulation suspend occurs after the start of a transmission, then the data will be shifted out to completion. When the start of transmission occurs is dependent on the baud rate used. 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.
4	FREE	R/W	0h	Emulation Free Run 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. Reset type: SYSRSn 0h (R/W) = Emulation mode is selected by the SOFT bit 1h (R/W) = Free run, continue SPI operation regardless of suspend or when the suspend occurred.
3-0	RESERVED	R	0h	Reserved

Serial Communications Interface (SCI)

This chapter describes the features and operation of the serial communication interface (SCI) module. SCI is a two−wire asynchronous serial port, commonly known as a UART. The SCI modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format. The SCI receiver and transmitter each have a 16-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.

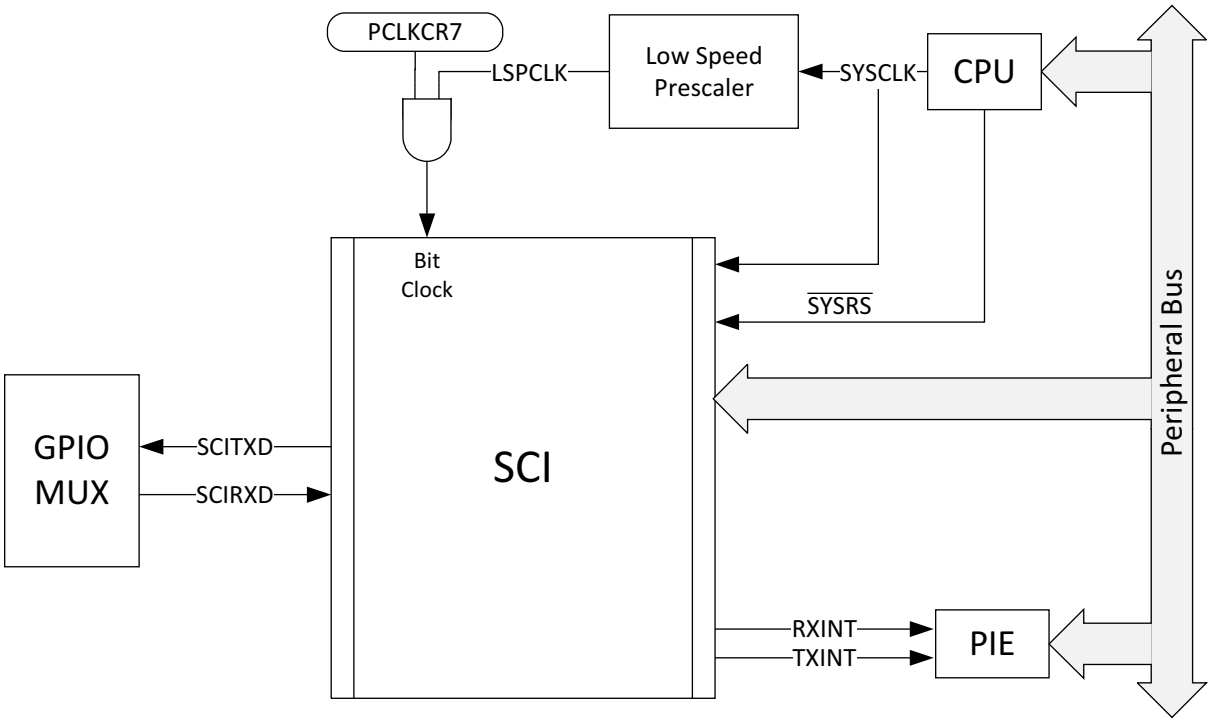
To specify data integrity, the SCI checks received data for break detection, parity, overrun, and framing errors. The bit rate is programmable to different speeds through a 16-bit baud-select register.

Topic **Page**

10.1 Introduction

The SCI interfaces are shown in Figure 10-1.

Figure 10-1. SCI CPU Interface



Features of the SCI module include:

- Two external pins:
  - SCITXD: SCI transmit-output pin
  - SCIRXD: SCI receive-input pin

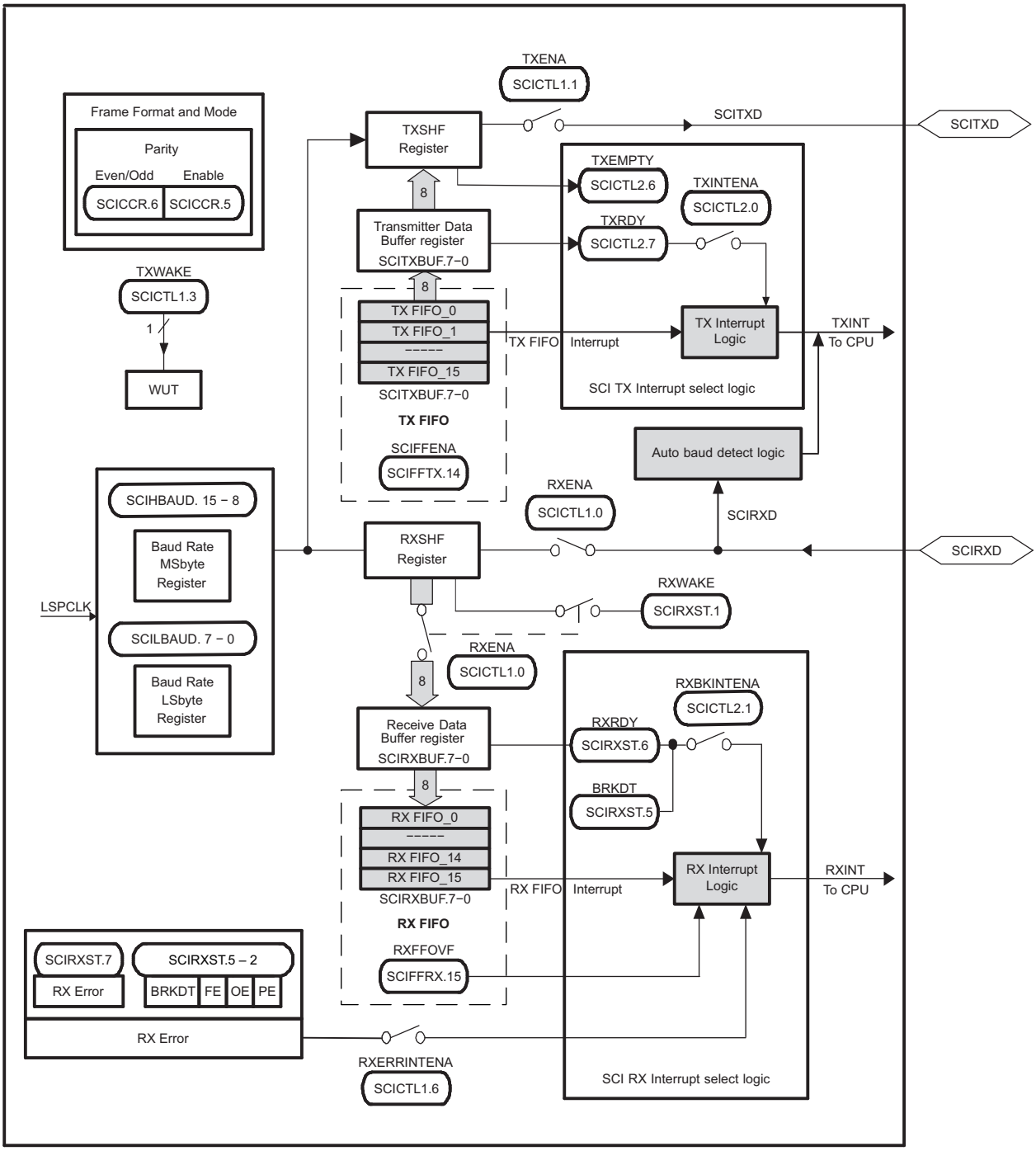
  Both pins can be used as GPIO if not used for SCI.
- Baud rate programmable to 64K different rates
- Data-word format
  - One start bit
  - Data-word length programmable from one to eight bits
  - Optional even/odd/no parity bit
  - One or two stop bits
  - An extra bit to distinguish addresses from data (address bit mode only)
- Four error-detection flags: parity, overrun, framing, and break detection
- Two wake-up multiprocessor modes: idle-line and address bit
- Half- or full-duplex operation
- Double-buffered receive and transmit functions
- Transmitter and receiver operations can be accomplished through interrupt-driven or polled algorithms with status flags.
- Separate enable bits for transmitter and receiver interrupts (except BRKDT)
- NRZ (non-return-to-zero) format

Enhanced features include:

- Auto-baud-detect hardware logic
- 16-level transmit/receive FIFO

Figure 10-2 shows the SCI module block diagram. The SCI port operation is configured and controlled by the registers listed in Section 10.14 of this chapter.

Figure 10-2. Serial Communications Interface (SCI) Module Block Diagram



10.2 Architecture

The major elements used in full-duplex operation are shown in Figure 10-2 and include:

- A transmitter (TX) and its major registers (upper half of Figure 10-2)
  - 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 10-2)
  - RXSHF register — receiver shift register. Shifts data in from SCIRXD pin, one bit at a time
  - SCIRXBUF — receiver data buffer register. Contains data to be read by the CPU. Data from a remote processor is loaded into register RXSHF and then into registers SCIRXBUF and SCIRXEMU
- A programmable baud generator
- Control and status registers

The SCI receiver and transmitter can operate either independently or simultaneously.

10.3 SCI Module Signal Summary

A summarized description of each SCI signal name is shown in Table 10-1.

Table 10-1. 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

10.4 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins. To avoid glitches on the pins, the GPyGMUX bits must be configured first (while keeping the corresponding GPyMUX bits at the default of zero), followed by writing the GPyMUX register to the desired value.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPyPUD register.

See the *GPIO* chapter for more details on GPIO mux and settings.

10.5 Multiprocessor and Asynchronous Communication Modes

The SCI has two multiprocessor protocols, the idle-line multiprocessor mode (see Section 10.8) and the address-bit multiprocessor mode (see Section 10.9). These protocols allow efficient data transfer between multiple processors.

The SCI offers the universal asynchronous receiver/transmitter (UART) communications mode for interfacing with many popular peripherals. The asynchronous mode (see Section 10.10) requires two lines to interface with many standard devices such as terminals and printers that use RS-232-C formats. Data transmission characteristics include:

- One start bit
- One to eight data bits

- An even/odd parity bit or no parity bit
- One or two stop bits

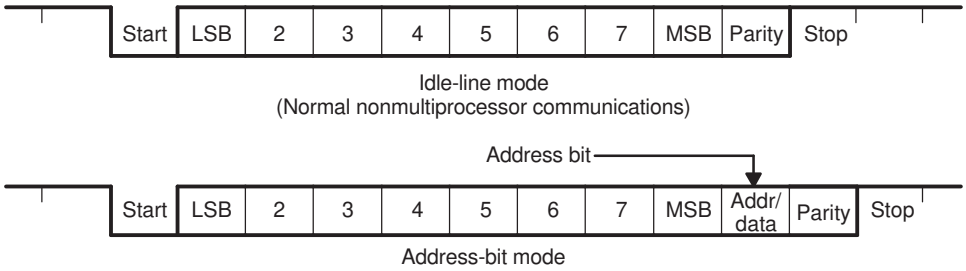
10.6 SCI Programmable Data Format

SCI data, both receive and transmit, is in NRZ (non-return-to-zero) format. The NRZ data format, shown in Figure 10-3, 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 10-3.

Figure 10-3. 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 10-2.

Table 10-2. 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.

10.7 SCI Multiprocessor Communication

The multiprocessor communication format allows one processor to efficiently send blocks of data to other processors on the same serial link. On one serial line, there should be only one transfer at a time. In other words, there can be only one talker on a serial line at a time.

Address Byte

The first byte of a block of information that the talker sends contains an address byte that is read by all listeners. Only listeners with the correct address can be interrupted by the data bytes that follow the address byte. The listeners with an incorrect address remain uninterrupted until the next address byte.

Sleep Bit

All processors on the serial link set the SCI SLEEP bit (bit 2 of SCICTL1) to 1 so that they are interrupted only when the address byte is detected. When a processor reads a block address that corresponds to the CPU device address as set by your application software, your program must clear the SLEEP bit to enable the SCI to generate an interrupt on receipt of each data byte.

Although the receiver still operates when the SLEEP bit is 1, it does not set RXRDY, RXINT, or any of the receiver error status bits to 1 unless the address byte is detected and the address bit in the received frame is a 1 (applicable to address-bit mode). The SCI does not alter the SLEEP bit; your software must alter the SLEEP bit.

10.7.1 Recognizing the Address Byte

A processor recognizes an address byte differently, depending on the multiprocessor mode used. For example:

- The idle-line mode (Section 10.8) leaves a quiet space before the address byte. This mode does not have an extra address/data bit and is more efficient than the address-bit mode for handling blocks that contain more than ten bytes of data. The idle-line mode should be used for typical non-multiprocessor SCI communication.
- The address-bit mode (Section 10.9) adds an extra bit (that is, an address bit) into every byte to distinguish addresses from data. This mode is more efficient in handling many small blocks of data because, unlike the idle mode, it does not have to wait between blocks of data. However, at a high transmit speed, the program is not fast enough to avoid a 10-bit idle in the transmission stream.

10.7.2 Controlling the SCI TX and RX Features

The multiprocessor mode is software selectable via the ADDR/IDLE MODE bit (SCICCR, bit 3). Both modes use the TXWAKE flag bit (SCICTL1, bit 3), RXWAKE flag bit (SCIRXST, bit1), and the SLEEP flag bit (SCICTL1, bit 2) to control the SCI transmitter and receiver features of these modes.

10.7.3 Receipt Sequence

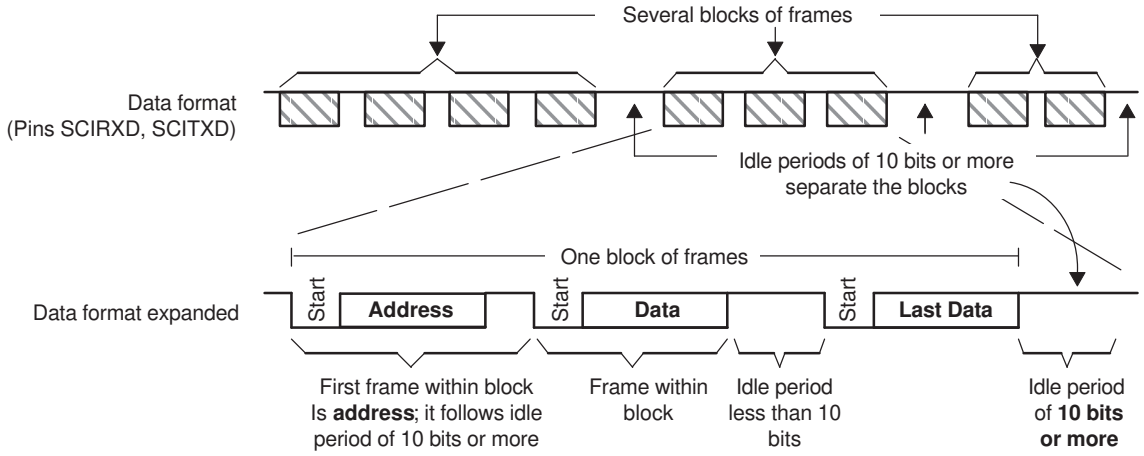
In both multiprocessor modes, the receive sequence is as follows:

1. At the receipt of an address block, the SCI port wakes up and requests an interrupt (bit number 1 RX/BK INT ENA-of SCICTL2 must be enabled to request an interrupt). It reads the first frame of the block, which contains the destination address.
2. A software routine is entered through the interrupt and checks the incoming address. This address byte is checked against its device address byte stored in memory.
3. If the check shows that the block is addressed to the device CPU, the CPU clears the SLEEP bit and reads the rest of the block. If not, the software routine exits with the SLEEP bit still set, and does not receive interrupts until the next block start.

10.8 Idle-Line Multiprocessor Mode

In the idle-line multiprocessor protocol (ADDR/IDLE MODE bit=0), blocks are separated by having a longer idle time between the blocks than between frames in the blocks. An idle time of ten or more high-level bits after a frame indicates the start of a new block. The time of a single bit is calculated directly from the baud value (bits per second). The idle-line multiprocessor communication format is shown in Figure 10-4 (ADDR/IDLE MODE bit is bit 3 of SCICCR).

Figure 10-4. Idle-Line Multiprocessor Communication Format



10.8.1 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.

10.8.2 Block Start Signal

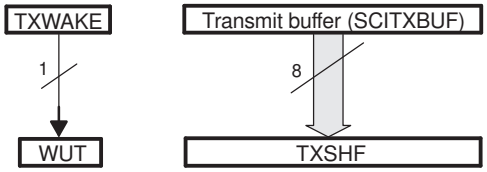
There are two ways to send a block-start signal:

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.)

10.8.3 Wake-UP Temporary (WUT) Flag

Associated with the TXWAKE bit is the wake-up temporary (WUT) flag. WUT is an internal flag, double-buffered with TXWAKE. When TXSHF is loaded from SCITXBUF, WUT is loaded from TXWAKE, and the TXWAKE bit is cleared to 0. This arrangement is shown in Figure 10-5.

Figure 10-5. Double-Buffered WUT and TXSHF



10.8.3.1 Sending a Block Start Signal

To send out a block-start signal of exactly one frame time during a sequence of block transmissions:

1. Write a 1 to the TXWAKE bit.
2. Write a data word (content not important: a don’t care) to the SCITXBUF register (transmit data buffer) to send a block-start signal. (The first data word written is suppressed while the block-start signal is sent out and ignored after that.) When the TXSHF (transmit shift register) is free again, SCITXBUF contents are shifted to TXSHF, the TXWAKE value is shifted to WUT, and then TXWAKE is cleared.

   Because TXWAKE was set to a 1, the start, data, and parity bits are replaced by an idle period of 11 bits transmitted following the last stop bit of the previous frame.
3. **Write a new address value to SCITXBUF**

   A don’t-care data word must first be written to register SCITXBUF so that the TXWAKE bit value can be shifted to WUT. After the don’t-care data word is shifted to the TXSHF register, the SCITXBUF (and TXWAKE if necessary) can be written to again because TXSHF and WUT are both double-buffered.

10.8.4 Receiver Operation

The receiver operates regardless of the SLEEP bit. However, the receiver neither sets RXRDY nor the error status bits, nor does it request a receive interrupt until an address frame is detected.

10.9 Address-Bit Multiprocessor Mode

In the address-bit protocol (ADDR/IDLE MODE bit=1), frames have an extra bit called an address bit that immediately follows the last data bit. The address bit is set to 1 in the first frame of the block and to 0 in all other frames. The idle period timing is irrelevant (see Figure 10-6).

10.9.1 Sending an Address

The TXWAKE bit value is placed in the address bit. During transmission, when the SCITXBUF register and TXWAKE are loaded into the TXSHF register and WUT respectively, TXWAKE is reset to 0 and WUT becomes the value of the address bit of the current frame. Thus, to send an address:

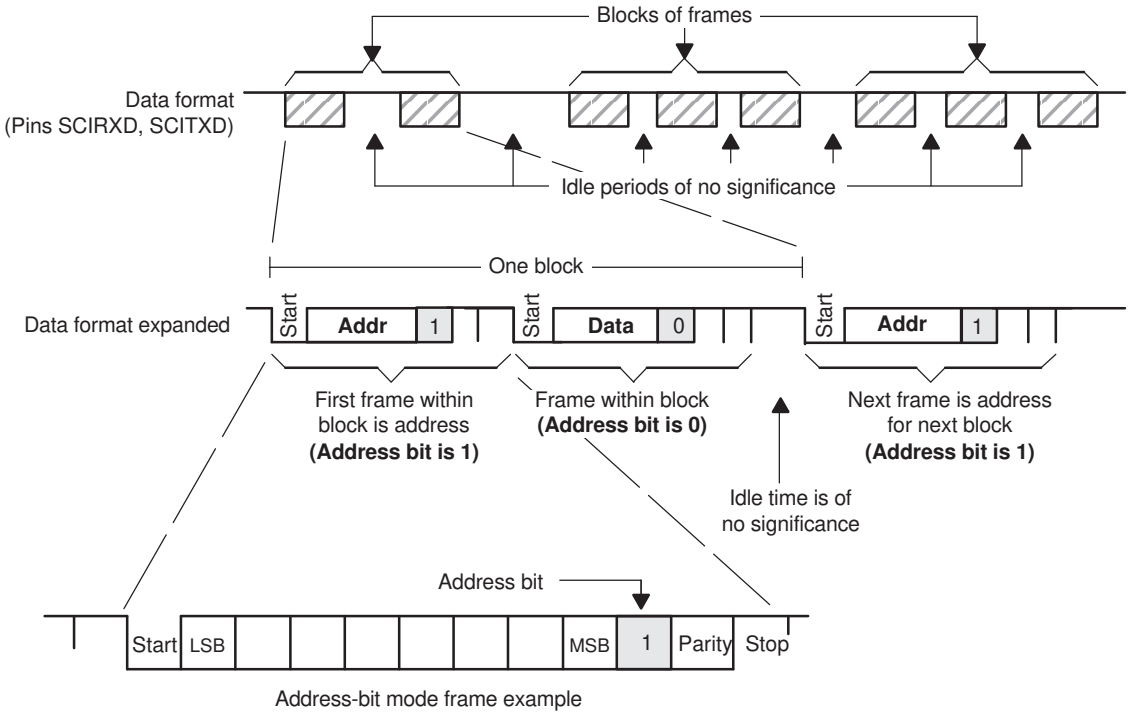
1. Set the TXWAKE bit to 1 and write the appropriate address value to the SCITXBUF register.

When this address value is transferred to the TXSHF register and shifted out, its address bit is sent as a 1. This flags the other processors on the serial link to read the address.

2. Write to SCITXBUF and TXWAKE after TXSHF and WUT are loaded. (Can be written to immediately since both TXSHF and WUT are both double-buffered.
3. Leave the TXWAKE bit set to 0 to transmit non-address frames in the block.

NOTE: As a general rule, the address-bit format is typically used for data frames of 11 bytes or less. This format adds one bit value (1 for an address frame, 0 for a data frame) to all data bytes transmitted. The idle-line format is typically used for data frames of 12 bytes or more.

Figure 10-6. Address-Bit Multiprocessor Communication Format



10.10 SCI Communication Format

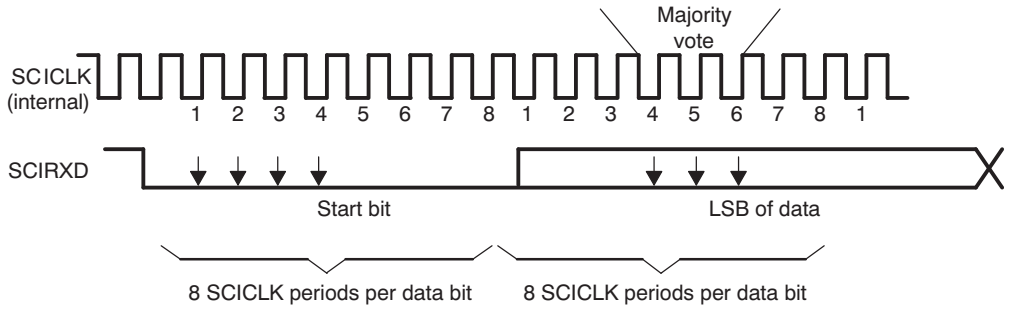
The SCI asynchronous communication format uses either single line (one way) or two line (two way) communications. In this mode, the frame consists of a start bit, one to eight data bits, an optional even/odd parity bit, and one or two stop bits (shown in Figure 10-7). There are eight SCICLK periods per 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 10-7. 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 10-7 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 10-7. SCI Asynchronous Communications Format

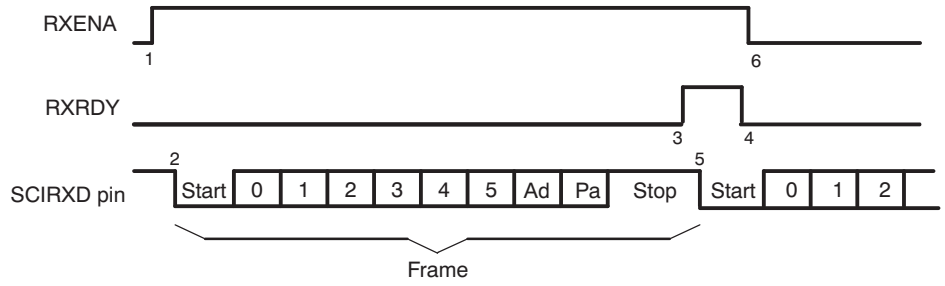


10.10.1 Receiver Signals in Communication Modes

Figure 10-8 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 10-8. SCI RX Signals in Communication Modes



(1) Data arrives on the SCIRXD pin, start bit detected.

(2) 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:

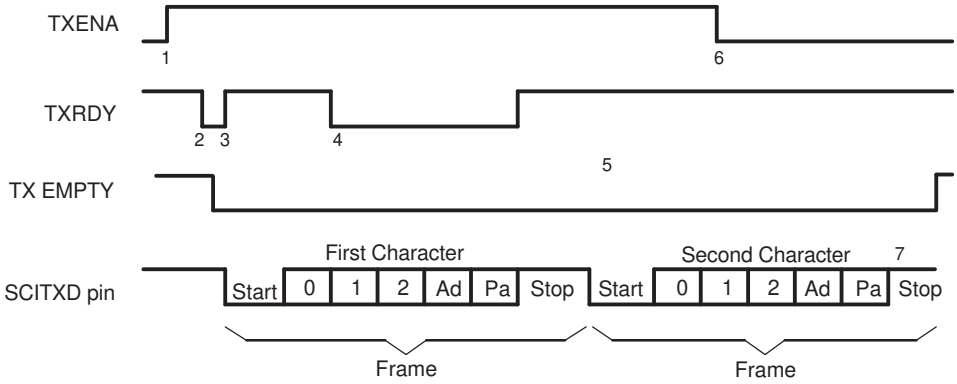
1. Flag bit RXENA (SCICTL1, bit 0) goes high to enable the receiver.
2. Data arrives on the SCIRXD pin, start bit detected.
3. Data is shifted from RXSHF to the receiver buffer register (SCIRXBUF); an interrupt is requested. Flag bit RXRDY (SCIRXST, bit 6) goes high to signal that a new character has 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.

10.10.2 Transmitter Signals in Communication Modes

Figure 10-9 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 10-9. 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.

10.11 SCI Port Interrupts

The SCI receiver and transmitter can be interrupt controlled. The SCICTL2 register has one flag bit (TXRDY) that indicates active interrupt conditions, and the SCIRXST register has two interrupt flag bits (RXRDY and BRKDT), plus the RX ERROR interrupt flag 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.

The operation of peripheral interrupts is described in the peripheral interrupt expansion controller section of the *External Peripheral Interface (ePIE)* chapter.

- If the RX/BK INT ENA bit (SCICTL2, bit 1) is set, the receiver peripheral interrupt request is asserted when one of the following events occurs:
  - The SCI receives a complete frame and transfers the data in the RXSHF register to the SCIRXBUF register. This action sets the RXRDY flag (SCIRXST, bit 6) and initiates an interrupt.
  - A break detect condition occurs (the SCIRXD is low for 9.625 bit periods following a missing stop bit). This action sets the BRKDT flag bit (SCIRXST, bit 5) and initiates an interrupt.
- If the TX INT ENA bit (SCICTL2.0) is set, the transmitter peripheral interrupt request is asserted whenever the data in the SCITXBUF register is transferred to the TXSHF register, indicating that the CPU can write to SCITXBUF; this action sets the TXRDY flag bit (SCICTL2, bit 7) and initiates an interrupt.

NOTE: Interrupt generation due to the RXRDY and BRKDT bits is controlled by the RX/BK INT ENA bit (SCICTL2, bit 1). Interrupt generation due to the RX ERROR bit is controlled by the RX ERR INT ENA bit (SCICTL1, bit 6).

10.12 SCI Baud Rate Calculations

The internally generated serial clock is determined by the low-speed peripheral clock LSPCLK) and the baud-select registers. The SCI uses the 16-bit value of the baud-select registers to select one of the 64K different serial clock rates possible for a given LSPCLK.

See the bit descriptions in the baud-select registers, for the formula to use when calculating the SCI asynchronous baud. Table 10-3 shows the baud-select values for common SCI bit rates.

Table 10-3. Asynchronous Baud Register Values for Common SCI Bit Rates

		LSPCLK Clock Frequency, 100 MHz	
Ideal Baud	**BRR**	**Actual Baud**	**% Error**
2400	5207 (1457h)	2400	0
4800	2603 (A2Bh)	4800	0
9600	1301 (515h)	9601	0.01
19200	650 (28Ah)	19201	0.01
38400	324 (144h)	38462	0.16

LSPCLK/16 is the maximum baud rate. For example, if LSPCLK is 100MHz, then the maximum baud rate is 6.25Mbps.

10.13 SCI Enhanced Features

The 28x SCI features autobaud detection and transmit/receive FIFO. The following section explains the FIFO operation.

10.13.1 SCI FIFO Description

The following steps explain the FIFO features and help with programming the SCI with FIFOs.

1. *Reset.* At reset the SCI powers up in standard SCI mode and the FIFO function is disabled. The FIFO registers SCIFFTX, SCIFFRX, and SCIFFCT remain inactive.
2. *Standard SCI.* The standard SCI modes will work normally with TXINT/RXINT interrupts as the interrupt source for the module.
3. *FIFO enable.* FIFO mode is enabled by setting the SCIFFEN bit in the SCIFFTX register. SCIRST can reset the FIFO mode at any stage of its operation.
4. Active registers. All the SCI registers and SCI FIFO registers (SCIFFTX, SCIFFRX, and SCIFFCT) are active.
5. *Interrupts.* FIFO mode has two interrupts; one for transmit FIFO, TXINT and one for receive FIFO, RXINT. RXINT is the common interrupt for SCI FIFO receive, receive error, and receive FIFO overflow conditions. The TXINT of the standard SCI will be disabled and this interrupt will service as SCI transmit FIFO interrupt.
6. *Buffers.* Transmit and receive buffers are supplemented with two 16-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 the 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. When FIFO mode is enabled for SCI, characters written to SCITXBUF are queued in to SCI-TXFIFO and the characters received in SCI-RXFIFO can be read using SCIRXBUF.
7. *Delayed transfer.* The rate at which words in the FIFO are transferred to the transmit shift register is programmable. The SCIFFCT register bits (7−0) FFTXDLY7−FFTXDLY0 define the delay between the word transfer. The delay is defined in the number SCI baud clock cycles. The 8 bit register can define

a minimum delay of 0 baud clock cycles and a maximum of 256-baud clock cycles. With zero delay, the SCI module can transmit data in continuous mode with the FIFO words shifting out back to back. With the 256 clock delay the SCI module can transmit data in a maximum delayed mode with the FIFO words shifting out with a delay of 256 baud clocks between each words. The programmable delay facilitates communication with slow SCI/UARTs with little CPU intervention.

8. *FIFO status bits.* Both the transmit and receive FIFOs have status bits TXFFST or RXFFST (bits 12−8) that define the number of words available in the FIFOs at any time. The transmit FIFO reset bit TXFIFO and receive reset bit RXFIFO reset the FIFO pointers to zero when these bits are cleared to 0. The FIFOs resumes operation from start once these bits are set to 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 10-10 and Table 10-4 explain the operation/configuration of SCI interrupts in nonFIFO/FFO mode.

Figure 10-10. SCI FIFO Interrupt Flags and Enable Logic

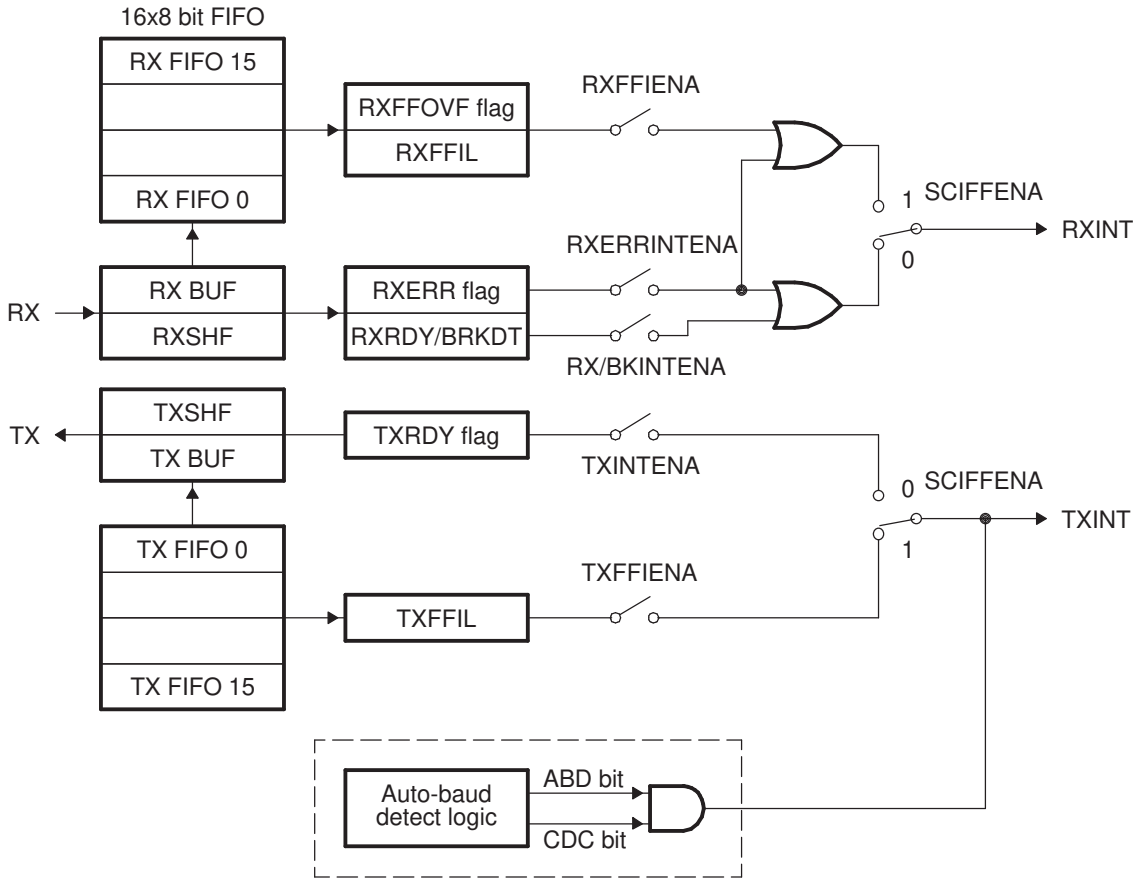


Table 10-4. SCI Interrupt Flags

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

10.13.2 SCI Auto-Baud

Most SCI modules do not have an auto-baud detect logic built-in hardware. These SCI modules are integrated with embedded controllers whose clock rates are dependent on PLL reset values. Often embedded controller clocks change after final design. In the enhanced feature set this module supports an autobaud-detect logic in hardware. The following section explains the enabling sequence for autobaud-detect feature.

10.13.3 Autobaud-Detect Sequence

Bits ABD and CDC in SCIFFCT control the autobaud logic. The SCIRST bit should be enabled to make autobaud logic work.

If ABD is set while CDC is 1, which indicates auto-baud alignment, SCI transmit FIFO interrupt will occur (TXINT). After the interrupt service CDC bit has to be cleared by software. If CDC remains set even after interrupt service, there should be no repeat interrupts.

1. Enable autobaud-detect mode for the SCI by setting the CDC bit (bit 13) in SCIFFCT and clearing the ABD bit (Bit 15) by writing a 1 to ABDCLR bit (bit 14).
2. Initialize the baud register to be 1 or less than a baud rate limit of 500 Kbps.
3. Allow SCI to receive either character "A" or "a" from a host at the desired baud rate. If the first character is either "A" or "a", the autobaud- detect hardware will detect the incoming baud rate and set the ABD bit.
4. The auto-detect hardware will update the baud rate register with the equivalent baud value hex. The logic will also generate an interrupt to the CPU.
5. Respond to the interrupt clear ADB bit by writing a 1 to ABD CLR (bit 14) of SCIFFCT register and disable further autobaud locking by clearing CDC bit by writing a 0.
6. Read the receive buffer for character "A" or "a" to empty the buffer and buffer status.
7. If ABD is set while CDC is 1, which indicates autobaud alignment, the SCI transmit FIFO interrupt will occur (TXINT). After the interrupt service CDC bit must be cleared by software.

NOTE: At higher baud rates, the slew rate of the incoming data bits can be affected by transceiver and connector performance. While normal serial communications may work well, this slew rate may limit reliable autobaud detection at higher baud rates (typically beyond 100k baud) and cause the auto-baudlock feature to fail.

To avoid this, the following is recommended:

- Achieve a baud-lock between the host and 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.

10.14 SCI Registers

The section describes the Serial Communication Interface module Registers.

10.14.1 SCI Base Addresses

10.14.2 SCI_REGS Registers

Table 10-5 lists the SCI_REGS registers. All register offset addresses not listed in Table 10-5 should be considered as reserved locations and the register contents should not be modified.

Table 10-5. SCI_REGS Registers

Offset	Acronym	Register Name	Write Protection	Section
0h	SCICCR	Communications control register		Go
1h	SCICTL1	Control register 1		Go
2h	SCIHBAUD	Baud rate (high) register		Go
3h	SCILBAUD	Baud rate (low) register		Go
4h	SCICTL2	Control register 2		Go
5h	SCIRXST	Receive status register		Go
6h	SCIRXEMU	Receive emulation buffer register		Go
7h	SCIRXBUF	Receive data buffer		Go
9h	SCITXBUF	Transmit data buffer		Go
Ah	SCIFFTX	FIFO transmit register		Go
Bh	SCIFFRX	FIFO receive register		Go
Ch	SCIFFCT	FIFO control register		Go
Fh	SCIPRI	SCI priority control		Go

Complex bit access types are encoded to fit into small table cells. Table 10-6 shows the codes that are used for access types in this section.

Table 10-6. SCI_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R -0	Read Returns 0s
Write Type		
W	W	Write
W1S	W 1S	Write 1 to set
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

10.14.2.1 SCICCR Register (Offset = 0h) [reset = 0h]

SCICCR is shown in Figure 10-11 and described in Table 10-7.

Return to the Summary Table.

SCICCR defines the character format, protocol, and communications mode used by the SCI.

Figure 10-11. SCICCR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
STOPBITS	PARITY	PARITYENA	LOOPBKENA	ADDRIDLE_MODE	SCICHAR		
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h		

Table 10-7. SCICCR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	STOPBITS	R/W	0h	SCI number of stop bits. This bit specifies the number of stop bits transmitted. The receiver checks for only one stop bit. Reset type: SYSRSn 0h (R/W) = One stop bit 1h (R/W) = Two stop bits
6	PARITY	R/W	0h	SCI parity odd/even selection. If the PARITY ENABLE bit (SCICCR, bit 5) is set, PARITY (bit 6) designates odd or even parity (odd or even number of bits with the value of 1 in both transmitted and received characters). Reset type: SYSRSn 0h (R/W) = Odd parity 1h (R/W) = Even parity
5	PARITYENA	R/W	0h	SCI parity enable. This bit enables or disables the parity function. If the SCI is in the addressbit multiprocessor mode (set using bit 3 of this register), the address bit is included in the parity calculation (if parity is enabled). For characters of less than eight bits, the remaining unused bits should be masked out of the parity calculation. Reset type: SYSRSn 0h (R/W) = Parity disabled no parity bit is generated during transmission or is expected during reception 1h (R/W) = Parity is enabled
4	LOOPBKENA	R/W	0h	Loop Back test mode enable. This bit enables the Loop Back test mode where the Tx pin is internally connected to the Rx pin. Reset type: SYSRSn 0h (R/W) = Loop Back test mode disabled 1h (R/W) = Loop Back test mode enabled

Table 10-7. SCICCR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3	ADDRIDLE_MODE	R/W	0h	SCI multiprocessor mode control bit. This bit selects one of the multiprocessor protocols.Multiprocessor communication is different from the other communication modes because it uses SLEEP and TXWAKE functions (bits SCICTL1, bit 2 and SCICTL1, bit 3, respectively). The idle-line mode is usually used for normal communications because the address-bit mode adds an extra bit to the frame. The idle-line mode does not add this extra bit and is compatible with RS-232 type communications. Reset type: SYSRSn 0h (R/W) = Idle-line mode protocol selected 1h (R/W) = Address-bit mode protocol selected
2-0	SCICHAR	R/W	0h	Character-length control bits 2-0. These bits select the SCI character length from one to eight bits. Characters of less than eight bits are right-justified in SCIRXBUF and SCIRXEMU and are padded with leading zeros in SCIRXBUF. SCITXBUF doesn't need to be padded with leading zeros. Reset type: SYSRSn 0h (R/W) = SCICHAR_LENGTH_1 1h (R/W) = SCICHAR_LENGTH_2 2h (R/W) = SCICHAR_LENGTH_3 3h (R/W) = SCICHAR_LENGTH_4 4h (R/W) = SCICHAR_LENGTH_5 5h (R/W) = SCICHAR_LENGTH_6 6h (R/W) = SCICHAR_LENGTH_7 7h (R/W) = SCICHAR_LENGTH_8

10.14.2.2 SCICTL1 Register (Offset = 1h) [reset = 0h]

SCICTL1 is shown in Figure 10-12 and described in Table 10-8.

Return to the Summary Table.

SCICTL1 controls the receiver/transmitter enable, TXWAKE and SLEEP functions, and the SCI software reset.

Figure 10-12. SCICTL1 Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	RXERRINTENA	SWRESET	RESERVED	TXWAKE	SLEEP	TXENA	RXENA
R-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 10-8. SCICTL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	RXERRINTENA	R/W	0h	SCI receive error interrupt enable. Setting this bit enables an interrupt if the RX ERROR bit (SCIRXST, bit 7) becomes set because of errors occurring. Reset type: SYSRSn 0h (R/W) = Receive error interrupt disabled 1h (R/W) = Receive error interrupt enabled
5	SWRESET	R/W	0h	SCI software reset (active low). Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition. The SW RESET bit does not affect any of the configuration bits. All affected logic is held in the specified reset state until a 1 is written to SW RESET (the bit values following a reset are shown beneath each register diagram in this section). Thus, after a system reset, re-enable the SCI by writing a 1 to this bit. Clear this bit after a receiver break detect (BRKDT flag, bit SCIRXST, bit 5). SW RESET affects the operating flags of the SCI, but it neither affects the configuration bits nor restores the reset values. Once SW RESET is asserted, the flags are frozen until the bit is deasserted. The affected flags are as follows: Value After SW SCI Flag Register Bit RESET 1 TXRDY SCICTL2, bit 7 1 TX EMPTY SCICTL2, bit 6 0 RXWAKE SCIRXST, bit 1 0 PE SCIRXST, bit 2 0 OE SCIRXST, bit 3 0 FE SCIRXST, bit 4 0 BRKDT SCIRXST, bit 5 0 RXRDY SCIRXST, bit 6 0 RX ERROR SCIRXST, bit 7 Reset type: SYSRSn 0h (R/W) = Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition. 1h (R/W) = After a system reset, re-enable the SCI by writing a 1 to this bit.

Table 10-8. SCICTL1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	RESERVED	R	0h	Reserved
3	TXWAKE	R/W	0h	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) Reset type: SYSRSn 0h (R/W) = Transmit feature is not selected. In idle-line mode: write a 1 to TXWAKE, then write data to register SCITXBUF to generate an idle period of 11 data bits In address-bit mode: write a 1 to TXWAKE, then write data to SCITXBUF to set the address bit for that frame to 1 1h (R/W) = Transmit feature selected is dependent on the mode, idle-line or address-bit: TXWAKE is not cleared by the SW RESET bit (SCICTL1, bit 5) it is cleared by a system reset or the transfer of TXWAKE to the WUT flag.
2	SLEEP	R/W	0h	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. Reset type: SYSRSn 0h (R/W) = Sleep mode disabled 1h (R/W) = Sleep mode enabled
1	TXENA	R/W	0h	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. Data written into SCITXBUF when TXENA is disabled will not be transmitted even if the TXENA is enabled later. Reset type: SYSRSn 0h (R/W) = Transmitter disabled 1h (R/W) = Transmitter enabled
0	RXENA	R/W	0h	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. Reset type: SYSRSn 0h (R/W) = Prevent received characters from transfer into the SCIRXEMU and SCIRXBUF receiver buffers 1h (R/W) = Send received characters to SCIRXEMU and SCIRXBUF

10.14.2.3 SCIHBAUD Register (Offset = 2h) [reset = 0h]

SCIHBAUD is shown in Figure 10-13 and described in Table 10-9.

Return to the Summary Table.

The values in SCIHBAUD and SCILBAUD specify the baud rate for the SCI.

Figure 10-13. SCIHBAUD Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
BAUD							
R/W-0h							

Table 10-9. SCIHBAUD Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	BAUD	R/W	0h	SCI 16-bit baud selection Registers SCIHBAUD (MSbyte). The internally-generated serial clock is determined by the low speed peripheral clock (LSPCLK) signal and the two baud-select registers. The SCI uses the 16-bit value of these registers to select one of 64K serial clock rates for the communication modes. BRR = (SCIHBAUD << 8) + (SCILBAUD) The SCI baud rate is calculated using the following equation: SCI Asynchronous Baud = LSPCLK / ((BRR + 1) *8) Alternatively, BRR = LSPCLK / (SCI Asynchronous Baud * 8) - 1 Note that the above formulas are applicable only when 0 < BRR < 65536. If BRR = 0, then SCI Asynchronous Baud = LSPCLK / 16 Where: BRR = the 16-bit value (in decimal) in the baud-select registers Reset type: SYSRSn

10.14.2.4 SCILBAUD Register (Offset = 3h) [reset = 0h]

SCILBAUD is shown in Figure 10-14 and described in Table 10-10.

Return to the Summary Table.

The values in SCIHBAUD and SCILBAUD specify the baud rate for the SCI.

Figure 10-14. SCILBAUD Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
BAUD							
R/W-0h							

Table 10-10. SCILBAUD Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	BAUD	R/W	0h	See SCIHBAUD Detailed Description Reset type: SYSRSn

10.14.2.5 SCICTL2 Register (Offset = 4h) [reset = C0h]

SCICTL2 is shown in Figure 10-15 and described in Table 10-11.

Return to the Summary Table.

SCICTL2 enables the receive-ready, break-detect, and transmit-ready interrupts as well as transmitter-ready and -empty flags.

Figure 10-15. SCICTL2 Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXRDY	TXEMPTY	RESERVED				RXBKINTENA	TXINTENA
R-1h	R-1h	R-0h				R/W-0h	R/W-0h

Table 10-11. SCICTL2 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	TXRDY	R	1h	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. Reset type: SYSRSn 0h (R/W) = SCITXBUF is full 1h (R/W) = SCITXBUF is ready to receive the next character
6	TXEMPTY	R	1h	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. Reset type: SYSRSn 0h (R/W) = Transmitter buffer or shift register or both are loaded with data 1h (R/W) = Transmitter buffer and shift registers are both empty
5-2	RESERVED	R	0h	Reserved
1	RXBKINTENA	R/W	0h	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. Reset type: SYSRSn 0h (R/W) = Disable RXRDY/BRKDT interrupt 1h (R/W) = Enable RXRDY/BRKDT interrupt

Table 10-11. SCICTL2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
0	TXINTENA	R/W	0h	SCITXBUF-register interrupt enable. This bit controls the interrupt request caused by the setting of TXRDY flag bit (SCICTL2.7). However, it does not prevent the TXRDY flag from being set (which indicates SCITXBUF is ready to receive another character). 0 Disable TXRDY interrupt 1 Enable TXRDY interrupt. In non-FIFO mode, a dummy (or a valid) data has to be written to SCITXBUF for the first transmit interrupt to occur. This is the case when you enable the transmit interrupt for the first time and also when you re-enable (disable and then enable) the transmit interrupt. If TXINTENA is enabled after writing the data to SCITXBUF, it will not generate an interrupt. Reset type: SYSRSn 0h (R/W) = Disable TXRDY interrupt 1h (R/W) = Enable TXRDY interrupt

10.14.2.6 SCIRXST Register (Offset = 5h) [reset = 0h]

SCIRXST is shown in Figure 10-16 and described in Table 10-12.

Return to the Summary Table.

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 10-16. SCIRXST Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RXERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	RESERVED
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

Table 10-12. SCIRXST Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	RXERROR	R	0h	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. Reset type: SYSRSn 0h (R/W) = No error flags set 1h (R/W) = Error flag(s) set
6	RXRDY	R	0h	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. Reset type: SYSRSn 0h (R/W) = No new character in SCIRXBUF 1h (R/W) = Character ready to be read from SCIRXBUF
5	BRKDT	R	0h	SCI break-detect flag. The SCI sets this bit when a break condition occurs. A break conditionoccurs 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. Reset type: SYSRSn 0h (R/W) = No break condition 1h (R/W) = Break condition occurred

Table 10-12. SCIRXST Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	FE	R	0h	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. Reset type: SYSRSn 0h (R/W) = No framing error detected 1h (R/W) = Framing error detected
3	OE	R	0h	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. Reset type: SYSRSn 0h (R/W) = No overrun error detected 1h (R/W) = Overrun error detected
2	PE	R	0h	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. Reset type: SYSRSn 0h (R/W) = No parity error or parity is disabled 1h (R/W) = Parity error is detected
1	RXWAKE	R	0h	Receiver wake-up-detect flag Reset type: SYSRSn 0h (R/W) = No detection of a receiver wake-up condition 1h (R/W) = A value of 1 in this bit indicates detection of a receiver wake-up condition. In the address-bit multiprocessor mode (SCICCR.3 = 1), RXWAKE reflects the value of the address bit for the character contained in SCIRXBUF. In the idle-line multiprocessor mode, RXWAKE is set if the SCIRXD data line is detected as idle. 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	R	0h	Reserved

10.14.2.7 SCIRXEMU Register (Offset = 6h) [reset = 0h]

SCIRXEMU is shown in Figure 10-17 and described in Table 10-13.

Return to the Summary Table.

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 10-17. SCIRXEMU Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
ERXDT							
R-0h							

Table 10-13. SCIRXEMU Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	ERXDT	R	0h	Receive emulation buffer data Reset type: SYSRSn

10.14.2.8 SCIRXBUF Register (Offset = 7h) [reset = 0h]

SCIRXBUF is shown in Figure 10-18 and described in Table 10-14.

Return to the Summary Table.

When the current data received is shifted from RXSHF to the receiver buffer, flag bit RXRDY is set and the data is ready to be read. If the RXBKINTENA bit (SCICTL2.1) is set, this shift also causes an interrupt. When SCIRXBUF is read, the RXRDY flag is reset. SCIRXBUF is cleared by a system reset.

Figure 10-18. SCIRXBUF Register

15	14	13	12	11	10	9	8
SCIFFFE	SCIFFPE	RESERVED					
R-0h	R-0h	R-0h					
7	6	5	4	3	2	1	0
SAR							
R-0h							

Table 10-14. SCIRXBUF Register Field Descriptions

Bit	Field	Type	Reset	Description
15	SCIFFFE	R	0h	SCIFFFE. SCI FIFO Framing error flag bit (applicable only if the FIFO is enabled) Reset type: SYSRSn 0h (R/W) = No frame error occurred while receiving the character, in bits 7-0. This bit is associated with the character on the top of the FIFO. 1h (R/W) = A frame error occurred while receiving the character in bits 7-0. This bit is associated with the character on the top of the FIFO.
14	SCIFFPE	R	0h	SCIFFPE. SCI FIFO parity error flag bit (applicable only if the FIFO is enabled) Reset type: SYSRSn 0h (R/W) = No parity error occurred while receiving the character, in bits 7-0. This bit is associated with the character on the top of the FIFO. 1h (R/W) = A parity error occurred while receiving the character in bits 7-0. This bit is associated with the character on the top of the FIFO.
13-8	RESERVED	R	0h	Reserved
7-0	SAR	R	0h	Receive Character bits Reset type: SYSRSn

10.14.2.9 SCITXBUF Register (Offset = 9h) [reset = 0h]

SCITXBUF is shown in Figure 10-19 and described in Table 10-15.

Return to the Summary Table.

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 TXINTENA (SCICTL2.0) is set, this data transfer also causes an interrupt.

Figure 10-19. SCITXBUF Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXDT							
R/W-0h							

Table 10-15. SCITXBUF Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	TXDT	R/W	0h	Transmit data buffer Reset type: SYSRSn

10.14.2.10 SCIFFTX Register (Offset = Ah) [reset = A000h]

SCIFFTX is shown in Figure 10-20 and described in Table 10-16.

Return to the Summary Table.

SCIFFTX controls the transmit FIFO interrupt, FIFO enhancements, and reset for the SCI transmit and receive channels.

Figure 10-20. SCIFFTX Register

15	14	13	12	11	10	9	8
SCIRST	SCIFFENA	TXFIFORESET	TXFFST				
R/W-1h	R/W-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL				
R-0h	R-0/W1S-0h	R/W-0h	R/W-0h				

Table 10-16. SCIFFTX Register Field Descriptions

Bit	Field	Type	Reset	Description
15	SCIRST	R/W	1h	SCI Reset 0 Write 0 to reset the SCI transmit and receive channels. SCI FIFO register configuration bits will be left as is. 1 SCI FIFO can resume transmit or receive. SCIRST should be 1 even for Autobaud logic to work. Reset type: SYSRSn
14	SCIFFENA	R/W	0h	SCI FIFO enable Reset type: SYSRSn 0h (R/W) = SCI FIFO enhancements are disabled 1h (R/W) = SCI FIFO enhancements are enabled
13	TXFIFORESET	R/W	1h	Transmit FIFO reset Reset type: SYSRSn 0h (R/W) = Reset the FIFO pointer to zero and hold in reset 1h (R/W) = Re-enable transmit FIFO operation
12-8	TXFFST	R	0h	FIFO status Reset type: SYSRSn 0h (R/W) = Transmit FIFO is empty 1h (R/W) = Transmit FIFO has 1 words 2h (R/W) = Transmit FIFO has 2 words 3h (R/W) = Transmit FIFO has 3 words 4h (R/W) = Transmit FIFO has 4 words 5h (R/W) = Transmit FIFO has 5 words 6h (R/W) = Transmit FIFO has 6 words 7h (R/W) = Transmit FIFO has 7 words 8h (R/W) = Transmit FIFO has 8 words 9h (R/W) = Transmit FIFO has 9 words Ah (R/W) = Transmit FIFO has 10 words Bh (R/W) = Transmit FIFO has 11 words Ch (R/W) = Transmit FIFO has 12 words Dh (R/W) = Transmit FIFO has 13 words Eh (R/W) = Transmit FIFO has 14 words Fh (R/W) = Transmit FIFO has 15 words 10h (R/W) = Transmit FIFO has 16 words
7	TXFFINT	R	0h	Transmit FIFO interrupt Reset type: SYSRSn 0h (R/W) = TXFIFO interrupt has not occurred, read-only bit 1h (R/W) = TXFIFO interrupt has occurred, read-only bit

Table 10-16. SCIFFTX Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
6	TXFFINTCLR	R-0/W1S	0h	Transmit FIFO clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero 1h (R/W) = Write 1 to clear TXFFINT flag in bit 7
5	TXFFIENA	R/W	0h	Transmit FIFO interrrupt enable Reset type: SYSRSn 0h (R/W) = TX FIFO interrupt is disabled 1h (R/W) = TX FIFO interrupt is enabled. This interrupt is triggered whenever the transmit FIFO status (TXFFST) bits match (equal to or less than) the interrupt trigger level bits TXFFIL (bits 4-0).
4-0	TXFFIL	R/W	0h	TXFFIL4-0 Transmit FIFO interrupt level bits. The transmit FIFO generates an interrupt whenever the FIFO status bits (TXFFST4-0) are less than or equal to the FIFO level bits (TXFFIL4-0). The maximum value that can be assigned to these bits to generate an interrupt cannot be more than the depth of the TX FIFO. The default value of these bits after reset is 00000b. Users should set TXFFIL to best fit their application needs by weighing between the CPU overhead to service the ISR and the best possible usage of SCI bus bandwidth. Reset type: SYSRSn

10.14.2.11 SCIFFRX Register (Offset = Bh) [reset = 201Fh]

SCIFFRX is shown in Figure 10-21 and described in Table 10-17.

Return to the Summary Table.

SCIFFTX controls the receive FIFO interrupt, receive FIFO reset, and status of the receive FIFO overflow.

Figure 10-21. SCIFFRX Register

15	14	13	12	11	10	9	8
RXFFOVF	RXFFOVRCLR	RXFIFORESET	RXFFST				
R-0h	R-0/W1S-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL				
R-0h	W-0h	R/W-0h	R/W-1Fh				

Table 10-17. SCIFFRX Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RXFFOVF	R	0h	Receive FIFO overflow. This will function as flag, but cannot generate interrupt by itself. This condition will occur while receive interrupt is active. Receive interrupts should service this flag condition. Reset type: SYSRSn 0h (R/W) = Receive FIFO has not overflowed, read-only bit 1h (R/W) = Receive FIFO has overflowed, read-only bit. More than 16 words have been received in to the FIFO, and the first received word is lost
14	RXFFOVRCLR	R-0/W1S	0h	RXFFOVF clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFFOVF flag bit, Bit reads back a zero 1h (R/W) = Write 1 to clear RXFFOVF flag in bit 15
13	RXFIFORESET	R/W	1h	Receive FIFO reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Re-enable receive FIFO operation
12-8	RXFFST	R	0h	FIFO status Reset type: SYSRSn 0h (R/W) = Receive FIFO is empty 1h (R/W) = Receive FIFO has 1 words 2h (R/W) = Receive FIFO has 2 words 3h (R/W) = Receive FIFO has 3 words 4h (R/W) = Receive FIFO has 4 words 5h (R/W) = Receive FIFO has 5 words 6h (R/W) = Receive FIFO has 6 words 7h (R/W) = Receive FIFO has 7 words 8h (R/W) = Receive FIFO has 8 words 9h (R/W) = Receive FIFO has 9 words Ah (R/W) = Receive FIFO has 10 words Bh (R/W) = Receive FIFO has 11 words Ch (R/W) = Receive FIFO has 12 words Dh (R/W) = Receive FIFO has 13 words Eh (R/W) = Receive FIFO has 14 words Fh (R/W) = Receive FIFO has 15 words 10h (R/W) = Receive FIFO has 16 words

Table 10-17. SCIFFRX Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
7	RXFFINT	R	0h	Receive FIFO interrupt Reset type: SYSRSn 0h (R/W) = RXFIFO interrupt has not occurred, read-only bit 1h (R/W) = RXFIFO interrupt has occurred, read-only bit
6	RXFFINTCLR	W	0h	Receive FIFO interrupt clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFIFINT flag bit. Bit reads back a zero. 1h (R/W) = Write 1 to clear RXFFINT flag in bit 7
5	RXFFIENA	R/W	0h	Receive FIFO interrupt enable Reset type: SYSRSn 0h (R/W) = RX FIFO interrupt is disabled 1h (R/W) = RX FIFO interrupt is enabled. This interrupt is triggered whenever the receive FIFO status (RXFFST) bits match (equal to or greater than) the interrupt trigger level bits RXFFIL (bits 4-0).
4-0	RXFFIL	R/W	1Fh	Receive FIFO interrupt level bits The receive FIFO generates an interrupt whenever the FIFO status bits (RXFFST4-0) are greater than or equal to the FIFO level bits (RXFFIL4-0). The maximum value that can be assigned to these bits to generate an interrupt cannot be more than the depth of the RX FIFO. The default value of these bits after reset is 11111b. Users should set RXFFIL to best fit their application needs by weighing between the CPU overhead to service the ISR and the best possible usage of received SCI data. Reset type: SYSRSn

10.14.2.12 SCIFFCT Register (Offset = Ch) [reset = 0h]

SCIFFCT is shown in Figure 10-22 and described in Table 10-18.

Return to the Summary Table.

SCIFFCT contains the status of auto-baud detect, clears the auto-baud flag, and calibrate for A-detect bit.

Figure 10-22. SCIFFCT Register

15	14	13	12	11	10	9	8
ABD	ABDCLR	CDC	RESERVED				
R-0h	W-0h	R/W-0h	R-0h				
7	6	5	4	3	2	1	0
FFTXDLY							
R/W-0h							

Table 10-18. SCIFFCT Register Field Descriptions

Bit	Field	Type	Reset	Description
15	ABD	R	0h	Auto-baud detect (ABD) bit Reset type: SYSRSn 0h (R/W) = Auto-baud detection is not complete. "A","a" character has not been received successfully. 1h (R/W) = Auto-baud hardware has detected "A" or "a" character on the SCI receive register. Auto-detect is complete.
14	ABDCLR	W	0h	ABD-clear bit Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on ABD flag bit. Bit reads back a zero. 1h (R/W) = Write 1 to clear ABD flag in bit 15.
13	CDC	R/W	0h	CDC calibrate A-detect bit Reset type: SYSRSn 0h (R/W) = Disables auto-baud alignment 1h (R/W) = Enables auto-baud alignment
12-8	RESERVED	R	0h	Reserved
7-0	FFTXDLY	R/W	0h	FIFO transfer delay. These bits define the delay between every transfer from FIFO transmit bufferto transmit shift register. The delay is defined in the number of SCI serial baud clock cycles. The 8 bit register could define a minimum delay of 0 baud clock cycles and a maximum of 256 baud clock cycles In FIFO mode, the buffer (TXBUF) between the shift register and the FIFO should be filled only after the shift register has completed shifting of the last bit. This is required to pass on the delay between transfers to the data stream. In FIFO mode, TXBUF should not be treated as one additional level of buffer. The delayed transmit feature will help to create an auto-flow scheme without RTS/CTS controls as in standard UARTS. When SCI is configured for one stop-bit, delay introduced by FFTXDLY between one frame and the next frame is equal to number of baud clock cycles that FFTXDLY is set to. When SCI is configured for two stop-bits, delay introduced by FFTXDLY between one frame and the next frame is equal to number of baud clock cycles that FFTXDLY is set to minus 1. Reset type: SYSRSn

10.14.2.13 SCIPRI Register (Offset = Fh) [reset = 0h]

SCIPRI is shown in Figure 10-23 and described in Table 10-19.

Return to the Summary Table.

SCIPRI determines what happens when an emulation suspend event occurs.

Figure 10-23. SCIPRI Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			FREESOFT		RESERVED		
R-0h			R/W-0h		R-0h		

Table 10-19. SCIPRI Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-5	RESERVED	R	0h	Reserved
4-3	FREESOFT	R/W	0h	These bits determine what occurs when an emulation suspend event occurs (for example, when the debugger hits a 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. Reset type: SYSRSn 0h (R/W) = Immediate stop on suspend 1h (R/W) = Complete current receive/transmit sequence before stopping 2h (R/W) = Free run 3h (R/W) = Free run
2-0	RESERVED	R	0h	Reserved

Inter-Integrated Circuit Module (I2C)

This chapter describes the features and operation of the inter-integrated circuit (I2C) module. The I2C module provides an interface between one of these devices and devices compliant with the NXP Semiconductors Inter-IC bus (I2C bus) specification version 2.1, and connected by way of an I2C bus. External components attached to this 2-wire serial bus can transmit/receive 1 to 8-bit data to/from the device through the I2C module. This guide assumes the reader is familiar with the I2C bus specification.

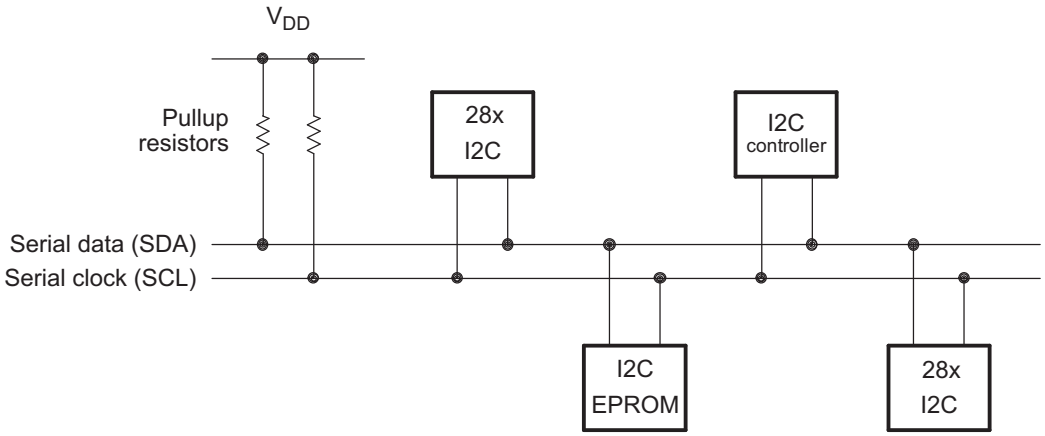
NOTE: A unit of data transmitted or received by the I2C module can have fewer than 8 bits; however, for convenience, a unit of data is called a data byte throughout this document. The number of bits in a data byte is selectable via the BC bits of the mode register, I2CMDR.

Topic **Page**

11.1 Introduction

The I2C module supports any slave or master I2C-compatible device. Figure 11-1 shows an example of multiple I2C modules connected for a two-way transfer from one device to other devices.

Figure 11-1. Multiple I2C Modules Connected



11.1.1 Features

The I2C module has the following features:

- Compliance with the NXP Semiconductors I2C bus specification (version 2.1):
  - Support for 8-bit format transfers
  - 7-bit and 10-bit addressing modes
  - General call
  - START byte mode
  - Support for multiple master-transmitters and slave-receivers
  - Support for multiple slave-transmitters and master-receivers
  - Combined master transmit/receive and receive/transmit mode
  - Data transfer rate from 10 kbps up to 400 kbps (Fast-mode)
- Receive FIFO and Transmitter FIFO (16-deep x 8-bit FIFO)
- Supports two ePIE interrupts:
  - I2Cx Interrupt – Any of the below events can be configured to generate an I2Cx interrupt:
    - Transmit-data ready
    - Receive-data ready
    - Register-access ready
    - No-acknowledgment received
    - Arbitration lost
    - Stop condition detected
    - Addressed as slave
  - I2Cx_FIFO interrupts:
    - Transmit FIFO interrupt
    - Receive FIFO interrupt
- Module enable/disable capability
- Free data format mode

11.1.2 Features Not Supported

The I2C module does not support:

- High-speed mode (Hs-mode)
- CBUS-compatibility mode

11.1.3 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 Section 11.6. 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: .

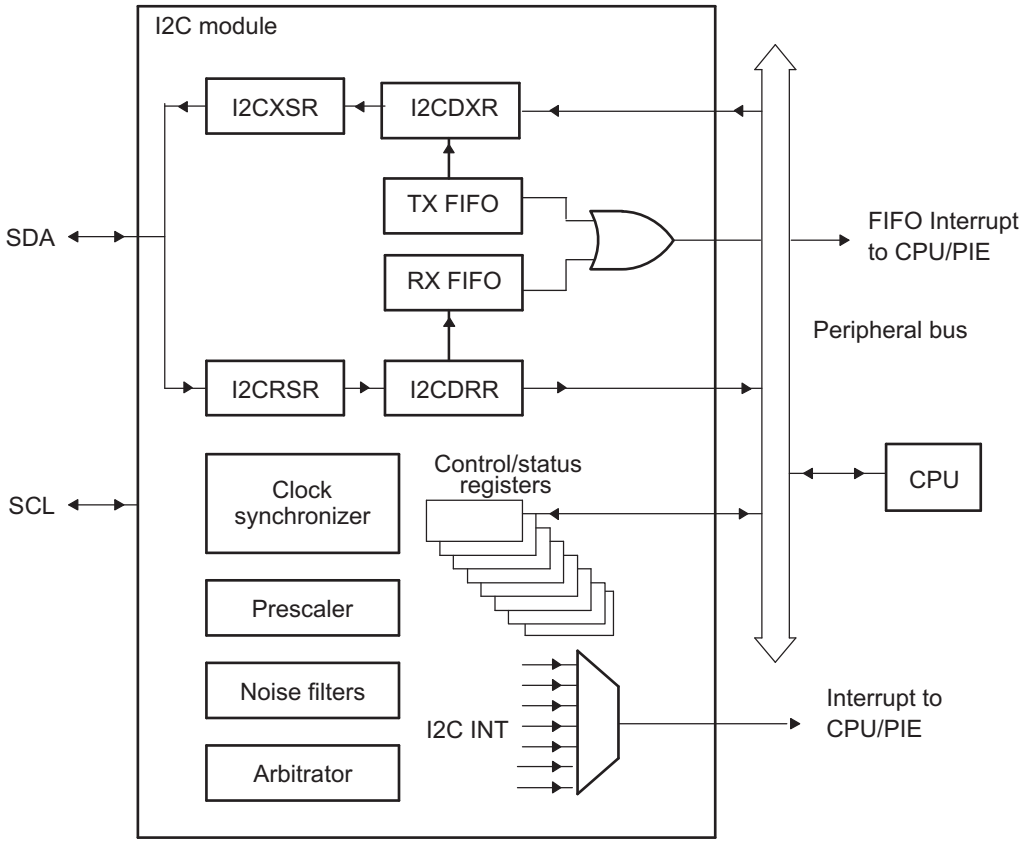
- Standard Mode: Send exactly n data values, where n is a value you program in an I2C module register. See Section 11.6 for more information.
- Repeat Mode: Keep sending data values until you use software to initiate a STOP condition or a new START condition. See *Registers* for RM bit information.

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 11-2 shows the four registers used for transmission and reception in non-FIFO mode. The CPU writes data for transmission to I2CDXR and reads received data from I2CDRR. When the I2C module is configured as a transmitter, data written to I2CDXR is copied to I2CXSR and shifted out on the SDA pin one bit at a time. When the I2C module is configured as a receiver, received data is shifted into I2CRSR and then copied to I2CDRR.

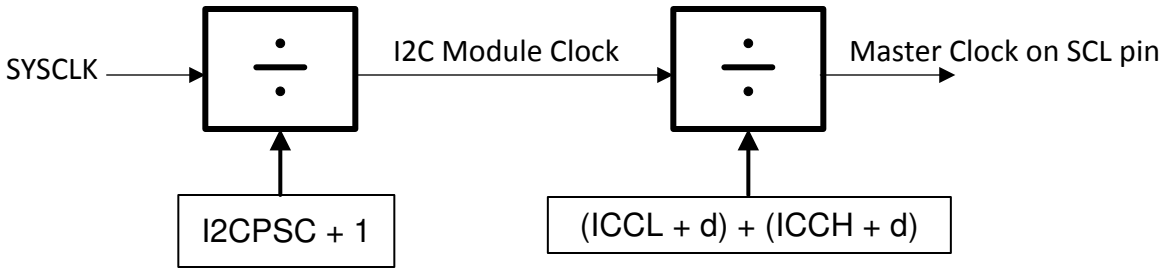
Figure 11-2. I2C Module Conceptual Block Diagram



11.1.4 Clock Generation

The I2C module clock determines the frequency at which the I2C module operates. A programmable prescaler in the I2C module divides down the SYSCLK to produce the I2C module clock and this I2C module clock is divided further to produce the I2C master clock on the SCL pin. Figure 11-3 shows the clock generation diagram for I2C module.

Figure 11-3. Clocking Diagram for the I2C Module



To specify the divide-down value, initialize the IPSC field of the prescaler register, I2CPSC. The resulting frequency is:

$$\text{I2C Module Clock (Fmod)} = \frac{\text{SYSCLK}}{(\text{I2CPSC} + 1)}$$

NOTE: To meet all of the I2C protocol timing specifications, the I2C module clock must be 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.

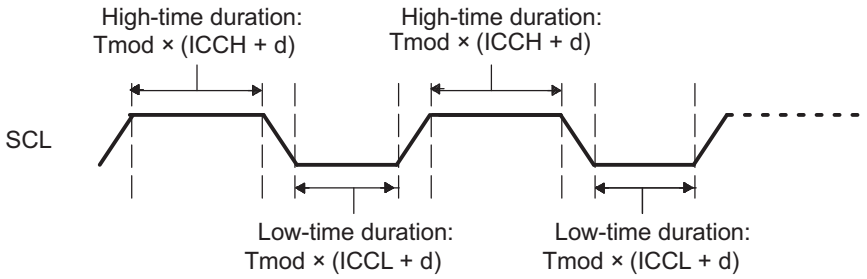
The master clock appears on the SCL pin when the I2C module is configured to be a master on the I2C bus. This clock controls the timing of communication between the I2C module and a slave. As shown in Figure 11-3, a second clock divider in the I2C module divides down the module clock to produce the master 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 11.1.5 for the master clock frequency equation.

11.1.5 I2C Clock Divider Registers (I2CCLKL and I2CCLKH)

As explained in Section 11.1.4, when the I2C module is a master, the I2C module clock is divided down further to use as the master clock on the SCL pin. As shown in Figure 11-4, the shape of the master clock depends on two divide-down values:

- ICCL in I2CCLKL. For each master clock cycle, ICCL determines the amount of time the signal is low.
- ICCH in I2CCKLH. For each master clock cycle, ICCH determines the amount of time the signal is high.

Figure 11-4. The Roles of the Clock Divide-Down Values (ICCL and ICCH)



11.1.5.1 Formula for the Master Clock Period

The master clock period (Tmst) is a multiple of the period of the I2C Module Clock (Tmod):

$$\text{Master Clock period (Tmst)} = \frac{[\,(\text{ICCH} + d) + (\text{ICCL} + d)\,]}{\text{I2C Module Clock (Fmod)}}$$

where d depends on the divide-down value IPSC, as shown in Table 11-1. IPSC is described in the I2CPSC register.

Table 11-1. Dependency of Delay d on the Divide-Down Value IPSC

IPSC	d
0	7
1	6
Greater than 1	5

11.2 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPyPUD register.

See the *GPIO* chapter for more details on GPIO mux and settings.

11.3 I2C Module Operational Details

This section provides an overview of the I2C bus protocol and how it is implemented.

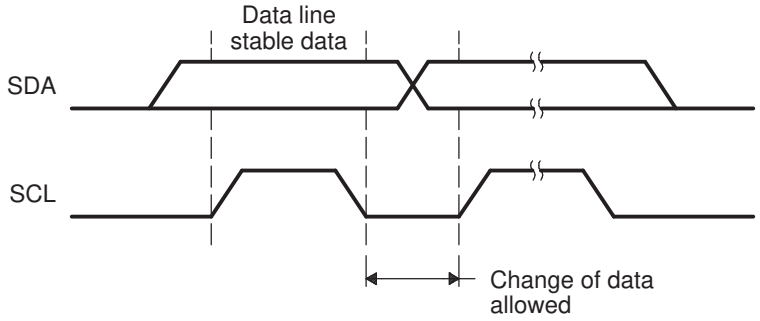
11.3.1 Input and Output Voltage Levels

One clock pulse is generated by the master device for each data bit transferred. Due to a variety of different technology devices that can be connected to the I2C bus, the levels of logic 0 (low) and logic 1 (high) are not fixed and depend on the associated level of V_{DD}. For details, see the data manual for your particular device.

11.3.2 Data Validity

The data on SDA must be stable during the high period of the clock (see Figure 11-5). The high or low state of the data line, SDA, should change only when the clock signal on SCL is low.

Figure 11-5. Bit Transfer on the I2C bus



11.3.3 Operating Modes

The I2C module has four basic operating modes to support data transfers as a master and as a slave. See Table 11-2 for the names and descriptions of the modes.

If the I2C module is a master, it begins as a master-transmitter and typically transmits an address for a particular slave. When giving data to the slave, the I2C module must remain a master-transmitter. To receive data from a slave, the I2C module must be changed to the master-receiver mode.

If the I2C module is a slave, it begins as a slave-receiver and typically sends acknowledgment when it recognizes its slave address from a master. If the master will be sending data to the I2C module, the module must remain a slave-receiver. If the master has requested data from the I2C module, the module must be changed to the slave-transmitter mode.

Table 11-2. 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 11.3.7 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 11.3.7 for more details.

Table 11-2. Operating Modes of the I2C Module (continued)

Operating Mode	Description
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 I2C module is a master and transmits control information and data to a slave.
	All masters begin in this mode. In this mode, data assembled in any of the 7-bit/10-bit addressing formats is shifted out on SDA. The bit shifting is synchronized with the clock pulses generated by the I2C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the device is required (XSMT = 0 in I2CSTR) after a byte has been transmitted.

To summarize, SCL will be held low in the following conditions:

- When an overrun condition is detected (RSFULL = 1), in Slave-receiver mode.
- When an underflow condition is detected (XSMT = 0), in Slave-transmitter mode.

I2C slave nodes have to accept and provide data when the I2C master node requests it.

- To release SCL in slave-receiver mode, read data from I2CDRR.
- To release SCL in slave-transmitter mode, write data to I2CDXR.
- To force a release without handling the data, reset the module using the I2CMDR.IRS bit.

Table 11-3. 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)

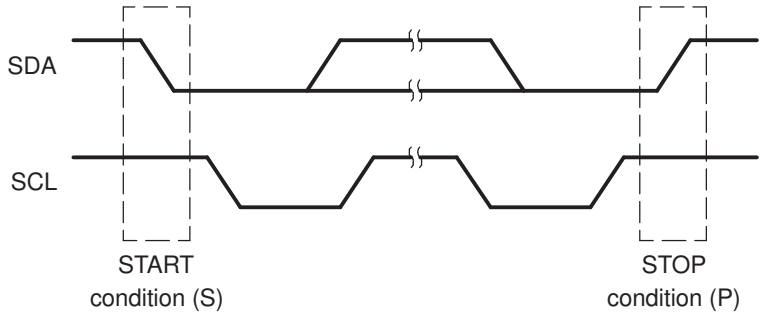
(1) S = START condition; A = Address; D = Data byte; P = STOP condition;

11.3.4 I2C Module START and STOP Conditions

START and STOP conditions can be generated by the I2C module when the module is configured to be a master on the I2C bus. As shown in Figure 11-6:

- 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 11-6. 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 *Registers* Section 11.6.

The I2C peripheral cannot detect a START or STOP condition while it is in reset (IRS = 0). The BB bit will remain in the cleared state (BB = 0) while the I2C peripheral is in reset (IRS = 0). When the I2C peripheral is taken out of reset (IRS set to 1) the BB bit will not correctly reflect the I2C bus status until a START or STOP condition is detected.

Follow these steps before initiating the first data transfer with I2C:

1. After taking the I2C peripheral out of reset by setting the IRS bit to 1, wait a period larger than the total time taken for the longest data transfer in the application. By waiting for a period of time after I2C comes out of reset, users can ensure that at least one START or STOP condition will have occurred on the I2C bus and been captured by the BB bit. After this period, the BB bit will correctly reflect the state of the I2C bus.
2. Check the BB bit and verify that BB = 0 (bus not busy) before proceeding.
3. Begin data transfers.

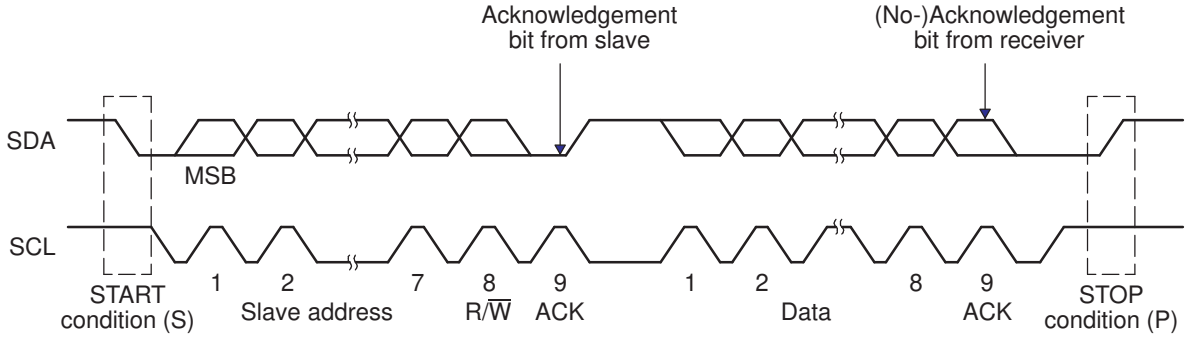
Not resetting the I2C peripheral in between transfers ensures that the BB bit reflects the actual bus status. If users must reset the I2C peripheral in between transfers, repeat steps 1 through 3 every time the I2C peripheral is taken out of reset.

11.3.5 Serial Data Formats

Figure 11-7 shows an example of a data transfer on the I2C bus. The I2C module supports 1 to 8-bit data values. In Figure 11-7, 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 11-7 is the 7-bit addressing format. The I2C module supports the formats shown in Figure 11-8 through Figure 11-10 and described in the paragraphs that follow the figures.

NOTE: In Figure 11-7 through Figure 11-10, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

Figure 11-7. I2C Module Data Transfer (7-Bit Addressing with 8-bit Data Configuration Shown)



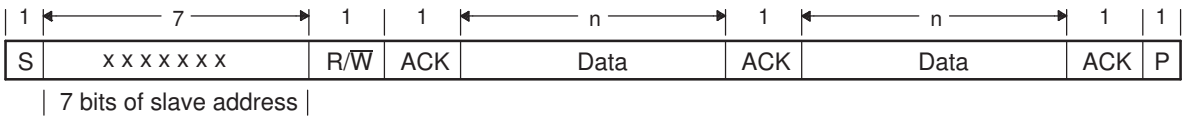
11.3.5.1 7-Bit Addressing Format

The 7-bit addressing format is the default format after reset. Disabling expanded address (I2CMDR.XA = 0) and free data format (I2CMDR.FDF = 0) enables 7-bit addressing format.

In this format (see Figure 11-8), 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 I2C master writes (transmits) data to the addressed slave. This can be achieved by setting I2CMDR.TRX = 1 (Transmitter mode)
- R/$\overline{W}$ = 1: The I2C master reads (receives) data from the slave. This can be achieved by setting I2CMDR.TRX = 0 (Receiver mode)

Figure 11-8. I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMDR)



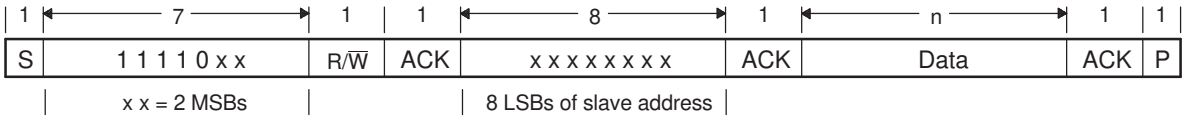
An extra clock cycle dedicated for acknowledgment (ACK) is inserted after each byte. If the ACK bit is inserted by the slave after the first byte from the master, it is followed by n bits of data from the transmitter (master or slave, depending on the R/$\overline{W}$ bit). n is a number from 1 to 8 determined by the bit count (BC) field of I2CMDR. After the data bits have been transferred, the receiver inserts an ACK bit.

11.3.5.2 10-Bit Addressing Format

The 10-bit addressing format can be enabled by setting expanded address (I2CMDR.XA = 1) and disabling free data format (I2CMDR.FDF = 0).

The 10-bit addressing format (see Figure 11-9) 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/$\overline{W}$. The second byte is the remaining 8 bits of the 10-bit slave address. The slave must send acknowledgment after each of the two byte transfers. Once the master has written the second byte to the slave, the master can either write data or use a repeated START condition to change the data direction. For more details about using 10-bit addressing, see the NXP Semiconductors I2C bus specification.

Figure 11-9. I2C Module 10-Bit Addressing Format (FDF = 0, XA = 1 in I2CMDR)

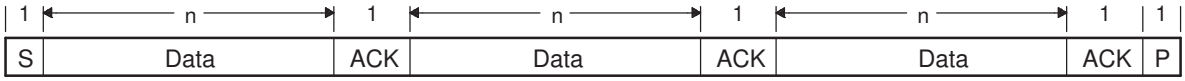


11.3.5.3 Free Data Format

The free data format can be enabled by setting I2CMDR. FDF = 1.

In this format (see Figure 11-10), the first byte after a START condition (S) is a data byte. An ACK bit is inserted after each data byte, which can be from 1 to 8 bits, depending on the BC field of I2CMDR. No address or data-direction bit is sent. Therefore, the transmitter and the receiver must both support the free data format, and the direction of the data must be constant throughout the transfer.

Figure 11-10. I2C Module Free Data Format (FDF = 1 in I2CMDR)



NOTE: The free data format is not supported in the digital loopback mode (I2CMDR.DLB = 1).

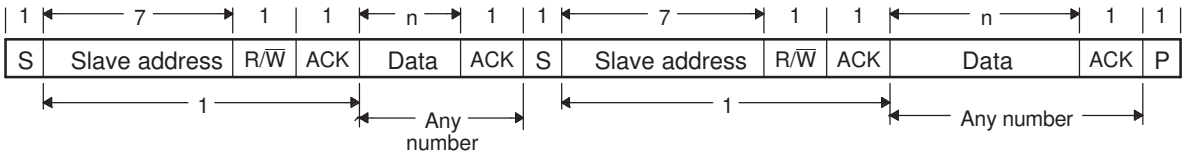
Table 11-4. How the MST and FDF Bits of I2CMDR Affect the Role of the TRX Bit of I2CMDR

MST	FDF	I2C Module State	Function of TRX
0	0	In slave mode but not free data format mode	TRX is a don't care. Depending on the command from the master, the I2C module responds as a receiver or a transmitter.
0	1	In slave mode and free data format mode	The free data format mode requires that the I2C module remains the transmitter or the receiver throughout the transfer. TRX identifies the role of the I2C module: TRX = 1: The I2C module is a transmitter. TRX = 0: The I2C module is a receiver.
1	0	In master mode but not free data format mode	TRX = 1: The I2C module is a transmitter. TRX = 0: The I2C module is a receiver.
1	1	In master mode and free data format mode	TRX = 0: The I2C module is a receiver. TRX = 1: The I2C module is a transmitter.

11.3.5.4 Using a Repeated START Condition

I2C master can communicate with multiple slave addresses without having to give up control of the I2C bus by driving a STOP condition. This can be achieved by driving another START condition at the end of each data type. The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. Figure 11-11 shows a repeated START condition in the 7-bit addressing format.

Figure 11-11. Repeated START Condition (in This Case, 7-Bit Addressing Format)



NOTE: In Figure 11-11, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

11.3.6 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 11-5 summarizes the various ways you can tell the I2C module to send a NACK bit.

Table 11-5. Ways to Generate a NACK Bit

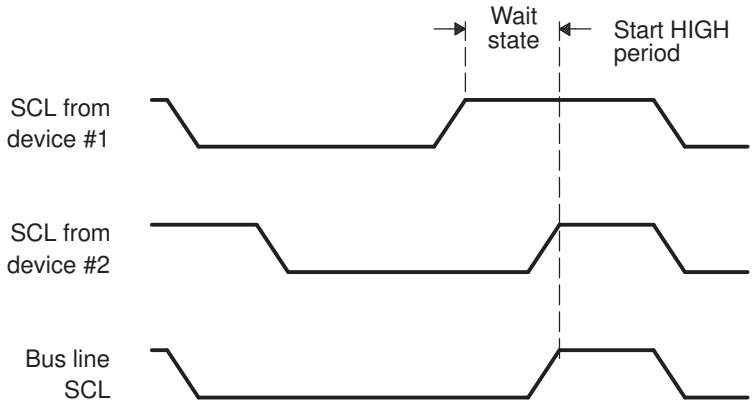
I2C Module Condition	NACK Bit Generation Options
Slave-receiver modes	• Allow an overrun condition (RSFULL = 1 in I2CSTR) • Reset the module (IRS = 0 in I2CMDR) • Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive
Master-receiver mode AND Repeat mode (RM = 1 in I2CMDR)	• Generate a STOP condition (STP = 1 in I2CMDR) • Reset the module (IRS = 0 in I2CMDR) • Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive
Master-receiver mode AND Nonrepeat mode (RM = 0 in I2CMDR)	• If STP = 1 in I2CMDR, allow the internal data counter to count down to 0 and thus force a STOP condition • If STP = 0, make STP = 1 to generate a STOP condition • Reset the module (IRS = 0 in I2CMDR). = 1 to generate a STOP condition • Set the NACKMOD bit of I2CMDR before the rising edge of the last data bit you intend to receive

11.3.7 Clock Synchronization

Under normal conditions, only one master device generates the clock signal, SCL. During the arbitration procedure, however, there are two or more masters and the clock must be synchronized so that the data output can be compared. Figure 11-12 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 11-12. Synchronization of Two I2C Clock Generators During Arbitration



11.3.8 Arbitration

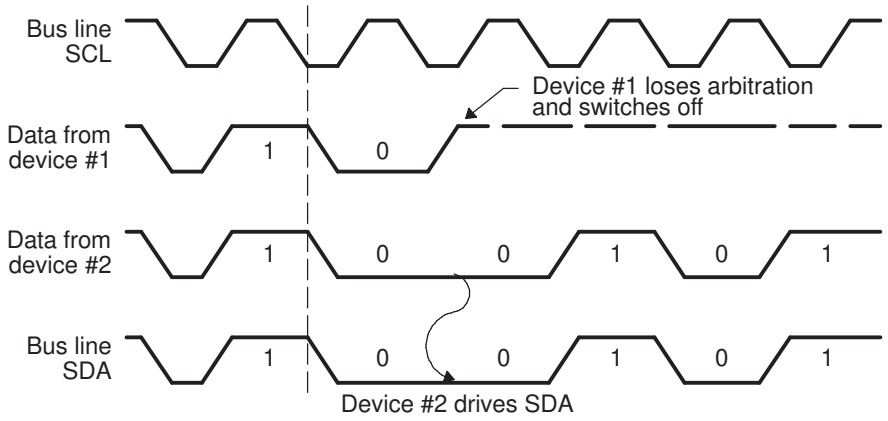
If two or more master-transmitters attempt to start a transmission on the same bus at approximately the same time, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. Figure 11-13 illustrates the arbitration procedure between two devices. The first master-transmitter that releases the SDA line high is overruled by another master-transmitter that drives the SDA low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If the I2C module is the losing master, it switches to the slave-receiver mode, sets the arbitration lost (ARBL) flag, and generates the arbitration-lost interrupt request.

If during a serial transfer the arbitration procedure is still in progress when a repeated START condition or a STOP condition is transmitted to SDA, the master-transmitters involved must send the repeated START condition or the STOP condition at the same position in the format frame. Arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

Figure 11-13. Arbitration Procedure Between Two Master-Transmitters



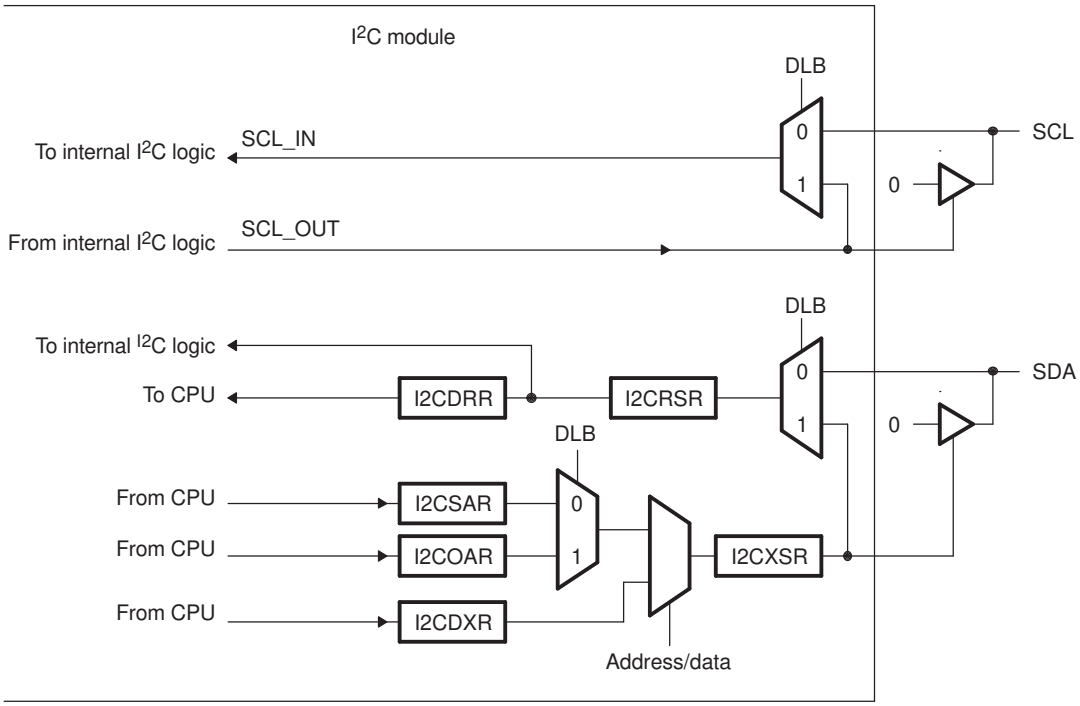
11.3.9 Digital Loopback Mode

The I2C module support a self-test mode called digital loopback, which is enabled by setting the DLB bit in the I2CMDR register. In this mode, data transmitted out of the I2CDXR register is received in the I2CDRR register. The data follows an internal path, and takes n cycles to reach I2CDRR, where:

n = 8 * (SYSCLK) / (I2C module clock (Fmod))

The transmit clock and the receive clock are the same. The address seen on the external SDA pin is the address in the I2COAR register. Figure 11-14 shows the signal routing in digital loopback mode.

Figure 11-14. Pin Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit



NOTE: The free data format (I2CMDR.FDF = 1) is not supported in digital loopback mode.

11.4 Interrupt Requests Generated by the I2C Module

Each I2C module can generate two CPU interrupts.

1. Basic I2C interrupt: Possible basic I2C interrupt sources which can trigger this interrupt are described in Section 11.4.1.
2. I2C FIFO interrupt: Possible I2C FIFO interrupt sources which can trigger this interrupt are described in Section 11.4.2

11.4.1 Basic I2C Interrupt Requests

The I2C module generates the interrupt requests described in Table 11-6. As shown in Figure 11-15, 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.

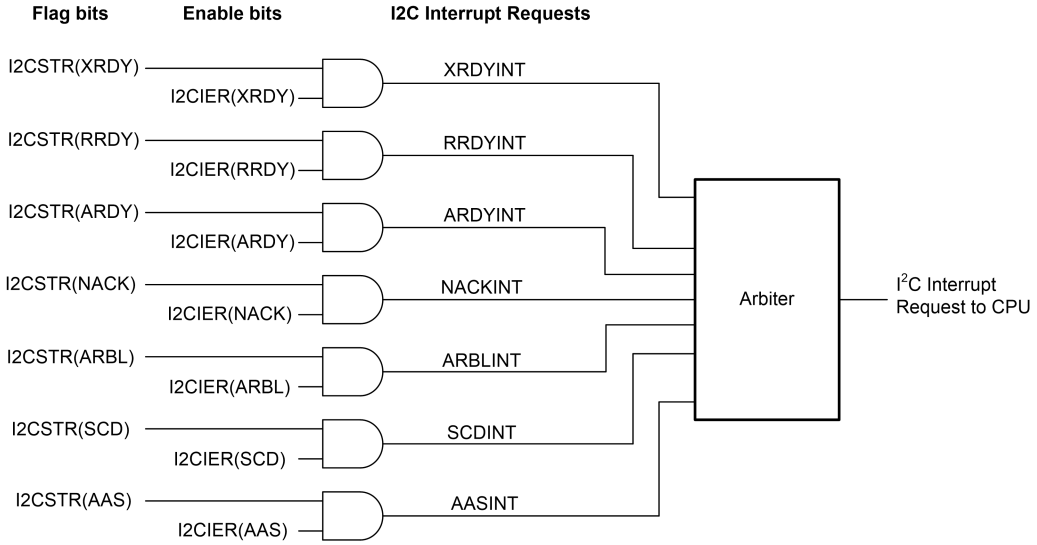
After the CPU reads I2CISRC, the following events occur:

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 11-6. Descriptions of the Basic I2C Interrupt Requests

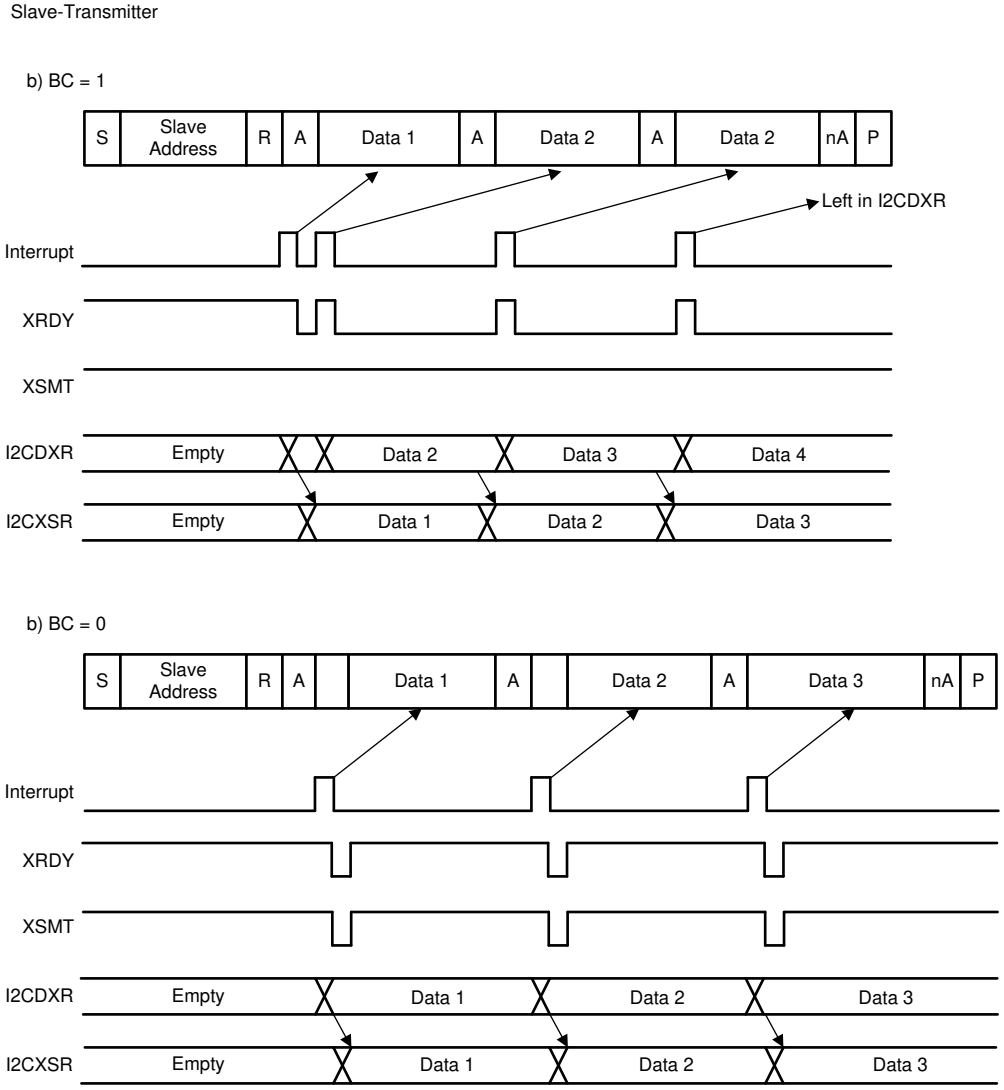
I2C Interrupt Request	Interrupt Source
XRDYINT	Transmit ready condition: The data transmit register (I2CDXR) is ready to accept new data because the previous data has been copied from I2CDXR to the transmit shift register (I2CXSR). As an alternative to using XRDYINT, the CPU can poll the XRDY bit of 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.
ARBLINT	Arbitration-lost condition: The I2C module has lost an arbitration contest with another master-transmitter. As an alternative to using ARBLINT, the CPU can poll the ARBL 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 11-15. Enable Paths of the I2C Interrupt Requests



The I2C module has a backwards compatibility bit (BC) in the I2CEMDR register. The timing diagram in Figure 11-16 demonstrates the effect the backwards compatibility bit has on I2C module registers and interrupts when configured as a slave-transmitter.

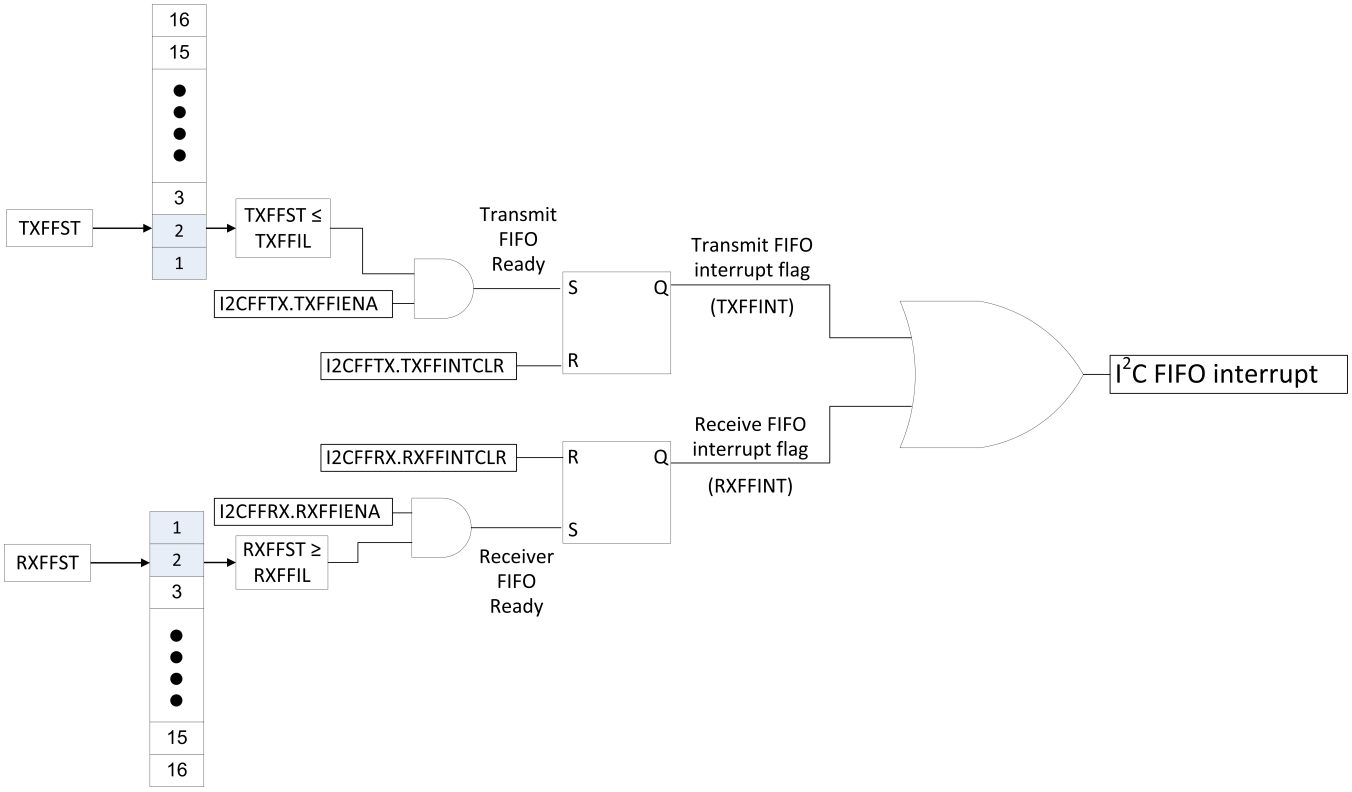
Figure 11-16. Backwards Compatibility Mode Bit, Slave Transmitter



11.4.2 I2C FIFO Interrupts

In addition to the seven basic I2C interrupts, the transmit and receive FIFOs each contain the ability to generate an interrupt (I2CINT2A). The transmit FIFO can be configured to generate an interrupt after transmitting a defined number of bytes, up to 16. The receive FIFO can be configured to generate an interrupt after receiving a defined number of bytes, up to 16. These two interrupt sources are ORed together into a single maskable CPU interrupt. Figure 11-17 shows the structure of I2C FIFO interrupt. The interrupt service routine can then read the FIFO interrupt status flags to determine from which source the interrupt came. See the I2C transmit FIFO register (I2CFFTX) and the I2C receive FIFO register (I2CFFRX) descriptions.

Figure 11-17. I2C_FIFO_interrupt



11.5 Resetting or Disabling the I2C Module

You can reset or disable the I2C module in two ways:

- Write 0 to the I2C reset bit (IRS) in the I2C mode register (I2CMDR). All status bits (in I2CSTR) are forced to their default values, and the I2C module remains disabled until IRS is changed to 1. The SDA and SCL pins are in the high-impedance state.
- Initiate a device reset by driving the $\overline{\text{XRS}}$ pin low. The entire device is reset and is held in the reset state until you drive the pin high. When the $\overline{\text{XRS}}$ pin is released, all I2C module registers are reset to their default values. The IRS bit is forced to 0, which resets the I2C module. The I2C module stays in the reset state until you write 1 to IRS.

The IRS must be 0 while you configure or reconfigure the I2C module. Forcing IRS to 0 can be used to save power and to clear error conditions.

11.6 I2C Registers

This section describes the C28x I2C Module Registers.

11.6.1 I2C Base Addresses

Table 11-7. I2C Base Address Table

Bit Field Name		Base Address
Instance	Structure	
I2caRegs	I2C_REGS	0x0000_7900

11.6.2 I2C_REGS Registers

Table 11-8 lists the I2C_REGS registers. All register offset addresses not listed in Table 11-8 should be considered as reserved locations and the register contents should not be modified.

Table 11-8. I2C_REGS Registers

Offset	Acronym	Register Name	Write Protection	Section
0h	I2COAR	I2C Own address		Go
1h	I2CIER	I2C Interrupt Enable		Go
2h	I2CSTR	I2C Status		Go
3h	I2CCLKL	I2C Clock low-time divider		Go
4h	I2CCLKH	I2C Clock high-time divider		Go
5h	I2CCNT	I2C Data count		Go
6h	I2CDRR	I2C Data receive		Go
7h	I2CSAR	I2C Slave address		Go
8h	I2CDXR	I2C Data Transmit		Go
9h	I2CMDR	I2C Mode		Go
Ah	I2CISRC	I2C Interrupt Source		Go
Bh	I2CEMDR	I2C Extended Mode		Go
Ch	I2CPSC	I2C Prescaler		Go
20h	I2CFFTX	I2C FIFO Transmit		Go
21h	I2CFFRX	I2C FIFO Receive		Go

Complex bit access types are encoded to fit into small table cells. Table 11-9 shows the codes that are used for access types in this section.

Table 11-9. I2C_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R -0	Read Returns 0s
Write Type		
W	W	Write
W1C	W 1C	Write 1 to clear
W1S	W 1S	Write 1 to set
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

11.6.2.1 I2COAR Register (Offset = 0h) [reset = 0h]

I2COAR is shown in Figure 11-18 and described in Table 11-10.

Return to the Summary Table.

The I2C own address register (I2COAR) is a 16-bit register. The I2C module uses this register to specify its own slave address, which distinguishes it from other slaves connected to the I2C-bus. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6-0 are used
write 0s to bits 9-7.

Figure 11-18. I2COAR Register

15	14	13	12	11	10	9	8
RESERVED						OAR	
R-0h						R/W-0h	
7	6	5	4	3	2	1	0
OAR							
R/W-0h							

Table 11-10. I2COAR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-10	RESERVED	R	0h	Reserved
9-0	OAR	R/W	0h	In 7-bit addressing mode (XA = 0 in I2CMDR): 00h-7Fh Bits 6-0 provide the 7-bit slave address of the I2C module. Write 0s to bits 9-7. In 10-bit addressing mode (XA = 1 in I2CMDR): 000h-3FFh Bits 9-0 provide the 10-bit slave address of the I2C module. Reset type: SYSRSn

11.6.2.2 I2CIER Register (Offset = 1h) [reset = 0h]

I2CIER is shown in Figure 11-19 and described in Table 11-11.

Return to the Summary Table.

I2CIER is used by the CPU to individually enable or disable I2C interrupt requests.

Figure 11-19. I2CIER Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	AAS	SCD	XRDY	RRDY	ARDY	NACK	ARBL
R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 11-11. I2CIER Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	AAS	R/W	0h	Addressed as slave interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
5	SCD	R/W	0h	Stop condition detected interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
4	XRDY	R/W	0h	Transmit-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
3	RRDY	R/W	0h	Receive-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
2	ARDY	R/W	0h	Register-access-ready interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
1	NACK	R/W	0h	No-acknowledgment interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
0	ARBL	R/W	0h	Arbitration-lost interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled

11.6.2.3 I2CSTR Register (Offset = 2h) [reset = 410h]

I2CSTR is shown in Figure 11-20 and described in Table 11-12.

Return to the Summary Table.

The I2C status register (I2CSTR) is a 16-bit register used to determine which interrupt has occurred and to read status information.

Figure 11-20. I2CSTR Register

15	14	13	12	11	10	9	8
RESERVED	SDIR	NACKSNT	BB	RSFULL	XSMT	AAS	AD0
R-0h	R/W1C-0h	R/W1C-0h	R-0h	R-0h	R-1h	R-0h	R-0h
7	6	5	4	3	2	1	0
RESERVED		SCD	XRDY	RRDY	ARDY	NACK	ARBL
R/W-0h		R/W1C-0h	R-1h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h

Table 11-12. I2CSTR Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reserved
14	SDIR	R/W1C	0h	Slave direction bit Reset type: SYSRSn 0h (R/W) = I2C is not addressed as a slave transmitter. SDIR is cleared by one of the following events: - It is manually cleared. To clear this bit, write a 1 to it. - Digital loopback mode is enabled. - A START or STOP condition occurs on the I2C bus. 1h (R/W) = I2C is addressed as a slave transmitter.
13	NACKSNT	R/W1C	0h	NACK sent bit. This bit is used when the I2C module is in the receiver mode. One instance in which NACKSNT is affected is when the NACK mode is used (see the description for NACKMOD in Reset type: SYSRSn 0h (R/W) = NACK not sent. NACKSNT bit is cleared by any one of the following events: - It is manually cleared. To clear this bit, write a 1 to it. - The I2C module is reset (either when 0 is written to the IRS bit of I2CMDR or when the whole device is reset). 1h (R/W) = NACK sent: A no-acknowledge bit was sent during the acknowledge cycle on the I2C-bus.
12	BB	R	0h	Bus busy bit. BB indicates whether the I2C-bus is busy or is free for another data transfer. See the paragraph following the table for more information Reset type: SYSRSn 0h (R/W) = Bus free. BB is cleared by any one of the following events: - The I2C module receives or transmits a STOP bit (bus free). - The I2C module is reset. 1h (R/W) = Bus busy: The I2C module has received or transmitted a START bit on the bus.

Table 11-12. I2CSTR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
11	RSFULL	R	0h	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. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = Overrun detected
10	XSMT	R	1h	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. Reset type: SYSRSn 0h (R/W) = Underflow detected (empty) 1h (R/W) = No underflow detected (not empty). XSMT is set by one of the following events: - Data is written to I2CDXR. - The I2C module is reset
9	AAS	R	0h	Addressed-as-slave bit Reset type: SYSRSn 0h (R/W) = In the 7-bit addressing mode, the AAS bit is cleared when receiving a NACK, a STOP condition, or a repeated START condition. In the 10-bit addressing mode, the AAS bit is cleared when receiving a NACK, a STOP condition, or by a slave address different from the I2C peripheral's own slave address. 1h (R/W) = The I2C module has recognized its own slave address or an address of all zeros (general call).
8	AD0	R	0h	Address 0 bits Reset type: SYSRSn 0h (R/W) = AD0 has been cleared by a START or STOP condition. 1h (R/W) = An address of all zeros (general call) is detected.
7-6	RESERVED	R/W	0h	Reserved
5	SCD	R/W1C	0h	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. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = A STOP condition has been detected on the I2C bus.

Table 11-12. I2CSTR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	XRDY	R	1h	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 When in FIFO mode, use TXFFINT instead. Reset type: SYSRSn 0h (R/W) = I2CDXR not ready. XRDY is cleared when data is written to I2CDXR. 1h (R/W) = I2CDXR ready: Data has been copied from I2CDXR to I2CXSR. XRDY is also forced to 1 when the I2C module is reset.
3	RRDY	R/W1C	0h	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 When in FIFO mode, use RXFFINT instead. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = I2CDRR ready: Data has been copied from I2CRSR to I2CDRR.
2	ARDY	R/W1C	0h	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. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = 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 11-12. I2CSTR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
1	NACK	R/W1C	0h	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 noacknowledge bit (NACK) from the receiver. The CPU can poll NACK or use the NACK interrupt request. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = 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	ARBL	R/W1C	0h	Arbitration-lost interrupt flag bit (only applicable when the I2C module is a master-transmitter). ARBL primarily indicates when the I2C module has lost an arbitration contest with another mastertransmitter. The CPU can poll ARBL or use the ARBL interrupt request. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = 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.

11.6.2.4 I2CCLKL Register (Offset = 3h) [reset = 0h]

I2CCLKL is shown in Figure 11-21 and described in Table 11-13.

Return to the Summary Table.

I2C Clock low-time divider

Figure 11-21. I2CCLKL Register

15	14	13	12	11	10	9	8
I2CCLKL							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCLKL							
R/W-0h							

Table 11-13. I2CCLKL Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	I2CCLKL	R/W	0h	Clock low-time divide-down value. To produce the low time duration of the master clock, the period of the module clock is multiplied by (ICCL + d). d is an adjustment factor based on the prescaler. See the Clock Divider Registers section of the Introduction for details. Note: These bits must be set to a non-zero value for proper I2C clock generation. Reset type: SYSRSn

11.6.2.5 I2CCLKH Register (Offset = 4h) [reset = 0h]

I2CCLKH is shown in Figure 11-22 and described in Table 11-14.

Return to the Summary Table.

I2C Clock high-time divider

Figure 11-22. I2CCLKH Register

15	14	13	12	11	10	9	8
I2CCLKH							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCLKH							
R/W-0h							

Table 11-14. I2CCLKH Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	I2CCLKH	R/W	0h	Clock high-time divide-down value. To produce the high time duration of the master clock, the period of the module clock is multiplied by (ICCL + d). d is an adjustment factor based on the prescaler. See the Clock Divider Registers section of the Introduction for details. Note: These bits must be set to a non-zero value for proper I2C clock generation. Reset type: SYSRSn

11.6.2.6 I2CCNT Register (Offset = 5h) [reset = 0h]

I2CCNT is shown in Figure 11-23 and described in Table 11-15.

Return to the Summary Table.

I2CCNT is a 16-bit register used to indicate how many data bytes to transfer when the I2C module is configured as a transmitter, or to receive when configured as a master receiver. In the repeat mode (RM = 1), I2CCNT is not used.

The value written to I2CCNT is copied to an internal data counter. The internal data counter is decremented by 1 for each byte transferred (I2CCNT remains unchanged). If a STOP condition is requested in the master mode (STP = 1 in I2CMDR), the I2C module terminates the transfer with a STOP condition when the countdown is complete (that is, when the last byte has been transferred).

Figure 11-23. I2CCNT Register

15	14	13	12	11	10	9	8
I2CCNT							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCNT							
R/W-0h							

Table 11-15. I2CCNT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	I2CCNT	R/W	0h	Data count value. I2CCNT 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. The start value loaded to the internal data counter is 65536. The start value loaded to internal data counter is 1-65535. Reset type: SYSRSn

11.6.2.7 I2CDRR Register (Offset = 6h) [reset = 0h]

I2CDRR is shown in Figure 11-24 and described in Table 11-16.

Return to the Summary Table.

I2CDRR is a 16-bit register used by the CPU to read received data. The I2C module can receive a data byte with 1 to 8 bits. The number of bits is selected with the bit count (BC) bits in 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 11-24. I2CDRR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
DATA							
R-0h							

Table 11-16. I2CDRR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	DATA	R	0h	Receive data Reset type: SYSRSn

11.6.2.8 I2CSAR Register (Offset = 7h) [reset = 3FFh]

I2CSAR is shown in Figure 11-25 and described in Table 11-17.

Return to the Summary Table.

The I2C slave address register (I2CSAR) is a 16-bit register for storing the next slave address that will be transmitted by the I2C module when it is a master. The SAR field of I2CSAR contains a 7-bit or 10-bit slave address. When the I2C module is not using the free data format (FDF = 0 in I2CMDR), it uses this address to initiate data transfers with a slave, or slaves. When the address is nonzero, the address is for a particular slave. When the address is 0, the address is a general call to all slaves. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6-0 of I2CSAR are used
write 0s to bits 9-7.

Figure 11-25. I2CSAR Register

15	14	13	12	11	10	9	8
RESERVED						SAR	
R-0h						R/W-3FFh	
7	6	5	4	3	2	1	0
SAR							
R/W-3FFh							

Table 11-17. I2CSAR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-10	RESERVED	R	0h	Reserved
9-0	SAR	R/W	3FFh	In 7-bit addressing mode (XA = 0 in I2CMDR): 00h-7Fh Bits 6-0 provide the 7-bit slave address that the I2C module transmits when it is in the master-transmitter mode. Write 0s to bits 9-7. In 10-bit addressing mode (XA = 1 in I2CMDR): 000h-3FFh Bits 9-0 provide the 10-bit slave address that the I2C module transmits when it is in the master transmitter mode. Reset type: SYSRSn

11.6.2.9 I2CDXR Register (Offset = 8h) [reset = 0h]

I2CDXR is shown in Figure 11-26 and described in Table 11-18.

Return to the Summary Table.

The CPU writes transmit data to I2CDXR. This 16-bit register accepts a data byte with 1 to 8 bits. Before writing to I2CDXR, specify how many bits are in a data byte by loading the appropriate value into the bit count (BC) bits of I2CMDR. When writing a data byte with fewer than 8 bits, make sure the value is right-aligned in I2CDXR.

After a data byte is written to I2CDXR, the I2C module copies the data byte to the transmit shift register (I2CXSR). The CPU cannot access I2CXSR directly. From I2CXSR, the I2C module shifts the data byte out on the SDA pin, one bit at a time.

When in the transmit FIFO mode, the I2CDXR register acts as the transmit FIFO buffer.

Figure 11-26. I2CDXR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
DATA							
R/W-0h							

Table 11-18. I2CDXR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	DATA	R/W	0h	Transmit data Reset type: SYSRSn

11.6.2.10 I2CMDR Register (Offset = 9h) [reset = 0h]

I2CMDR is shown in Figure 11-27 and described in Table 11-19.

Return to the Summary Table.

The I2C mode register (I2CMDR) is a 16-bit register that contains the control bits of the I2C module.

Figure 11-27. I2CMDR Register

15	14	13	12	11	10	9	8
NACKMOD	FREE	STT	RESERVED	STP	MST	TRX	XA
R/W-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
RM	DLB	IRS	STB	FDF	BC		
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h		

Table 11-19. I2CMDR Register Field Descriptions

Bit	Field	Type	Reset	Description
15	NACKMOD	R/W	0h	NACK mode bit. This bit is only applicable when the I2C module is acting as a receiver. Reset type: SYSRSn 0h (R/W) = In the slave-receiver mode: The I2C module sends an acknowledge (ACK) bit to the transmitter during each acknowledge cycle on the bus. The I2C module only sends a no-acknowledge (NACK) bit if you set the NACKMOD bit. 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 1h (R/W) = In either slave-receiver or master-receiver mode: The I2C module sends a NACK bit to the transmitter during the next acknowledge cycle on the bus. Once the NACK bit has been sent, NACKMOD is cleared. 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	R/W	0h	This bit controls the action taken by the I2C module when a debugger breakpoint is encountered. Reset type: SYSRSn 0h (R/W) = 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. 1h (R/W) = The I2C module runs free that is, it continues to operate when a breakpoint occurs.
13	STT	R/W	0h	START condition bit (only applicable when the I2C module is a master). The RM, STT, and STP bits determine when the I2C module starts and stops data transmissions (see Table 9-6). Note that the STT and STP bits can be used to terminate the repeat mode, and that this bit is not writable when IRS = 0. Reset type: SYSRSn 0h (R/W) = In the master mode, STT is automatically cleared after the START condition has been generated. 1h (R/W) = In the master mode, setting STT to 1 causes the I2C module to generate a START condition on the I2C-bus
12	RESERVED	R	0h	Reserved

Table 11-19. I2CMDR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
11	STP	R/W	0h	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. Note that the STT and STP bits can be used to terminate the repeat mode, and that this bit is not writable when IRS=0. When in non-repeat mode, at least one byte must be transferred before a stop condition can be generated. The I2C module delays clearing of this bit until after the I2CSTR[SCD] bit is set. To avoid disrupting the I2C state machine, the user must wait until this bit is clear before initiating a new message. Reset type: SYSRSn 0h (R/W) = STP is automatically cleared after the STOP condition has been generated 1h (R/W) = STP has been set by the device to generate a STOP condition when the internal data counter of the I2C module counts down to 0.
10	MST	R/W	0h	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 Reset type: SYSRSn 0h (R/W) = Slave mode. The I2C module is a slave and receives the serial clock from the master. 1h (R/W) = Master mode. The I2C module is a master and generates the serial clock on the SCL pin.
9	TRX	R/W	0h	Transmitter mode bit. When relevant, TRX selects whether the I2C module is in the transmitter mode or the receiver mode. Reset type: SYSRSn 0h (R/W) = Receiver mode. The I2C module is a receiver and receives data on the SDA pin. 1h (R/W) = Transmitter mode. The I2C module is a transmitter and transmits data on the SDA pin.
8	XA	R/W	0h	Expanded address enable bit. Reset type: SYSRSn 0h (R/W) = 7-bit addressing mode (normal address mode). The I2C module transmits 7-bit slave addresses (from bits 6-0 of I2CSAR), and its own slave address has 7 bits (bits 6-0 of I2COAR). 1h (R/W) = 10-bit addressing mode (expanded address mode). The I2C module transmits 10-bit slave addresses (from bits 9-0 of I2CSAR), and its own slave address has 10 bits (bits 9-0 of I2COAR).
7	RM	R/W	0h	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 Reset type: SYSRSn 0h (R/W) = Nonrepeat mode. The value in the data count register (I2CCNT) determines how many bytes are received/transmitted by the I2C module. 1h (R/W) = Repeat mode. A data byte is transmitted each time the I2CDXR register is written to (or until the transmit FIFO is empty when in FIFO mode) until the STP bit is manually set. The value of I2CCNT is ignored. The ARDY bit/interrupt can be used to determine when the I2CDXR (or FIFO) is ready for more data, or when the data has all been sent and the CPU is allowed to write to the STP bit.

Table 11-19. I2CMDR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
6	DLB	R/W	0h	Digital loopback mode bit. Reset type: SYSRSn 0h (R/W) = Digital loopback mode is disabled. 1h (R/W) = 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) x 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	R/W	0h	I2C module reset bit. Reset type: SYSRSn 0h (R/W) = The I2C module is in reset/disabled. When this bit is cleared to 0, all status bits (in I2CSTR) are set to their default values. 1h (R/W) = The I2C module is enabled. This has the effect of releasing the I2C bus if the I2C peripheral is holding it.
4	STB	R/W	0h	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. Reset type: SYSRSn 0h (R/W) = The I2C module is not in the START byte mode. 1h (R/W) = The I2C module is in the START byte mode. When you set the START condition bit (STT), the I2C module begins the transfer with more than just a START condition. Specifically, it generates: 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	R/W	0h	Free data format mode bit. Reset type: SYSRSn 0h (R/W) = Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit. 1h (R/W) = Free data format mode is enabled. Transfers have the free data (no address) format described in Section 9.2.5. The free data format is not supported in the digital loopback mode (DLB=1).

Table 11-19. I2CMDR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
2-0	BC	R/W	0h	Bit count bits. BC defines the number of bits (1 to 8) in the next data byte that is to be received or transmitted by the I2C module. The number of bits selected with BC must match the data size of the other device. Notice that when BC = 000b, a data byte has 8 bits. BC does not affect address bytes, which always have 8 bits. Note: If the bit count is less than 8, receive data is right-justified in I2CDRR(7-0), and the other bits of I2CDRR(7-0) are undefined. Also, transmit data written to I2CDXR must be right-justified Reset type: SYSRSn 0h (R/W) = 8 bits per data byte 1h (R/W) = 1 bit per data byte 2h (R/W) = 2 bits per data byte 3h (R/W) = 3 bits per data byte 4h (R/W) = 4 bits per data byte 5h (R/W) = 5 bits per data byte 6h (R/W) = 6 bits per data byte 7h (R/W) = 7 bits per data byte

11.6.2.11 I2CISRC Register (Offset = Ah) [reset = 0h]

I2CISRC is shown in Figure 11-28 and described in Table 11-20.

Return to the Summary Table.

The I2C interrupt source register (I2CISRC) is a 16-bit register used by the CPU to determine which event generated the I2C interrupt.

Figure 11-28. I2CISRC Register

15	14	13	12	11	10	9	8
RESERVED				WRITE_ZEROS			
R-0h				R/W-0h			
7	6	5	4	3	2	1	0
RESERVED					INTCODE		
R-0h					R-0h		

Table 11-20. I2CISRC Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reserved
11-8	WRITE_ZEROS	R/W	0h	TI internal testing bits These reserved bit locations should always be written as zeros. Reset type: SYSRSn
7-3	RESERVED	R	0h	Reserved
2-0	INTCODE	R	0h	Interrupt code bits. The binary code in INTCODE indicates the event that generated an I2C interrupt. A CPU read will clear this field. If another lower priority interrupt is pending and enabled, the value corresponding to that interrupt will then be loaded. Otherwise, the value will stay cleared. In the case of an arbitration lost, a no-acknowledgment condition detected, or a stop condition detected, a CPU read will also clear the associated interrupt flag bit in the I2CSTR register. Emulator reads will not affect the state of this field or of the status bits in the I2CSTR register. Reset type: SYSRSn 0h (R/W) = None 1h (R/W) = Arbitration lost 2h (R/W) = No-acknowledgment condition detected 3h (R/W) = Registers ready to be accessed 4h (R/W) = Receive data ready 5h (R/W) = Transmit data ready 6h (R/W) = Stop condition detected 7h (R/W) = Addressed as slave

11.6.2.12 I2CEMDR Register (Offset = Bh) [reset = 1h]

I2CEMDR is shown in Figure 11-29 and described in Table 11-21.

Return to the Summary Table.

I2C Extended Mode

Figure 11-29. I2CEMDR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							BC
R-0h							R/W-1h

Table 11-21. I2CEMDR Register Field Descriptions

Bit	Field	Type	Reset	Description
15-1	RESERVED	R	0h	Reserved
0	BC	R/W	1h	Backwards compatibility mode. This bit affects the timing of the transmit status bits (XRDY and XSMT) in the I2CSTR register when in slave transmitter mode. Check Backwards Compatibility Mode Bit, Slave Transmitter diagram for more details. Reset type: SYSRSn 0h (R/W) = See the "Backwards Compatibility Mode Bit, Slave Transmitter" Figure for details. 1h (R/W) = See the "Backwards Compatibility Mode Bit, Slave Transmitter" Figure for details.

11.6.2.13 I2CPSC Register (Offset = Ch) [reset = 0h]

I2CPSC is shown in Figure 11-30 and described in Table 11-22.

Return to the Summary Table.

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.

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 11-30. I2CPSC Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
IPSC							
R/W-0h							

Table 11-22. I2CPSC Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	IPSC	R/W	0h	I2C prescaler divide-down value. IPSC determines how much the CPU clock is divided to create the module clock of the I2C module: module clock frequency = I2C input clock frequency/(IPSC + 1) Note: IPSC must be initialized while the I2C module is in reset (IRS = 0 in I2CMDR). Reset type: SYSRSn

11.6.2.14 I2CFFTX Register (Offset = 20h) [reset = 0h]

I2CFFTX is shown in Figure 11-31 and described in Table 11-23.

Return to the Summary Table.

The I2C transmit FIFO register (I2CFFTX) is a 16-bit register that contains the I2C FIFO mode enable bit as well as the control and status bits for the transmit FIFO mode of operation on the I2C peripheral.

Figure 11-31. I2CFFTX Register

15	14	13	12	11	10	9	8
RESERVED	I2CFFEN	TXFFRST	TXFFST				
R-0h	R/W-0h	R/W-0h	R-0h				
7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL				
R-0h	R-0/W1S-0h	R/W-0h	R/W-0h				

Table 11-23. I2CFFTX Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reserved
14	I2CFFEN	R/W	0h	I2C FIFO mode enable bit. This bit must be enabled for either the transmit or the receive FIFO to operate correctly. Reset type: SYSRSn 0h (R/W) = Disable the I2C FIFO mode. 1h (R/W) = Enable the I2C FIFO mode.
13	TXFFRST	R/W	0h	Transmit FIFO Reset Reset type: SYSRSn 0h (R/W) = Reset the transmit FIFO pointer to 0000 and hold the transmit FIFO in the reset state. 1h (R/W) = Enable the transmit FIFO operation.
12-8	TXFFST	R	0h	Contains the status of the transmit FIFO: xxxxx Transmit FIFO contains xxxxx bytes. 00000 Transmit FIFO is empty. Note: Since these bits are reset to zero, the transmit FIFO interrupt flag will be set when the transmit FIFO operation is enabled and the I2C is taken out of reset. This will generate a transmit FIFO interrupt if enabled. To avoid any detrimental effects from this, write a one to the TXFFINTCLR once the transmit FIFO operation is enabled and the I2C is taken out of reset. Reset type: SYSRSn
7	TXFFINT	R	0h	Transmit FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the TXFFINTCLR bit. If the TXFFIENA bit is set, this bit will generate an interrupt when it is set. Reset type: SYSRSn 0h (R/W) = Transmit FIFO interrupt condition has not occurred. 1h (R/W) = Transmit FIFO interrupt condition has occurred.
6	TXFFINTCLR	R-0/W1S	0h	Transmit FIFO Interrupt Flag Clear Reset type: SYSRSn 0h (R/W) = Writes of zeros have no effect. Reads return a 0. 1h (R/W) = Writing a 1 to this bit clears the TXFFINT flag.
5	TXFFIENA	R/W	0h	Transmit FIFO Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disabled. TXFFINT flag does not generate an interrupt when set. 1h (R/W) = Enabled. TXFFINT flag does generate an interrupt when set.

Table 11-23. I2CFFTX Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4-0	TXFFIL	R/W	0h	Transmit FIFO interrupt level. These bits set the status level that will set the transmit interrupt flag. When the TXFFST4-0 bits reach a value equal to or less than these bits, the TXFFINT flag will be set. This will generate an interrupt if the TXFFIENA bit is set. Because the I2C on this device has a 16-level transmit FIFO, these bits cannot be configured for an interrupt of more than 16 FIFO levels. Reset type: SYSRSn

11.6.2.15 I2CFFRX Register (Offset = 21h) [reset = 0h]

I2CFFRX is shown in Figure 11-32 and described in Table 11-24.

Return to the Summary Table.

The I2C receive FIFO register (I2CFFRX) is a 16-bit register that contains the control and status bits for the receive FIFO mode of operation on the I2C peripheral.

Figure 11-32. I2CFFRX Register

15	14	13	12	11	10	9	8
RESERVED		RXFFRST	RXFFST				
R-0h		R/W-0h	R-0h				
7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL				
R-0h	R-0/W1S-0h	R/W-0h	R/W-0h				

Table 11-24. I2CFFRX Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	RESERVED	R	0h	Reserved
13	RXFFRST	R/W	0h	I2C receive FIFO reset bit Reset type: SYSRSn 0h (R/W) = Reset the receive FIFO pointer to 0000 and hold the receive FIFO in the reset state. 1h (R/W) = Enable the receive FIFO operation.
12-8	RXFFST	R	0h	Contains the status of the receive FIFO: xxxxx Receive FIFO contains xxxxx bytes 00000 Receive FIFO is empty. Reset type: SYSRSn
7	RXFFINT	R	0h	Receive FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the RXFFINTCLR bit. If the RXFFIENA bit is set, this bit will generate an interrupt when it is set Reset type: SYSRSn 0h (R/W) = Receive FIFO interrupt condition has not occurred. 1h (R/W) = Receive FIFO interrupt condition has occurred.
6	RXFFINTCLR	R-0/W1S	0h	Receive FIFO interrupt flag clear bit. Reset type: SYSRSn 0h (R/W) = Writes of zeros have no effect. Reads return a zero. 1h (R/W) = Writing a 1 to this bit clears the RXFFINT flag.
5	RXFFIENA	R/W	0h	Receive FIFO interrupt enable bit. Reset type: SYSRSn 0h (R/W) = Disabled. RXFFINT flag does not generate an interrupt when set. 1h (R/W) = Enabled. RXFFINT flag does generate an interrupt when set.

Table 11-24. I2CFFRX Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4-0	RXFFIL	R/W	0h	Receive FIFO interrupt level. These bits set the status level that will set the receive interrupt flag. When the RXFFST4-0 bits reach a value equal to or greater than these bits, the RXFFINT flag is set. This will generate an interrupt if the RXFFIENA bit is set. Note: Since these bits are reset to zero, the receive FIFO interrupt flag will be set if the receive FIFO operation is enabled and the I2C is taken out of reset. This will generate a receive FIFO interrupt if enabled. To avoid this, modify these bits on the same instruction as or prior to setting the RXFFRST bit. Because the I2C on this device has a 16-level receive FIFO, these bits cannot be configured for an interrupt of more than 16 FIFO levels. Reset type: SYSRSn

Multichannel Buffered Serial Port (McBSP)

This document describes the multichannel buffered serial port (McBSP) of this device.

Topic **Page**

12.1 Overview

This device provides up to two high-speed multichannel buffered serial ports (McBSPs) that allow direct interface to codecs and other devices in a system. The McBSP consists of a data-flow path and a control path connected to external devices by six pins as shown in Figure 12-1.

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 the DX pin 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 receive buffer registers (RBRs) is then copied to the data receive registers (DRRs), which can be read by the CPU or the DMA controller. This allows simultaneous movement of internal and external data communications.

If the serial word length is 8 bits, 12 bits, or 16 bits, the DRR2, RBR2, RSR2, DXR2, and XSR2 registers are not used (written, read, or shifted) 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.

12.1.1 Features of the McBSPs

The McBSPs feature:

- 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

12.1.2 McBSP Pins/Signals

Table 12-1 describes the McBSP interface pins and some internal signals.

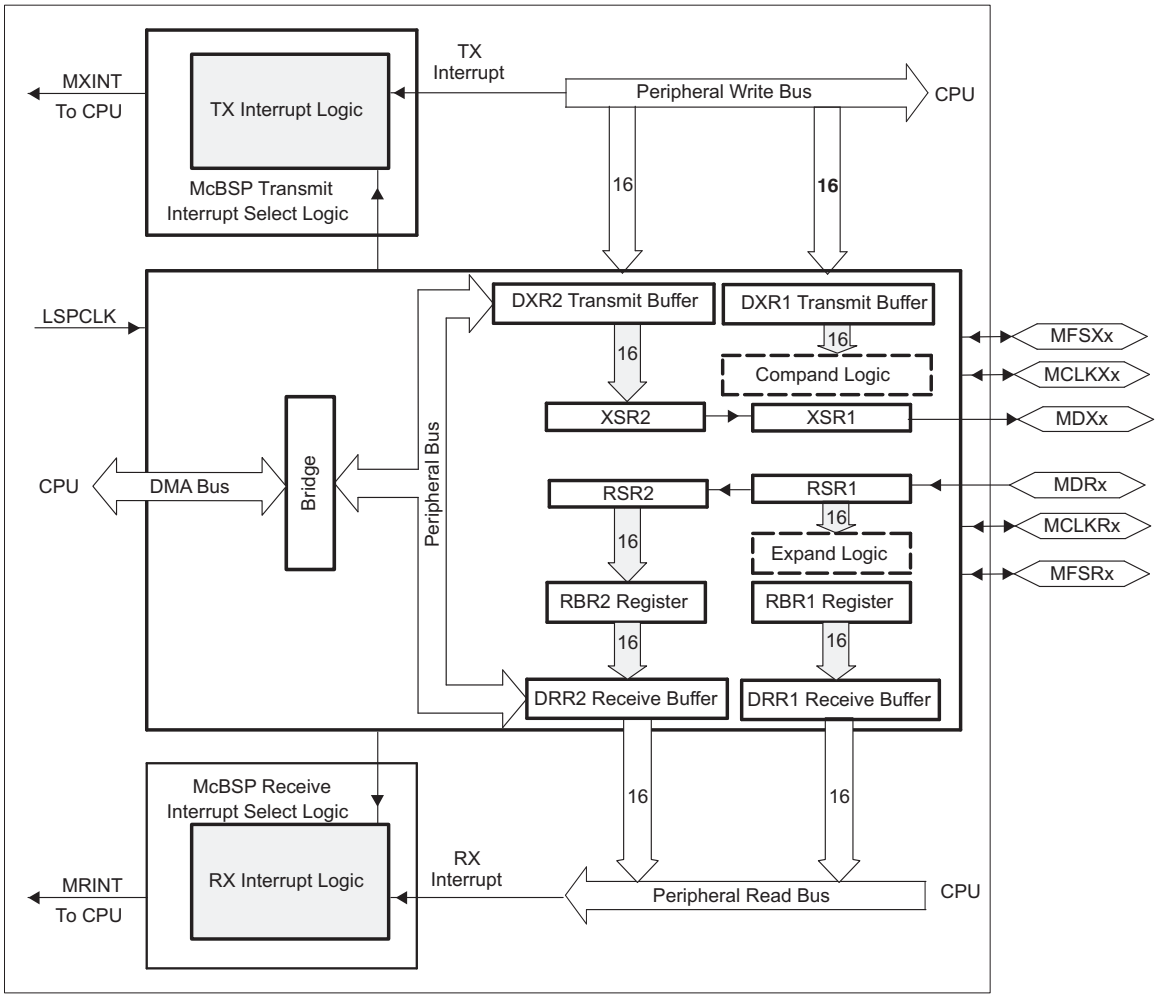
Table 12-1. McBSP Interface Pins/Signals

McBSP-A Pin	McBSP-B Pin	Type	Description
MCLKRA	MCLKRB	I/O	Supplies or reflects the receive clock; supplys the input clock of the sample rate generator
MCLKXA	MCLKXB	I/O	Supplys or reflects the transmit clock; supplys the input clock of the sample rate generator
MDRA	MDRB	I	Serial data receive pin
MDXA	MDXB	O	Serial data transmit pin
MFSRA	MFSRB	I/O	Supplys or reflects the receive frame-sync signal; controlsg sample rate generator synchronization when GSYNC = 1 (see Section 12.4.3)
MFSXA	MFSXB	I/O	Supplys or reflects 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

12.1.2.1 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 12-1. The figure and the text in this section use generic pin names.

Figure 12-1. Conceptual Block Diagram of the McBSP



A Not available in all devices. See the device-specific data sheet

12.2 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins. To avoid glitches on the pins, the GPyGMUX bits must be configured first (while keeping the corresponding GPyMUX bits at the default of zero), followed by writing the GPyMUX register to the desired value.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPyPUD register.

See the *GPIO* chapter for more details on GPIO mux and settings.

12.3 McBSP Operation

This section addresses the following topics:

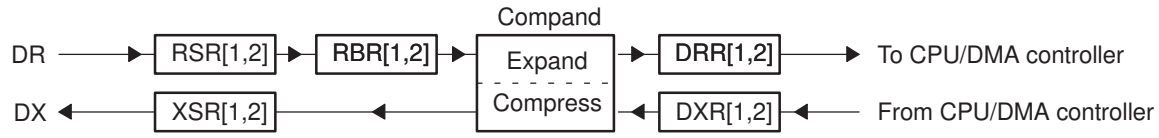
- Data transfer process
- Companding (compressing and expanding) data
- Clocking and framing data

- Frame phases
- McBSP reception
- McBSP transmission
- Interrupts and DMA events generated by McBSPs

12.3.1 Data Transfer Process of McBSPs

Figure 12-2 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 12-2. McBSP Data Transfer Paths



12.3.1.1 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 the receive buffer register 1 (RBR1), that is, if RBR1 is not full with previous data. RBR1 is then copied to the 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 12.3.5.

Transmit data is written by the CPU or the DMA controller to the data transmit register 1 (DXR1). If there is no previous data in the 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 12.3.6.

12.3.1.2 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 12.3.5.

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 12.3.6.

12.3.2 Companding (Compressing and Expanding) Data

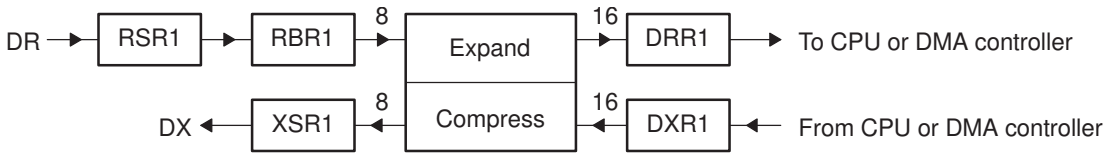
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM 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 12-3 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 12-3. Companding Processes

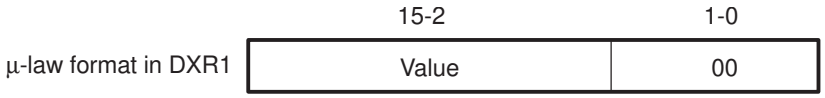


12.3.2.1 Companding Formats

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.

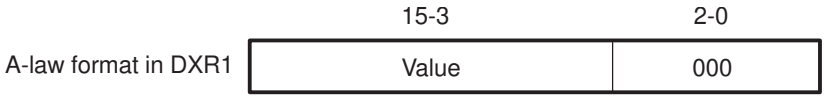
For transmission using μ-law compression, the 14 data bits must be left-justified in DXR1 with the remaining two low-order bits filled with 0s as shown in Figure 12-4.

Figure 12-4. μ-Law Transmit Data Companding Format



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 12-5.

Figure 12-5. A-Law Transmit Data Companding Format



12.3.2.2 Capability to Compand Internal Data

If the McBSP is unused (the serial port transmit and receive sections are reset), the companding hardware can compand internal data. This can be used to:

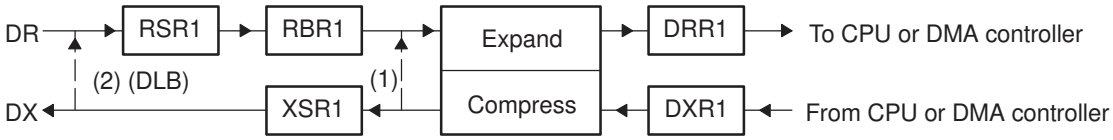
- Convert linear to the appropriate μ-law or A-law format
- Convert μ-law or A-law to the linear format
- Observe the quantization effects in companding by transmitting linear data and compressing and re-expanding this data. This is useful only if both XCOMPAND and RCOMPAND enable the same companding format.

Figure 12-6 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 12-6. Two Methods by Which the McBSP Can Compand Internal Data



12.3.2.3 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.

12.3.3 Clocking and Framing Data

This section explains basic concepts and terminology important for understanding how McBSP data transfers are timed and delimited.

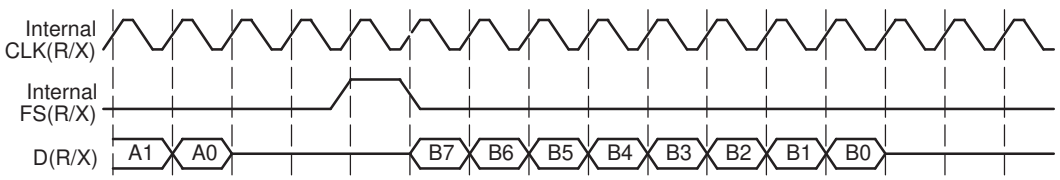
12.3.3.1 Clocking

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.

Figure 12-7 shows how the clock signal controls the timing of each bit transfer on the pin.

Figure 12-7. 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) (that is, be certain that CLKX or CLKR ≤ LSPCLK/2).

12.3.3.2 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 exmaple in Figure 12-7, an 8-bit word size was defined (see bits 7 through 0 of word B being transferred).

12.3.3.3 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 the DR data pin. 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 Figure 12-7, 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.

12.3.3.4 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).

12.3.3.4.1 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.

12.3.3.5 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 12.5.3)
- *Unexpected Transmit Frame-Synchronization Pulse* (see Section 12.5.6)

You can also use the frame-synchronization ignore function for data packing (for more details, see Section 12.11.2).

12.3.3.6 Frame Frequency

The frame frequency is determined by the period between frame-synchronization pulses and is defined as shown by Example 1.

Example 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.

12.3.3.7 Maximum Frame Frequency

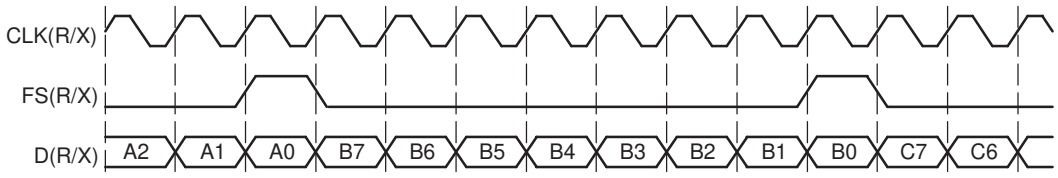
The minimum number of clock cycles between frame synchronization pulses is equal to the number of bits transferred per frame. The maximum frame frequency is defined as shown by Example 2.

Example 2: McBSP Maximum Frame Frequency

$$\text{Maximum Frame Frequency} = \frac{\text{Clock Frequency}}{\text{Number of Bits Per Frame}}$$

Figure 12-8 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 12-8. 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 12.9.13.

12.3.4 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.

12.3.4.1 Number of Phases, Words, and Bits Per Frame

Table 12-2 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 12-2. Register Bits That Determine the Number of Phases, Words, and Bits

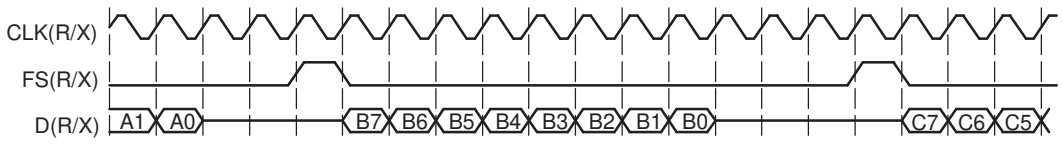
Operation	Number of Phases	Words per Frame Set With	Bits per Word Set With
Reception	1 (RPHASE = 0)	RFRLEN1	RWDLEN1
Reception	2 (RPHASE = 1)	RFRLEN1 and RFRLEN2	RWDLEN1 for phase 1
			RWDLEN2 for phase 2
Transmission	1 (XPHASE = 0)	XFRLEN1	XWDLEN1
Transmission	2 (XPHASE = 1)	XFRLEN1 and XFRLEN2	XWDLEN1 for phase 1
			XWDLEN2 for phase 2

12.3.4.2 Single-Phase Frame Example

Figure 12-9 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

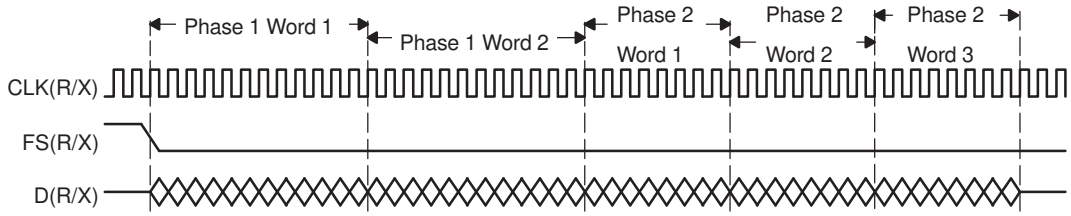
Figure 12-9. Single-Phase Frame for a McBSP Data Transfer



12.3.4.3 Dual-Phase Frame Example

Figure 12-10 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 12-10. Dual-Phase Frame for a McBSP Data Transfer

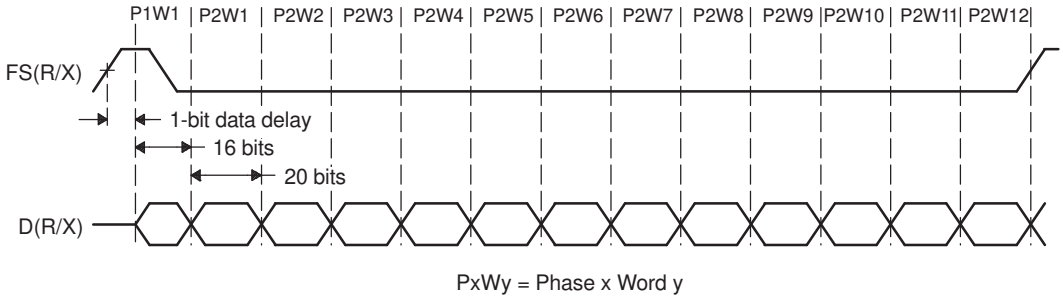


A XRDY gets asserted once per phase. So, if there are 2 phases, XRDY gets asserted twice (once per phase).

12.3.4.4 Implementing the AC97 Standard With a Dual-Phase Frame

Figure 12-11 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.

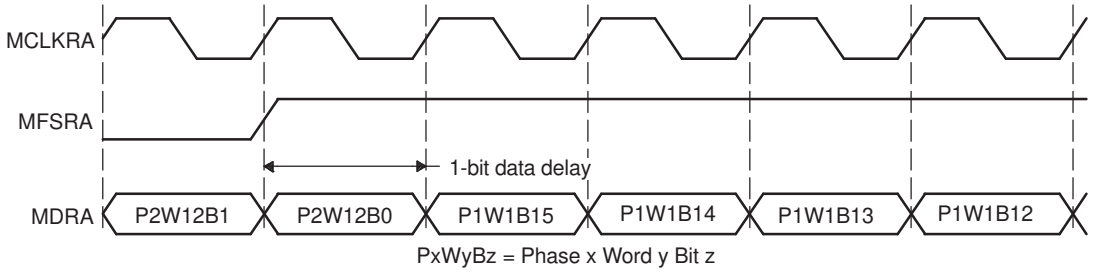
Figure 12-11. Implementing the AC97 Standard With a Dual-Phase Frame



- (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 12-12 shows the timing of an AC97-standard data transfer near frame synchronization. In this figure, individual bits are shown on D(R/X). Specifically, it 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 12-12. Timing of an AC97-Standard Data Transfer Near Frame Synchronization

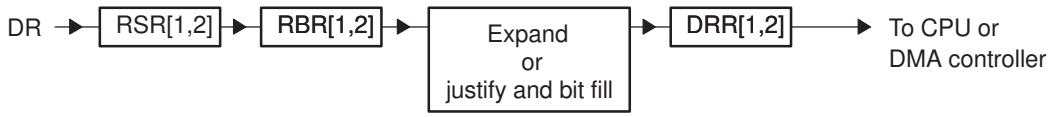


12.3.5 McBSP Reception

This section explains the fundamental process of reception in the McBSP. For details about how to program the McBSP receiver, see Section 12.8.

Figure 12-13 and Figure 12-14 show how reception occurs in the McBSP. Figure 12-13 shows the physical path for the data. Figure 12-14 is a timing diagram showing signal activity for one possible reception scenario. A description of the process follows the figures.

Figure 12-13. McBSP Reception Physical Data Path

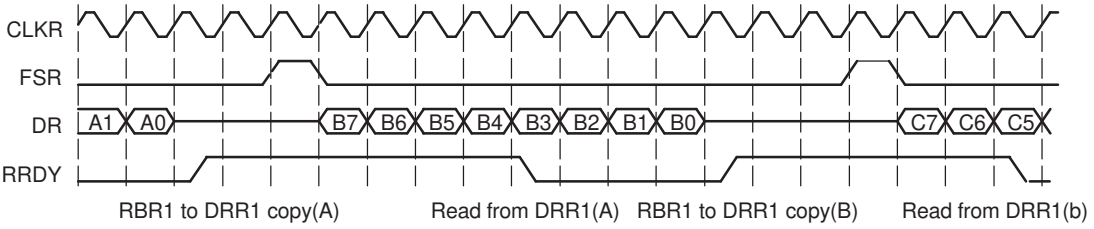


A RSR[1,2]: Receive shift registers 1 and 2

B RBR[1,2]: Receive buffer registers 1 and 2

C DRR[1,2]: Data receive registers 1 and 2

Figure 12-14. 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, 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 12.8.8, *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:

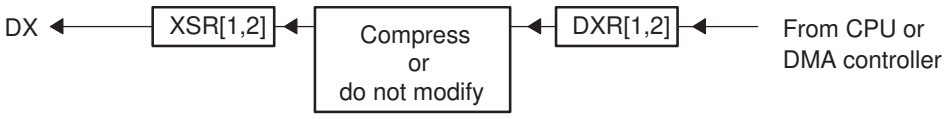
- *Overrun in the Receiver* (see Section 12.5.2)
- *Unexpected Receive Frame-Synchronization Pulse* (see Section 12.5.3)

12.3.6 McBSP Transmission

This section explains the fundamental process of transmission in the McBSP. For details about how to program the McBSP transmitter, see Section 12.9.

Figure 12-15 and Figure 12-16 show how transmission occurs in the McBSP. Figure 12-15 shows the physical path for the data. Figure 12-16 is a timing diagram showing signal activity for one possible transmission scenario. A description of the process follows the figures.

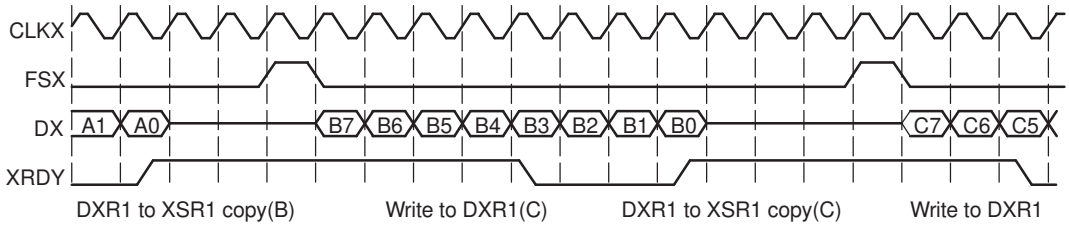
Figure 12-15. McBSP Transmission Physical Data Path



A XSR[1,2]: Transmit shift registers 1 and 2

B DXR[1,2]: Data transmit registers 1 and 2

Figure 12-16. 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 12.9.9.

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 12-16), 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 12.5.4)
- *Underflow in the Transmitter* (Section 12.5.5)
- *Unexpected Transmit Frame-Synchronization Pulse* (Section 12.5.6)

12.3.7 Interrupts and DMA Events Generated by a McBSP

The McBSP sends notification of important events to the CPU and DMA via the internal signals shown in Table 12-3.

Table 12-3. Interrupts and DMA Events Generated by a McBSP

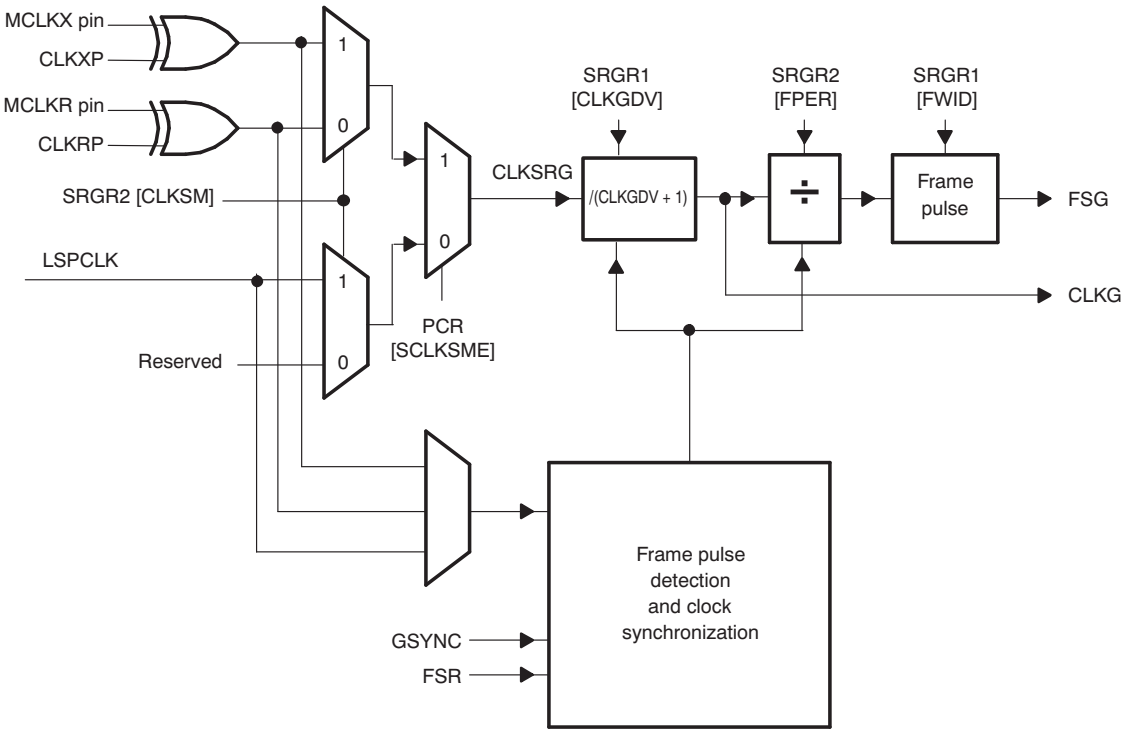
Internal Signal	Description
RINT	Receive interrupt
	The McBSP sends a receive interrupt request to the CPU based upon a selected condition in the receiver of the McBSP (a condition selected by the RINTM bits of SPCR1).
XINT	Transmit interrupt
	The McBSP sends a transmit interrupt request to the CPU based 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.

12.4 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 12-17 is a conceptual block diagram of the sample rate generator.

12.4.1 Block Diagram

Figure 12-17. 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. You must 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 12.4.4.

12.4.1.1 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 12-4. The digital loopback mode (described in Section 12.8.4) is selected with the DLB bit of SPCR1. The clock stop mode (described in Section 12.7.2) 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 12-4. 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.

12.4.1.2 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 12-5). When CLKSM = 1, the minimum divide down value in CLKGDV bits is 1. CLKGDV is described in Section 12.4.1.4.

Table 12-5. 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

12.4.1.3 Choosing a Polarity for the Input Clock

As shown in Figure 12-18, 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 12-6.

Figure 12-18. Possible Inputs to the Sample Rate Generator and the Polarity Bits

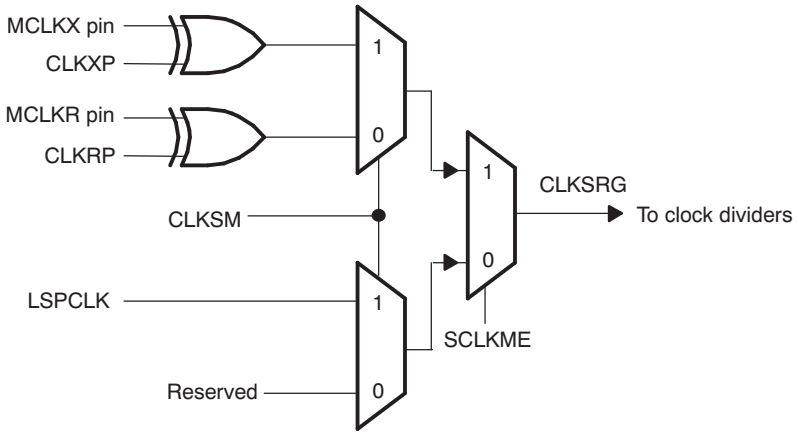


Table 12-6. Polarity Options for the Input to the Sample Rate Generator

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.

12.4.1.4 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 12-1) 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 the equation below.

Equation 1: CLKG Frequency

$$\text{CLKG frequency} = \frac{\text{Input clock frequency}}{(\text{CLKGDV} + 1)}$$

12.4.1.4.1 CLKG Frequency

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.

12.4.1.5 Keeping CLKG Synchronized to External MCLKR

When the MCLKR pin is used to drive the sample rate generator (see Section 12.4.1.2), 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 12.4.3.

12.4.2 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 = 1).

12.4.2.1 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.

12.4.2.2 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.

12.4.2.3 Keeping FSG Synchronized to an External Clock

When an external signal is selected to drive the sample rate generator (see Section 12.4.1.2) , 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 12.4.3 for more details about synchronization.

12.4.3 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.

12.4.3.1 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).

12.4.3.2 Synchronization Examples

Figure 12-19 and Figure 12-20 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. Figure 12-20 has a slower CLKG frequency (it has a larger divide-down value in the CLKGDV bits of SRGR1).

Figure 12-19. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 1

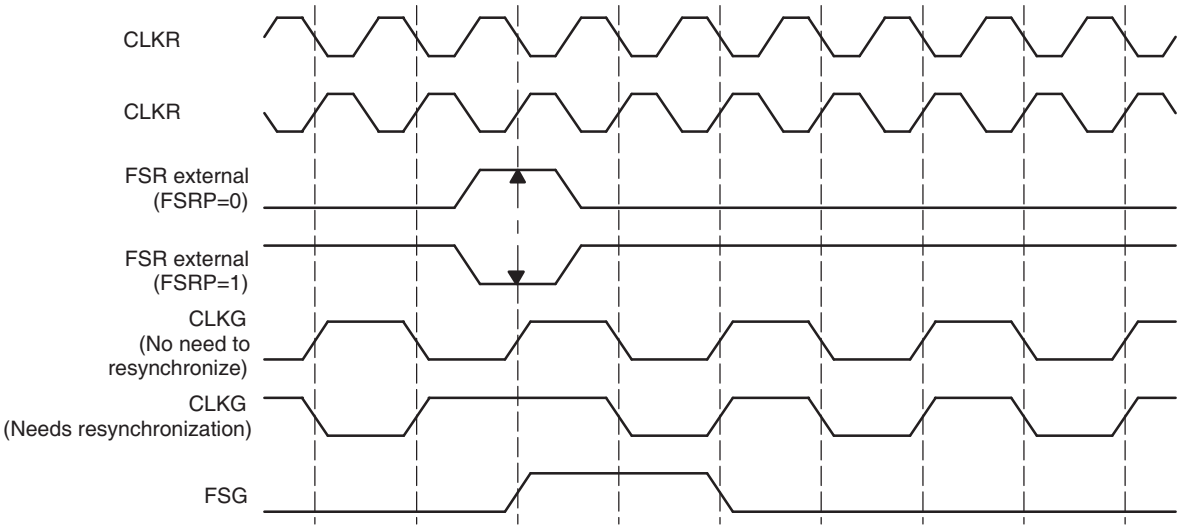
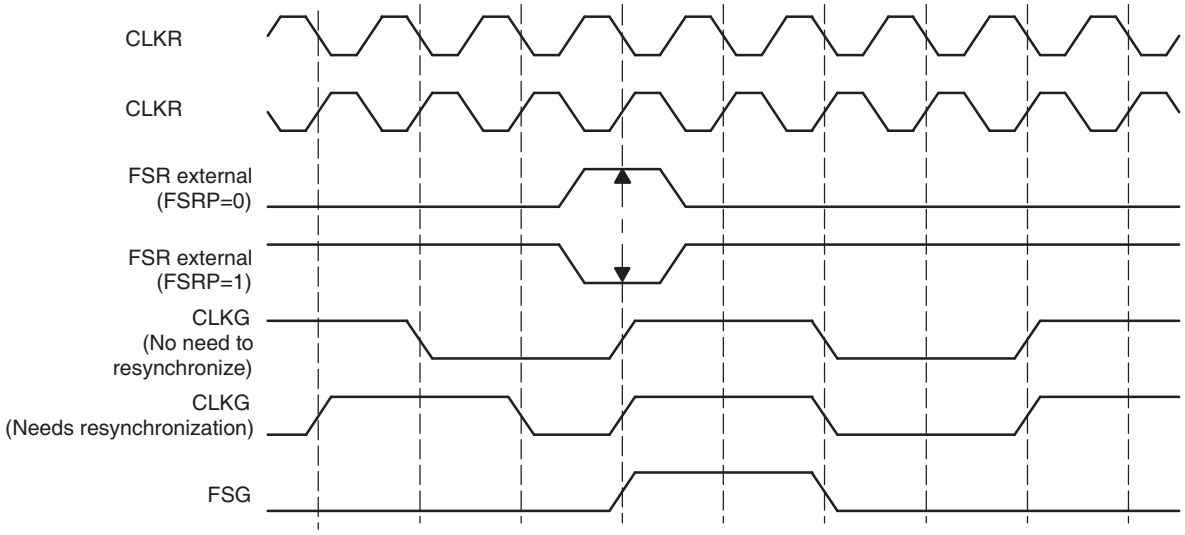


Figure 12-20. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 3



12.4.4 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 seting 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 the CLKG Frequency equation below.

Table 12-7. 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 2: 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 12-7.

12.5 McBSP Exception/Error Conditions

This chapter describes exception/error conditions and how to handle them.

12.5.1 Types of Errors

There are five serial port events that can constitute a system error:

- Receiver overrun (RFULL = 1)

  This error 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 12.5.2.
- Unexpected receive frame-synchronization pulse (RSYNCERR = 1)

  This error 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 12.5.3.
- Transmitter data overwrite

  This error 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 12.5.4.
- 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 12.5.5.
- Unexpected transmit frame-synchronization pulse (XSYNCERR = 1)

  This error 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 12.5.6.

12.5.2 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 Section 12.3.5, 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:

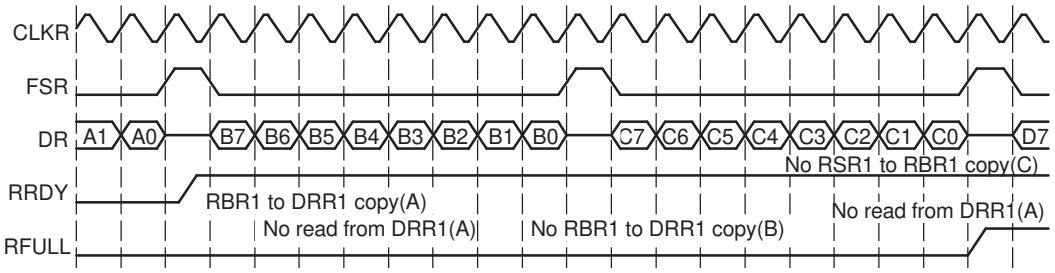
- The CPU or DMA controller reads DRR1.
- The receiver is reset individually (RRST = 0) or as part of a device reset.

Another frame-synchronization pulse is required to restart the receiver.

12.5.2.1 Example of Overrun Condition

Figure 12-21 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.

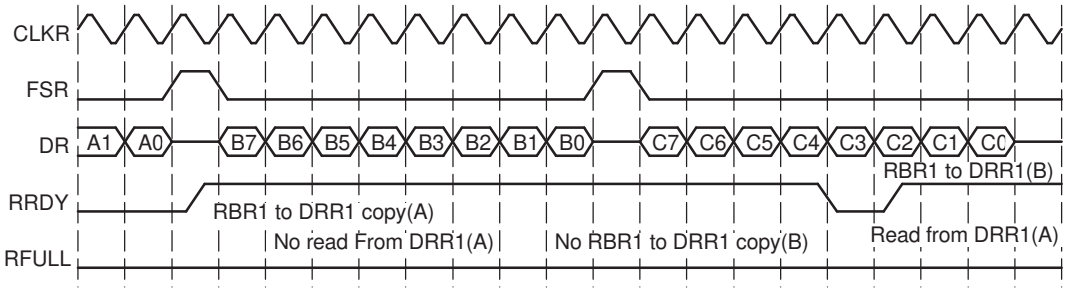
Figure 12-21. Overrun in the McBSP Receiver



12.5.2.2 Example of Preventing Overrun Condition

Figure 12-22 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 12-22. Overrun Prevented in the McBSP Receiver



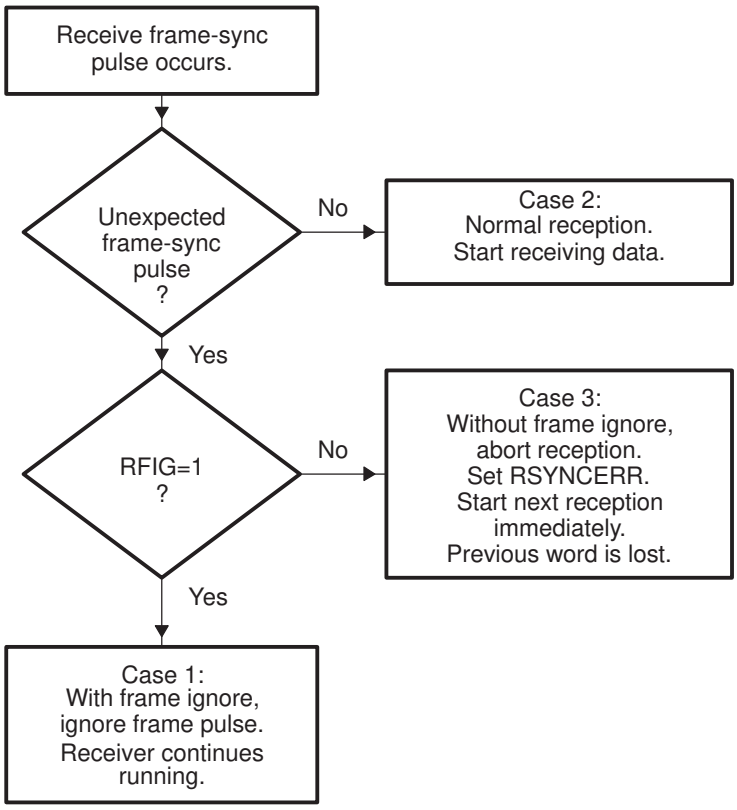
12.5.3 Unexpected Receive Frame-Synchronization Pulse

Section 12.5.3.1 shows how the McBSP responds to any receive frame-synchronization pulses, including an unexpected pulse. Section 12.5.3.2 and Section 12.5.3.3 show an example of a frame-synchronization error and an example of how to prevent such an error, respectively.

12.5.3.1 Possible Responses to Receive Frame-Synchronization Pulses

Figure 12-23 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 shows where an error occurs.

Figure 12-23. Possible Responses to Receive Frame-Synchronization Pulses



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.

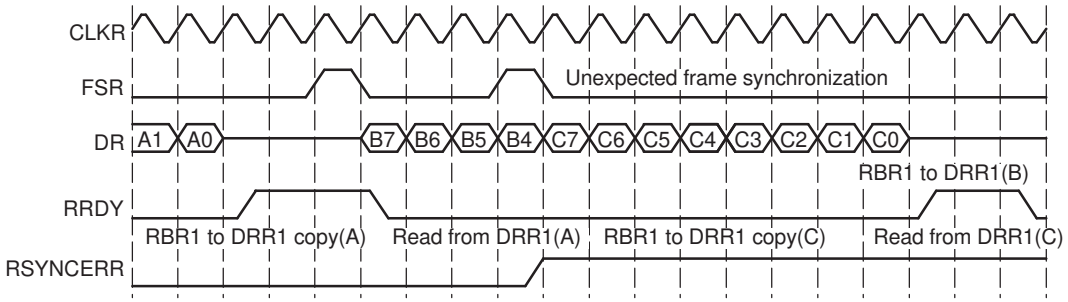
  If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully received, this pulse is treated as an unexpected frame-synchronization pulse, and the receiver sets the receive frame-synchronization error bit (RSYNCERR) in SPCR1. RSYNCERR can be cleared only by a 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.

12.5.3.2 Example of Unexpected Receive Frame-Synchronization Pulse

Figure 12-30 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.

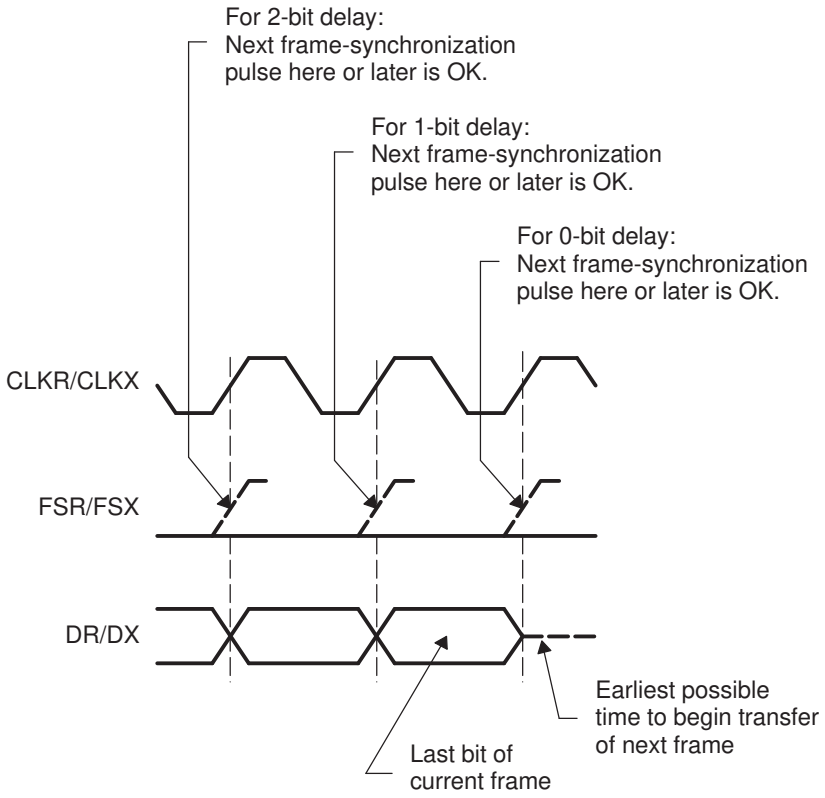
Figure 12-24. An Unexpected Frame-Synchronization Pulse During a McBSP Reception



12.5.3.3 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 12-25 shows when a new frame-synchronization pulse on FSR can safely occur relative to the last bit of the current frame.

Figure 12-25. Proper Positioning of Frame-Synchronization Pulses



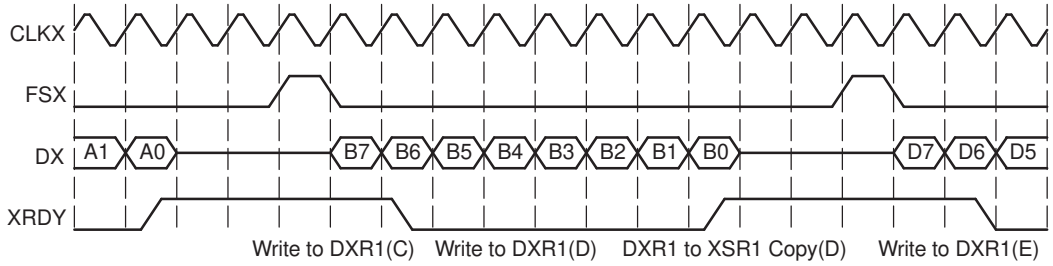
12.5.4 Overwrite in the Transmitter

As described in Section 12.3.6), 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.

12.5.4.1 Example of Overwrite Condition

Figure 12-26 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 12-26. Data in the McBSP Transmitter Overwritten and Thus Not Transmitted



12.5.4.2 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.

12.5.5 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.

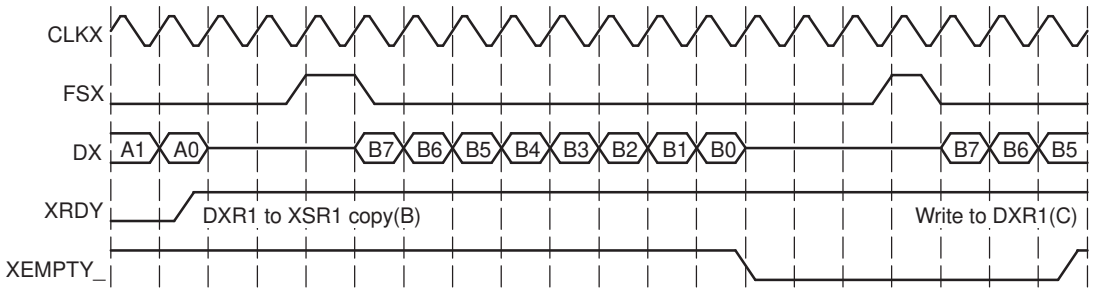
XEMPTY is deactivated (XEMPTY = 1) when a new word in DXR1 is transferred to XSR1. If FSXM = 1 in PCR and FSGM = 0 in SRGR2, the transmitter generates a single internal FSX pulse in response to a DXR-to-XSR copy. Otherwise, the transmitter waits for the next frame-synchronization pulse before sending 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.

12.5.5.1 Example of the Underflow Condition

Figure 12-27 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.

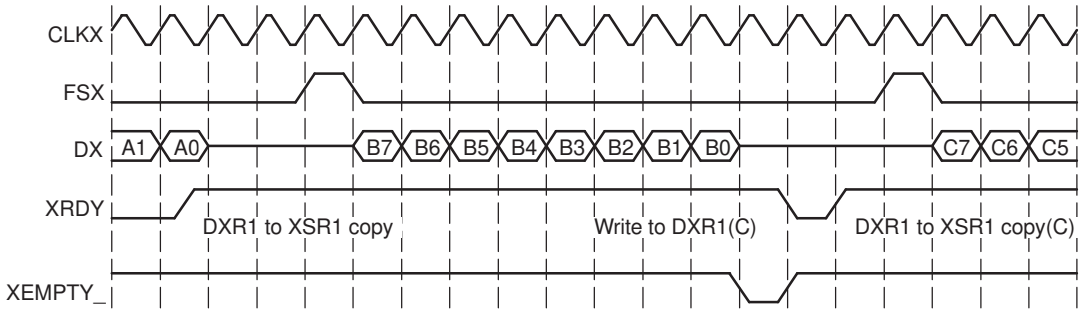
Figure 12-27. Underflow During McBSP Transmission



12.5.5.2 Example of Preventing Underflow Condition

Figure 12-28 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 12-28. Underflow Prevented in the McBSP Transmitter



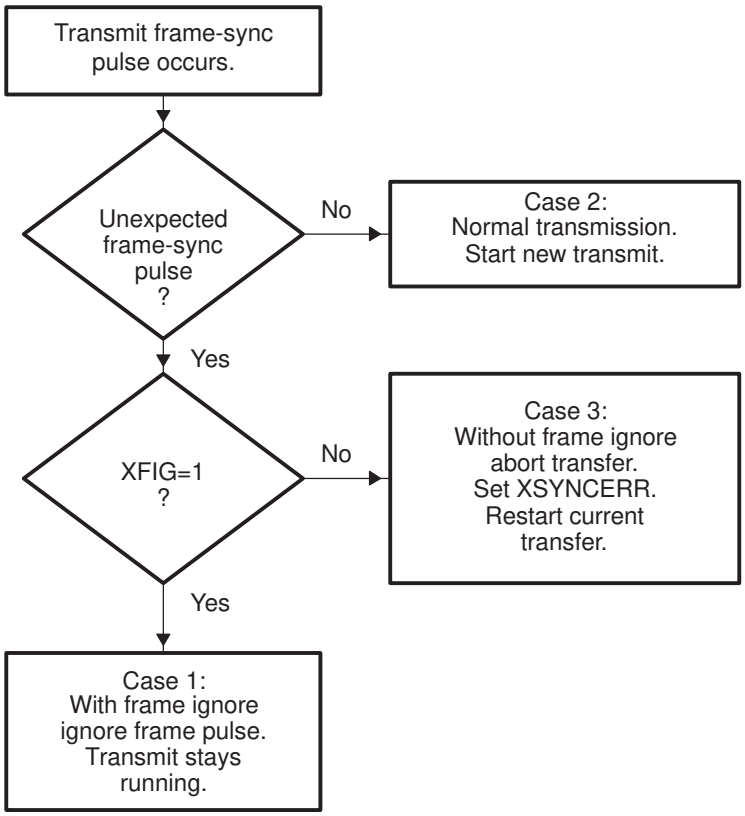
12.5.6 Unexpected Transmit Frame-Synchronization Pulse

Section 12.5.6.1 shows how the McBSP responds to any transmit frame-synchronization pulses, including an unexpected pulse. Section 12.5.6.2 and Section 12.5.6.3 show examples of a frame-synchronization error and an example of how to prevent such an error, respectively.

12.5.6.1 Possible Responses to Transmit Frame-Synchronization Pulses

Figure 12-29 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 shows where an error occurs.

Figure 12-29. Possible Responses to Transmit Frame-Synchronization Pulses



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.

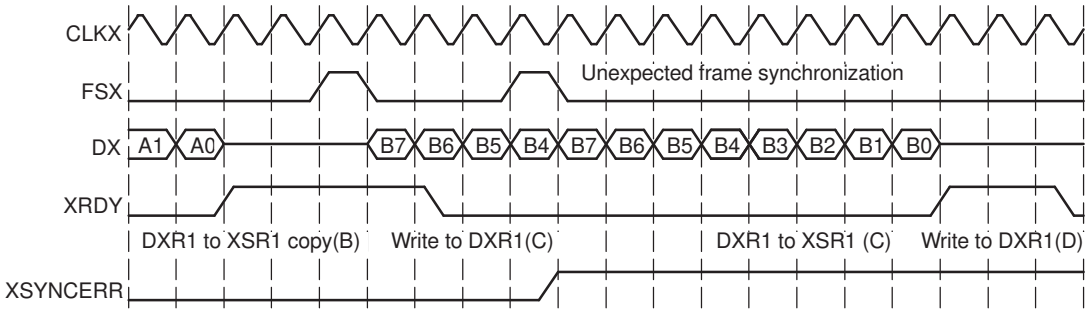
  If a frame-synchronization pulse starts the transfer of a new frame before the current frame is fully transmitted, this pulse is treated as an unexpected frame-synchronization pulse, and the transmitter sets the transmit frame-synchronization error bit (XSYNCERR) in SPCR2. XSYNCERR can be cleared only 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.

12.5.6.2 Example of Unexpected Transmit Frame-Synchronization Pulse

Section 12.5.3.2 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.

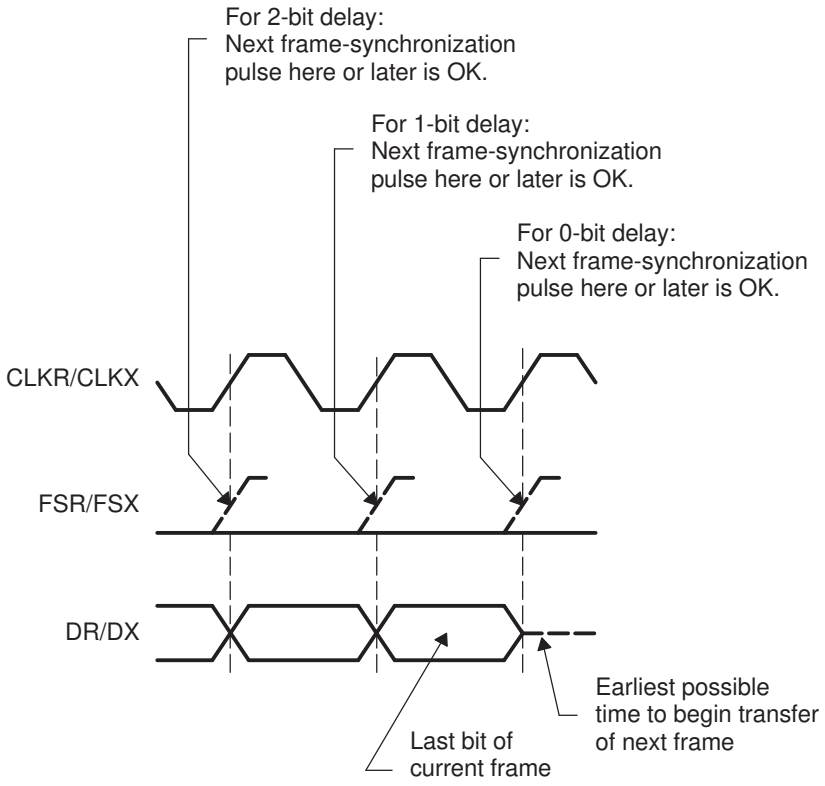
Figure 12-30. An Unexpected Frame-Synchronization Pulse During a McBSP Transmission



12.5.6.3 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 12-31 shows when a new frame-synchronization pulse on FSX can safely occur relative to the last bit of the current frame.

Figure 12-31. Proper Positioning of Frame-Synchronization Pulses



12.6 Multichannel Selection Modes

This section discusses the multichannel selection modes for the McBSP.

12.6.1 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 12-8 through Table 12-10) :

- 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 12-8. 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 12.6.4), 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 12.6.5), blocks 0 through 7 are automatically assigned to partitions, A through H, respectively.

Table 12-9. 2-Partition Mode

Partition	Blocks
A	0 or 2 or 4 or 6
B	1 or 3 or 5 or 7

Table 12-10. 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).

12.6.2 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 12.6.6) and three transmit multichannel selection modes (described in Section 12.6.7).

12.6.3 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.

12.6.4 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 12.6.5). 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.

12.6.4.1 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:

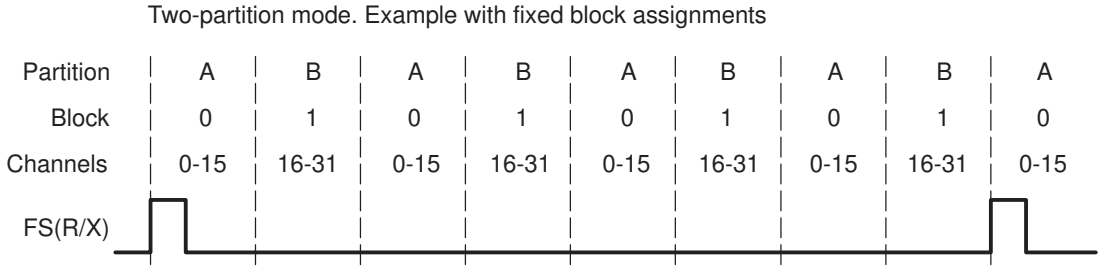
- Assign an even-numbered channel block (0, 2, 4, or 6) to receive partition A by writing to the RPABLK bits. In the receive multichannel selection mode (described in Section 12.6.6), the channels in this partition are controlled by receive channel enable register A (RCERA).
- Assign an odd-numbered 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 12.6.7), 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 12-32 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 12-32. Alternating Between the Channels of Partition A and the Channels of Partition B



As explained in Section 12.6.4.2, you can dynamically change which blocks of channels are assigned to the partitions.

12.6.4.2 Reassigning Blocks During Reception/Transmission

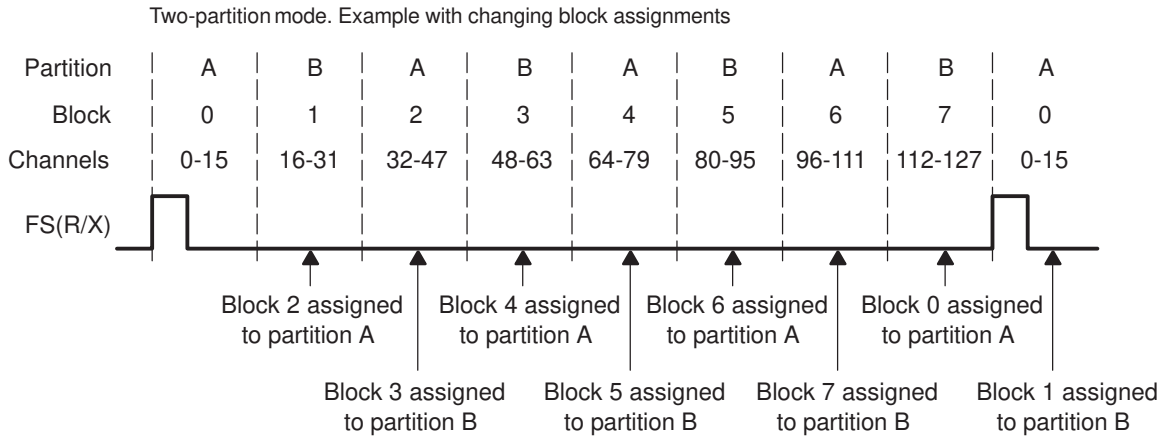
If you want to use more than 32 channels, you can change which channel blocks are assigned to partitions A and B during the course of a data transfer. However, these changes must be carefully timed. While a partition is being transferred, its associated block assignment bits cannot be modified and its associated channel enable register cannot be modified. For example, if block 3 is being transferred and block 3 is assigned to partition A, you cannot modify (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 12.6.8.

Figure 12-33 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 12-33. Reassigning Channel Blocks Throughout a McBSP Data Transfer



12.6.5 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 12.6.4). 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 12-11 and Table 12-12. These assignments cannot be changed. The tables also show the registers used to control the channels in the partitions.

Table 12-11. Receive Channel Assignment and Control With Eight Receive Partitions

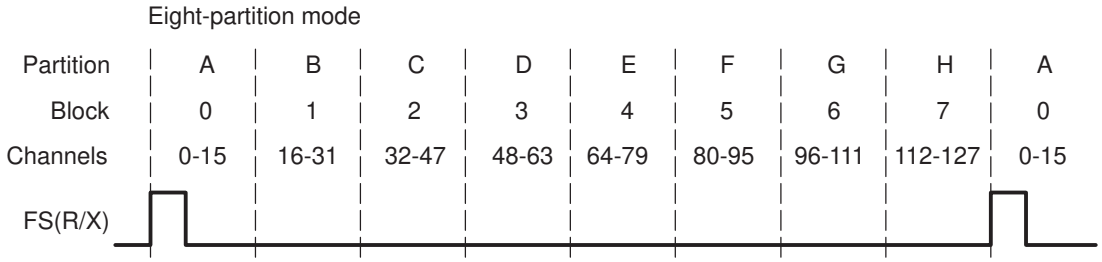
Receive Partition	Assigned Block of Receive Channels	Register Used For Channel Control
A	Block 0: channels 0 through 15	RCERA
B	Block 1: channels 16 through 31	RCERB
C	Block 2: channels 32 through 47	RCERC
D	Block 3: channels 48 through 63	RCERD
E	Block 4: channels 64 through 79	RCERE
F	Block 5: channels 80 through 95	RCERF
G	Block 6: channels 96 through 111	RCERG
H	Block 7: channels 112 through 127	RCERH

Table 12-12. 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 12-34 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 12-34. McBSP Data Transfer in the 8-Partition Mode



12.6.6 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

12.6.7 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 12.6.7.1. The McBSP has three transmit multichannel selection modes (XMCM = 01b, XMCM = 10b, and XMCM = 11b), which are described in the following table.

Table 12-13. 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

12.6.7.1 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.

12.6.7.2 Activity on McBSP Pins for Different Values of XMCM

Figure 12-35 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.

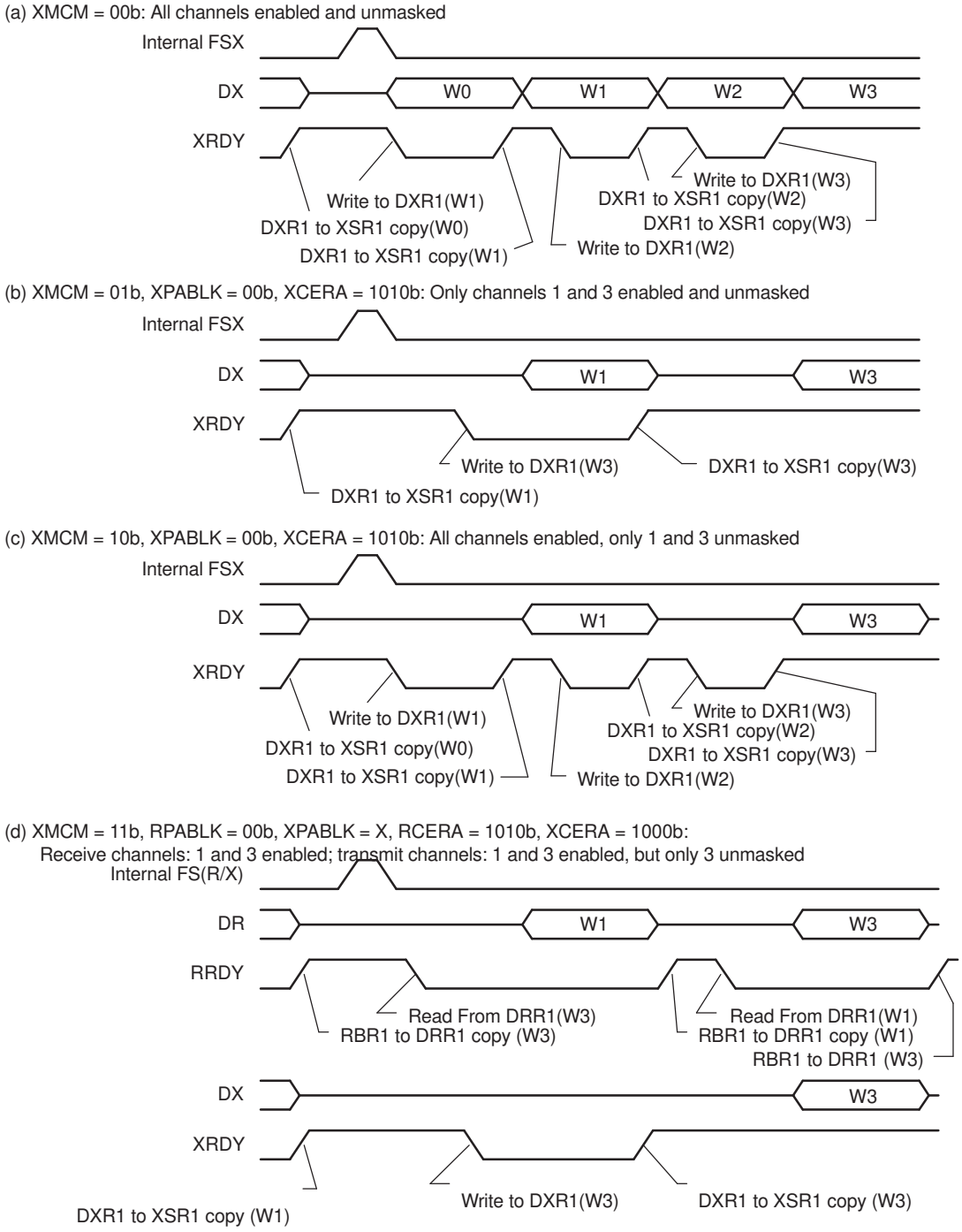
12.6.8 Using Interrupts Between Block Transfers

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 12.6.4) and you want to know when you can assign a different block of channels to partition A or B.

Figure 12-35. Activity on McBSP Pins for the Possible Values of XMCM



12.7 SPI Operation Using the Clock Stop Mode

This chapter explains how to use the McBSP in SPI mode.

12.7.1 SPI Protocol

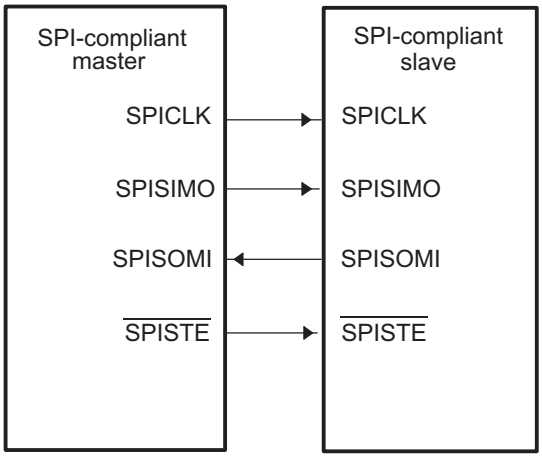
The SPI protocol is a master-slave configuration with one master device and one or more slave devices. The interface consists of the following four signals:

- Serial data input (also referred to as master in/slave out, or SPISOMI)

- Serial data output (also referred to as master out/slave in, or SPISIMO)
- 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 12-36.

Figure 12-36. 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.

12.7.2 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.

12.7.3 Enable and Configure the Clock Stop Mode

The bits required to configure the McBSP as an SPI device are introduced in Table 12-14. Table 12-15 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 12.7.4 show the effects of CLKSTP, CLKXP, and CLKRP.

Table 12-14. 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 12-15.)
CLKXP bit of PCR	This bit determines the polarity of the CLKX signal. (See also Table 12-15.)
CLKRP bit of PCR	This bit determines the polarity of the MCLKR signal. (See also Table 12-15.)
CLKXM bit of PCR	This bit determines whether CLKX is an input signal (McBSP as slave) or an output signal (McBSP as master).

Table 12-14. 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 12-15. Effects of CLKSTP, CLKXP, and CLKRP on the Clock Scheme

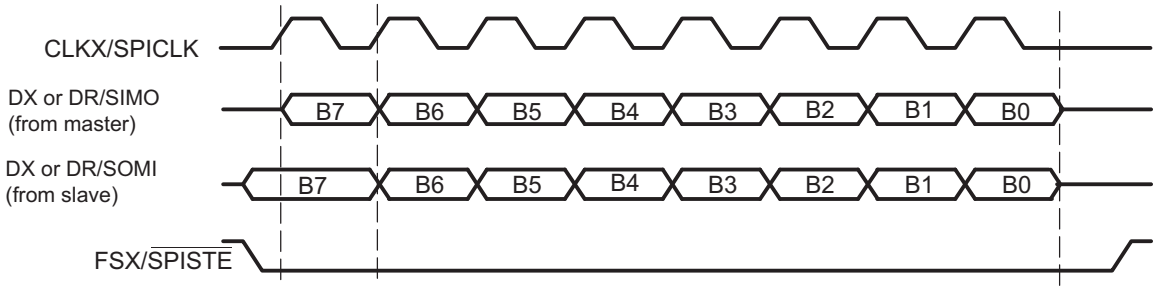
Bit Settings	Clock Scheme
CLKSTP = 00b or 01b CLKXP = 0 or 1 CLKRP = 0 or 1	Clock stop mode disabled. Clock enabled for non-SPI mode.
CLKSTP = 10b CLKXP = 0 CLKRP = 0	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of MCLKR.
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.

12.7.4 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.

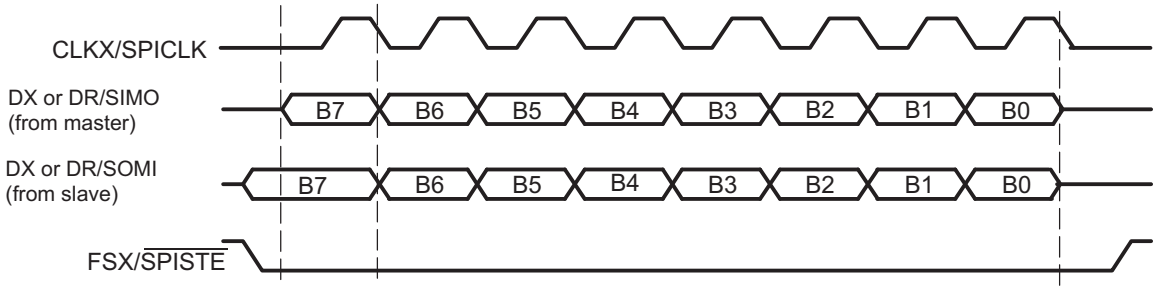
Figure 12-37. SPI Transfer With CLKSTP = 10b (No Clock Delay), CLKXP = 0, and CLKRP = 0



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.

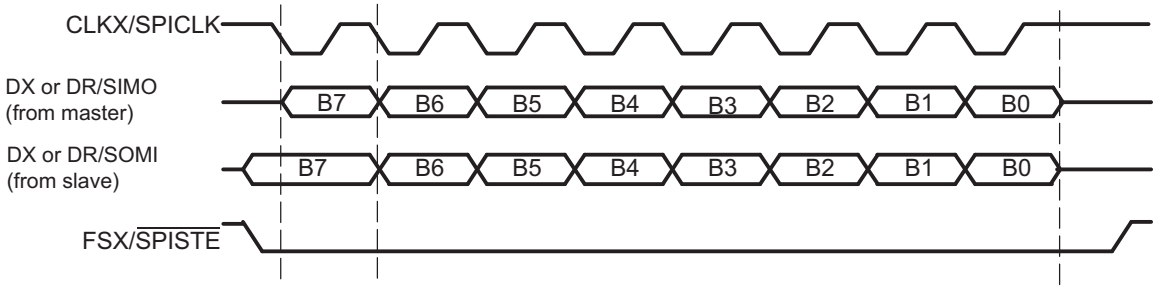
Figure 12-38. SPI Transfer With CLKSTP = 11b (Clock Delay), CLKXP = 0, CLKRP = 1



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.

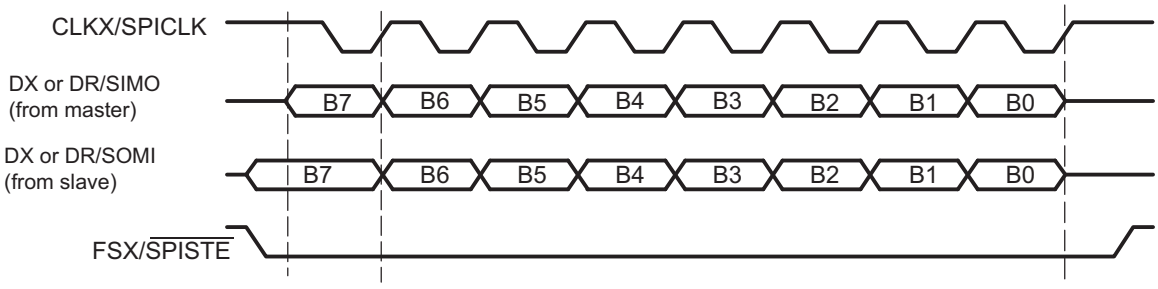
Figure 12-39. SPI Transfer With CLKSTP = 10b (No Clock Delay), CLKXP = 1, and CLKRP = 0



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 12-40. 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.

12.7.5 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 12.7.6)
- *McBSP as an SPI Slave* (Section 12.7.7)

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).

12.7.6 McBSP as the SPI Master

An SPI interface with the McBSP used as the master is shown in Figure 12-41. When the McBSP is configured as a master, the transmit output signal (DX) is used as the SPISIMO signal of the SPI protocol and the receive input signal (DR) is used as the SPISOMI signal.

The register bit values required to configure the McBSP as a master are listed in Table 12-16. After the table are more details about the configuration requirements.

Figure 12-41. SPI Interface with McBSP Used as Master

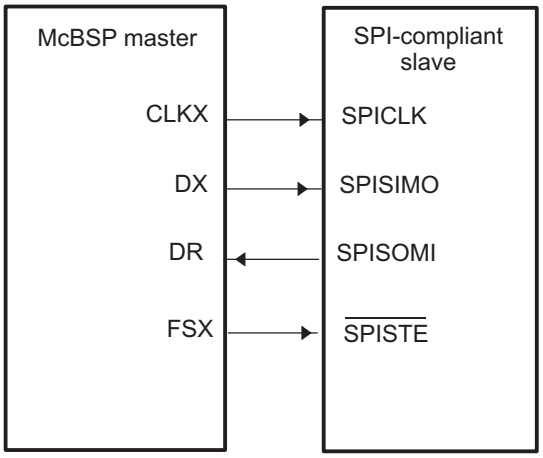


Table 12-16. Bit Values Required to Configure the McBSP as an SPI Master

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.
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 (SS_) 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 12.7.4. 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.

12.7.7 McBSP as an SPI Slave

An SPI interface with the McBSP used as a slave is shown in Figure 12-42. When the McBSP is configured as a slave, DX is used as the SPISOMI signal and DR is used as the SPISIMO signal.

The register bit values required to configure the McBSP as a slave are listed in Table 12-17. Following the table are more details about configuration requirements.

Figure 12-42. SPI Interface With McBSP Used as Slave

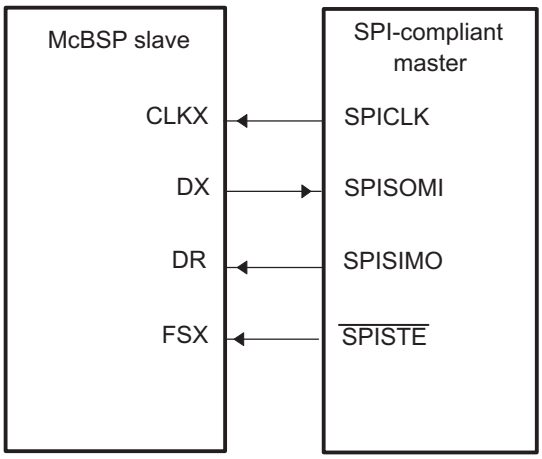


Table 12-17. Bit Values Required to Configure the McBSP as an SPI Slave

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	These bits must be 0s for SPI slave operation.
RDATDLY = 00b	

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.

12.8 Receiver Configuration

To configure the McBSP receiver, perform the following procedure:

1. Place the McBSP/receiver in reset (see Section 12.8.2).
2. Program McBSP registers for the desired receiver operation (see Section 12.8.1).
3. Take the receiver out of reset (see Section 12.8.2).

12.8.1 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.

12.8.2 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 12-18 describes the bits used for both of these steps.

Table 12-18. 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.

12.8.2.1 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 12-19 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 12.10.2.

Table 12-19. 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

12.8.3 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.

12.8.4 Digital Loopback Mode

The DLB bit determines whether the digital loopback mode is on. DLB is described in Table 12-20.

Table 12-20. 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.		

12.8.4.1 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 12-21. This mode allows testing of serial port code with a single DSP device; the McBSP receives the data it transmits.

Table 12-21. 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)

12.8.5 Clock Stop Mode

The CLKSTP bits determine whether the clock stop mode is on. CLKSTP is described in Table 12-22.

Table 12-22. 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		

12.8.5.1 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 12-23 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 12-23. 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.

12.8.6 Receive Multichannel Selection Mode

The RMCM bit determines whether the receive multichannel selection mode is on. RMCM is described in Table 12-24. For more details, see Section 12.6.6.

Table 12-24. 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.		

12.8.7 Receive Frame Phases

The RPHASE bit (see Table 12-25) determines whether the receive data frame has one or two phases.

Table 12-25. 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		

12.8.8 Receive Word Length(s)

The RWDLEN1 and RWDLEN2 bit fields (see Table 12-26) determine how many bits are in each serial word in phase 1 and in phase 2, respectively, of the receive data frame.

Table 12-26. 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		

12.8.8.1 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.

12.8.9 Receive Frame Length

The RFRLEN1 and RFRLEN2 bit fields (see Table 12-27) determine how many serial words are in phase 1 and in phase 2, respectively, of the receive data frame.

Table 12-27. 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 12-27. 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		

12.8.9.1 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 12-28 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 12-28. 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

12.8.10 Receive Frame-Synchronization Ignore Function

The RFIG bit (see Table 12-29) controls the receive frame-synchronization ignore function.

Table 12-29. 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.		

12.8.10.1 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:

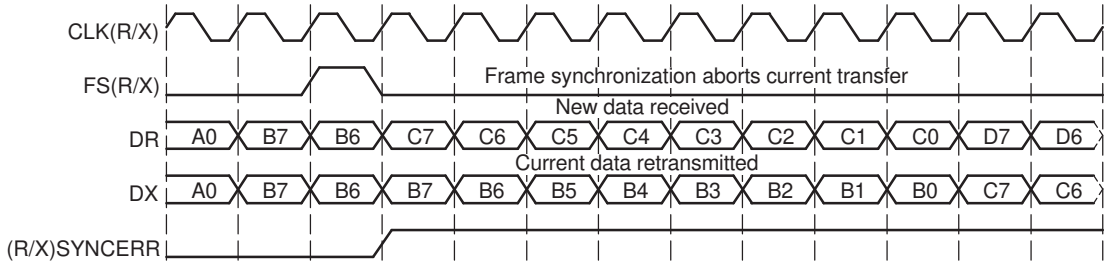
1. Aborts the current data transfer
2. Sets RSYNCERR in SPCR1 to 1
3. Begins the transfer of a new data word

For more details about the frame-synchronization error condition, see Section 12.5.3.

12.8.10.2 Examples of Effects of RFIG

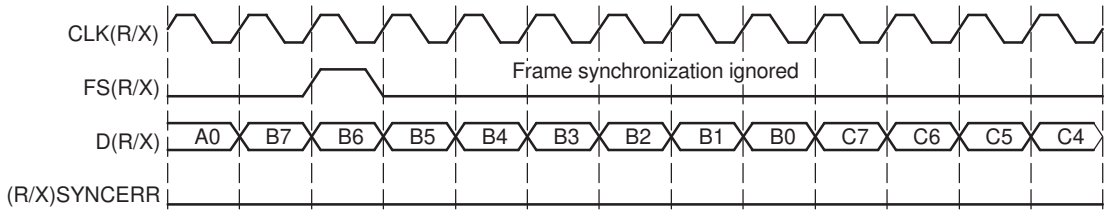
Figure 12-43 shows an example in which word B is interrupted by an unexpected frame-synchronization pulse when (R/X)FIG = 0. In the case of reception, the reception of B is aborted (B is lost), and a new data word C in this example) is received after the appropriate data delay. This condition is a receive synchronization error, which sets the RSYNCERR bit.

Figure 12-43. Unexpected Frame-Synchronization Pulse With (R/X)FIG = 0



In contrast with Figure 12-43, Figure 12-44 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 12-44. Unexpected Frame-Synchronization Pulse With (R/X)FIG = 1



12.8.11 Receive Companding Mode

The RCOMPAND bits (see Table 12-30) determine whether companding or another data transfer option is chosen for McBSP reception.

Table 12-30. 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 12.8.11.4).		
			RCOMPAND = 10 μ-law companding, 8-bit data, MSB received first		
			RCOMPAND = 11 A-law companding, 8-bit data, MSB received first		

12.8.11.1 Companding

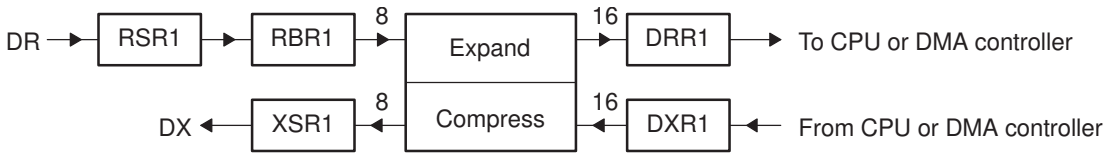
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM 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 12-45 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 12-45. Companding Processes for Reception and for Transmission



12.8.11.2 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.

12.8.11.3 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 12.3.2.2.

12.8.11.4 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.

12.8.12 Receive Data Delay

The RDATDLY bits (see Table 12-31) determine the length of the data delay for the receive frame.

Table 12-31. 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		
			RDATDLY = 11	Reserved		

12.8.12.1 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.

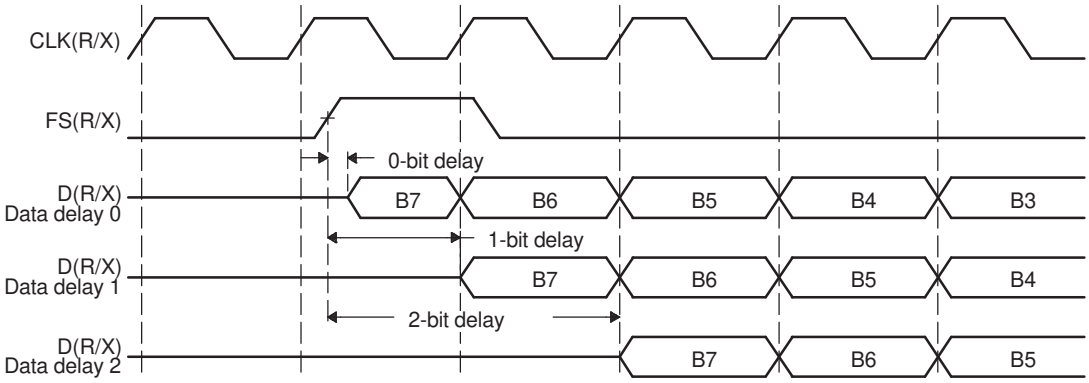
RDATDLY specifies the data delay for reception. The range of programmable data delay is zero to two bit-clocks (RDATDLY = 00b-10b), as described in Table 12-31 and shown in Figure 12-46. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on. Typically a 1-bit delay is selected, because data often follows a 1-cycle active frame-synchronization pulse.

12.8.12.2 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.

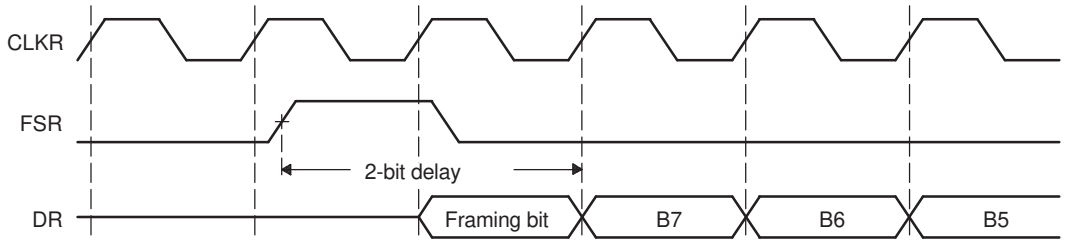
Figure 12-46. Range of Programmable Data Delay



12.8.12.3 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 12-47. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on.

Figure 12-47. 2-Bit Data Delay Used to Skip a Framing Bit



12.8.13 Receive Sign-Extension and Justification Mode

The RJUST bits (see Table 12-32) determine whether data received by the McBSP is sign-extended and how it is justified.

Table 12-32. 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		

12.8.13.1 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 12-33 and Table 12-34 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 12-33. 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 12-34. 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

12.8.14 Receive Interrupt Mode

The RINTM bits (see Table 12-35) 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 12-35. 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 12.6.8. 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 12.5.3.		

12.8.15 Receive Frame-Synchronization Mode

The bits described in Table 12-36 determine the source for receive frame synchronization and the function of the FSR pin.

12.8.15.1 Receive Frame-Synchronization Modes

Table 12-37 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 12-36. 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 12-36. 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 12.4.3.		
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 12-37. 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 12-37. 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.

12.8.16 Receive Frame-Synchronization Polarity

The FSRP bit (see Table 12-38) determines whether frame-synchronization pulses are active high or active low on the FSR pin.

Table 12-38. 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.		

12.8.16.1 Frame-Synchronization Pulses, Clock Signals, and Their Polarities

Receive frame-synchronization pulses can be generated internally by the sample rate generator (see Section 12.4.2) 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 12.8.15. Similarly, receive clocks can be selected to be inputs or outputs by programming the mode bit, CLKRM, in the PCR (see Section 12.8.17).

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.

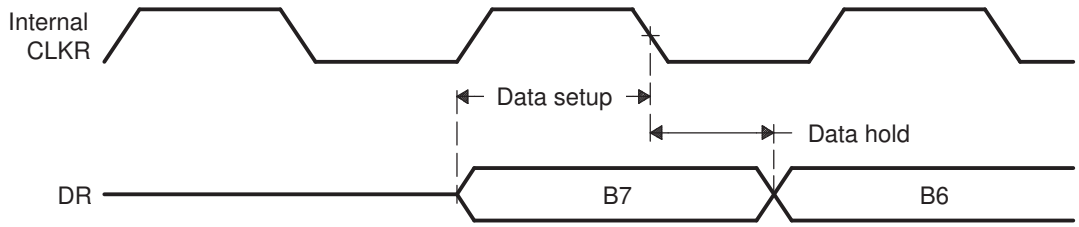
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, MCLKR, and CLKX signals, respectively. All frame-synchronization signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external 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 12-48 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 12-48. 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

12.8.16.2 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 12-39 shows settings for FPER and FWID.

Figure 12-49 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).

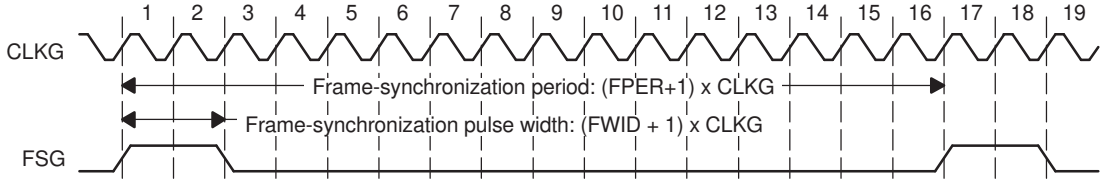
Table 12-39. Register Bits Used to Set the SRG Frame-Synchronization Period and Pulse Width

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 12-39. 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 12-49. Frame of Period 16 CLKG Periods and Active Width of 2 CLKG Periods



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.

12.8.17 Receive Clock Mode

Table 12-40 shows the settings for bits used to set receive clock mode.

Table 12-40. 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 12-40. 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.		

12.8.17.1 Selecting a Source for the Receive Clock and a Data Direction for the MCLKR Pin

Table 12-41 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 12-41. 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 12.9.19.	High impedance
1	1	Internal CLKX drives internal MCLKR. To configure CLKX, see Section 12.9.19.	Output. Internal MCLKR (same as internal CLKX) is inverted as determined by CLKRP before being driven out on the MCLKR pin.

12.8.18 Receive Clock Polarity

Table 12-42. 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		

12.8.18.1 Frame Synchronization Pulses, Clock Signals, and Their Polarities

Receive frame-synchronization pulses can be generated internally by the sample rate generator (see Section 12.4.2) 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 12.8.15. Similarly, receive clocks can be selected to be inputs or outputs by programming the mode bit, CLKRM, in the PCR (see Section 12.8.17).

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.

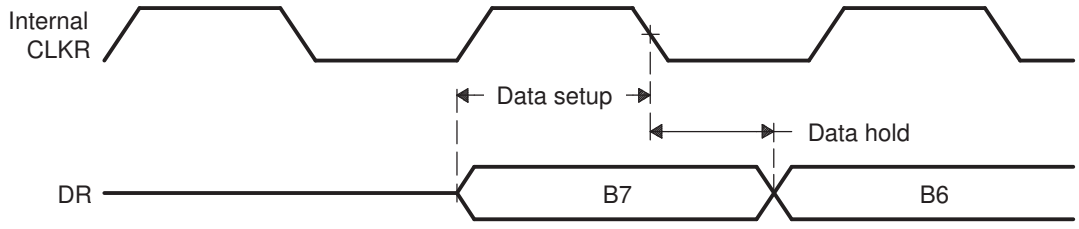
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, MCLKR, and CLKX signals, respectively. All frame-synchronization signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external 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 12-50 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 12-50. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



12.8.19 SRG Clock Divide-Down Value

Table 12-43. 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

12.8.19.1 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.

12.8.20 SRG Clock Synchronization Mode

For more details on using the clock synchronization feature, see Section 12.4.3.

Table 12-44. 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. 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.	R/W	0

12.8.21 SRG Clock Mode (Choose an Input Clock)

Table 12-45. 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		
			SCLKME = 0 CLKSM = 1	Sample rate generator clock derived from LSPCLK (default)		
			SCLKME = 1 CLKSM = 0	Sample rate generator clock derived from MCLKR pin		
			SCLKME = 1 CLKSM = 1	Sample rate generator clock derived from MCLKX pin		

12.8.21.1 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 12-45 shows the four possible sources of the input clock. For more details on generating CLKG, see Section 12.4.1.1.

12.8.22 SRG Input Clock Polarity

Table 12-46. 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.		

12.8.22.1 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.

12.9 Transmitter Configuration

To configure the McBSP transmitter, perform the following procedure:

1. Place the McBSP/transmitter in reset (see Section 12.9.2).
2. Program the McBSP registers for the desired transmitter operation (see Section 12.9.1).
3. Take the transmitter out of reset (see Section 12.9.2).

12.9.1 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.

12.9.2 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 12-47 describes the bits used for both of these steps.

Table 12-47. 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.

12.9.2.1 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 12.10.2.

12.9.3 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.

12.9.4 Digital Loopback Mode

The DLB bit determines whether the digital loopback mode is on. DLB is described in Table 12-48.

Table 12-48. 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.		

12.9.4.1 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 12-49. This mode allows testing of serial port code with a single DSP device; the McBSP receives the data it transmits.

Table 12-49. 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)

12.9.5 Clock Stop Mode

The CLKSTP bits determine whether the clock stop mode is on. CLKSTP is described in Table 12-50.

Table 12-50. 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		

12.9.5.1 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 12-51 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 12-51. 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.

12.9.6 Transmit Multichannel Selection Mode

For more details, see Section 12.6.7.

Table 12-52. 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.		

12.9.7 XCERs Used in the 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 12-96. 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 12-53. 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)

Table 12-53. Use of the Transmit Channel Enable Registers (continued)

Number of Selectable Channels	Block Assignments		Channel Assignments	
	XCERx	Block Assigned	Bit in XCERx	Channel Assigned
	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
	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

Table 12-53. Use of the Transmit Channel Enable Registers (continued)

Number of Selectable Channels	Block Assignments		Channel Assignments	
	XCERx	Block Assigned	Bit in XCERx	Channel Assigned
	XCERH	Block 7	XCE0	Channel 112
			XCE1	Channel 113
			XCE2	Channel 114
			:	:
			XCE15	Channel 127

12.9.8 Transmit Frame Phases

Table 12-54. 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		

12.9.9 Transmit Word Length(s)

Table 12-55. 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		

12.9.9.1 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.

12.9.10 Transmit Frame Length

Table 12-56. 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		

12.9.10.1 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 12-57 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 12-57. 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

12.9.11 Enable/Disable the Transmit Frame-Synchronization Ignore Function

Table 12-58. 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.		

12.9.11.1 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:

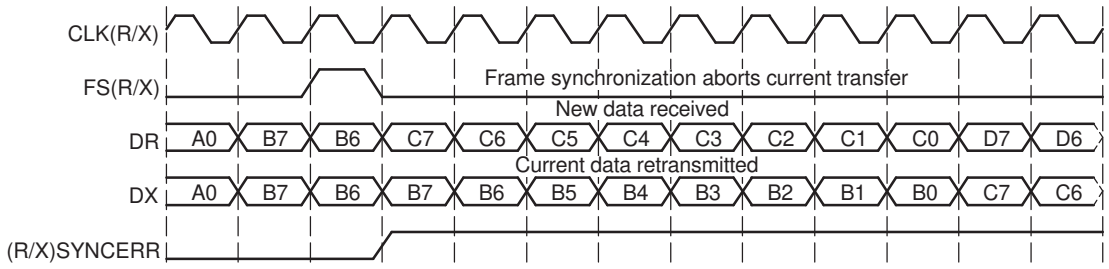
1. Aborts the present transmission
2. Sets XSYNCERR to 1 in SPCR2
3. Reinitiates transmission of the current word that was aborted

For more details about the frame-synchronization error condition, see Section 12.5.6.

12.9.11.2 Examples Showing the Effects of XFIG

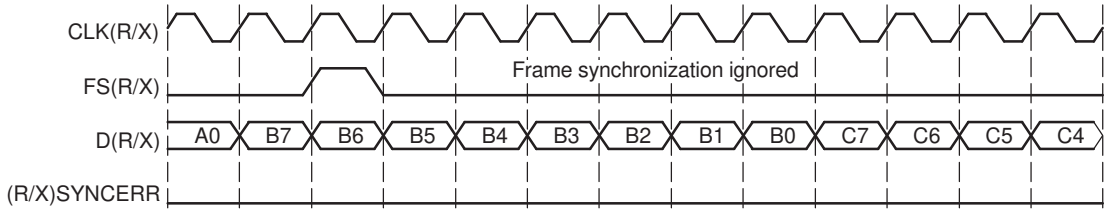
Figure 12-51 shows an example in which word B is interrupted by an unexpected frame-synchronization pulse when (R/X)FIG = 0. In the case of transmission, the transmission of B is aborted (B is lost). This condition is a transmit synchronization error, which sets the XSYNCERR bit. No new data has been written to DXR[1,2]; therefore, the McBSP transmits B again.

Figure 12-51. Unexpected Frame-Synchronization Pulse With (R/X) FIG = 0



In contrast with Figure 12-51, Figure 12-52 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 12-52. Unexpected Frame-Synchronization Pulse With (R/X) FIG = 1



12.9.12 Transmit Companding Mode

Table 12-59. 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 12.8.11.4, *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		

12.9.12.1 Companding

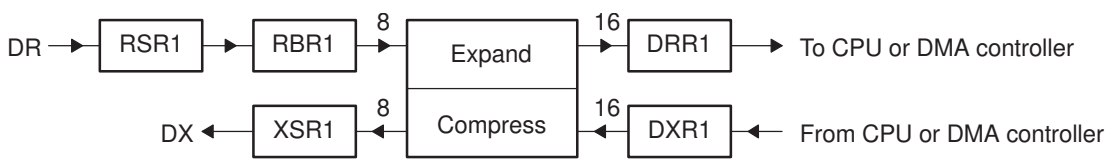
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM 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 12-53 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.

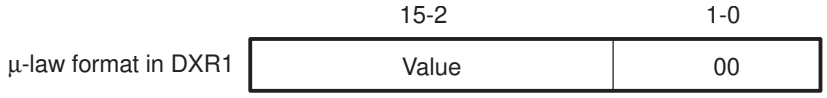
Figure 12-53. Companding Processes for Reception and for Transmission



12.9.12.2 Format for Data To Be Compressed

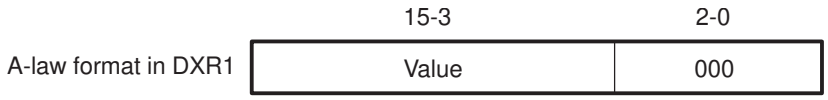
For transmission using μ-law compression, make sure the 14 data bits are left-justified in DXR1, with the remaining two low-order bits filled with 0s as shown in Figure 12-54.

Figure 12-54. 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 12-55.

Figure 12-55. A-Law Transmit Data Companding Format



12.9.12.3 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 12.3.2.2, *Capability to Compand Internal Data*.

12.9.12.4 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.

12.9.13 Transmit Data Delay

Table 12-60. Register Bits Used to Set the Transmit Data Delay

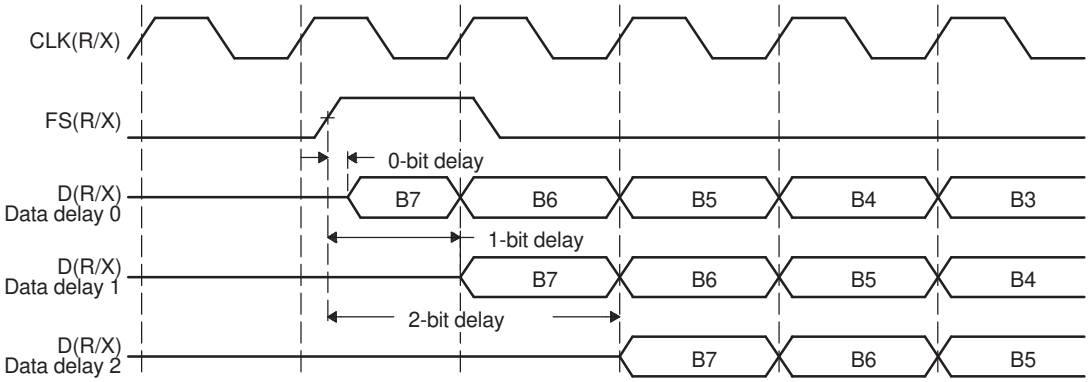
Register	Bit	Name	Function		Type	Reset Value
XCR2	1-0	XDATDLY	Transmitter data delay		R/W	00
			XDATDLY = 00	0-bit data delay		
			XDATDLY = 01	1-bit data delay		
			XDATDLY = 10	2-bit data delay		
			XDATDLY = 11	Reserved		

12.9.13.1 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 12-60 and Figure 12-56. 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 12-56. Range of Programmable Data Delay



12.9.13.2 0-Bit Data Delay

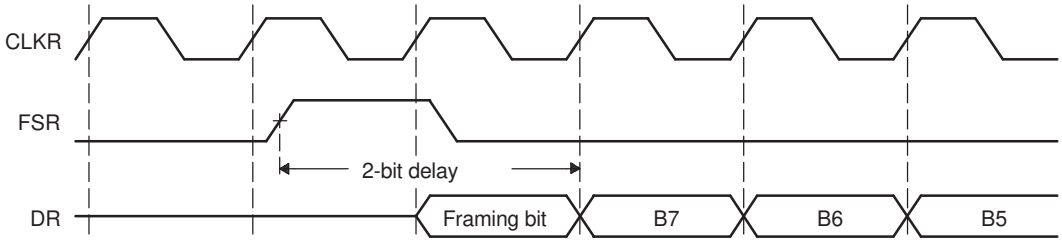
Normally, a frame-synchronization pulse is detected or sampled with respect to an edge of serial clock internal CLK(R/X). Thus, on the following cycle or later (depending on the data delay value), data can be received or transmitted. However, in the case of 0-bit data delay, the data must be ready for 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.

12.9.13.3 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 12-57. 2-Bit Data Delay Used to Skip a Framing Bit



12.9.14 Transmit DXENA Mode

Table 12-61 shows which register bit enables the Transmit DXENA (DX Delay Enabler) Mode.

Table 12-61. 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.		

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.

12.9.15 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 12-62. 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 12.6.8. 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 12.5.6.		

12.9.16 Transmit Frame-Synchronization Mode

Table 12-63 shows which register bits enable the Transmit Frame-Synchronization Mode.

Table 12-63. 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.		

Table 12-64 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 12-64 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 12-64. 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.

12.9.16.1 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 12.4.3.

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.

12.9.17 Transmit Frame-Synchronization Polarity

Table 12-65 shows which register bits enable the Transmit Frame-Synchronization Polarity.

Table 12-65. 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.		

12.9.17.1 Frame Synchronization Pulses, Clock Signals, and Their Polarities

Transmit frame-synchronization pulses can be generated internally by the sample rate generator (see Section 12.4.2) 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 12.9.16). Similarly, transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLKXM, in the PCR (see Section 12.9.19).

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.

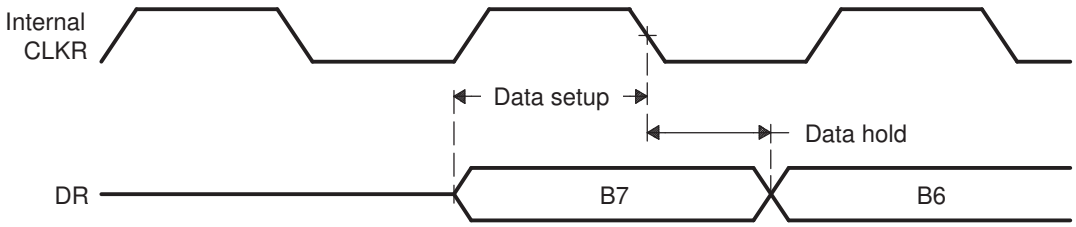
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, MCLKR, and CLKX signals, respectively. All frame-synchronization signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external 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 12-58 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 12-58. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



12.9.18 SRG Frame-Synchronization Period and Pulse Width

Table 12-66 shows which register bits set the SRG Frame-Synchronization Period and Pulse Width.

Table 12-66. 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	R/W	0000 0000 0000
			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.		
SRGR1	15-8	FWID	Sample rate generator frame-synchronization pulse width	R/W	0000 0000
			This field plus 1 determines the width of each frame-synchronization pulse on FSG.		
			Range for (FWID + 1): 1 to 256 CLKG cycles.		

12.9.18.1 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.

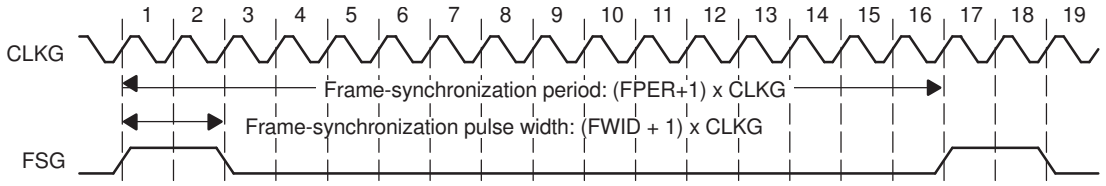
On FSG, the period from the start of a frame-synchronization pulse to the start of the next pulse is (FPER + 1) CLKG cycles. The 12 bits of FPER allow a frame-synchronization period of 1 to 4096 CLKG cycles, which allows up to 4096 data bits per frame. When GSYNC = 1, FPER is a don't care value.

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 12-59 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 12-59. Frame of Period 16 CLKG Periods and Active Width of 2 CLKG Periods



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.

12.9.19 Transmit Clock Mode

Table 12-67 shows which register bits can set the Transmit Clock Mode.

Table 12-67. 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.		

12.9.19.1 Selecting a Source for the Transmit Clock and a Data Direction for the MCLKX pin

Table 12-68 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 12-68. 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.

12.9.19.2 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 12.4.3.

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.

12.9.20 Transmit Clock Polarity

Table 12-69 shows which register bits set the Transmit Clock Polarity.

Table 12-69. 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.		

12.9.20.1 Frame Synchronization Pulses, Clock Signals, and Their Polarities

Transmit frame-synchronization pulses can be either generated internally by the sample rate generator (see Section 12.4.2) 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 12.9.16). Similarly, transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLKXM, in the PCR (see Section 12.9.19).

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.

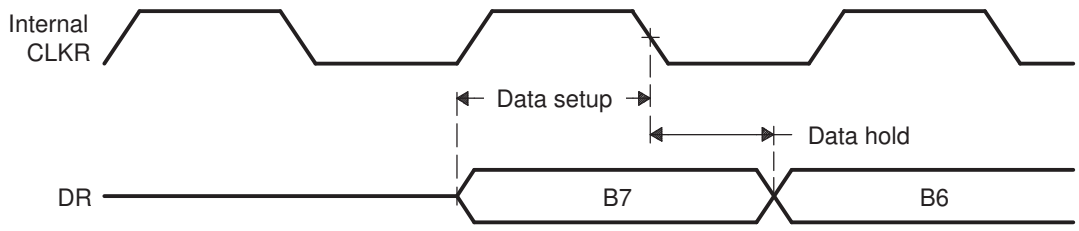
On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1 and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input 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 12-58).

Figure 12-60 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 12-60. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



12.10 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 12.10.1)
- How to reset and initialize the various parts of the McBSP (see Section 12.10.2)

12.10.1 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 12-70.

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 12-70. 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.

12.10.2 Resetting and Initializing McBSPs

This section discusses in greater depth the MCBSP Reset and Initialization configurations.

12.10.2.1 McBSP Pin States: DSP Reset Versus Receiver/Transmitter Reset

Table 12-71 shows the state of McBSP pins when the serial port is reset due to direct receiver or transmitter reset on the 2833x device.

Table 12-71. 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.

12.10.2.2 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.

12.10.2.3 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 12.3.2.2).
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.

12.10.2.4 Resetting the Transmitter While the Receiver is Running

Example 12-1 shows values in the control registers that reset and configure the transmitter while the receiver is running.

Example 12-1. 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.
```

12.11 Data Packing Examples

This section shows two ways to implement data packing in the McBSP.

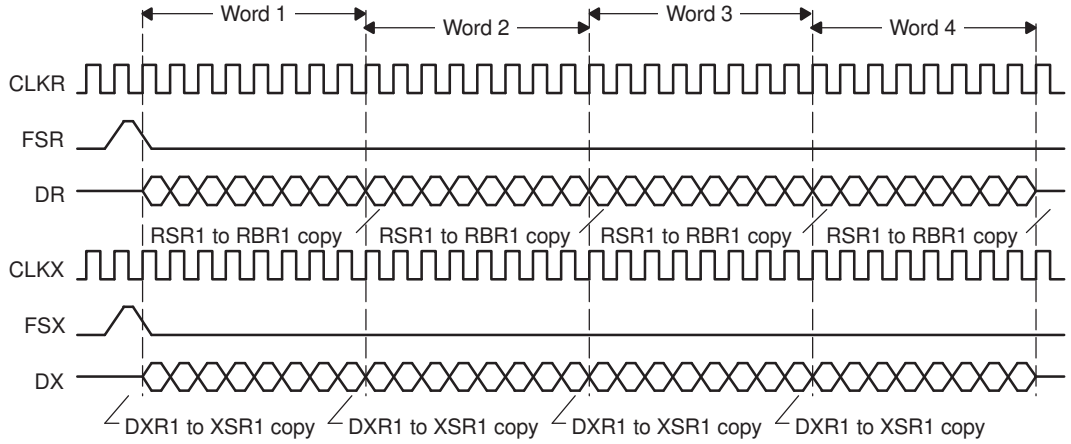
12.11.1 Data Packing Using Frame Length and Word Length

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 12-61. 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 12-61. Four 8-Bit Data Words Transferred To/From the McBSP



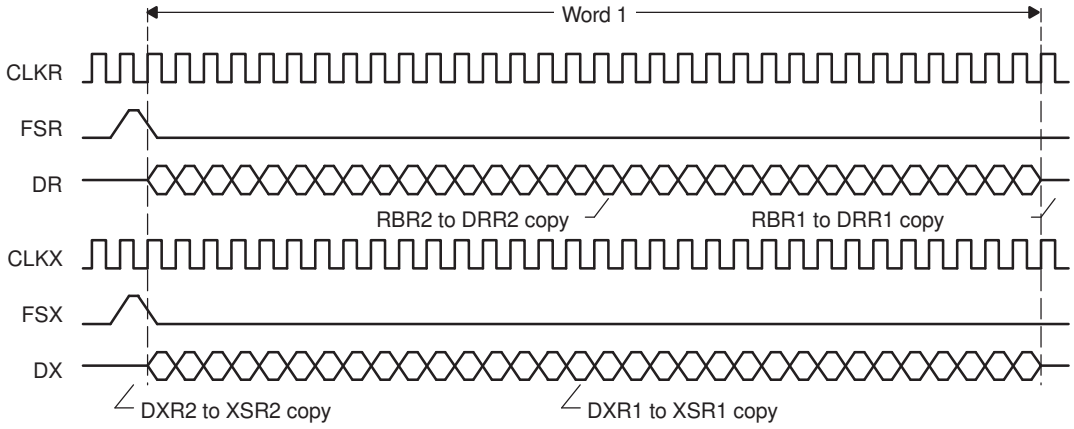
This data can also be treated as a single-phase frame consisting of one 32-bit data word, as shown in Figure 12-62. In this case:

- (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 12-62. One 32-Bit Data Word Transferred To/From the McBSP



12.11.2 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 12-63, 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 12-63. 8-Bit Data Words Transferred at Maximum Packet Frequency

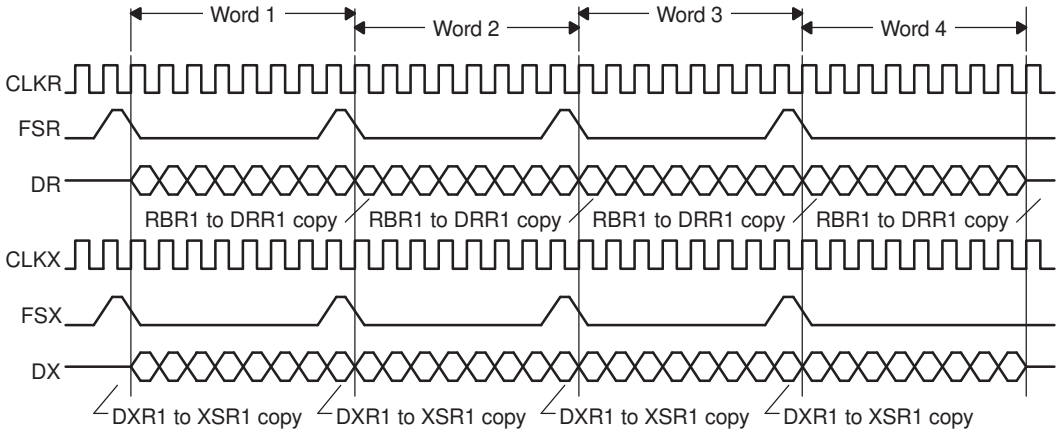
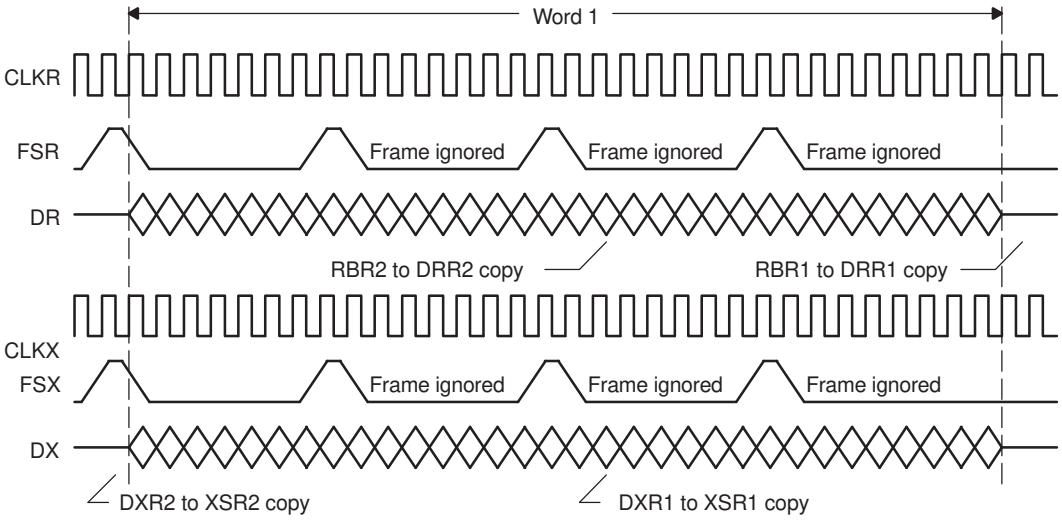


Figure 12-64 shows the McBSP configured to treat this data stream as a continuous 32-bit word. In this example, the McBSP responds to an initial frame-synchronization pulse. However, (R/X)FIG = 1 so that the McBSP ignores subsequent pulses. Only two read transfers or two write transfers are needed every 32 bits. This configuration effectively reduces the required bus bandwidth to half the bandwidth needed to transfer four 8-bit words.

Figure 12-64. Configuring the Data Stream of Figure 12-63 as a Continuous 32-Bit Word



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.

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 12-65. Receive Interrupt Generation

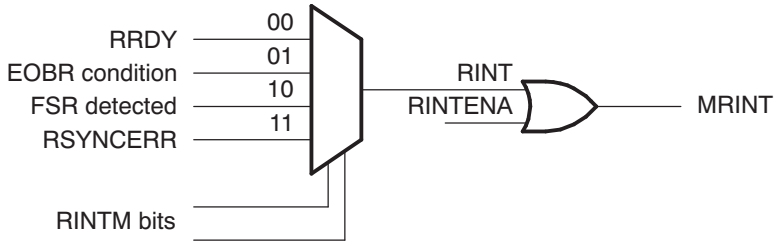


Table 12-72. Receive Interrupt Sources and Signals

McBSP Interrupt Signal	Interrupt Flags	Interrupt Enables in SPCR1	Interrupt Enables	Type of Interrupt	Interrupt Line
		RINTM Bits			
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.

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 12-66. Transmit Interrupt Generation

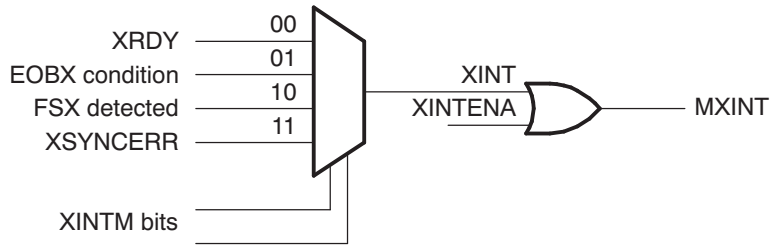


Table 12-73. Transmit Interrupt Sources and Signals

McBSP Interrupt Signal	Interrupt Flags	Interrupt Enables in SPCR2	Interrupt Enables	Type of Interrupt	Interrupt Line
		XINTM Bits			
XINT	XRDY	0	XINTENA	Every word transmit	MXINT
	EOBX	1	XINTENA	Every 16-channel block boundary	

Table 12-73. Transmit Interrupt Sources and Signals (continued)

McBSP Interrupt Signal	Interrupt Flags	Interrupt Enables in SPCR2	Interrupt Enables	Type of Interrupt	Interrupt Line
	FSX	10	XINTENA	On every FSX	
	XSYNCERR	11	XINTENA	Frame sync error	

12.12.3 Error Flags

The McBSP has several error flags both on receive and transmit channel. Table 12-74 explains the error flags and their meaning.

Table 12-74. 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.

12.13 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 12-75 provides a quick reference to McBSP mode selection.

Table 12-75. 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 XCERs
		1	1	1	10	All channels are enabled,but masked unless

Table 12-75. McBSP Mode Selection (continued)

No.	McBSP Word Size	Register Bits Used for Mode Selection				Mode and Function Description
		MCR1 bit 9,0		MCR2 bit 9,1,0		
		RMCME	RMCM	XMCME	XMCM	
						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

12.14 Special Case: External Device is the Transmit Frame Master

Care must be taken if the transmitter expects a frame sync from an external device. After the transmitter comes out of reset (XRST = 1), it waits for a frame sync from the external device. If the first frame sync arrives very shortly after the transmitter is enabled, the CPU or DMA controller may not have a chance to service DXR. In this case, the transmitter shifts out the default data in XSR instead of the desired value, which has not yet arrived in DXR. This causes problems in some applications, as the first data element in the frame is invalid. The data stream appears element-shifted (the first data word may appear in the second channel instead of the first).

To ensure proper operation when the external device is the frame master, you must assure that DXR is already serviced with the first word when a frame sync occurs. To do so, you can keep the transmitter in reset until the first frame sync is detected. Upon detection of the first frame sync, the McBSP generates an interrupt to the CPU. Within the interrupt service routine, the transmitter is taken out of reset (XRST = 1). This assures that the transmitter does not begin data transfers at the data pin during the first frame sync period. This also provides almost an entire frame period for the DSP to service DXR with the first word before the second frame sync occurs. The transmitter only begins data transfers upon receiving the second frame sync. At this point, DXR is already serviced with the first word.

The interrupt service routine must first be setup according to the description in . Then follow this modified procedure for proper initialization:

1. Ensure that no portion of the McBSP is using the internal sample rate generator signal CLKG and the internal frame sync generator signal FSG (GRST = FRST = 0). The respective portion of the McBSP needs to be in reset (XRST = 0 and/or RRST = 0).
2. Program SRGR and other control registers as required. Ensure the internal sample rate generator and the internal frame sync generator are still in reset (GRST = FRST = 0). Also ensure the respective portion of the McBSP is still in reset in this step (XRST = 0 and/or RRST = 0).
3. Program the XINTM bits to 2h in SPCR to generate an interrupt to the CPU upon detection of a transmit frame sync. Do not enable the XINT interrupt in the interrupt enable register (IER) in this step.
4. Wait for proper internal synchronization. If the external device provides the bit clock, wait for two CLKR or CLKX cycles. If the McBSP generates the bit clock as a clock master, wait for two CLKSRG cycles. In this case, the clock source to the sample rate generator (CLKSRG) is selected by the CLKSM bit in SRGR.
5. Skip this step if the bit clock is provided by the external device. This step only applies if the McBSP is the bit clock master and the internal sample rate generator is used.
   a. Start the sample rate generator by setting the GRST bit to 1. Wait two CLKG bit clocks for synchronization. CLKG is the output of the sample rate generator.
   b. On the next rising edge of CLKSRG, CLKG transitions to 1 and starts clocking with a frequency equal to 1/(CLKGDV + 1) of the sample rate generator source clock CLKSRG.
6. A transmit sync error (XSYNCERR) may occur when it is enabled for the first time after device reset. The purpose of this step is to clear any potential XSYNCERR that occurs on the transmitter at this time:
   a. Set the XRST bit to 1 to enable the transmitter.
   b. Wait for any unexpected frame sync error to occur. If the external device provides the bit clock,

wait for two CLKR or CLKX cycles. If the McBSP generates the bit clock as a clock master, wait for two CLKG cycles. The unexpected frame sync error (XSYNCERR), if any, occurs within this time period.

c. Disable the transmitter (XRST = 0). This clears any outstanding XSYNCERR.

7. Setup data acquisition as required:
   a. If the DMA controller is used to service the McBSP, setup data acquisition as desired and start the DMA controller in this step, before the McBSP is taken out of reset.
   b. If CPU interrupt is used to service the McBSP, no action is required in this step.
   c. If CPU polling is used to service the McBSP, no action is required in this step.
8. Enable the XINT interrupt by setting the corresponding bit in the interrupt enable register (IER). In this step, the McBSP transmitter is still in reset. Upon detection of the first transmit frame sync from the external device, the McBSP generates an interrupt to the CPU and the DSP enters the interrupt service routine (ISR). The ISR needs to perform these tasks in this order:
   a. Modify the XINTM bits to the value desired for normal McBSP operations. If CPU interrupt is used to service the McBSP in normal operations, ensure that the XINTM bits are modified to 0 to detect the McBSP XRDY event. If no McBSP interrupt is desired in normal operations, disable future McBSP-to-CPU interrupt in the interrupt enable register (IER).
   b. Set the XRST bit and/or the RRST bit to 1 to enable the respective portion of the McBSP. The McBSP is now ready to transmit and/or receive.
9. Service the McBSP:
   a. If CPU polling is used to service the McBSP in normal operations, it can do so upon exit from the ISR.
   b. If CPU interrupt is used to service the McBSP in normal operations, upon XRDY interrupt service routine is entered. The ISR should be setup to verify that XRDY = 1 and service the McBSP accordingly.
   c. If DMA controller is used to service the McBSP in normal operations, it services the McBSP automatically upon receiving the XEVT and/or REVT.
10. Upon detection of the second frame sync, DXR is already serviced and the transmitter is ready to transmit the valid data. The receiver is also serviced properly by the DSP.

12.15 McBSP Registers

This section describes the McBSP registers.

12.15.1 McBSP Base Addresses

Table 12-76 shows the registers accessible on each McBSP. Section 12.15.2 through Section 12.15.11 describe the register bits.

Table 12-76. McBSP Register Summary

Name	McBSP-A Address	McBSP-B Address	Type	Reset Value	Description
Data Registers, Receive, Transmit					
DRR2	0x6000	0x6040	R	0x0000	McBSP Data Receive Register 2
DRR1	0x6001	0x6041	R	0x0000	McBSP Data Receive Register 1
DXR2	0x6002	0x6042	W	0x0000	McBSP Data Transmit Register 2
DXR1	0x6003	0x6043	W	0x0000	McBSP Data Transmit Register 1
McBSP Control Registers					
SPCR2	0x6004	0x6044	R/W	0x0000	McBSP Serial Port Control Register 2
SPCR1	0x6005	0x6045	R/W	0x0000	McBSP Serial Port Control Register 1
RCR2	0x6006	0x6046	R/W	0x0000	McBSP Receive Control Register 2
RCR1	0x6007	0x6047	R/W	0x0000	McBSP Receive Control Register 1
XCR2	0x6008	0x6048	R/W	0x0000	McBSP Transmit Control Register 2
XCR1	0x6009	0x6049	R/W	0x0000	McBSP Transmit Control Register 1
SRGR2	0x600A	0x604A	R/W	0x0000	McBSP Sample Rate Generator Register 2
SRGR1	0x600B	0x604B	R/W	0x0000	McBSP Sample Rate Generator Register 1
Multichannel Control Registers					
MCR2	0x600C	0x604C	R/W	0x0000	McBSP Multichannel Register 2
MCR1	0x600D	0x604D	R/W	0x0000	McBSP Multichannel Register 1
RCERA	0x600E	0x604E	R/W	0x0000	McBSP Receive Channel Enable Register Partition A
RCERB	0x600F	0x604F	R/W	0x0000	McBSP Receive Channel Enable Register Partition B
XCERA	0x6010	0x6050	R/W	0x0000	McBSP Transmit Channel Enable Register Partition A
XCERB	0x6011	0x6051	R/W	0x0000	McBSP Transmit Channel Enable Register Partition B
PCR	0x6012	0x6052	R/W	0x0000	McBSP Pin Control Register
RCERC	0x6013	0x6053	R/W	0x0000	McBSP Receive Channel Enable Register Partition C
RCERD	0x6014	0x6054	R/W	0x0000	McBSP Receive Channel Enable Register Partition D
XCERC	0x6015	0x6055	R/W	0x0000	McBSP Transmit Channel Enable Register Partition C
XCERD	0x6016	0x6056	R/W	0x0000	McBSP Transmit Channel Enable Register Partition D
RCERE	0x6017	0x6057	R/W	0x0000	McBSP Receive Channel Enable Register Partition E
RCERF	0x6018	0x6058	R/W	0x0000	McBSP Receive Channel Enable Register Partition F
XCERE	0x6019	0x6059	R/W	0x0000	McBSP Transmit Channel Enable Register Partition E
XCERF	0x601A	0x605A	R/W	0x0000	McBSP Transmit Channel Enable Register Partition F
RCERG	0x601B	0x605B	R/W	0x0000	McBSP Receive Channel Enable Register Partition G
RCERH	0x601C	0x605C	R/W	0x0000	McBSP Receive Channel Enable Register Partition H
XCERG	0x601D	0x605D	R/W	0x0000	McBSP Transmit Channel Enable Register Partition G
XCERH	0x601E	0x605E	R/W	0x0000	McBSP Transmit Channel Enable Register Partition H
MFFINT	0x6023	0x6063	R/W	0x0000	McBSP Interrupt Enable Register

12.15.2 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 12-67). 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 12-67. 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

12.15.2.1 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.

12.15.3 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 12-68). 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 12-68. 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

12.15.3.1 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.

12.15.4 Serial Port Control Registers (SPCR[1,2])

Each McBSP has two serial port control registers, SPCR1 (Table 12-77) and SPCR2 (Table 12-78). 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)

12.15.4.1 Serial Port Control 1 Register (SPCR1)

The serial port control 1 register (SPCR1) is shown in Figure 12-69 and described in Table 12-77.

Figure 12-69. Serial Port Control 1 Register (SPCR1)

15	14	13	12	11	10	9	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 12-77. 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.

Table 12-77. Serial Port Control 1 Register (SPCR1) Field Descriptions (continued)

Bit	Field	Value	Description
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 12.8.13.
		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)
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 12.8.5.
		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 12.9.14.
		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 12.5.3.

Table 12-77. Serial Port Control 1 Register (SPCR1) Field Descriptions (continued)

Bit	Field	Value	Description
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 12.5.2.
		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.
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 12.10.2.
		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.

12.15.4.2 Serial Port Control 2 Register (SPCR2)

The serial port control 2 register (SPCR2) is shown in Figure 12-70 and described in Table 12-78.

Figure 12-70. 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 12-78. 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 12.10.2.
		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 12-78. 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.
		1h	The McBSP sends an XINT request to the CPU when 16 enabled bits have been transmitted on the DX pin. 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 12.5.6.
		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 12.5.5.
		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 12-78. 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 12.10.2.
		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.

12.15.5 Receive Control Registers (RCR[1, 2])

Each McBSP has two receive control registers, RCR1 (Table 12-79) and RCR2 (Table 12-81). 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)

12.15.5.1 Receive Control Register 1 (RCR1)

The receive control register 1 (RCR1) is shown in Figure 12-71 and described in Table 12-79.

Figure 12-71. 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 12-79. 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 12-80 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 12-79. 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 12-80. Frame Length Formula for Receive Control 1 Register (RCR1)

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

12.15.5.2 Receive Control Register 2 (RCR2)

The receive control register 2 (RCR2) is shown in Figure 12-72 and described in Table 12-81.

Figure 12-72. Receive Control Register 2 (RCR2)

15	14	13	12	11	10	9	8
RPHASE	RFRLEN2						
R/W-0	R/W-0						

7	6	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 12-81. 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 12-82 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 12-81. 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 12.3.2.
		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 Figure 12-30. Setting RFIG causes the serial port to ignore unexpected frame-synchronization signals during reception. For more details on the effects of RFIG, see Section 12.8.10.1.
		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 12.8.12.
		0	0-bit data delay
		1h	1-bit data delay
		2h	2-bit data delay
		3h	Reserved (do not use)

Table 12-82. 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

12.15.6 Transmit Control Registers (XCR1 and XCR2)

Each McBSP has two transmit control registers, XCR1 (Table 12-83) and XCR2 (Table 12-85). 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)

12.15.6.1 Transmit Control 1 Register (XCR1)

The transmit control 1 register (XCR1) is shown in Figure 12-73 and described in Table 12-83.

Figure 12-73. 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 12-83. 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 12-84 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 12-84. 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

12.15.6.2 Transmit Control 2 Register (XCR2)

The transmit control 2 register (XCR2) is shown in Figure 12-74 and described in Table 12-85.

Figure 12-74. 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 12-85. 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 12-86 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	XCOMPAND	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 12.3.2.
			XCOMPAND allows you to choose one of the following companding modes for the McBSP transmitter. For more details about these companding modes, see Section 12.3.2.
		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 12-85. 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. Setting XFIG causes the serial port to ignore unexpected frame-synchronization pulses during transmission.
		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 12.9.13.
		0	0-bit data delay
		1h	1-bit data delay
		2h	2-bit data delay
		3h	Reserved (do not use)

Table 12-86. 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

12.15.7 Sample Rate Generator Registers (SRGR1 and SRGR2)

Each McBSP has two sample rate generator registers, SRGR1 (Table 12-87) and SRGR2 (Table 12-88). 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.

12.15.7.1 Sample Rate Generator 1 Register (SRGR1)

The sample rate generator 1 register is shown in Figure 12-75 and described in Table 12-87.

Figure 12-75. 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 12-87. 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 ≤ FWID ≤ 255 1 ≤ (FWID + 1) ≤ 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

12.15.7.2 Sample Rate Generator 2 Register (SRGR2)

The sample rate generator 2 register (SRGR2) is shown in Figure 12-76 and described in Table 12-88.

Figure 12-76. 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 12-88. 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 12.4.3.
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.

12.15.8 Multichannel Control Registers (MCR[1,2])

Each McBSP has two multichannel control registers. MCR1 (Table 12-89) has control and status bits (with an R prefix) for multichannel selection operation in the receiver. MCR2 (Table 12-90) 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)

12.15.8.1 Multichannel Control 1 Register (MCR1)

The multichannel control 1 register (MCR1) is shown in Figure 12-77 and described in Table 12-89.

Figure 12-77. 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 12-89. 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 12-89. 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.

12.15.8.2 Multichannel Control 2 Register (MCR2)

The multichannel control 2 register (MCR2) is shown in Figure 12-78 and described in Table 12-90.

Figure 12-78. 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 12-90. 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 12-90. 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 12.6.7 .
		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.

12.15.9 Pin Control Register (PCR)

Each McBSP has one pin control register (PCR). Table 12-91 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 12-79 and described in Table 12-91.

Figure 12-79. 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 12-91. 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 12-91. 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 12-91. 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 12-92. 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

12.15.10 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 12-80. There is one enable register for each of the receive partitions: A, B, C, D, E, F, G, and H. Table 12-93 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 12.15.10.1.

The receive channel enable registers (RCERA...RCERH) are shown in Figure 12-80 and described in Table 12-93.

Figure 12-80. 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 12-93. 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.

12.15.10.1 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 12-94 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 12-94. 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 12-94. 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

12.15.11 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 12-81. There is one for each of the transmit partitions: A, B, C, D, E, F, G, and H. Table 12-95 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 12-81 and described in Table 12-95.

Figure 12-81. 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 12-95. 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.

12.15.12 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 12-96. 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 12-96. 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 12-96. 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

12.15.13 McBSP Interrupt Enable Register

Figure 12-82. 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 12-97. 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.

12.16 Register to Driverlib Function Mapping

Controller Area Network (CAN)

The enhanced Controller Area Network (eCAN) module 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 a time−stamping feature, the eCAN module provides a versatile and robust serial communication interface. Refer to the*C2000 Real-Time Control Peripheral Reference Guide* for a list of devices with the eCAN module. Some devices have a second CAN module, eCAN-B. The word eCAN is generically used to refer to the CAN modules. The specific module reference (A or B) is used where appropriate. For a given eCAN module, the same address space is used for the module registers in all applicable 28xx /28xxx devices. Refer to *Programming Examples for the TMS320x28xx eCAN* that provides many useful examples that illustrate how to program the eCAN module.

Topic **Page**

13.1 CAN Overview

13.1.1 Features

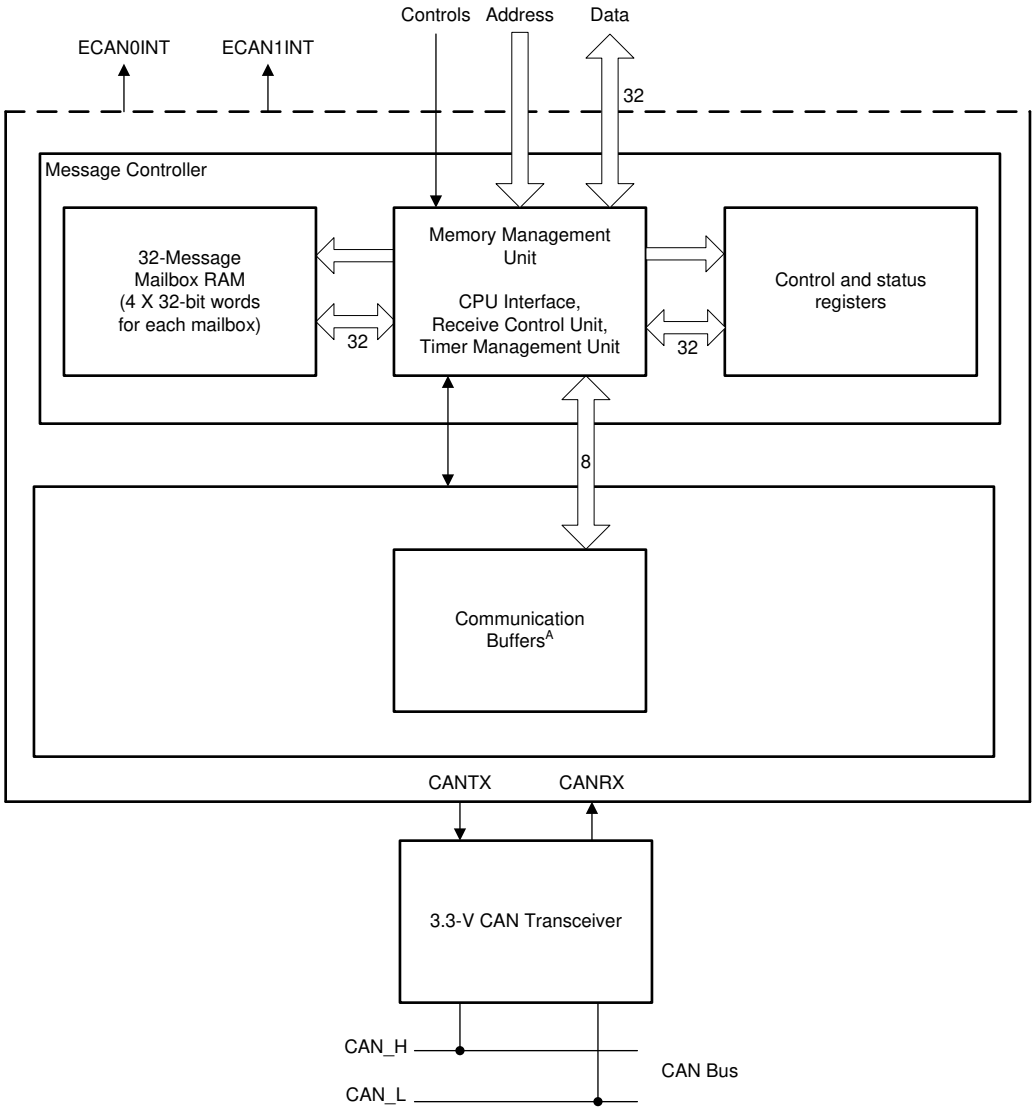
The eCAN module has the following features:

- 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 self-generated acknowledge is provided, thereby eliminating the need for another node to provide the acknowledge bit.

13.1.2 Block Diagram

Figure 13-1 shows the block diagram of eCAN.

Figure 13-1. eCAN Block Diagram and Interface Circuit



A The communication buffers are transparent to the user and are not accessible by user code.

13.2 eCAN Compatibility With Other TI CAN Modules

The eCAN module is identical to the High-end CAN Controller (HECC) used in the TMS470™ series of 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, and so on) over the CAN module featured in the LF240xA™ series of devices. For this reason, code written for LF240xA CAN modules cannot be directly ported to eCAN. However, eCAN follows the same register bit-layout structure and bit functionality as that of LF240xA CAN (for registers that exist in both devices) that is, 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.

13.3 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 high level of reliability, and a communication rate of up to 1 Mbps. The CAN bus is ideal for applications operating in electrically noisy 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.

13.3.1 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 node(s).
- 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 upon detecting an error condition.
- Overload frames that provide an extra delay between the preceding and the succeeding data frames or remote frames.

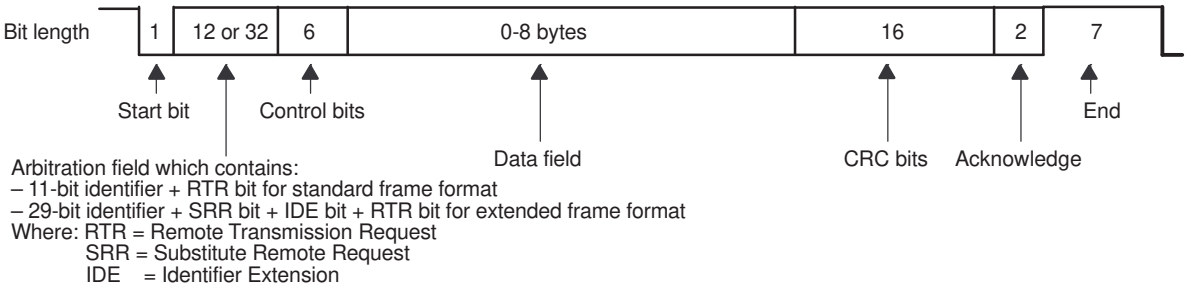
In addition, CAN specification version 2.0B defines two different formats that differ in the length of the identifier field: standard frames with an 11-bit identifier and extended frames with 29-bit identifier.

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 or extended data frames, along with their position as shown in Figure 13-2 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 13-2. CAN Data Frame

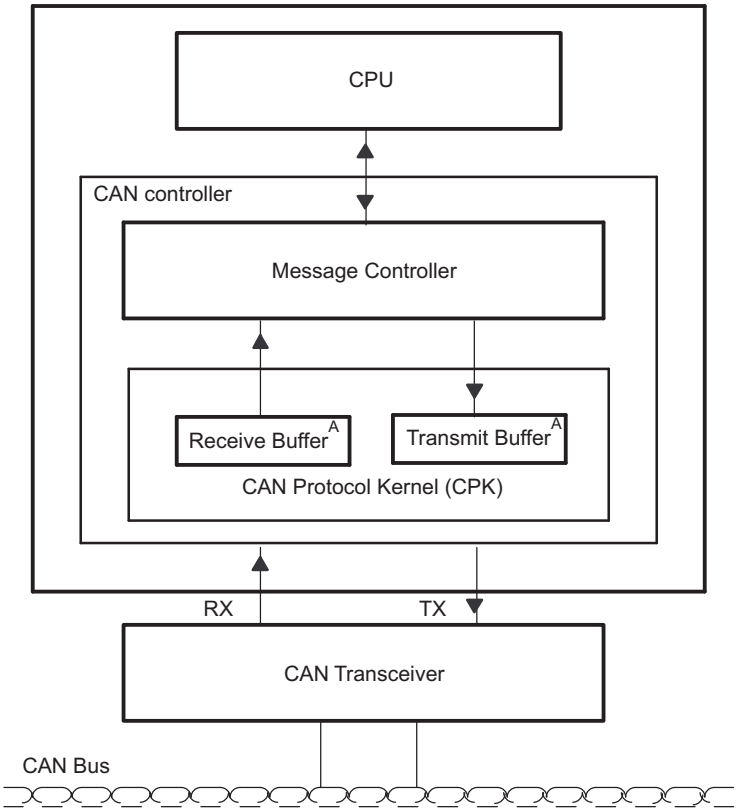


Note: Unless otherwise noted, numbers are amount of bits in field.

The eCAN controller provides the CPU with full functionality of the CAN protocol. 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 13-3, is composed of a CAN protocol kernel (CPK) and a message controller.

Figure 13-3. Architecture of the eCAN Module



A The receive and transmit buffers are transparent to the user and are not accessible by user code.

One function of the CPK is to decode all messages received on the CAN bus and to transfer these messages into a receive buffer. Another 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 (that is, copied into the mailbox RAM) 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.

13.4 eCAN Controller Overview

The eCAN has 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) must be set. 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.

13.4.1 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).

13.4.2 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 13-4, 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.

13.4.2.1 32-bit Access to Control and Status Registers

As indicated in Section 13.4.2, 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 C2000Ware 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 and writes:

Example 13-1. 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 or bits 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. It is good practice to initialize the shadow registers to zero before step 1.

Example 13-2. 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 13-4. eCAN-A Memory Map

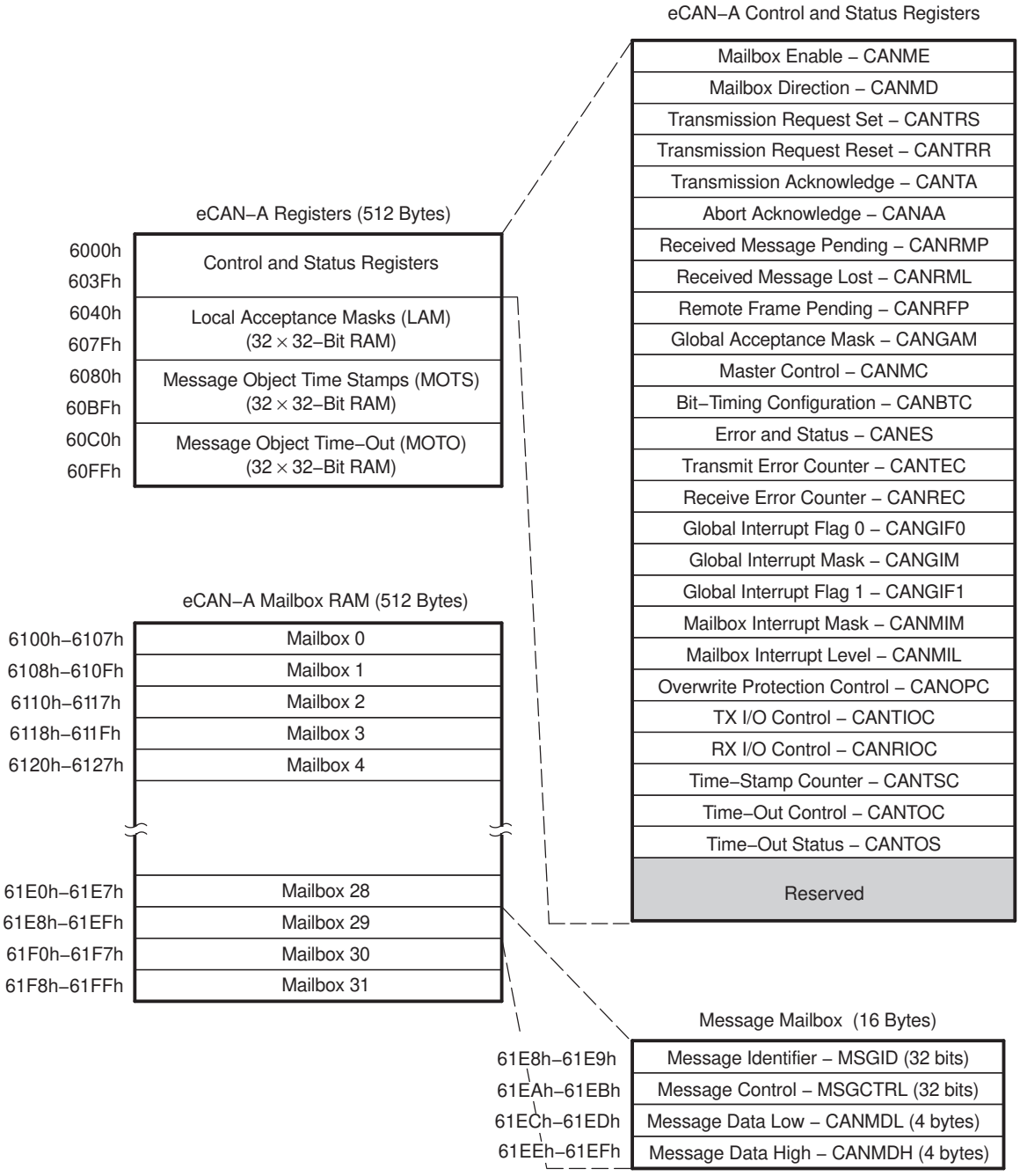
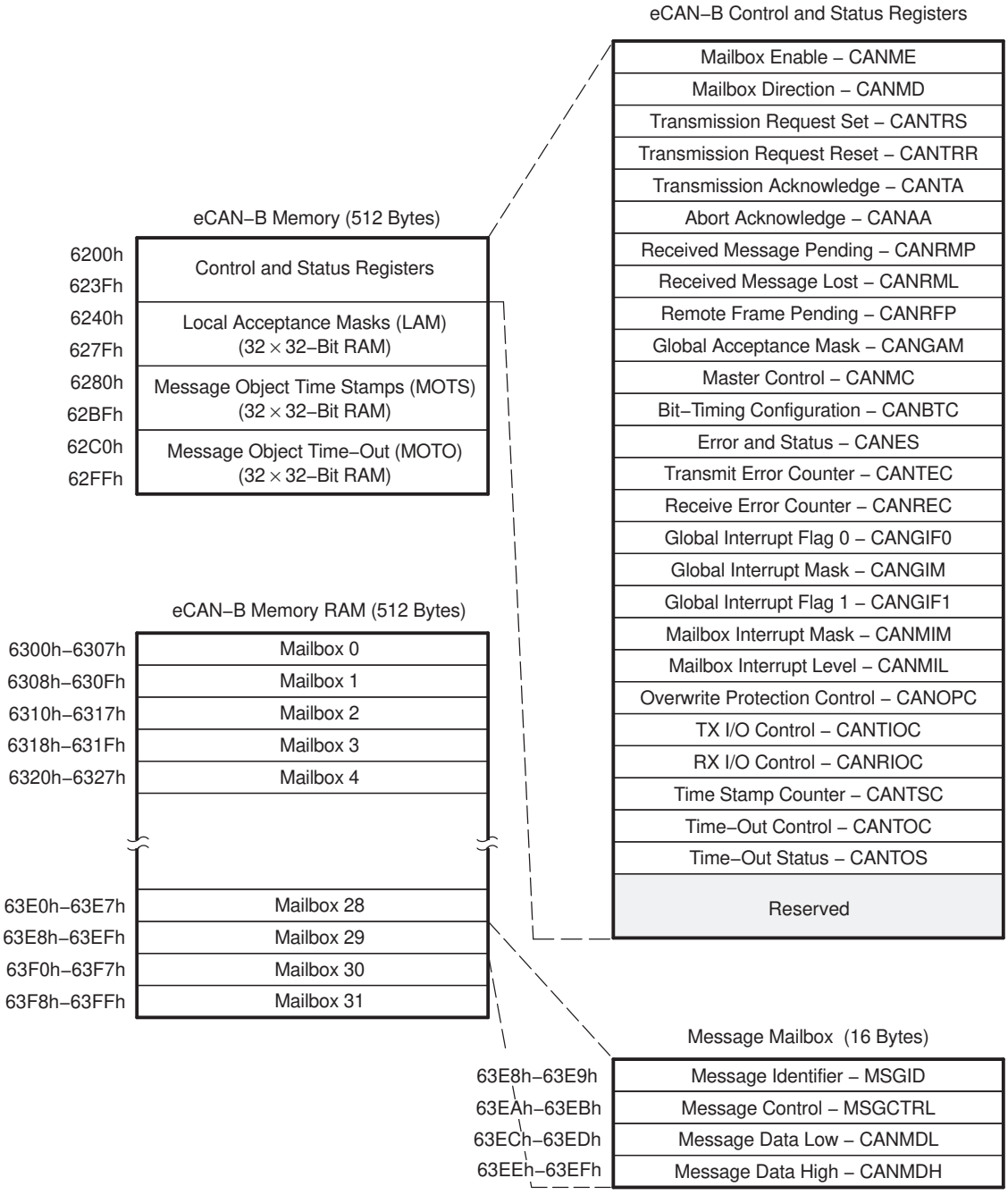


Figure 13-5. eCAN-B Memory Map



13.5 Message Objects

The eCAN module has 32 message objects (mailboxes).

Each message object can be configured to either transmit or receive. Each message object has its own acceptance mask (in the eCAN mode; not in SCC mode).

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.

13.6 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 (see Table 13-1). 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 13-2 lists the mailbox RAM layout.

Table 13-1. 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
Remote-Request message object	1	0	1
Auto-Reply message object	0	1	0

Table 13-2. 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 13-3. 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 13-4. eCAN-B Mailbox RAM Layout

MB	MSGID MSGIDL-MSGIDH	MSGCTRL MSGCTRL - Rsvd	CANMDL CANMDL_L - CANMDL_H	CANMDH CANMDH_L - CANMDH_H
0	6300-6301h	6302-6303h	6304-6305h	6306-6307h
1	6308-6309h	630A-630Bh	630C-630Dh	630E-630Fh
2	6310-6311h	6312-6313h	6314-6315h	6316-6317h
3	6318-6319h	631A-631Bh	631C-631Dh	631E-631Fh
4	6320-6321h	6322-6323h	6324-6325h	6326-6327h
5	6328-6329h	632A-632Bh	632C-632Dh	632E-632Fh
6	6330-6331h	6332-6333h	6334-6335h	6336-6337h
7	6338-6339h	633A-633Bh	633C-633Dh	633E-633Fh
8	6340-6341h	6342-6343h	6344-6345h	6346-6347h
9	6348-6349h	634A-634Bh	634C-634Dh	634E-634Fh
10	6350-6351h	6352-6353h	6354-6355h	6356-6357h
11	6358-6359h	635A-635Bh	635C-635Dh	635E-635Fh
12	6360-6361h	6362-6363h	6364-6365h	6366-6367h
13	6368-6369h	636A-636Bh	636C-636Dh	636E-636Fh
14	6370-6371h	6372-6373h	6374-6375h	6376-6377h
15	6378-6379h	637A-637Bh	637C-637Dh	637E-637Fh
16	6380-6381h	6382-6383h	6384-6385h	6386-6387h
17	6388-6389h	638A-638Bh	638C-638Dh	638E-638Fh
18	6390-6391h	6392-6393h	6394-6395h	6396-6397h
19	6398-6399h	639A-639Bh	639C-639Dh	639E-639Fh
20	63A0-63A1h	63A2-63A3h	63A4-63A5h	63A6-63A7h
21	63A8-63A9h	63AA-63ABh	63AC-63ADh	63AE-63AFh
22	63B0-63B1h	63B2-63B3h	63B4-63B5h	63B6-63B7h
23	63B8-63B9h	63BA-63BBh	63BC-63BDh	63BE-63BFh
24	63C0-63C1h	63C2-63C3h	63C4-63C5h	63C6-63C7h
25	63C8-63C9h	63CA-63CBh	63CC-63CDh	63CE-63CFh
26	63D0-63D1h	63D2-63D3h	63D4-63D5h	63D6-63D7h
27	63D8-63D9h	63DA-63DBh	63DC-63DDh	63DE-63DFh
28	63E0-63E1h	63E2-63E3h	63E4-63E5h	63E6-63E7h
29	63E8-63E9h	63EA-63EBh	63EC-63EDh	63EE-63EFh
30	63F0-63F1h	63F2-63F3h	63F4-63F5h	63F6-63F7h
31	63F8-63F9h	63FA-63FBh	63FC-63FDh	63FE-63FFh

Table 13-5. Addresses of LAM, MOTS, and MOTO Registers for Mailboxes (eCAN-B)

Mailbox	LAM	MOTS	MOT0
0	6240h−6241h	6280h−6281h	62C0h−62C1h
1	6242h− 6243h	6282h−6283h	62C2h−62C3h
2	6244h− 6245h	6284h−6285h	62C4h−62C5h
3	6246h−6247h	6286h−6287h	62C6h−62C7h
4	6248h−6249h	6288h−6289h	62C8h−62C9h
5	624Ah−624Bh	628Ah−628Bh	62CAh−62CBh
6	624Ch−624Dh	628Ch−628Dh	62CCh−62CDh
7	624Eh−624Fh	628Eh−628Fh	62CEh−62CFh
8	6250h−6251h	6290h−6291h	62D0h−62D1h
9	6252h−6253h	6292h−6293h	62D2h−62D3h
10	6254h−6255h	6294h−6295h	62D4h−62D5h
11	6256h−6257h	6296h−6297h	62D6h−62D7h
12	6258h−6259h	6298h−6299h	62D8h−62D9h
13	625Ah−625Bh	629Ah−629Bh	62DAh−62DBh
14	625Ch−625Dh	629Ch−629Dh	62DCh−62DDh
15	625Eh−625Fh	629Eh−629Fh	62DEh−62DFh
16	6260h−6261h	62A0h−62A1h	62E0h−62E1h
17	6262h−6263h	62A2h−62A3h	62E2h−62E3h
18	6264h−6265h	62A4h−62A5h	62E4h−62E5h
19	6266h−6267h	62A6h−62A7h	62E6h−62E7h
20	6268h−6269h	62A8h−62A9h	62E8h−62E9h
21	626Ah−626Bh	62AAh−62ABh	62EAh−62EBh
22	626Ch−626Dh	62ACh−62ADh	62ECh−62EDh
23	626Eh−626Fh	62AEh−62AFh	62EEh−62EFh
24	6270h−6271h	62B0h−62B1h	62F0h−62F1h
25	6272h−6273h	62B2h−62B3h	62F2h−62F3h
26	6274h−6275h	62B4h−62B5h	62F4h−62F5h
27	6276h−6277h	62B6h−62B7h	62F6h−62F7h
28	6278h−6279h	62B8h−62B9h	62F8h−62F9h
29	627Ah−627Bh	62BAh−62BBh	62FAh−62FBh
30	627Ch−627Dh	62BCh−62BDh	62FCh−62FDh
31	627Eh−627Fh	62BEh−62BFh	62FEh−62FFh

13.6.1 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. If the TRS bit of a higher-priority (determined either by the MBX number or by the associated TPL value) mailbox had been set before the message in the transmit-buffer has lost arbitration, the transmit-buffer contents will be replaced with that of the higher-priority mailbox and the higher priority mailbox will be transmitted after the arbitration loss. However, if that TRS bit was set after the message in the transmit-buffer has lost arbitration, the higher priority MBX will be transmitted only after the current message in the transmit-buffer has been transmitted.

13.6.2 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.

13.6.3 CAN Module Operation in Normal Configuration

If the CAN module is being used in normal configuration (that is, 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 into the mailbox RAM 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 the eCAN module.

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).

13.7 eCAN Configuration

This section explains the process of initialization and describes the procedures to configure the eCAN module.

13.7.1 CAN Module Initialization

The CAN module must be initialized before the utilization. Initialization is only possible if the module is in initialization mode. Figure 13-6 is a flow chart showing the process.

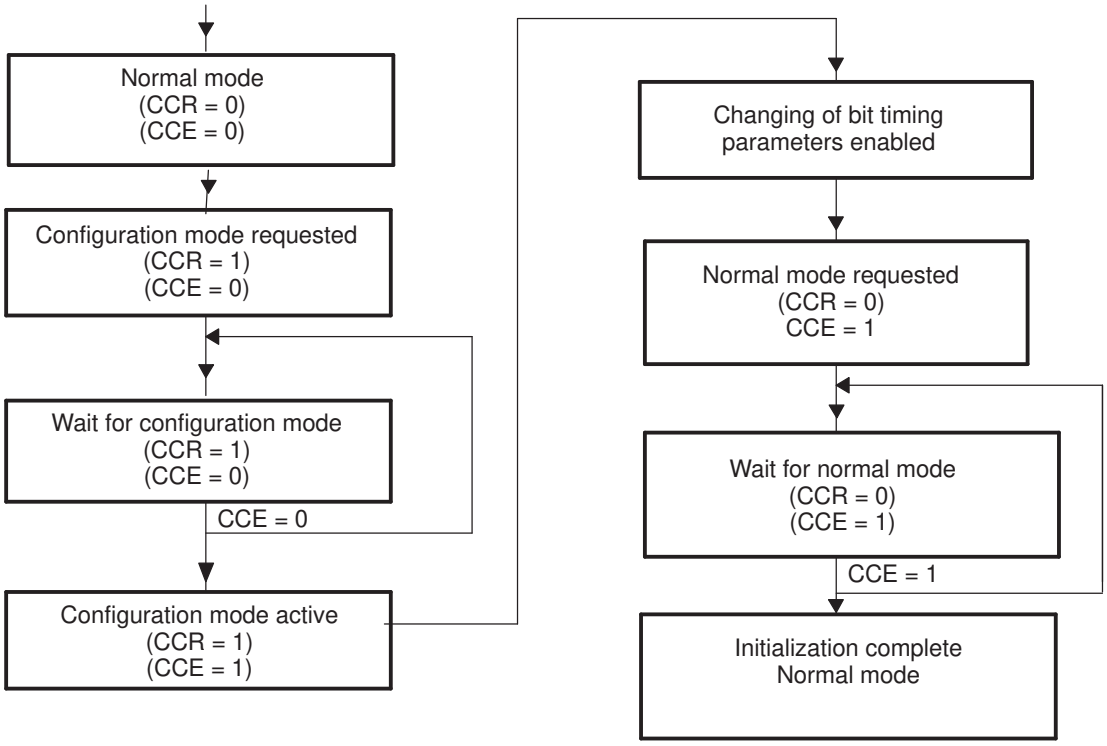
Programming CCR (CANMC.12) = 1 sets the initialization mode. The initialization can be performed only when CCE (CANES.4) = 1. Afterwards, the configuration registers can be written.

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, that is CCE (CANES.4) bit remains at 1 when clearing the CCR bit.

Figure 13-6. 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.

13.7.1.1 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).

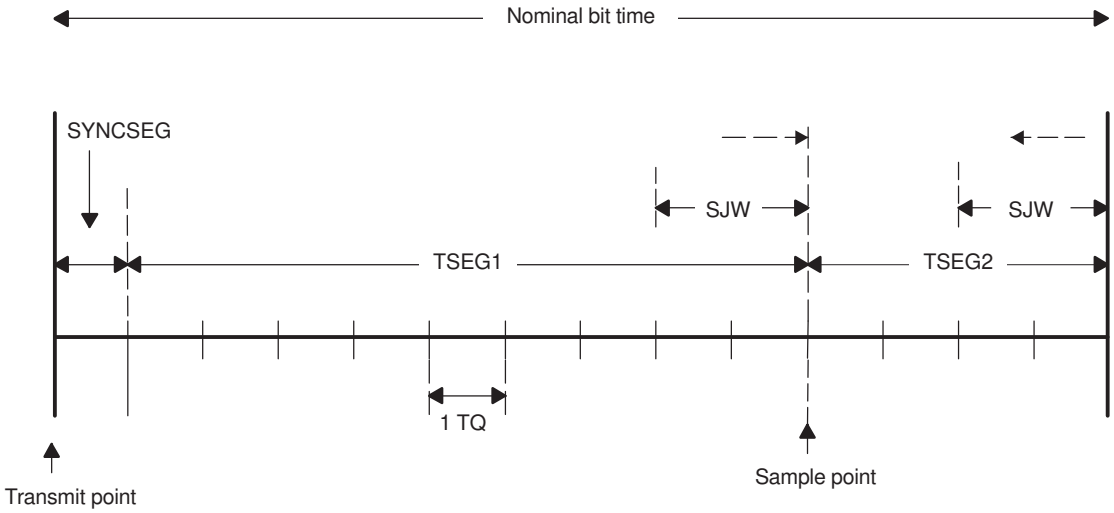
TSEG1 combines the two time segments PROP_SEG and PHASE_SEG1 as defined by the CAN protocol. TSEG2 defines the length of the time segment PHASE_SEG2.

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 13-7. CAN Bit Timing



A TSEG1 can be lengthened or TSEG2 shortened by the SJW

13.7.1.2 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 device 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 13-7.

Bit-time = TSEG1 + TSEG2 + 1

13.7.1.3 Bit Configuration Parameters for 75 -MHz CAN Clock

This section provides example values for the CANBTC bit fields for some bit rates and sampling points. Note that these values are for illustrative purposes only. In a real-world application, parameters such as the oscillator accuracy and 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 13-6 shows how the BRP_{reg} field may be changed to achieve different bit rates with a BT of 15 for a 80% SP.

Table 13-6. 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 = 5	15 MHz
500 kbps	BRP_{reg} +1 = 10	7.5 MHz
250 kbps	BRP_{reg} +1 = 20	3.75 MHz
125 kbps	BRP_{reg} +1 = 40	1.875 MHz
100 kbps	BRP_{reg} +1 = 50	1.5 MHz
50 kbps	BRP_{reg} +1 = 100	0.75 MHz

Table 13-7 shows how to achieve different sampling points with a BT of 15.

Table 13-7. 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%

NOTE: For a SYSCLKOUT of 150 MHz, the lowest bit-rate that can be achieved is 11.719 kbps.

13.7.1.4 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, 7:6]
- CANBTC
- CANGIM
- CANMIM[31..0]
- CANTSC[31..0]
- CANTIOC[3]
- CANRIOC[3]

13.7.2 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:

1. Clear CCR (CANMC.12) = 0
2. Clear PDR (CANMC.11) = 0
3. Clear DBO (CANMC.10) = 0
4. Clear WUBA (CANMC.9)= 0
5. Clear CDR (CANMC.8) = 0
6. Clear ABO (CANMC.7) = 0
7. Clear STM (CANMC.6) = 0
8. Clear SRES (CANMC.5) = 0
9. 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.

13.7.2.1 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.1 = 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:
   1. Write MSGID(1) = 0x15AC0000
   2. 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).
   3. Set the mailbox direction by clearing the corresponding bit in the CANMD register.
   4. 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.

13.7.2.2 Transmitting a Message

To start a transmission (in this example, for mailbox1):

1. Write the message data into the mailbox data field.
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.

13.7.2.3 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.

13.7.2.4 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).

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. 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.

13.7.2.5 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 signaling 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 (that is, more than one message). In this case, all data needed for the message can be collected in the mailboxes and be read at once.

13.7.3 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.

13.7.3.1 Requesting Data From Another Node

In this case the mailbox is configured as receive mailbox as mentioned above. To request data from another node, the CPU needs to do the following:

1. Set the RTR bit in the message control field register (CANMSGCTRL) to 1.
2. Write the correct identifier into the message identifier register (MSGID).
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.
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.
5. Read the received message.

13.7.3.2 Answering a Remote Request

To answer a remote request, the following is needed:

1. Configure the object as a transmit mailbox.
2. Set the auto answer mode (AAM) (MSGID.29) bit in the MSGID register before the mailbox is enabled.
3. Update the data field.
   MDL, MDH(1) = xxxxxxxxh
4. Enable the mailbox by setting the CANME bit to 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.

13.7.3.3 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) (CANMC.8) bit and the mailbox number (MBNR) of that object in the master control register (CANMC). This signals the CAN module that the CPU wants to change the data field. For example, for object 1:
   Write CANMC = 0x0000101
2. Write the message data into the mailbox data register. For example:
   Write CANMDL(1) = xxxx0000h
3. Clear the CDR bit (CANMC.8) to enable the object.
   Write CANMC = 0x00000000

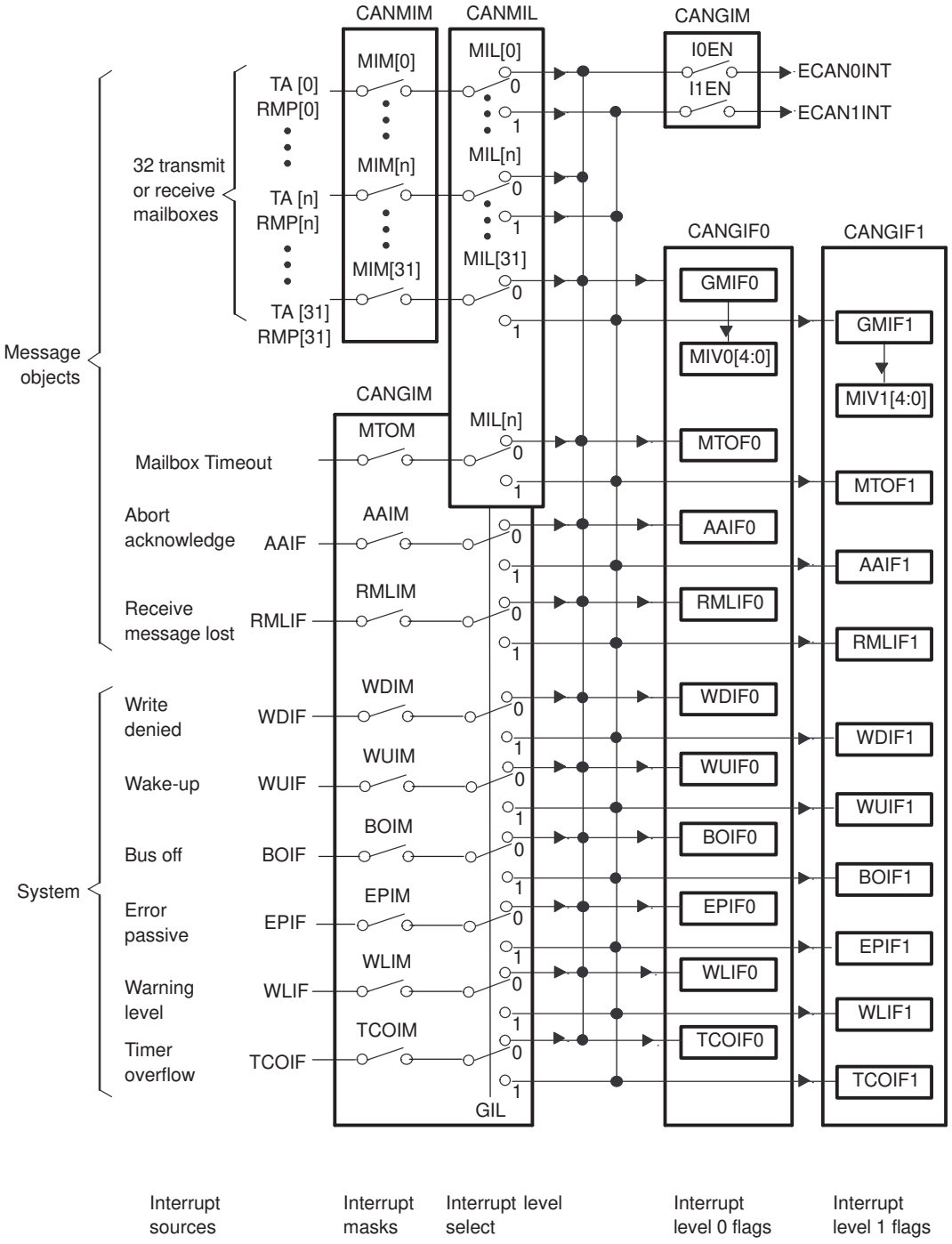
13.7.4 Interrupts

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 13-8.

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 13-8. Interrupts Scheme



13.7.4.1 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.

13.7.4.2 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 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.

13.7.4.3 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 13-8. This also releases the interrupt line if no other interrupt is pending.

Table 13-8. 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.

13.7.4.3.1 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 or reception 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 should always be set (enabling these interrupts). The GIL bit (CANGIM.2) can be cleared to have the global interrupts on another level than the mailbox interrupts. Both the I1EN (CANGIM.1) and I0EN (CANGIM.0) flags should be set to enable both interrupt lines. The flag RMLIM (CANGIM.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.

13.7.4.3.2 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.

13.7.4.3.3 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 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.

13.7.5 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.

13.7.5.1 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 (that is, 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.

13.7.5.2 Precautions for Entering and Exiting Device Low-Power Modes (LPM)

The 28x device 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.

13.7.5.3 Enabling or 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 PCLKCR0 register for eCAN-A module and bit 15 of PCLKCR0 register for eCAN-B module. 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.

13.7.5.4 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

13.8 eCAN Registers

This chapter contains the registers and bit descriptions.

Figure 13-9. Mailbox-Enable Register (CANME)

31	0
CANME[31:0]	
R/W-0	

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

13.8.1 Mailbox Enable Register (CANME)

This register is used to enable/disable individual mailboxes.

Figure 13-10. Mailbox-Enable Register (CANME)

31	0
CANME[31:0]	
R/W-0	

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

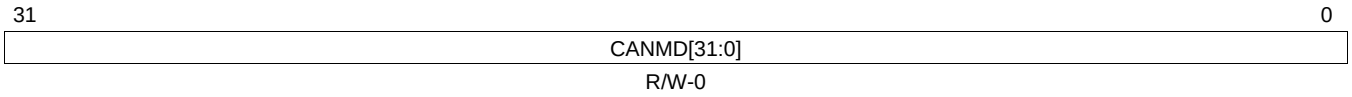
Table 13-9. Mailbox-Enable Register (CANME) Field Descriptions

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. 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 and an interrupt (write-denied interrupt) generated, if enabled.
		0	The corresponding mailbox RAM area is disabled for the eCAN; however, it is accessible to the CPU as normal RAM.

13.8.2 Mailbox-Direction Register (CANMD)

This register is used to configure a mailbox for transmit or receive operation.

Figure 13-11. Mailbox-Direction Register (CANMD)



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

Table 13-10. 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.

13.8.3 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 13-12. Transmission-Request Set Register (CANTRS)

31	0
TRS[31:0]	
RS-0	

LEGEND: RS = Read/Set; -*n* = value after reset

Table 13-11. Transmission-Request Set Register (CANTRS) Field Descriptions

Bit	Field	Value	Description
31:0	TRS[31:0]		Transmit-request-set bits
		1	Setting TRS*n* commences the transmission of the message in that mailbox. Several bits can be set simultaneously with all messages transmitted in turn.
		0	No operation

13.8.4 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 13-13. Transmission-Request-Reset Register (CANTRR)

31	0
TRR[31:0]	
RS-0	

LEGEND: RS = Read/Set; -*n* = value after reset

Table 13-12. 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

13.8.5 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 (CANGIF0.15/CANGIF1.15) 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 13-14. Transmission-Acknowledge Register (CANTA)

31	0
TA[31:0]	
RC-0	

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 13-13. 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.

13.8.6 Abort-Acknowledge Register (CANAA)

If the transmission of the message in mailbox *n* was aborted, the bit AA[*n*] is set and the AAIF (CANGIF.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 13-15. Abort-Acknowledge Register (CANAA)

31	0
AA[31:0]	
RC-0	

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 13-14. Abort-Acknowledge Register (CANAA) Field Descriptions

Bit	Field	Value	Description
31:0	AA[31:0]		Abort-acknowledge bits
		1	If the transmission of the message in mailbox *n* is aborted, the bit *n* of this register is set.
		0	The transmission is not aborted.

13.8.7 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 13-16. Received-Message-Pending Register (CANRMP)

31	0
RMP[31:0]	
RC-0	

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 13-15. Received-Message-Pending Register (CANRMP) Field Descriptions

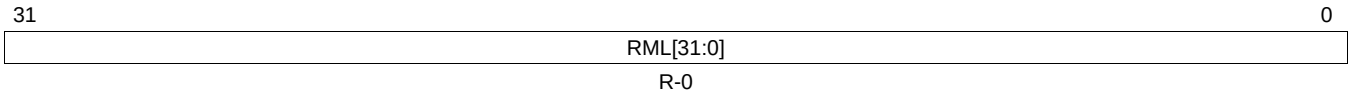
Bit	Field	Value	Description
31:0	RMP[31:0]		Received-message-pending bits
		1	If mailbox *n* contains a received message, bit RMP[*n*] of this register is set.
		0	The mailbox does not contain a message.

13.8.8 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 13-17. Received-Message-Lost Register (CANRML)



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

Table 13-16. 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.

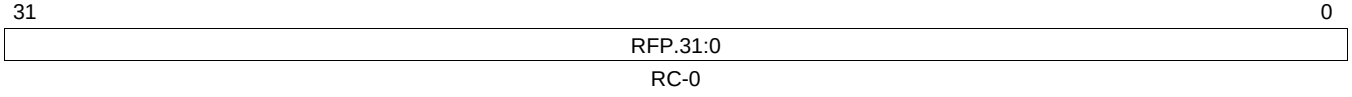
13.8.9 Remote-Frame-Pending Register (CANRFP)

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 13-18. Remote-Frame-Pending Register (CANRFP)



LEGEND: RC = Read/Clear; -*n* = value after reset

Table 13-17. 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 not 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.

13.8.9.1 Handling of Remote Frames

If a remote frame is received (the incoming message has the RTR bit set), 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 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 13.8.7.

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 (CDR) bit [CANMC.8] first. The CPU can then write the new data and clear the CDR bit to indicate to 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 remote-request message object is able to transmit a remote request frame and to wait for the corresponding data frame.
4. A auto-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.

13.8.10 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 13-19. 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 13-18. Global Acceptance Mask Register (CANGAM) Field Descriptions

Bit	Field	Value	Description
31	AMI		Global 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.
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. A value of "0" for a bit position means received identifier bit value must match the corresponding identifier bit of the MSGID register. A value of "1" means "don't care".

13.8.11 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 13-20. 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 13-19. 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 EALLOW protected.
		1	Select eCAN mode.
		0	The eCAN is in SCC mode. Only mailboxes 15 to 0 can be used. Timestamping feature is not available.
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 configure 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.

Table 13-19. Master Control Register (CANMC) Field Descriptions (continued)

Bit	Field	Value	Description
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.
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 (CANMC.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 bit 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	0 - 31	Mailbox for which the CPU requests a write access to the data field. This field is used in conjunction with the CDR bit. The bit MBNR.4 is for eCAN only, and is reserved in the SCC mode.

13.8.11.1 CAN Module Action in SUSPEND

The following points describe the behavior of the module in SUSPEND mode.

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, that is, 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.

13.8.12 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 13.7.1).

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 13.7.1.1.

Figure 13-21. 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 13-20. 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: $$TQ = \frac{1}{SYSCLKOUT/2} \times \left(BRP_{reg} + 1\right)$$ where SYSCLKOUT is the frequency of the CAN module clock. BRP_{reg} denotes the "register value" of the prescaler; that is, 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.
15:10	Reserved		Reads are undefined. Must be written with all zeroes only.
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;" that is, 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 13-20. Bit-Timing Configuration Register (CANBTC) Field Descriptions (continued)

Bit	Field	Value	Description
6:3	$TSEG1_{reg}$		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;" that is, 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 13.7.1.1.
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;" that is, 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. $TSEG2 = TSEG2_{reg} + 1$

13.8.13 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, that is, 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 13-22. 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 13-21. 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 13-21. 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.

13.8.14 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 13-23. Transmit-Error-Counter Register (CANTEC)

31	8	7	0
Reserved		TEC	
R-x		R-0	

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

Figure 13-24. 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) (CANMC.7) set). All internal flags are reset and the error counters are cleared. After leaving initialization mode, the error counters are cleared.

13.8.15 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.

13.8.15.1 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 13-25. 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 13-26. 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 13-22. 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 13-22. 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.

13.8.15.2 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.

The GMIF has no corresponding bit in the CANGIM because the mailboxes have individual mask bits in the CANMIM register.

Figure 13-27. 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 13-23. 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

Table 13-23. Global Interrupt Mask Register (CANGIM) Field Descriptions (continued)

Bit	Field	Value	Description
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.
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.

13.8.15.3 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 13-28. Mailbox Interrupt Mask Register (CANMIM)

31	0
MIM.31:0	
R/W-0	

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

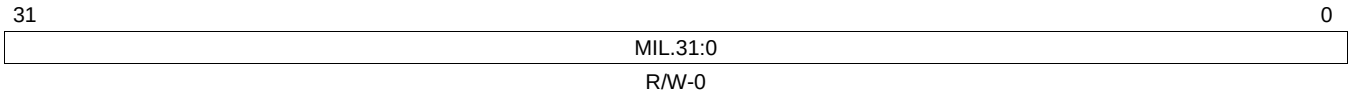
Table 13-24. Mailbox Interrupt Mask Register (CANMIM) Field Descriptions

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.

13.8.15.4 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 13-29. Mailbox Interrupt Level Register (CANMIL)



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

Table 13-25. Mailbox Interrupt Level Register (CANMIL) Field Descriptions

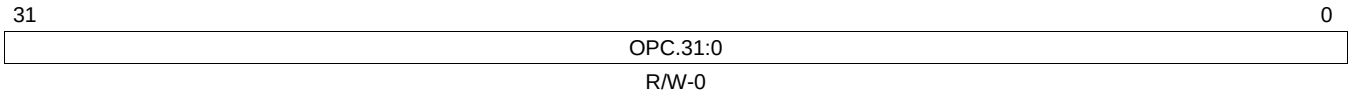
Bit	Field	Value	Description
31:0	MIL.31:0		Mailbox interrupt level. These bits allow any mailbox interrupt level to be selected individually.
		1	The mailbox interrupt is generated on interrupt line 1.
		0	The mailbox interrupt is generated on interrupt line 0.

13.8.16 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 13-30. Overwrite Protection Control Register (CANOPC)



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

Table 13-26. 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.

13.8.17 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 13-31. 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 13-27. 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 13-32. 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 13-28. 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

13.8.18 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.

13.8.18.1 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.

13.8.18.1.1 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 13-33. 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 13-29. 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 time-out functions.

13.8.18.2 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.

13.8.18.2.1 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 13-34. 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 13-30. 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.

13.8.18.2.2 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 13-35. 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 13-31. Message Object Time Stamp Registers (MOTS) Field Descriptions

Bit	Field	Value	Description
31:0	MOTS31:0		Value of the time stamp counter (TSC) when the message has been actually received or transmitted.

13.8.18.2.3 Time-Out Control Register (CANTOC)

This register controls whether or not time-out functionality is enabled for a given mailbox.

Figure 13-36. Time-Out Control Register (CANTOC)

31	0
TOC31:0	
R/W-0	

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

Table 13-32. 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.

13.8.18.2.4 Time-Out Status Register (CANTOS)

This register holds the status information of mailboxes that have timed out.

Figure 13-37. Time-Out Status Register (CANTOS)

31	0
TOS31:0	
R/C-0	

LEGEND: R/C = Read/Clear; -*n* = value after reset

Table 13-33. 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 (in the case of a transmit mailbox).

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.

13.8.18.3 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 MTOF.*n* bit and TOS.*n* bits are set. Communication is never successful; that is, the frame was never transmitted (or received). An interrupt is asserted. Application should handle this issue as desired and clear TOC.*n* bit which clears TOS.*n* bit which in turn clears the MTOF.*n* bit.
2. Time-out condition occurs. Both MTOF.*n* bit and TOS.*n* bits are set. However, communication is eventually successful; that is, the frame gets transmitted (or received). Both MTOF.*n* 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. This is handled per the application requirements.
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 CANES 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.

13.8.19 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

13.8.19.1 Message Identifier Register (MSGID)

This register contains the message ID and other control bits for a given mailbox.

Figure 13-38. 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). The reset-state of IDE, AME and AAM bits are undefined. As part of module initialization, these bits must be initialized as appropriate. Otherwise, they may assume random values and lead to improper operation of the mailboxes.

Table 13-34. Message Identifier Register (MSGID) Field Descriptions

Bit	Field	Value	Description
31	IDE		Identifier extension bit.
		1	Receive Mailbox -> The received message has an extended identifier (29 bits). Transmit Mailbox-> The message to be sent has an extended identifier (29 bits).
		0	Receive Mailbox -> The received message has a standard identifier (11 bits). Transmit Mailbox-> The message to be sent has a standard identifier (11 bits).
30	AME		Acceptance mask enable bit. AME is only used for receive mailboxes. 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.
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.

13.8.19.2 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 (CANMC.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.

13.8.19.3 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 13-39. 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 13-35. 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.

13.9 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 (CANMC.10) determines the ordering of stored data. The data is transmitted or received from the CAN bus, starting with byte 0.

- When DBO (CANMC.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 (CANMC.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 (CANMC.8)=1, with MBNR (CANMC.4-0) set to *n*. These settings also apply for a message object configured in reply mode (AAM (MSGID.29)=1).

Figure 13-40. 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 13-41. 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 13-42. 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 13-43. 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.

13.10 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 13-44.

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 13.7.1).

In eCAN mode, each of the 32 mailboxes has its own local-acceptance mask LAM(0) to LAM(31). There is no global-acceptance mask in the eCAN mode.

The selection of the mask to be used for the comparison depends on which mode (SCC or eCAN) is used.

13.10.1 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 eCAN mode, 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 13-44. Local-Acceptance-Mask Register (LAM*n*)

31	30	29	28	16
LAMI	Reserved		LAM*n*[28:16]	
R/W-x	R/W-x		R/W-x	

15	0
LAM*n*[15:0]	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset (x:Undefined)

Table 13-36. 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 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.

External Interface (XINTF)

The external interface (XINTF) is an asynchronous bus that enables access to off-chip memories. This memory access serves to supplement the on-chip FLASH and SARAM storage capacities.

Topic **Page**

14.1 Functional Description

The XINTF is mapped into three fixed memory-mapped zones as defined in Figure 14-1.

Each of the 28x XINTF zones has a chip-select signal that is toggled when an access is made to that particular zone. On some devices the chip-select signals for two zones may be internally ANDed together to form a single shared chip select. In this manner, the same memory is connected to both zones or external decode logic can be used to separate the two.

Each of the three zones can also be programmed with a specified number of wait states, strobe signal set-up and hold timing. The number of wait states, set-up and hold timing is separately specified for a read access and a write access. In addition, each zone can be programmed for extending wait states externally using the XREADY signal or not. The programmable wait-state, chip-select and programmable strobe timing enables glueless interface to external memories and peripherals.

The Set-up, Hold, and Access wait states for each XINTF zone are user-configurable using the associated XTIMINGx registers. The access timing is based on an internal clock called XTIMCLK. XTIMCLK can be set to the same rate as the SYSCLKOUT or to one-half of SYSCLKOUT. The rate of XTIMCLK applies to all of the XINTF zones. XINTF bus cycles begin on the rising edge of XCLKOUT and all timings and events are generated with respect to the rising edge of XTIMCLK.

14.1.1 Differences from the TMS320x281x XINTF

The XINTF described in this chapter is functionally very similar to the TMS320x281x XINTF. The main differences are:

- **Data Bus Width:**

  Each XINTF zone can be configured individually to use a 16-bit or 32-bit data bus. Using the 32-bit mode improves performance since 32 bits of data can be read or written in a single access. The data bus width does not change the size of the XINTF zones or memory reach. In 32-bit mode, the lowest address line XA0 becomes a 2nd write enable. The 281x XINTF is limited to a 16-bit data bus.

- **Address Bus Reach:**

  The address reach has been extended to 20 address lines. Zone 6 and Zone 7 both use the full address reach of 1M x 16 words each. The 281x address reach is 512k x 16 words.

- **Direct Memory Access (DMA):**

  All three XINTF zones are connected to the on-chip DMA module. The DMA can be used to copy code and data to or from the XINTF while the CPU is processing other data. The 281x devices do not include a DMA.

- **XINTF Clock Enable:**

  The XINTF clock (XTIMCLK) is disabled by default to save power. XTIMCLK can be enabled by writing a 1 to bit 12 of the PCLKCR3 register. PCLKCR3 is documented in the device-specific system control and interrupts user's guide. For the F2833x devices, it is *TMS320F2833x System Control and Interrupts Reference Guide* (literature number SPRUFB0). Turning off XTIMCLK does not turn off XCLKOUT. There is a separate control to turn off XCLKOUT. On the 281x, XTIMCLK is always enabled.

- **XINTF Pin MUXing:**

  Many of the XINTF pins are MUXed with general purpose I/O. The GPIO mux registers must be configured for XINTF operation before using the XINTF. On the 281x, the XINTF has dedicated pins.

- **Number of Zones and Chip Select Signals:**

  The number of XINTF zones has been reduced to 3: Zone 0, Zone 6, and Zone 7. Each of these zones has a dedicated chip select signal. Zone 0 is still read-followed-by write protected as described in Section 14.1.4. On the 2812 devices, some zone chip-select signals are shared between zones. Zone 0 and Zone 1 share XZCS0AND1, and Zone 6 and Zone 7 share XZCS6AND7.

- **Zone 7 Memory Mapping:**

  Zone 7 is always mapped. On the 281x devices, the MPNMC input signal determines if Zone 7 is mapped. Zone 6 and 7 do not share any locations. On 281x devices, Zone 7 is mirrored within Zone 6.

- **Zone Memory Map Locations:**

  Zone 0 starts at address 0x4000 and is 4K x 16. On 281x devices, Zone 0 starts at address 0x2000 and is 8K x 16. Zone 6 and 7 are both 1M x 16 and start at 0x100000 and 0x200000, respectively. On 281x devices, these two zones are 512K x 16 and 16K x 16.

- **EALLOW protection:**

  The XINTF registers are now EALLOW protected. On 281x devices, the XINTF registers were not EALLOW protected.

Refer to the latest data manual for the device for timing information.

14.1.2 Differences from the TMS320x2834x XINTF

The XINTF described in this chapter is functionally very similar to the TMS320x2834x XINTF. The main differences are:

- **XA0 and $\overline{\text{WE1}}$**

  For the F2833x/F2823x devices, XA0 and $\overline{\text{WE1}}$ share a single pin; however, for the C2834x device, they are separate pins.

- **XBANK Cycle Selection**

  The number of delay cycles must be configured based on the ratio of XTIMCLK and XCLKOUT. Refer to Section 14.5. C2834x device do not have this requirement.

Refer to the latest data manual for the device for timing information.

14.1.3 Accessing XINTF Zones

An XINTF zone is a region in the 28x memory map that is directly connected to the external interface. Figure 14-1 shows zone locations. The memory or peripheral attached to a zone can be accessed directly with the CPU or Code Composer Studio.

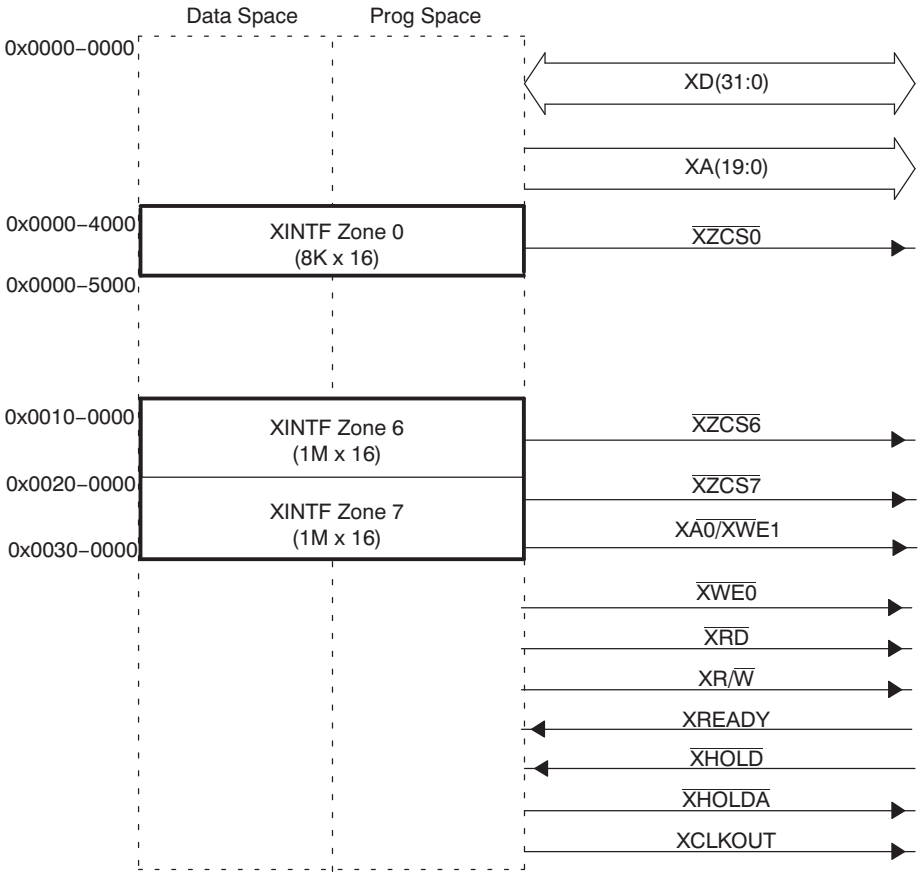
Each XINTF zone can be individually configured with unique read and write access timing and each has an associated zone chip-select signal. This chip-select signal is pulled low so that an access to that zone is currently taking place. On 2833x, 2823x devices , all zone chip select signals are independent.

The external address bus, XA, is 20 bits wide and is shared by all of the zones. What external addresses are generated depends on which zones are being accessed, as follow:

- Zone 0 uses external addresses 0xXX000 - 0xXXFFF. That is, an access to the first location in Zone 0 will issue external addresses 0xXX000 along with chip select 0 (XZCS0). An access to the last location in the zone will issue address 0xXXFFF with XZCS0. If the upper address lines, indicated by 'XX', are configured for XINTF functionality, they will be driven but the value is undefined.

- Zone 6 and 7 both use external addresses 0x00000 - 0xFFFFF. Depending on which zone is accessed, the appropriate zone chip select signal ($\overline{XZCS6}$ or $\overline{XZCS7}$) will also go low.

Figure 14-1. External Interface Block Diagram



A Each zone can be programmed with different wait states, setup and hold timings. A dedicated zone chip select ($\overline{XZCS}$) signal toggles when an access to a particular zone is performed. These features enable glueless connection to many external memories and peripherals.

B Zones 1 − 5 are reserved for future expansion.

C When the XINTF clock is enabled in PCLKCR3, all zones are enabled.

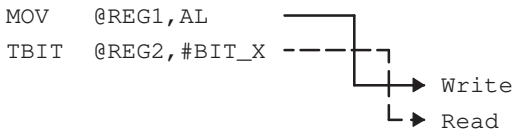
14.1.4 Write-Followed-by-Read Pipeline Protection

In the 28x CPU pipeline, the read phase of an operation occurs before the write phase. Due to this ordering, a write followed by a read access can actually occur in the opposite order: read followed by write.

For example, the following lines of code perform a write to one location followed by a read from another. Due to the 28x CPU pipeline, the read operation will be issued before the write as shown:

```
MOV   @REG1,AL
TBIT  @REG2,#BIT_X  ----> Read
                    ----> Write
```

On 28x devices, regions of memory where peripheral registers are common are protected from this order reversal by hardware. These regions of memory are said to be read-followed-by-write pipeline protected. XINTF Zone 0 is by default read-followed-by-write pipeline protected. Write and read accesses to Zone 0 are executed in the same order that they are written. For example, a write followed by a read is executed in the same order it was written as shown below:



The 28x CPU automatically protects writes followed by reads to the same memory location. The protection mechanism described above is for cases where the address is not the same, but within a given region of protected memory. In this case, the order of execution is preserved by the CPU automatically inserting enough NOP cycles for the write to complete before the read occurs.

This execution ordering becomes a concern only when peripherals are mapped to the XINTF. A write to one register may update status bits in another register. In this case, the write to the first register must finish before the read to the second register takes place. If the write and read operations are performed in the natural pipeline order, the wrong status may be read since the write would happen after the read. This reversal is not a concern when memory is mapped to the XINTF. Thus, Zone 0 would not typically be used to access memory but instead would be used only to access external peripherals.

If other zones are used to access peripherals that require write-followed-by-read instruction order to be preserved the following solutions can be used:

- Add up to 3 NOP assembly instructions between a write and read instructions. Fewer than three can be used if the code is analyzed and it is found that the pipeline stalls for other reasons.
- Move other instructions before the read to make sure that the write and read are at least three CPU cycles apart.
- Use the -mv compiler option to automatically insert NOP assembly instructions between write and read accesses. This option should be used with caution because this out-of-order execution is a concern only when accessing peripherals mapped to XINTF and not normal memory accesses.

14.2 XINTF Configuration Overview

This section is an overview of the XINTF parameters that can be configured to fit particular system requirements. The exact configuration used depends on the operating frequency of the 28x, switching characteristics of the XINTF, and the timing requirements of the external devices. Detailed information on each of these parameters is given in the following sections.

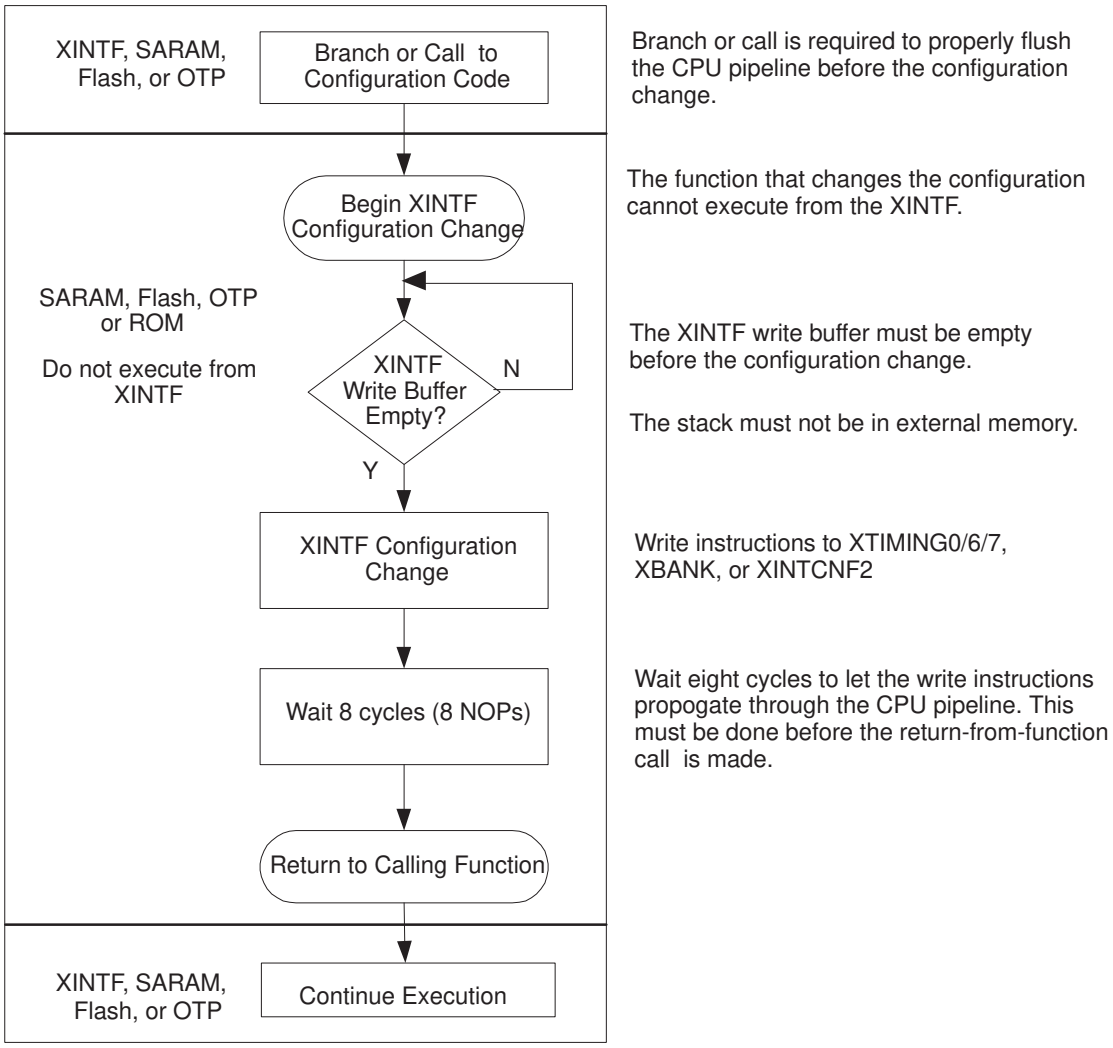
Because a change to the XINTF configuration parameters will cause a change to the access timing, code that configures these parameters should not execute from the XINTF itself.

14.2.1 Procedure to Change the XINTF Configuration and Timing Registers

During an XINTF configuration or timing change no accesses to the XINTF can be in progress. This includes instructions still in the CPU pipeline, write accesses in the XINTF write buffer, data reads or writes, instruction pre-fetch operations and DMA accesses. To be sure that no access takes place during the configuration follow these steps:

1. Make sure that the DMA is not accessing the XINTF.
2. Follow the procedure shown in Figure 14-2 to safely modify the XTIMING0/6/7, XBANK, or XINTCNF2 registers.

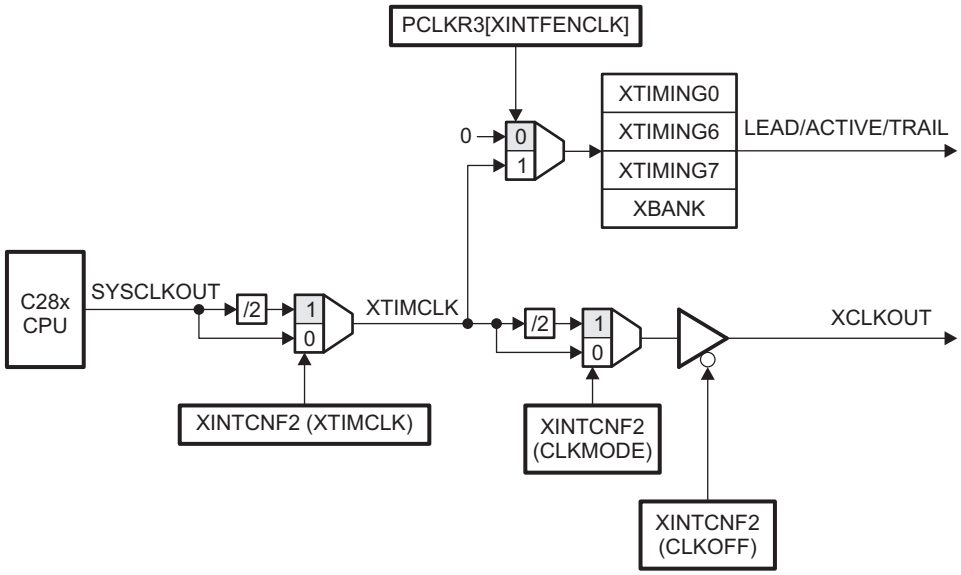
Figure 14-2. Access Flow Diagram



14.2.2 XINTF Clocking

There are two clocks used by the XINTF module: XTIMCLK and XCLKOUT. Figure 14-3 shows the relationship between these two clocks and the CPU clock, SYSCLKOUT.

Figure 14-3. Relationship Between XTIMCLK and SYSCLKOUT



All accesses to all of the XINTF zones are based on the frequency of the internal XINTF clock, XTIMCLK. XTIMCLK can be configured to be either equal or one half of SYSCLKOUT by writing to the XTIMCLK bit in the XINTFCNF2 register. By default XTIMCLK is one-half of SYSCLKOUT.

All XINTF accesses begin on the rising edge of the external clock out, XCLKOUT. In addition, external logic may be clocked off of XCLKOUT. The frequency of XCLKOUT can be configured as a ratio of the internal XINTF clock, XTIMCLK. XCLKOUT can be configured to be either equal or one-half of XTIMCLK by writing to the CLKMODE bit in the XINTFCNF2 register. By default, XCLKOUT is one-half of XTIMCLK, or one-fourth of the CPU clock, SYSCLKOUT.

To reduce system noise, the XCLKOUT output may be disabled. This is done by writing a 1 to the XINTCNF2[CLKOFF] bit.

14.2.3 Write Buffer

By default, write access buffering is disabled. In most cases, to improve performance of the XINTF, write buffering should be enabled. Up to three writes to the XINTF can be buffered without stalling the CPU. The write buffer depth is configured in the XINTCNF2 register.

14.2.4 XINTF Access Lead/Active/Trail Wait-State Timing Per Zone

An XINTF zone is a region of memory-mapped addresses that directly access the external interface. The timing of any read or write access to an XINTF zone can be divided into the following three portions: Lead, Active, and Trail. The number of XTIMCLK cycle wait states for each portion of an access can be configured for each XINTF zone in the corresponding zone XTIMING register. Timing for read accesses can be configured separately from timing for write accesses. In addition, to facilitate connections to slow external devices the X2TIMING bit can be used to double the specified lead/active and trail wait states for a particular zone.

During the lead portion, the chip-select signal for the zone being accessed is taken low and the address is placed on the address bus (XA). The total lead period, in XTIMCLK cycles can be configured in the zone's XTIMING register. By default, the lead period is set to the maximum six XTIMCLK cycles for both read and write accesses.

During the active period, the access to the external device is made. For a read access, the read strobe ($\overline{\text{XRD}}$) is brought low and data is latched into the device. For a write access, the write enable ($\overline{\text{XWE0}}$) strobe is brought low and data is placed on the data bus (XD). If the zone is configured to sample the XREADY signal, the external device can control the XREADY signal to further extend the active period beyond the programmed wait states.

The total active period for any access that does not sample XREADY is 1 XTIMCLK cycle plus the number wait states specified in the corresponding XTIMING register. By default, the active wait states are set to the 14 XTIMCLK cycles for both read and write accesses.

The trail period serves as a hold time in which the chip-select signal remains low but the read and write strobes are brought back high. The total trail period, in XTIMCLK cycles can be configured in the zone's XTIMING register. By default the trail period is set to the maximum six XTIMCLK cycles for both read and write accesses.

Based on system requirements, the lead, active and trail wait state values can be configured to best fit the devices connected to a particular XINTF zone. The following should be considered when selecting the timing parameters:

- Minimum wait state requirements as described in Section 14.4
- The timing characteristics of the XINTF, as described in the device data manual
- The timing requirements of the external device
- Any additional delays between the 28x device and the external device

14.2.5 XREADY Sampling For Each Zone

By sampling XREADY, the external device can extend the active portion of the access. All of the XINTF zones on a device share the same XREADY input signal but each XINTF zone can individually be configured to either sample or ignore the XREADY signal. In addition, the sampling can be specified as synchronous or asynchronous for each zone.

- Synchronous sampling

  If XREADY is sampled synchronously, then the XREADY signal must meet set-up and hold timing relative to one XTIMCLK edge before the end of the active period. That is, XREADY will be sampled one XTIMCLK cycle before the total lead + active cycles specified for the access.
- Asynchronous sampling

  If XREADY is sampled asynchronously, then the XREADY signal must meet set-up and hold timing relative to three XTIMCLK cycles before the end of the active period. That is, XREADY will be sampled three XTIMCLK cycles before the total lead + active cycles specified for the access.

In both the synchronous and asynchronous case if the XREADY sample is found to be low, the active portion of the cycle is extended by one XTIMCLK cycle and XREADY is sampled again during the next XTIMCLK cycle. This pattern continues until XREADY is sampled high at which time the access will complete normally.

If a zone is configured to sample XREADY, then it is done so for both read and write accesses to that zone. By default each XINTF zone is configured to sample XREADY in the asynchronous mode. When using the XREADY signal, the minimum XINTF wait state requirements must be satisfied as described in Section 14.4. The minimum requirements are different when sampling XREADY in the synchronous mode vs the asynchronous mode, depending on the following:

- The timing characteristics of the XINTF, as described in the device data sheet.
- The timing requirements of the external device.
- Any additional delays between the 28x device and the external device.

14.2.6 Bank Switching

When jumping from one XINTF zone to another XINTF zone, a slow device may require extra cycles in order to release the bus in time for another device to gain access. Bank switching allows the XINTF to add extra cycles for any access that crosses into or out of the specified zone. The zone and number of cycles is configured in the XBANK register. The number of cycles must meet the requirements described in Section 14.5.

14.2.7 Zone Data Bus Width

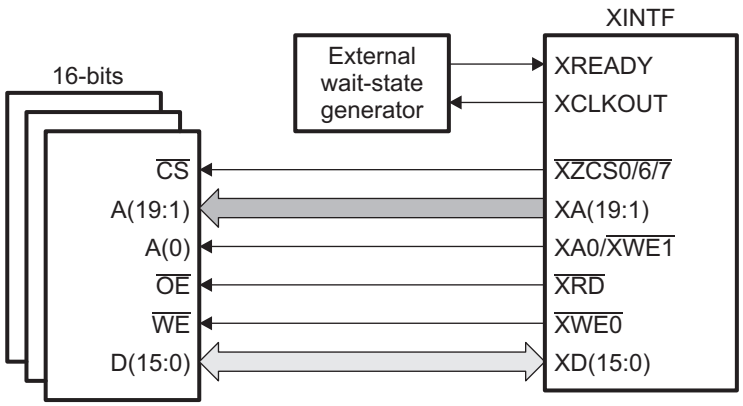
Each XINTF zone can individually be configured for a 16-bit or 32-bit wide data bus. The functionality of the XA0/$\overline{\text{XWE1}}$ signal changes depending on the configuration. When an XINTF zone is configured for 16-bit mode (XTIMINGx[XSIZE] = 3), then the XA0/$\overline{\text{XWE1}}$ signal takes on the role of least-significant address line (XA0). In this case, a typical XINTF bus connection looks as shown in Figure 14-4. The behavior of the $\overline{\text{XWE0}}$ and XA0/$\overline{\text{XWE1}}$ signals is summarized in Table 14-1 and Table 14-2.

If the width of the three zones (configured by XTIMINGx[XSIZE]) are different from each other, and there is a possibility for back-to-back accesses between two zones of different widths, then at least one delay cycle between the zone accesses should be added using the XBANK configuration discussed in Section 14.5. For instance, given the zones are configured as follows:

- Zone 0 configured for 16-bit mode (XTIMING0[XSIZE] = 3)
- Zone 6 configured for 32-bit mode (XTIMING6[XSIZE] = 1)
- Zone 7 configured for 32-bit mode (XTIMING7[XSIZE] = 1)

If there is a possibility for back-to-back accesses between Zone 0 and Zone 6 or Zone 0 and Zone 7, then at least one bank switching delay cycle should be added to Zone 0 (i.e. XBANK[BANK] = 0 and XBANK[BCYC] = 1). See Section 14.5 for configuring XBANK cycles.

Figure 14-4. Typical 16-bit Data Bus XINTF Connections



When an XINTF zone is configured for 32-bit mode (XTIMINGx[XSIZE] = 1), the XA0/$\overline{\text{XWE1}}$ signal is the active low write strobe $\overline{\text{XWE1}}$. $\overline{\text{XWE1}}$ is used, along with $\overline{\text{XWE0}}$ for 32-bit bus operation as shown in Figure 14-5.

Figure 14-5. Typical 32-bit Data Bus XINTF Connections

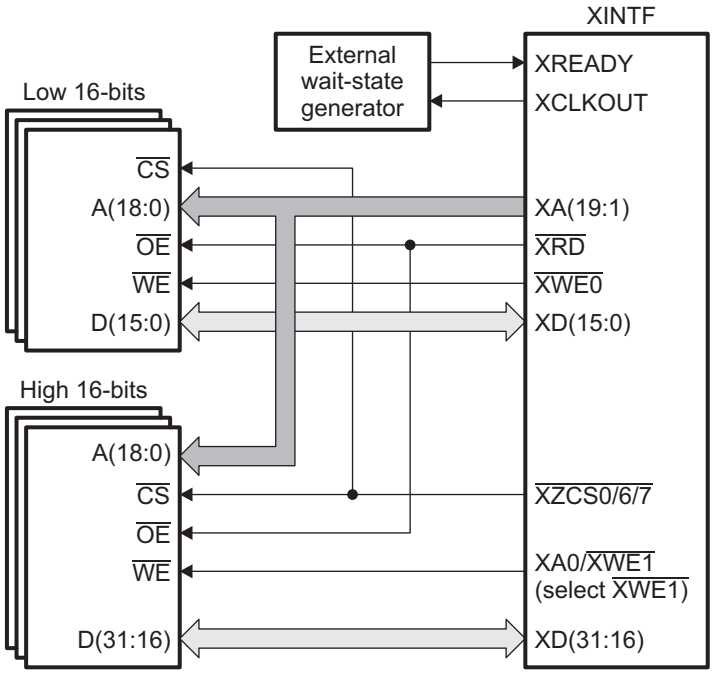


Table 14-1. 16-bit Mode Behavior

16-bit Mode Write Access	XA0/$\overline{\text{XWE1}}$	$\overline{\text{XWE0}}$
no access	1	1
16-bit value at even address	0	0
16-bit value at odd address	1	0

Table 14-2. 32-bit Mode Behavior

32-bit Mode Write Access	XA0/$\overline{\text{XWE1}}$	$\overline{\text{XWE0}}$
no access	1	1
16-bit value at even address	1	0
16-bit value at odd address	0	1
32-bit value	0	0

14.3 External DMA Support (XHOLD, XHOLDA)

The XINTF supports direct memory access (DMA) to its local (off-chip) program and data spaces. This is accomplished with the XHOLD signal input and XHOLDA output. When XHOLD is asserted (low active) a request to the external interface is generated to hold all outputs from the external interface a high impedance state. Upon completion of all outstanding accesses to the external interface, XHOLDA is asserted (low active). XHOLDA signals external devices that the external interface has its outputs in high-impedance state and that another device can control access to external memory or peripherals.

The HOLD Mode bit in XINTCNF2 register enables the automatic generation of a XHOLDA signal and granting access of the external bus, when a valid XHOLD signal is detected. While in HOLD mode, the CPU can continue to execute code from on-chip memory attached to the memory bus. If an attempt is made to access the external interface while XHOLDA is low, a not ready condition is generated, halting the processor. Status bits in the XINTCNF2 register will indicate the state of the XHOLD and XHOLDA signals.

If XHOLD is active, and the CPU attempts a write to the XINTF, the write is not buffered and the CPU will stall. The write buffer is disabled.

The HOLD mode bit in XINTCNF2 register bit will take precedence over the XHOLD input signal. Thus enabling customer code to determine when or not a XHOLD request is to be honored.

The XHOLD input signal is synchronized at the input to the XINTF before any actions are taken. Synchronization is with respect to XTIMCLK.

The HOLDS bit in XINTCNF2 register reflects the current synchronized state of the XHOLD input.

On reset, the HOLD mode bit is enabled, allowing for bootload of external memory using an XHOLD request. If XHOLD signal is active low during reset, the XHOLDA signal is driven low as per normal operation.

During power up, any undefined values in the XHOLD synchronizing latches are ignored and would eventually be flushed out when the clock stabilizes. Hence, synchronizing latches do not need to be reset.

If an XHOLD active low signal is detected, the XHOLDA signal is only driven low after all pending XINTF cycles are completed. Any pending CPU cycles are blocked and the CPU is held in a not-ready state if they are targeted for the XINTF.

Definitions:

Pending XINTF Cycle— Any cycle that is currently in the XINTF FIFO queue.

Pending CPU Cycle— Any cycle that is not in the FIFO queue but is active on the core memory bus.

The XHOLD signal should not be removed until the XHOLDA signal becomes active. Unpredictable results will occur if this rule should be violated.

The state of the XINTF external signals is as follows in HOLD mode:

Signal	HOLD Granted Mode
XA(19:1)	High-impedance
XD(31:0)	High-impedance
XA0/ XWE1	High-impedance
XRD, XWE0, XR/W	High-impedance
XZCS0	High-impedance
XZCS6	High-impedance
XZCS7	High-impedance

14.4 Configuring Lead, Active, and Trail Wait States

XINTF signal timing can be tuned to match specific external device requirements such as setup and hold times for both read and write accesses. The timing parameters can be configured individually for each XINTF zone in the XTIMING registers. Each zone can also be configured to either ignore the XREADY signal or sample it. This allows for optimal efficiency of the XINTF based on the memory or peripheral being accessed.

Table 14-3 shows the relationship between the parameters that can be configured in the XTIMING registers and the duration of the pulse in terms of XTIMCLK cycles, $t_{c(XTIM)}$.

Table 14-3. Pulse Duration in Terms of XTIMCLK Cycles

		Duration (ns) (1) (2)	
	Description	**X2TIMING = 0**	**X2TIMING = 1**
LR	Lead period, read access	XRDLEAD x $t_{c(xtim)}$	(XRDLEADx2) x $t_{c(xtim)}$
AR	Active period, read access	(XRDACTIVE+WS+1) x $t_{c(xtim)}$	(XRDACTIVEx2+WS+1) x $t_{c(xtim)}$
TR	Trail period, read access	XRDTRAIL x $t_{c(xtim)}$	(XRDTRAILx2) x $t_{c(xtim)}$
LW	Lead period, write access	XWRLEAD x $t_{c(xtim)}$	(XWRLEADx2) x $t_{c(xtim)}$
AW	Active period, write access	(XWRACTIVE+WS+1) x $t_{c(xtim)}$	(XWRACTIVEx2+WS+1) x $t_{c(xtim)}$
TW	Trail period, write access	XWRTRAIL x $t_{c(xtim)}$	(XWRTRAILx2) x $t_{c(xtim)}$

(1) $t_{c(xtim)}$ - Cycle time, XTIMCLK

(2) WS refers to the number of wait states inserted by hardware when using XREADY. If the zone is configured to ignore XREADY (USEREADY= 0) then WS = 0.

NOTE: Minimum wait-state configurations must be used for each zone's XTIMING register. These wait-state requirements are in addition to any timing requirements as specified by the device to which it is interfaced. For information on requirements for a particular device, see the data sheet for that device.

No internal device hardware is included to detect illegal settings.

14.4.1 USEREADY = 0

If the XREADY signal is ignored (USEREADY = 0), then the following requirement must be met:

Lead: $LR \geq t_{c(XTIM)}$
$LW \geq t_{c(XTIM)}$

These requirements result in the following XTIMING register configuration restrictions:

XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL	X2TIMING
≥ 1	≥ 0	≥ 0	≥ 1	≥ 0	≥0	0, 1

Examples of valid and invalid timings when not sampling XREADY:

	XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL	X2TIMING
Invalid[1]	0	0	0	0	0	0	0, 1
Valid	1	0	0	1	0	0	0, 1

(1) No hardware to detect illegal XTIMING configurations

14.4.2 Synchronous Mode (USEREADY = 1, READYMODE = 0)

If the XREADY signal is sampled in the synchronous mode (USEREADY = 1, READYMODE = 0), then the following requirements must be met:

1 Lead: $LR \geq \times t_{c(XTIM)}$
$LW \geq t_{c(XTIM)}$
2 Active: $AR \geq 2 \times t_{c(XTIM)}$
$AW \geq 2 \times t_{c(XTIM)}$

NOTE: Restriction does not include external hardware wait states.

These requirements result in the following XTIMING register configuration restrictions:

XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL	X2TIMING
≥ 1	≥ 1	≥ 0	≥ 1	≥ 1	≥ 0	0, 1[1]

(1) No hardware to detect illegal XTIMING configurations

Examples of valid and invalid timings when using synchronous XREADY:

	XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL	X2TIMING
Invalid[1]	0	0	0	0	0	0	0, 1
Invalid[1]	1	0	0	1	0	0	0, 1
Valid	1	1	0	1	1	0	0, 1

(1) No hardware to detect illegal XTIMING configurations

14.4.3 Asynchronous Mode (USEREADY = 1, READYMODE = 1)

If the XREADY signal is sampled in asynchronous mode (USEREADY = 1, READYMODE = 1), then the following requirements must be met:

1	Lead:	LR ≥ × $t_{c(XTIM)}$
		LW ≥ $t_{c(XTIM)}$
2	Active:	AR ≥ 2 × $t_{c(XTIM)}$
		AW ≥ 2 × $t_{c(XTIM)}$
3	Lead + Active:	LR + AR ≥ 4 × $t_{c(XTIM)}$
		LW + AW ≥ 4 × $t_{c(XTIM)}$

These requirements result in the following three possible XTIMING register configurations:

NOTE: Restrictions do not include external hardware wait states.

These requirements result in the following XTIMING register configuration restrictions:

XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL	X2TIMING
≥ 1	≥ 2	0	≥ 1	≥ 2	0	0, 1

or

XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL	X2TIMING
≥ 2	≥ 1	0	≥ 2	≥ 1	0	0, 1

Examples of valid and invalid timings when using asynchronous XREADY:

	XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL	X2TIMING
Invalid[(1)]	0	0	0	0	0	0	0, 1
Invalid[(1)]	1	0	0	1	0	0	0, 1
Invalid[(1)]	1	1	0	1	1	0	0
Valid	1	1	0	1	1	0	1
Valid	1	2	0	1	2	0	0, 1
Valid	2	1	0	2	1	0	0, 1

(1) No hardware to detect illegal XTIMING configurations

Table 14-4 and Table 14-5 show the relationship between Lead/Active/Trail values and the XTIMCLK/X2TIMING modes.

Table 14-4. Relationship Between Lead/Trail Values and the XTIMCLK/X2TIMING Modes

Lead/Trail Value	[XTIMCLK] (1)	X2TIMING (2)	Lead SYSCLKOUT Cycles	Trail SYSCLKOUT Cycles
Formula	0	0	Lead Value * 1	Trail Value * 1
	0	1	Lead Value * 2	Trail Value * 2
	1	0	Lead Value * 2	Trail Value * 2
	1	1	Lead Value * 4	Trail Value * 4
0	X	X	Not a valid value (do not use)	0
1	0	0	1	1
	0	1	2	2
	1	0	2	2
	1	1	4	4
2	0	0	2	2
	0	1	4	4
	1	0	4	4
	1	1	8	8
3	0	0	3	3
	0	1	6	6
	1	0	6	6
	1	1	12	12

(1) XINTCNF2[XTIMCLK] configures the ratio between SYSCLKOUT and XTIMCLK.
(2) X2TIMING is configured per zone in the zone specific XTIMING register.

Table 14-5. Relationship Between Active Values and the XTIMCLK/X2TIMING Modes

Active Value	XTIMCLK [1]	X2TIMING [2]	Total Active SYSCLKOUT Cycles (includes 1 implied active cycle)
Formula	0	0	Active Value * 1 + 1
	0	1	Active Value * 2 + 1
	1	0	Active Value * 2 + 2
	1	1	Active Value * 4 + 2
0	0	X	1 or Invalid if XREADY used (USEREADY = 1)
	1	X	2 or Invalid if XREADY used (USEREADY = 1)
1	0	0	2
	0	1	3
	1	0	4
	1	1	6
2	0	0	3
	0	1	5
	1	0	6
	1	1	10
3	0	0	4
	0	1	7
	1	0	8
	1	1	14
4	0	0	5
	0	1	9
	1	0	10
	1	1	18
5	0	0	6
	0	1	11
	1	0	12
	1	1	22
6	0	0	7
	0	1	13
	1	0	14
	1	1	26
7	0	0	8
	0	1	15
	1	0	16
	1	1	30

(1) XINTCNF2[XTIMCLK] configures the ratio between SYSCLKOUT and XTIMCLK.
(2) X2TIMING is configured per zone in the zone specific XTIMING register.

14.5 Configuring XBANK Cycles

When jumping from one XINTF zone to another XINTF zone, a slow device may require extra cycles in order to release the bus in time for another device to gain access. Bank switching allows the XINTF to add extra cycles for any access that crosses into or out of the specified zone. The zone and number of cycles is configured in the XBANK register.

Select the number of delay cycles based on the ratio of XTIMCLK and XCLKOUT. There are three cases:

- **Case 1: XTIMCLK = SYSCLKOUT**

  When XTIMCLK is equal to SYSCLKOUT, there are no restrictions on the selection of XBANK[BCYC].
- **Case 2: XTIMCLK = 1/2 SYSCLKOUT and XCLKOUT = 1/2 XTIMCLK**

  In this case, XBANK[BCYC] must not be 4 or 6. Any other value is valid.
- **Case 3: XTIMCLK = 1/2 SYSCLKOUT and XCLKOUT = XTIMCLK**
- **Case 4: XTIMCLK = 1/4 SYSCLKOUT**

  When delay cycles are inserted between two accesses, there is a zone access before the delay cycles and a zone access after the delay cycles. In order for bank switching to correctly insert the correct number of delay cycles the total access time to the first zone in the sequence must be greater than the number of bank cycles specified. To make sure this happens select XBANK[BCYC] such that it is smaller than the total access to the zone.

  Consider this example: Bank switching is enabled for Zone 7 (XBANK[BANK] = 7) . Delay cycles will be added to any access into or out of Zone 7. This means:

  – If an access to Zone 0 is followed by Zone 7

    The access to Zone 0 must be longer than the number of bank cycles specified.

  – If an access to Zone 1 is followed by Zone 7

    The access to Zone 1must be longer than the number of bank cycles specified.

  – If an access to Zone 7 is followed by Zone 0:

    The access to Zone 7 must be longer than the number of bank cycles specified.

  The lead, active and trail values can be used to make sure the access time is longer than the number of delay cycles. Since XREADY can only extend the access longer, it need not be considered.

  If X2TIMING is 0, then select:

  – XBANK[BCYC] < XWRLEAD + XWRACTIVE + 1 + XWRTRAIL and
  – XBANK[BCYC] < XRDLEAD + XRDACTIVE + 1 + XRDTRAIL

  If X2TIMING = 1, then select:

  – XBANK[BCYC] < XWRLEADx2 + XWRACTIVEx2 + 1 + XWRTRAILx2 and
  – XBANK[BCYC] < XRDLEADx2 + XRDACTIVEx2 + 1 + XRDTRAILx2

Table 14-6 lists valid XBANK[BCYC] values for different timing configurations. The lead, active and trail values are specified in the zones XTIMING register. When determining the proper XBANK[BCYC] values, use the timing that yields the longest access time. This may be the read or the write timing.

Table 14-6. Valid XBANK Configurations

Valid XBANK[BCYC]	Total Access Time	XRDLEAD or XWRLEAD	XRDACTIVE or XWRACTIVE	XRDTRAIL or XWRTRAIL	X2TIMING
< 5	1 + (2+1) + 1 = 5	1	2	1	0
< 6	1 + (3+1) + 1 = 6	1	3	1	0
< 7	2 + (3+1) + 1 = 7	2	3	1	0
< 5	1x2 + 0x2 + 1 + 1x2	1	0	1	1
< 5	1x2 + 1x2 + 1 + 0x2	1	1	0	1

14.6 XINTF Registers

Table 14-7 lists the memory-mapped registers for the XINTF_CONFIGURATION_AND_CONTROL_REGISTER_MAPPING. All register offset addresses not listed in Table 14-7 should be considered as reserved locations and the register contents should not be modified.

Table 14-7. XINTF Configuration and Control Register Mapping

Offset	Acronym	Register Name	Size (x16)	Section
83Dh	XRESET	XINTF Reset Register	2	Section 14.6.1
B20h	XTIMING0	XINTF Timing Register, Zone 0	2	Section 14.6.2
B2Ch	XTIMING6[(1)]	XINTF Timing Register, Zone 6	2	Section 14.6.3
B2Eh	XTIMING7	XINTF Timing Register, Zone 7	2	Section 14.6.4
B38h	XBANK	XINTF Bank Control Register	1	Section 14.6.5
B3Ah	XREVISION	XINTF Revision Register	1	Section 14.6.6

(1) XTIMING1 - XTIMING5 are reserved for future expansion and are not currently used.

The individual timing parameters can be programmed into the XTIMING registers described in Section 14.2.

14.6.1 XRESET Register (Offset = 83Dh) [reset = 0h]

XRESET is shown in Figure 14-6 and described in Table 14-8.

This register is EALLOW protected.

Figure 14-6. XRESET Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							XHARDRESET
R-0h							R/W-0h

Table 14-8. XRESET Register Field Descriptions

Bit	Field	Type	Reset	Description
15-1	RESERVED	R	0h	
0	XHARDRESET	R/W	0h	A hard reset may be used in cases where the CPU detects the XREADY signal is stuck low during a DMA transfer. 0h = Writing a 0 has no effect. This bit always reads back a 0. 1h = Force an XINTF hard reset. The XTIMING, XBANK, and XINTCNF2 registers will return to their default state and all XINTF signals will go to their inactive state. Any pending access will be lost including data in the write buffer. Any stall condition to DMA will be released.

14.6.2 XTIMING0 Register (Offset = B20h) [reset = 41D2A5h]

XTIMING0 is shown in Figure 14-7 and described in Table 14-9.

Each XINTF zone has one timing register. Changes to this register will affect the timing of that particular zone. Changes to a zone's timing register should be made only by code executing outside of that zone.

This register is EALLOW protected.

NOTE:

- Minimum wait-state requirements for different modes are shown in Section 14.2.
- The external device to which the 28x is interfaced may have additional timing constraints. See the vendor documentation for details.
- No logic is included to detect illegal settings.

Figure 14-7. XTIMING0 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED	X2TIMING	RESERVED				XSIZE	
R-0h	R/W-1h	R-0h				R/W-1h	
15	14	13	12	11	10	9	8
READYMODE	USEREADY	XRDLEAD		XRDACTIVE			XRDTRAIL
R/W-1h	R/W-1h	R/W-1h		R/W-1h			R/W-1h
7	6	5	4	3	2	1	0
XRDTRAIL	XWRLEAD		XWRACTIVE			XWRTRAIL	
R/W-1h	R/W-1h		R/W-1h			R/W-1h	

Table 14-9. XTIMING0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-23	RESERVED	R	0h	Reserved
22	X2TIMING	R/W	1h	This bit specifies the scaling factor of the XRDLEAD, XRDACTIVE, XRDTRAIL, XWRLEAD, XWRACTIVE, and XWRTRAIL values for the zone. 0h = The values are scaled 1:1 1h = The values are scaled 2:1 (doubled). This the default mode of operation on power up and reset.
21-18	RESERVED	R	0h	
17-16	XSIZE	R/W	1h	These two bits must always be written to as either 0, 1 (32-bit data bus) or 1, 1 (16-bit data bus). Any other combination is reserved and will result in incorrect XINTF behavior. 0h = Reserved - results in incorrect XINTF behavior 1h = 32-bit interface. In this mode the zone will use all 32 data lines. The XA0/$\overline{WE1}$ signal will behave as $\overline{WE1}$ as described in Section 14.2.7. 2h = Reserved - results in incorrect XINTF behavior 3h = 16-bit interface. In this mode the zone will only use 16 data lines. The XA0/$\overline{WE1}$ signal will behave as XA0. (default at reset)
15	READYMODE	R/W	1h	Sets the XREADY input sampling for the zone as synchronous or asynchronous. This bit is ignored if XREADY is not sampled (USEREADY = 0). 0h = XREADY input is synchronous for the zone. 1h = XREADY input is asynchronous for the zone. (default at reset)

Table 14-9. XTIMING0 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
14	USEREADY	R/W	1h	Determines if accesses to the zone will sample or ignore the XREADY input signal. 0h = The XREADY signal is ignored when accesses are made to the zone. 1h = The XREADY signal can further extend the active portion of an access to the zone past the minimum defined by the XRDACTIVE and XWRACTIVE fields.
13-12	XRDLEAD	R/W	1h	Two-bit field that defines the read cycle lead wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-12.
11-9	XRDACTIVE	R/W	1h	Three-bit field that defines the read cycle active wait state period, in XTIMCLK cycles. The active period is by default 1 XTIMCLK cycle. Therefore, the total active period is (1 + XWRACTIVE) XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-13.
8-7	XRDTRAIL	R/W	1h	Two-bit field that defines the read cycle trail wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-14.
6-5	XWRLEAD	R/W	1h	Two-bit field that defines the write cycle lead wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-15.
4-2	XWRACTIVE	R/W	1h	Three-bit field that defines the write cycle active wait state period, in XTIMCLK cycles. The active period is by default 1 XTIMCLK cycle. Therefore, the total active period is (1 + XWRACTIVE) XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-16.
1-0	XWRTRAIL	R/W	1h	Two-bit field that defines the write cycle trail wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-17.

14.6.3 XTIMING6 Register (Offset = B2Ch) [reset = 41D2A5h]

XTIMING6 is shown in Figure 14-8 and described in Table 14-10.

Each XINTF zone has one timing register. Changes to this register will affect the timing of that particular zone. Changes to a zone's timing register should be made only by code executing outside of that zone.

This register is EALLOW protected.

NOTE:

- Minimum wait-state requirements for different modes are shown in Section 14.2.
- The external device to which the 28x is interfaced may have additional timing constraints. See the vendor documentation for details.
- No logic is included to detect illegal settings.

Figure 14-8. XTIMING6 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED	X2TIMING	RESERVED				XSIZE	
R-0h	R/W-1h	R-0h				R/W-1h	
15	14	13	12	11	10	9	8
READYMODE	USEREADY	XRDLEAD		XRDACTIVE			XRDTRAIL
R/W-1h	R/W-1h	R/W-1h		R/W-1h			R/W-1h
7	6	5	4	3	2	1	0
XRDTRAIL	XWRLEAD		XWRACTIVE			XWRTRAIL	
R/W-1h	R/W-1h		R/W-1h			R/W-1h	

Table 14-10. XTIMING6 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-23	RESERVED	R	0h	Reserved
22	X2TIMING	R/W	1h	This bit specifies the scaling factor of the XRDLEAD, XRDACTIVE, XRDTRAIL, XWRLEAD, XWRACTIVE, and XWRTRAIL values for the zone. 0h = The values are scaled 1:1 1h = The values are scaled 2:1 (doubled). This the default mode of operation on power up and reset.
21-18	RESERVED	R	0h	
17-16	XSIZE	R/W	1h	These two bits must always be written to as either 0, 1 (32-bit data bus) or 1, 1 (16-bit data bus). Any other combination is reserved and will result in incorrect XINTF behavior. 0h = Reserved - results in incorrect XINTF behavior 1h = 32-bit interface. In this mode the zone will use all 32 data lines. The XA0/$\overline{WE1}$ signal will behave as $\overline{WE1}$ as described in Section 14.2.7. 2h = Reserved - results in incorrect XINTF behavior 3h = 16-bit interface. In this mode the zone will only use 16 data lines. The XA0/$\overline{WE1}$ signal will behave as XA0. (default at reset)
15	READYMODE	R/W	1h	Sets the XREADY input sampling for the zone as synchronous or asynchronous. This bit is ignored if XREADY is not sampled (USEREADY = 0). 0h = XREADY input is synchronous for the zone. 1h = XREADY input is asynchronous for the zone. (default at reset)

Table 14-10. XTIMING6 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
14	USEREADY	R/W	1h	Determines if accesses to the zone will sample or ignore the XREADY input signal. 0h = The XREADY signal is ignored when accesses are made to the zone. 1h = The XREADY signal can further extend the active portion of an access to the zone past the minimum defined by the XRDACTIVE and XWRACTIVE fields.
13-12	XRDLEAD	R/W	1h	Two-bit field that defines the read cycle lead wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-12.
11-9	XRDACTIVE	R/W	1h	Three-bit field that defines the read cycle active wait state period, in XTIMCLK cycles. The active period is by default 1 XTIMCLK cycle. Therefore, the total active period is (1 + XWRACTIVE) XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-13.
8-7	XRDTRAIL	R/W	1h	Two-bit field that defines the read cycle trail wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-14.
6-5	XWRLEAD	R/W	1h	Two-bit field that defines the write cycle lead wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-15.
4-2	XWRACTIVE	R/W	1h	Three-bit field that defines the write cycle active wait state period, in XTIMCLK cycles. The active period is by default 1 XTIMCLK cycle. Therefore, the total active period is (1 + XWRACTIVE) XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-16.
1-0	XWRTRAIL	R/W	1h	Two-bit field that defines the write cycle trail wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-17.

14.6.4 XTIMING7 Register (Offset = B2Eh) [reset = 41D2A5h]

XTIMING7 is shown in Figure 14-9 and described in Table 14-11.

Each XINTF zone has one timing register. Changes to this register will affect the timing of that particular zone. Changes to a zone's timing register should be made only by code executing outside of that zone.

This register is EALLOW protected.

NOTE:

- Minimum wait-state requirements for different modes are shown in Section 14.2.
- The external device to which the 28x is interfaced may have additional timing constraints. See the vendor documentation for details.
- No logic is included to detect illegal settings.

Figure 14-9. XTIMING7 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED	X2TIMING	RESERVED				XSIZE	
R-0h	R/W-1h	R-0h				R/W-1h	
15	14	13	12	11	10	9	8
READYMODE	USEREADY	XRDLEAD		XRDACTIVE			XRDTRAIL
R/W-1h	R/W-1h	R/W-1h		R/W-1h			R/W-1h
7	6	5	4	3	2	1	0
XRDTRAIL	XWRLEAD		XWRACTIVE			XWRTRAIL	
R/W-1h	R/W-1h		R/W-1h			R/W-1h	

Table 14-11. XTIMING7 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-23	RESERVED	R	0h	Reserved
22	X2TIMING	R/W	1h	This bit specifies the scaling factor of the XRDLEAD, XRDACTIVE, XRDTRAIL, XWRLEAD, XWRACTIVE, and XWRTRAIL values for the zone. 0h = The values are scaled 1:1 1h = The values are scaled 2:1 (doubled). This the default mode of operation on power up and reset.
21-18	RESERVED	R	0h	
17-16	XSIZE	R/W	1h	These two bits must always be written to as either 0, 1 (32-bit data bus) or 1, 1 (16-bit data bus). Any other combination is reserved and will result in incorrect XINTF behavior. 0h = Reserved - results in incorrect XINTF behavior 1h = 32-bit interface. In this mode the zone will use all 32 data lines. The XA0/$\overline{WE1}$ signal will behave as $\overline{WE1}$ as described in Section 14.2.7. 2h = Reserved - results in incorrect XINTF behavior 3h = 16-bit interface. In this mode the zone will only use 16 data lines. The XA0/$\overline{WE1}$ signal will behave as XA0. (default at reset)
15	READYMODE	R/W	1h	Sets the XREADY input sampling for the zone as synchronous or asynchronous. This bit is ignored if XREADY is not sampled (USEREADY = 0). 0h = XREADY input is synchronous for the zone. 1h = XREADY input is asynchronous for the zone. (default at reset)

Table 14-11. XTIMING7 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
14	USEREADY	R/W	1h	Determines if accesses to the zone will sample or ignore the XREADY input signal. 0h = The XREADY signal is ignored when accesses are made to the zone. 1h = The XREADY signal can further extend the active portion of an access to the zone past the minimum defined by the XRDACTIVE and XWRACTIVE fields.
13-12	XRDLEAD	R/W	1h	Two-bit field that defines the read cycle lead wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-12.
11-9	XRDACTIVE	R/W	1h	Three-bit field that defines the read cycle active wait state period, in XTIMCLK cycles. The active period is by default 1 XTIMCLK cycle. Therefore, the total active period is (1 + XWRACTIVE) XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-13.
8-7	XRDTRAIL	R/W	1h	Two-bit field that defines the read cycle trail wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-14.
6-5	XWRLEAD	R/W	1h	Two-bit field that defines the write cycle lead wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-15.
4-2	XWRACTIVE	R/W	1h	Three-bit field that defines the write cycle active wait state period, in XTIMCLK cycles. The active period is by default 1 XTIMCLK cycle. Therefore, the total active period is (1 + XWRACTIVE) XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-16.
1-0	XWRTRAIL	R/W	1h	Two-bit field that defines the write cycle trail wait state period, in XTIMCLK cycles. If the X2TIMING bit is set, then the number of wait states are doubled. See Section 14.4 for minimum requirements in different modes. See Table 14-17.

Table 14-12. XRDLEAD

	X2TIMING	Read Lead Period
00	X	Invalid
01	0	1 XTIMCLK cycle
	1	2 XTIMCLK cycles
10	0	2 XTIMCLK cycles
	1	4 XTIMCLK cycles
11	0	3 XTIMCLK cycles
	1	6 XTIMCLK cycles (default)

Table 14-13. XRDACTIVE

	X2TIMING	Read Active Period Waitstates
000	0	0
001	0	1 XTIMCLK cycle
	1	2 XTIMCLK cycles
010	0	2 XTIMCLK cycles
	1	4 XTIMCLK cycles
011	0	3 XTIMCLK cycles
	1	6 XTIMCLK cycles
100	0	4 XTIMCLK cycles
	1	8 XTIMCLK cycles
101	0	5 XTIMCLK cycles
	1	10 XTIMCLK cycles
110	0	6 XTIMCLK cycles
	1	12 XTIMCLK cycles
111	0	7 XTIMCLK cycles
	1	14 XTIMCLK cycles (default)

Table 14-14. XRDTRAIL

	X2TIMING	Read Trail Period
00	0	0
01	0	1 XTIMCLK cycle
	1	2 XTIMCLK cycles
10	0	2 XTIMCLK cycles
	1	4 XTIMCLK cycles
11	0	3 XTIMCLK cycles
	1	6 XTIMCLK cycles (default)

Table 14-15. XWRLEAD

	X2TIMING	Write Lead Period
00	0	0
01	0	1 XTIMCLK cycle
	1	2 XTIMCLK cycles
10	0	2 XTIMCLK cycles
	1	4 XTIMCLK cycles
11	0	3 XTIMCLK cycles
	1	6 XTIMCLK cycles (default)

Table 14-16. XWRACTIVE

	X2TIMING	Write Active Period Waitstates
000	0	0
001	0	1 XTIMCLK cycle
	1	2 XTIMCLK cycles
010	0	2 XTIMCLK cycles
	1	4 XTIMCLK cycles
011	0	3 XTIMCLK cycles
	1	6 XTIMCLK cycles

Table 14-16. XWRACTIVE (continued)

	X2TIMING	Write Active Period Waitstates
100	0	4 XTIMCLK cycles
	1	8 XTIMCLK cycles
101	0	5 XTIMCLK cycles
	1	10 XTIMCLK cycles
110	0	6 XTIMCLK cycles
	1	12 XTIMCLK cycles
111	0	7 XTIMCLK cycles
	1	14 XTIMCLK cycles (default)

Table 14-17. XWRTRAIL

	X2TIMING	Write Trail Period
00	x	
01	0	1 XTIMCLK cycle
	1	2 XTIMCLK cycles
10	0	2 XTIMCLK cycles
	1	4 XTIMCLK cycles
11	0	3 XTIMCLK cycles
	1	6 XTIMCLK cycles (default)

14.6.5 XBANK Register (Offset = B38h) [reset = 9h]

XBANK is shown in Figure 14-10 and described in Table 14-18.

This register is EALLOW protected.

Figure 14-10. XBANK Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED		BCYC			BANK		
R-0h		R/W-1h			R/W-1h		

Table 14-18. XBANK Register Field Descriptions

Bit	Field	Type	Reset	Description
15-6	RESERVED	R	0h	
5-3	BCYC	R/W	1h	These bits specify the number of XTIMCLK cycles to add between any consecutive access that crosses into or out of the specified zone, be it a read or write, program or data space. The number of XTIMCLK cycles can be 0 to 7. On a reset ($\overline{XRS}$), the value defaults to 7 XTIMCLK cycles (14 SYSCLKOUT cycles). 0h = 0 cycle 1h = 1 XTIMCLK cycle 2h = 2 XTIMCLK cycles 3h = 3 XTIMCLK cycles 4h = 4 XTIMCLK cycles 5h = 5 XTIMCLK cycles 6h = 6 XTIMCLK cycles 7h = 7 XTIMCLK cycles (default)
2-0	BANK	R/W	1h	These bits specify the XINTF zone for which bank switching is enabled, ZONE 0 to ZONE 7. At reset, XINTF Zone 7 is selected. 0h = Zone 0 1h = Reserved 2h = Reserved 3h = Reserved 4h = Reserved 5h = Reserved 6h = Zone 6 7h = Zone 7 (selected at reset by default)

14.6.6 XREVISION Register (Offset = B3Ah) [reset = X]

XREVISION is shown in Figure 14-11 and described in Table 14-19.

Figure 14-11. XREVISION Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REVISION															
R-X															

Table 14-19. XREVISION Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	REVISION	R	X	Current XINTF Revision. For internal use/reference. Test purposes only. Subject to change.

14.7 Signal Descriptions

Table 14-20. XINTF Signal Descriptions

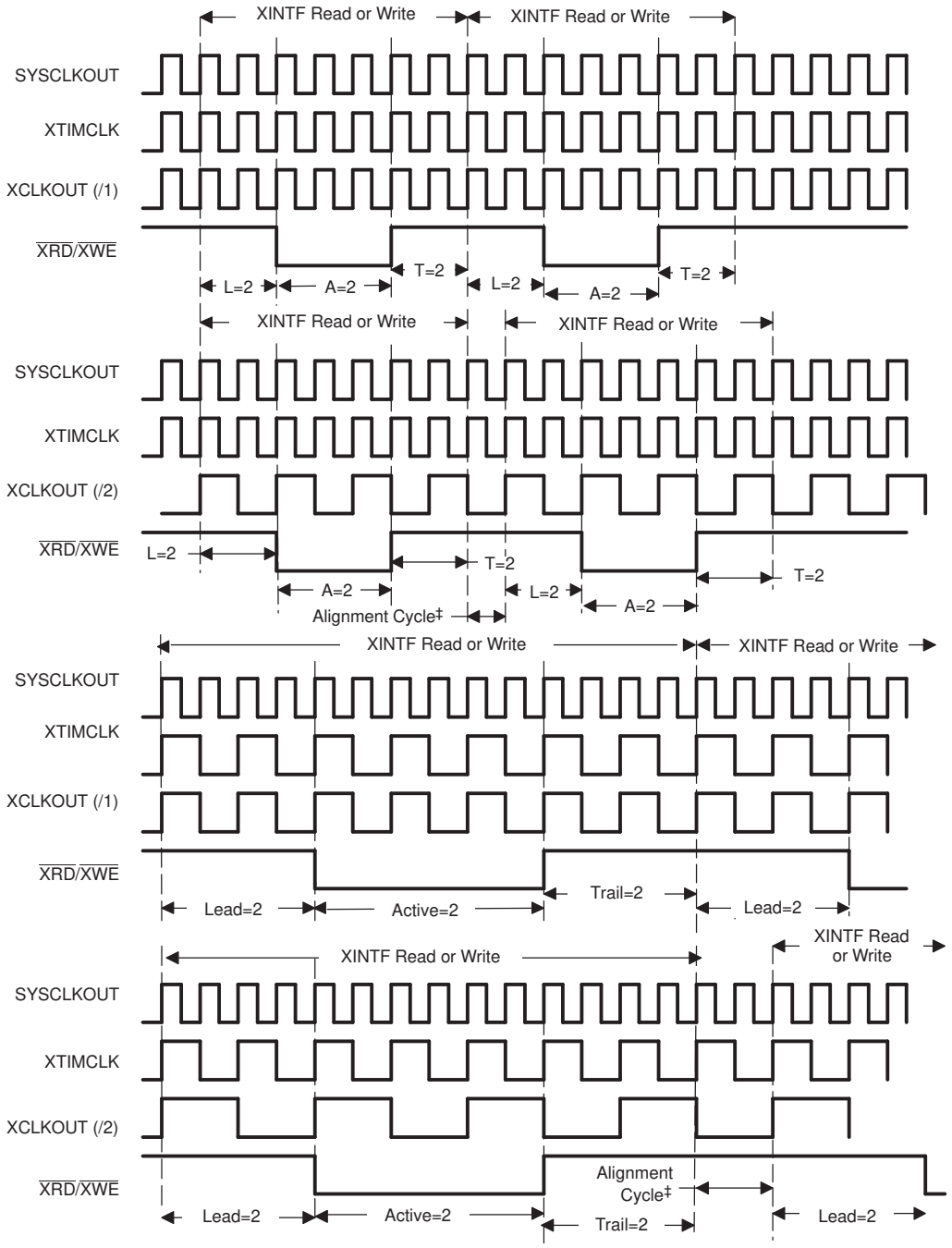
Name	Type	Description
XD(31:0)	I/O/Z	Bidirectional 32-bit data bus. In 16-bit mode only XD(15:0) are used.
XA(31:1)	O/Z	Address bus. The address is placed on the bus on the rising edge of XCLKOUT and held on the bus until the next access. Specific devices may not have all 32 address lines. See the data sheet for a specific device.
XA0/ $\overline{\text{XWE1}}$	O/Z	In 16-bit data mode (see), this signal is the least significant address line (XA0). In 32-bit data mode, this signal is the active low write strobe $\overline{\text{XWE1}}$. $\overline{\text{XWE1}}$ is used, along with $\overline{\text{XWE0}}$, for 32-bit bus operation as shown in Section 14.2.7.
XCLKOUT	O/Z	Single output clock derived from the XTIMCLK to be used for on-chip and off-chip wait-state generation and as a general-purpose clock source. XCLKOUT is either the same frequency or ½ the frequency of XTIMCLK, as defined by the CLKMODE bit in the XINTCNF2 register. At reset XCLKOUT = XTIMCLK/2 XTIMCLK = SYSCLKOUT/2
$\overline{\text{XWE0}}$	O/Z	Active low write strobe. In 16-bit mode, this signal is driven low on all bus modes and data size types. In 32-bit mode, it is driven as shown in Figure 14-5. The write strobe waveform is specified, per zone basis, by the Lead, Active, Trail periods in the XTIMINGx registers.
$\overline{\text{XRD}}$	O/Z	Active low read strobe. This signal is driven low on all bus modes and data size types. The read strobe waveform is specified, per zone basis, by the Lead, Active, Trail periods in the XTIMINGx registers. Note: The $\overline{\text{XRD}}$ and $\overline{\text{XWE0}}$ signals are mutually exclusive.
XR/$\overline{\text{W}}$	O/Z	Read-not-write control. When high, this signal indicates a read cycle is active, when low, it indicates a write cycle is active. This signal is normally held high. The XR/$\overline{\text{W}}$ signal performs similar functions to the $\overline{\text{XRD}}$ and $\overline{\text{XWE0}}$ signals. Generally, users opt to use $\overline{\text{XWE0}}$ and $\overline{\text{XWE1}}$ because they are cleaner and easier to use.
$\overline{\text{XZCS0}}$ $\overline{\text{XZCS6}}$ $\overline{\text{XZCS7}}$	O	Zone chip-selects. These signals are active when an access to the addressed zone is performed.
XREADY	I	Indicates peripheral is READY to complete the access when asserted to 1. For each XINTF zone, this can be configured to be a synchronous or an asynchronous input. In synchronous mode, the XINTF interface block requires XREADY to be valid one XTIMCLK clock cycle before the end of the active period. In asynchronous mode, The XINTF interface block samples XREADY three XTIMCLK clock cycles before the end of the active period. XREADY is sampled at the XTIMCLK rate independent of the XCLKOUT mode.
$\overline{\text{XHOLD}}$	I	This signal, when active low, requests the XINTF to release the external bus (place all busses and strobes into high-impedance state). The XINTF releases the bus when any current access is complete and there are no pending accesses on the XINTF. This signal is an asynchronous input and is synchronized by XTIMCLK.
$\overline{\text{XHOLDA}}$	O/Z	This signal is driven active low, when the XINTF has granted an $\overline{\text{XHOLD}}$ request. All XINTF busses and strobe signals will be in a high-impedance state. This signal is released when the $\overline{\text{XHOLD}}$ signal is released. External devices should only drive the external bus when this signal is active low.

14.8 Waveforms

Figure 14-12 shows example timing waveforms for various XTIMCLK and XCLKOUT modes assuming X2TIMING = 0 and Lead = 2, Active = 2 and Trail = 2.

NOTE: The diagrams included in this document are conceptual, cycle-by-cycle representations of the XINTF behavior. They do not take into account any buffer delays and additional setup times that will be found on a physical device. For more exact device-specific timing information for the XINTF, see the data sheet electrical timing specifications for that device.

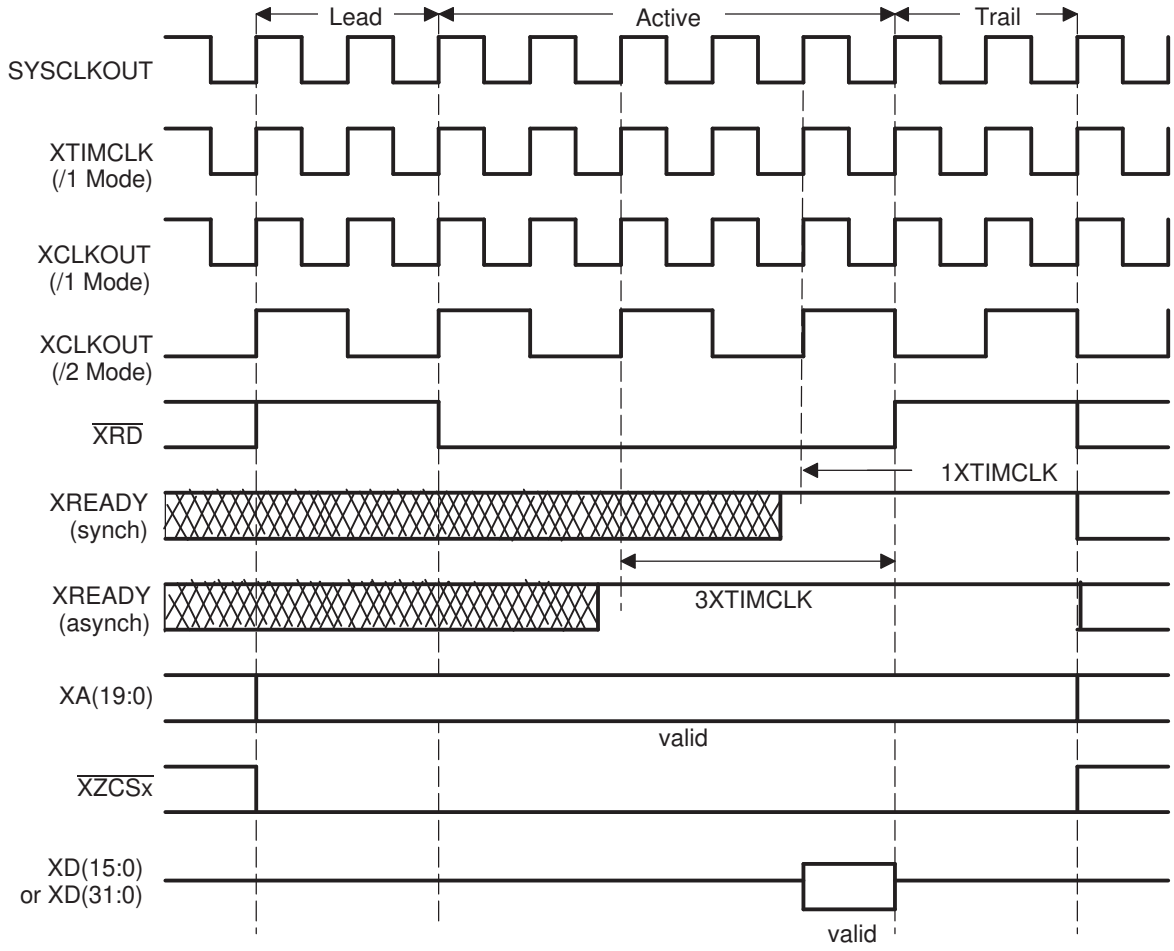
Figure 14-12. XTIMCLK and XCLKOUT Mode Waveforms



A X2TIMING = 0, XRDLEAD/XWRLEAD = 2, XRDACTIVE/XWRACTIVE = 2, XRDTRAIL/XWRTRAIL = 2

B Alignment cycle. Necessary to make sure all bus cycles start on rising edge of XCLKOUT.

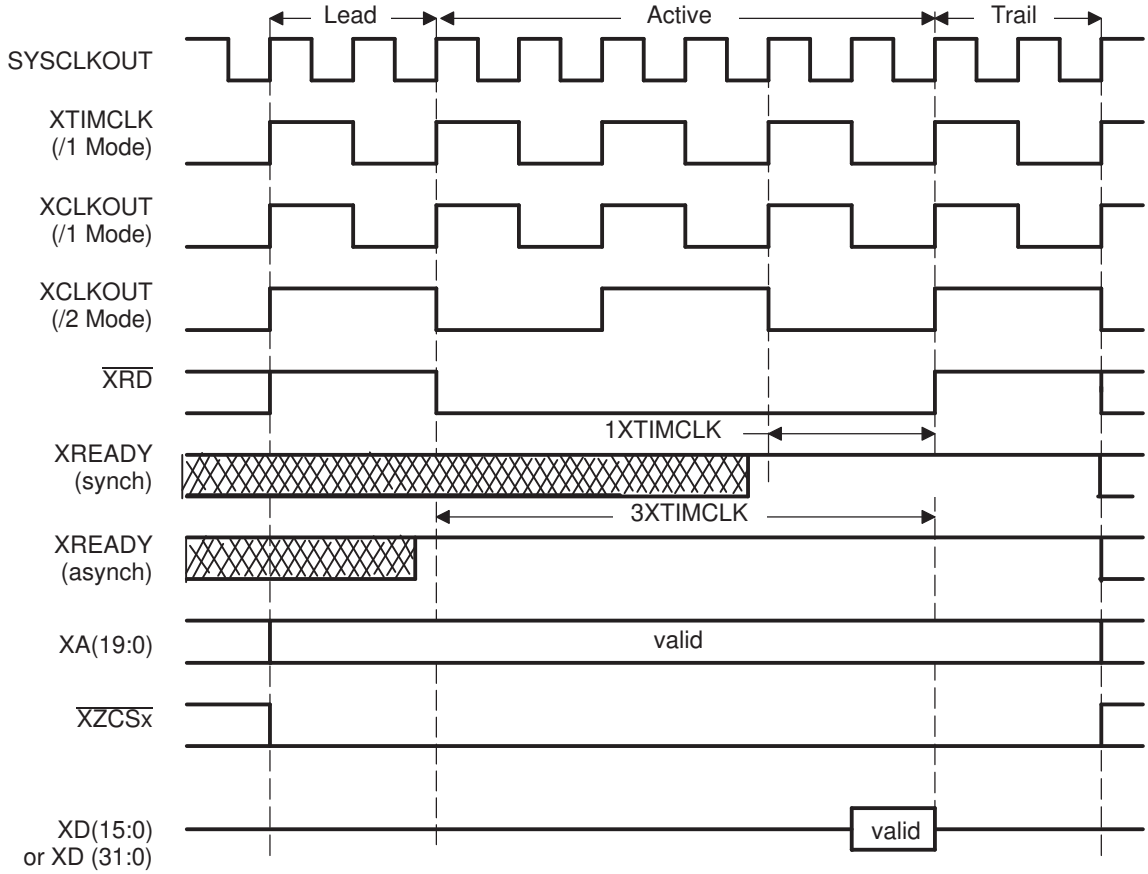
Figure 14-13. Generic Read Cycle (XTIMCLK = SYSCLKOUT mode)



A XRDLEAD = 2, XRDACTIVE = 4, XRDTRAIL = 2

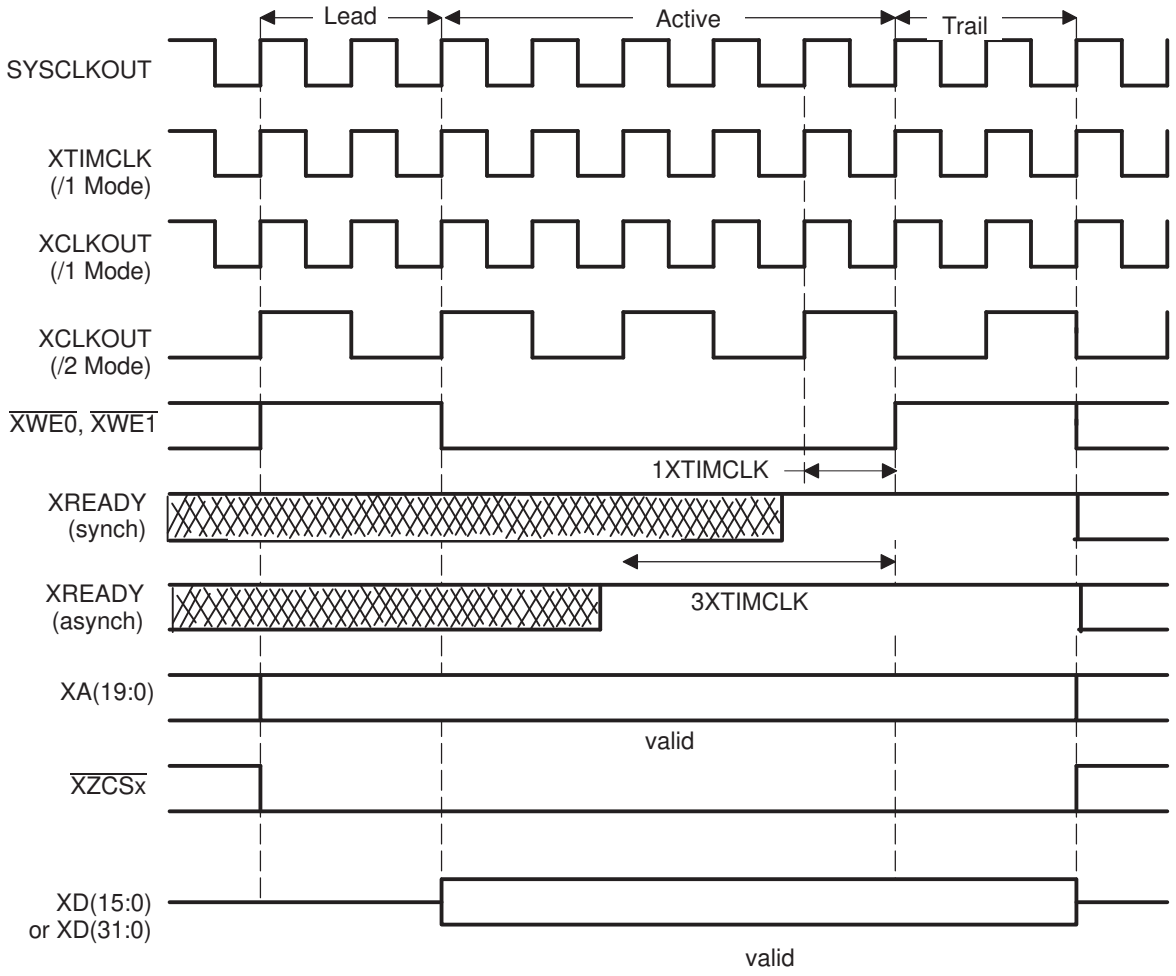
The XREADY signal can be sampled synchronously or asynchronously or ignored by each zone. If it is sampled synchronously, then the XREADY signal MUST meet set-up and hold timing relative to one XTIMCLK edge before the end of the active period. If it is sampled asynchronously, then the XREADY signal MUST meet set-up and hold timing relative to three XTIMCLK edges before the end of the active period. If XREADY is low at the sampling interval, an extra XTIMCLK period is added to the active phase and the XREADY input is sampled again on the next rising edge of XTIMCLK. XCLKOUT has no effect on the sampling interval.

Figure 14-14. Generic Read Cycle (XTIMCLK = ½ SYSCLKOUT mode)



A XRDLEAD = 1, XRDACTIVE = 3, XRDTRAIL = 1

Figure 14-15. Generic Write Cycle (XTIMCLK = SYSCLKOUT mode)



A XWRACTIVE = 2, XWRACTIVE = 4, XWRTRAIL = 2

B If the lead and active timing parameters are set low enough, it may not be possible to generate a valid XREADY signal. No hardware is added to detect this.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265