TMS320F2803x Microcontrollers

Technical Reference Manual

Literature Number: SPRUI10A
DECEMBER 2018 – REVISED JUNE 2022

Table of Contents

List of Figures

List of Tables

About This Manual

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

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

Notational Conventions

This document uses the following conventions.

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

Glossary

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

Related Documentation From Texas Instruments

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

Additionally, the *TMS320C28x DSP CPU and Instruction Set Reference Guide* and the *TMS320C28x Floating Point Unit and Instruction Set Reference Guide* must be used in conjunction with this TRM.

Support Resources

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

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

Trademarks

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

This page intentionally left blank.

Chapter 1

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

1.1 Flash and OTP Memory

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

1.1.1 Flash Memory

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

- **Multiple sectors**

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

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

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

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

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

1.1.2 OTP Memory

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

1.1.3 Flash and OTP Power Modes

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

- **Reset or Sleep State**

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

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

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

Note

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

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

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

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

  To move from a lower power state to a higher power state, there are two options.
  1. Change the FPWR register from a lower state to a higher state. This access brings the flash/OTP memory to the higher state.
  2. Access the flash or OTP memory by a read access or program opcode fetch access. This access automatically brings the flash/OTP memory to the active state.

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

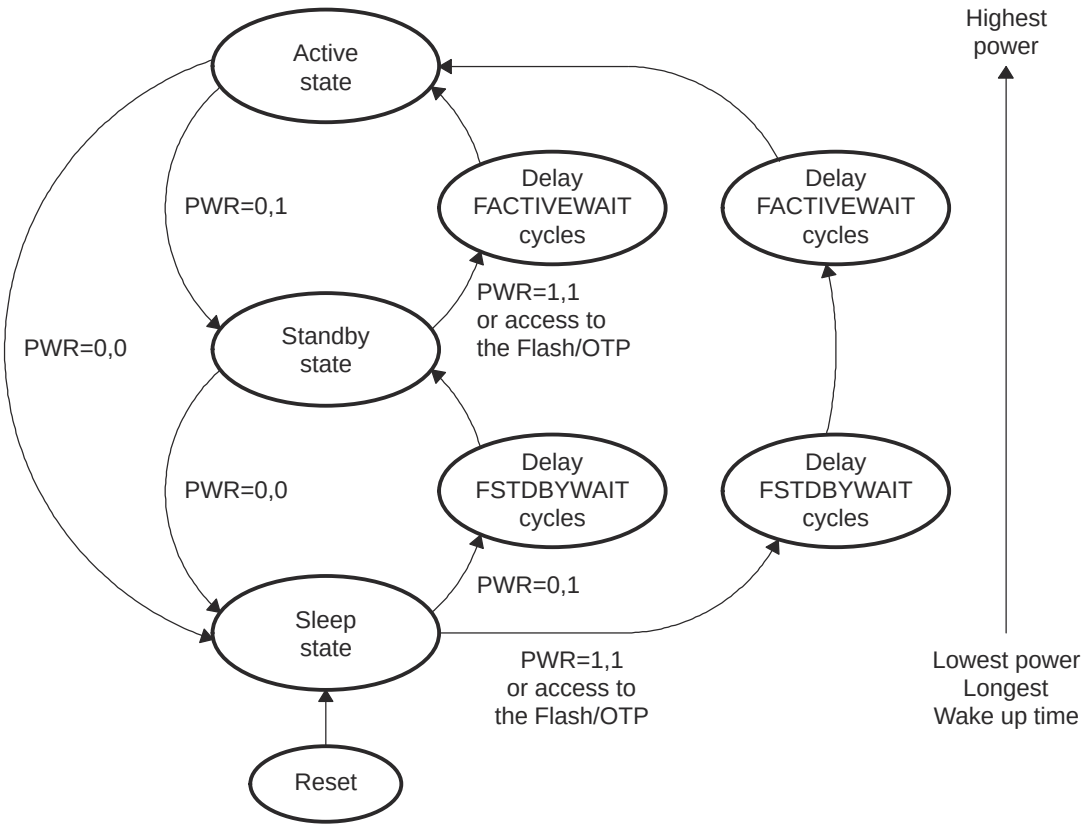


Figure 1-1. Flash Power Mode State Diagram

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

1.1.3.1 Flash and OTP Performance

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

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

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

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

1. **Flash Memory Random Access**

   The first access to a 2048 bit row is considered a random access.
2. **Flash Memory Paged Access**

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

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

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

3. **OTP Access**

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

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

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

1.1.3.2 Flash Pipeline Mode

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

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

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

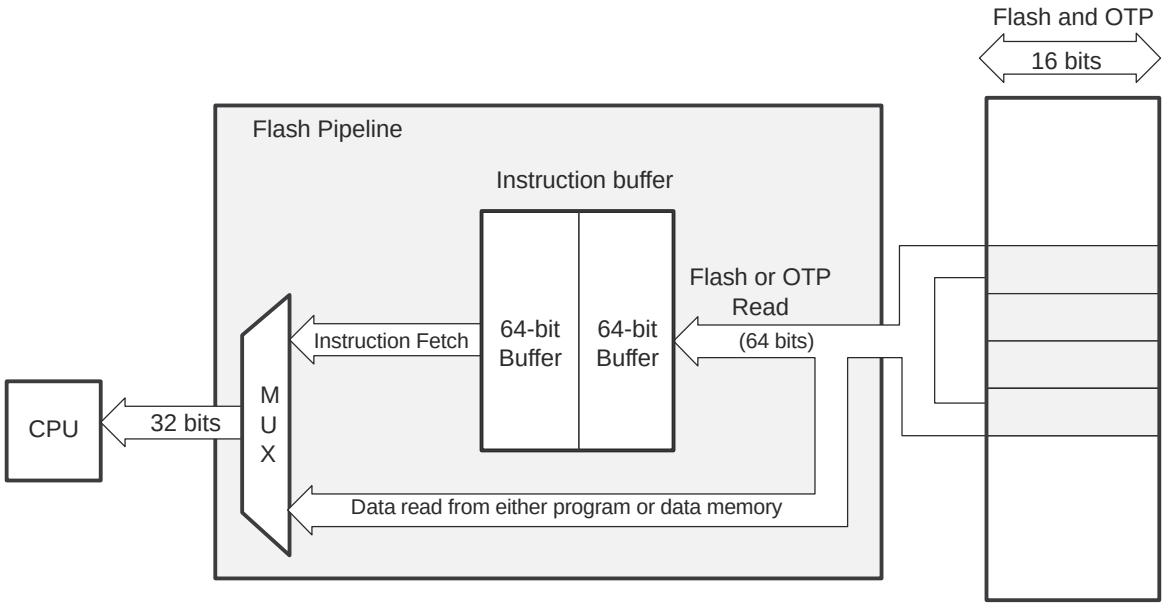


Figure 1-2. Flash Pipeline

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

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

1.1.3.3 Reserved Locations Within Flash and OTP

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

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

1.1.3.4 Procedure to Change the Flash Configuration Registers

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

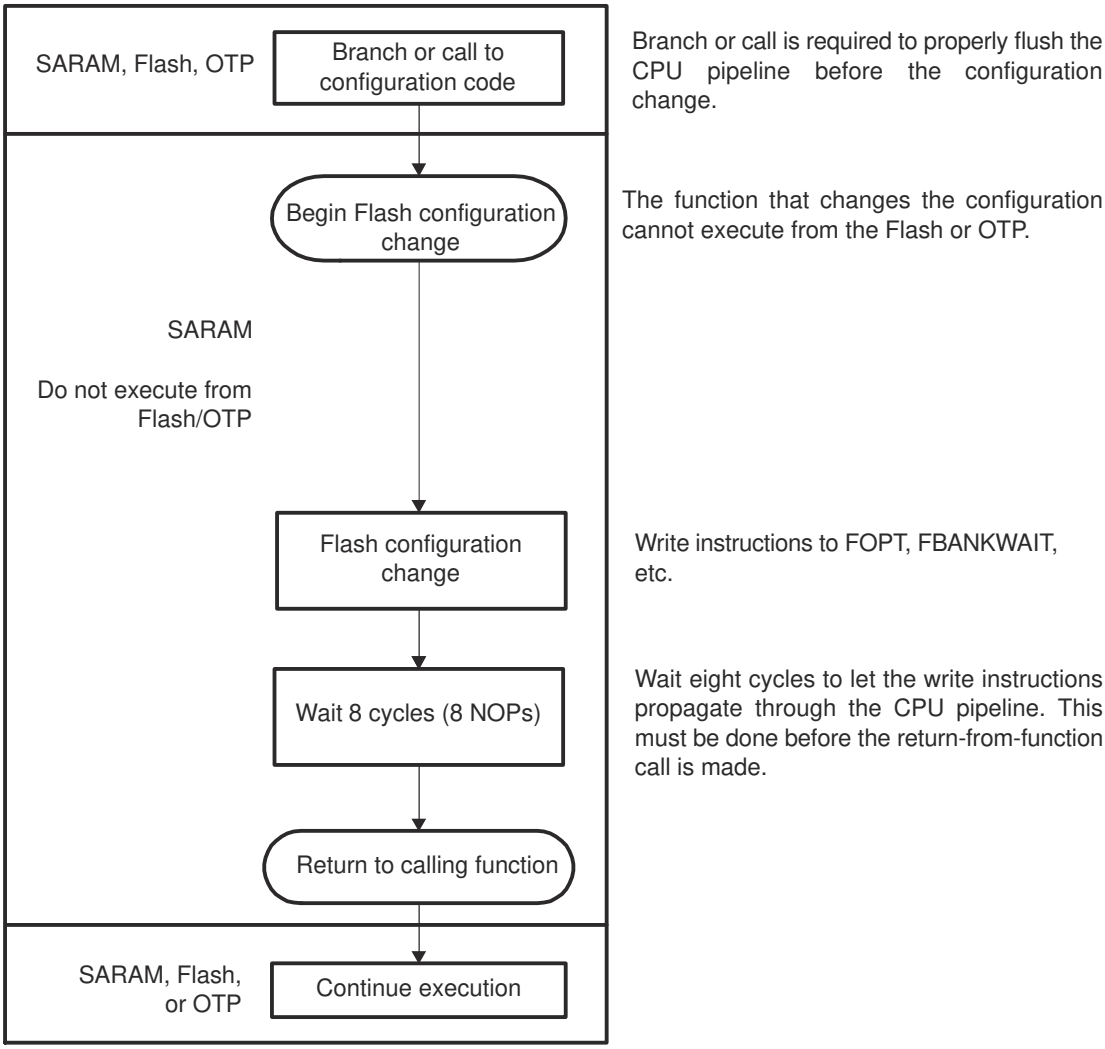


Figure 1-3. Flash Configuration Access Flow Diagram

1.1.4 Flash and OTP Registers

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

Table 1-1. Flash/OTP Configuration Registers

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

(1) These registers are EALLOW protected. See Section 1.5.2 for information.
(2) These registers are protected by the Code Security Module (CSM). See Section 1.2 for more information.
(3) These registers should be left in their default state.

Note

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

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

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

1.1.4.1 Flash Options Register (FOPT)

Figure 1-4. Flash Options Register (FOPT)

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

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

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

Bit	Field	Value	Description (1) (2) (3)
15-1	Reserved		Any writes to these bit(s) must always have a value of 0.
0	ENPIPE		Enable Flash Pipeline Mode Bit. Flash pipeline mode is active when this bit is set. The pipeline mode improves performance of instruction fetches by prefetching instructions. See 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.

1.1.4.2 Flash Power Register (FPWR)

Figure 1-5. Flash Power Register (FPWR)

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

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

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

Bit	Field	Value	Description (1) (2)
15-2	Reserved		Any writes to these bit(s) must always have a value of 0.
1-0	PWR		Flash Power Mode Bits. Writing to these bits changes the current power mode of the flash bank and pump. See 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.

1.1.4.3 Flash Status Register (FSTATUS)

Figure 1-6. Flash Status Register (FSTATUS)

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

7	4	3	2	1	0
Reserved		ACTIVEWAITS	STDBYWAITS	PWRS	
R-0		R-0	R-0	R-0	

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

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

Bit	Field	Value	Description (1) (2)
15-9	Reserved		Any writes to these bit(s) must always have a value of 0.
8	3VSTAT		Flash Voltage (V_{DD3VFL}) Status Latch Bit. When set, this bit indicates that the 3VSTAT signal from the pump module went to a high level. This signal indicates that the flash 3.3-V supply went out of the allowable range.
		0	Writes of 0 are ignored.
		1	When this bit reads 1, it indicates that the flash 3.3-V supply went out of the allowable range. Clear this bit by writing a 1.
7-4	Reserved		Any writes to these bit(s) must always have a value of 0.
3	ACTIVEWAITS		Bank and Pump Standby To Active Wait Counter Status Bit. This bit indicates whether the respective wait counter is timing out an access.
		0	The counter is not counting.
		1	The counter is counting.
2	STDBYWAITS		Bank and Pump Sleep To Standby Wait Counter Status Bit. This bit indicates whether the respective wait counter is timing out an access.
		0	The counter is not counting.
		1	The counter is counting.
1-0	PWRS		Power 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.

1.1.4.4 Flash Standby Wait Register (FSTDBYWAIT)

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

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

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

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

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

(1) This register is EALLOW protected. See Section 1.5.2 for more information.
(2) This register is protected by the Code Security Module (CSM). See Section 1.2 for more information.

1.1.4.5 Flash Standby to Active Wait Counter Register (FACTIVEWAIT)

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

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

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

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

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

(1) This register is EALLOW protected. See Section 1.5.2 for more information.
(2) This register is protected by the Code Security Module (CSM). See Section 1.2 for more information.

1.1.4.6 Flash Wait-State Register (FBANKWAIT)

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

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

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

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

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

1.1.4.7 OTP Wait-State Register (FOTPWAIT)

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

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

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

1.2 Code Security Module (CSM)

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

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

1.2.1 Functional Description

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

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

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

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

Table 1-9. Security Levels

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

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

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

Note

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

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

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

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

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

Note

Reserved Flash Locations When Using Code Security

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

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

Note

Code Security Module Disclaimer

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

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

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

1.2.2 CSM Impact on Other On-Chip Resources

The CSM affects access to the on-chip resources listed in Table 1-10.

Table 1-10. Resources Affected by the CSM

Address	Block
0x00 0A80 - 0x00 0A87	Flash Configuration Registers
0x00 8000 - 0x00 87FF	L0 SARAM (2K X 16)
0x00 8800 - 0x00 8BFF	L1 DPSARAM - CLA Data RAM 0 (1K X 16)
0x00 8C00 - 0x00 8FFF	L2 DPSARAM - CLA Data RAM 1 (1K X 16)
0x00 9000 - 0x00 9FFF	L3 DPSARAM - CLA Program RAM (4K X 16)
0x3E 8000 - 0x3F 7FFF or 0x3F 0000 - 0x3F 7FFF	Flash (64K X 16) Flash (32K X 16)
0x3D 7800 - 0x3D 7BFF	User One-Time Programmable (OTP) (1K X 16)
0x3D 7C00 - 0x3D 7FFF	TI One-Time Programmable (OTP)(1)(1K X 16)
0x3F 8000 - 0x3F 87FF	L0 SARAM (2K 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)
0x3F E000 0x3F FFFF	Boot ROM (4K x 16)

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

1.2.3 Incorporating Code Security in User Applications

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

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

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

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

1.2.3.1 Environments That Require Security Unlocking

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

- Code development using debuggers (such as Code Composer Studio).

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

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

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

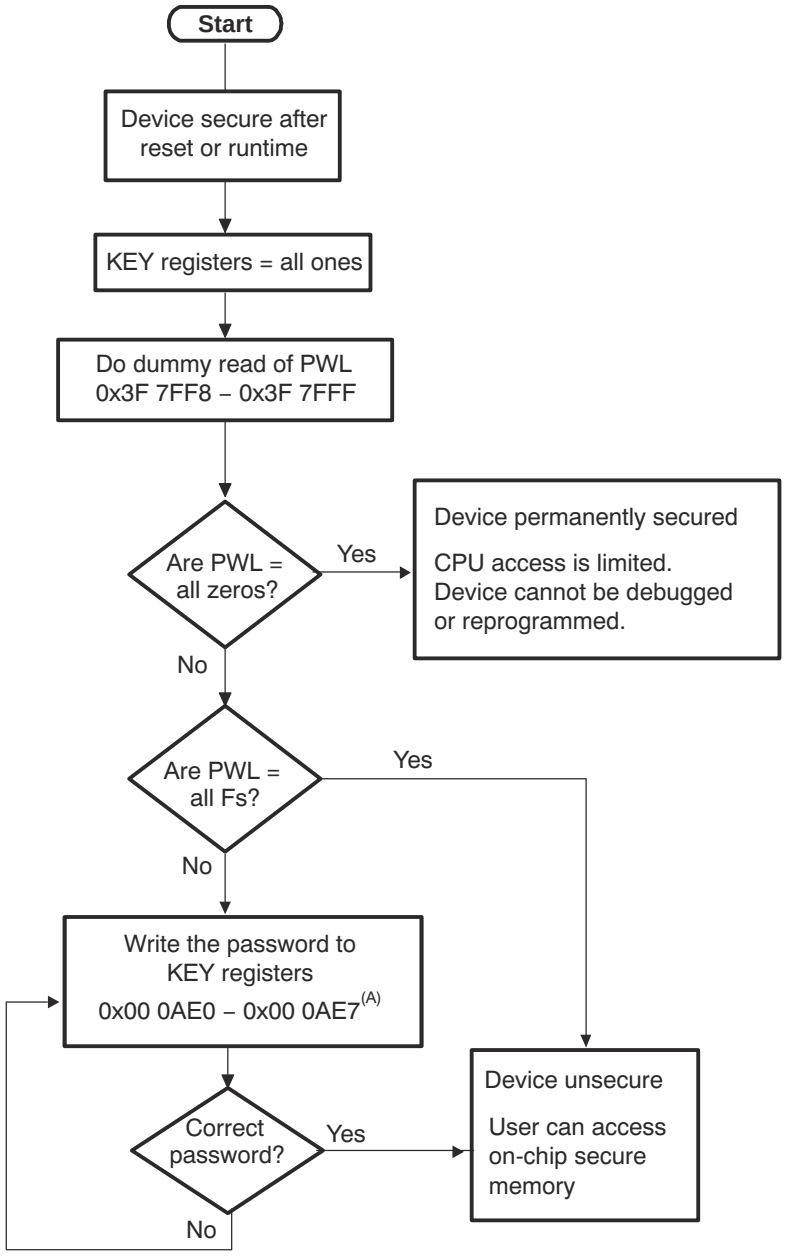
- Using the on-chip bootloader to load code or data into secure SARAM or to erase/program the flash.
- Executing code from on-chip unsecure memory and requiring access to secure memory for lookup table. This is not a suggested operating condition as supplying the password from external code could compromise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

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

1.2.3.2 Password Match Flow

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

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



A. The KEY registers are EALLOW protected.

Figure 1-11. Password Match Flow (PMF)

Note

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

1.2.3.3 Unsecuring Considerations for Devices With and Without Code Security

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

Case 1: Device With Code Security

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

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

Case 2: Device Without Code Security

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

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

Note

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

1.2.3.3.1 C Code Example to Unsecure

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

1.2.3.3.2 C Code Example to Resecure

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

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

1.2.4.1 Do's

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

1.2.4.2 Don'ts

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

1.2.5 CSM Features - Summary

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

1.2.6 CSM Status and Control Registers

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

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

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

1.2.6.1 CSM Status and Control Register (CSMSCR)

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

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

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

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

Bits	Field	Value	Description [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.

1.3 Clocking

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

1.3.1 Clocking and System Control

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

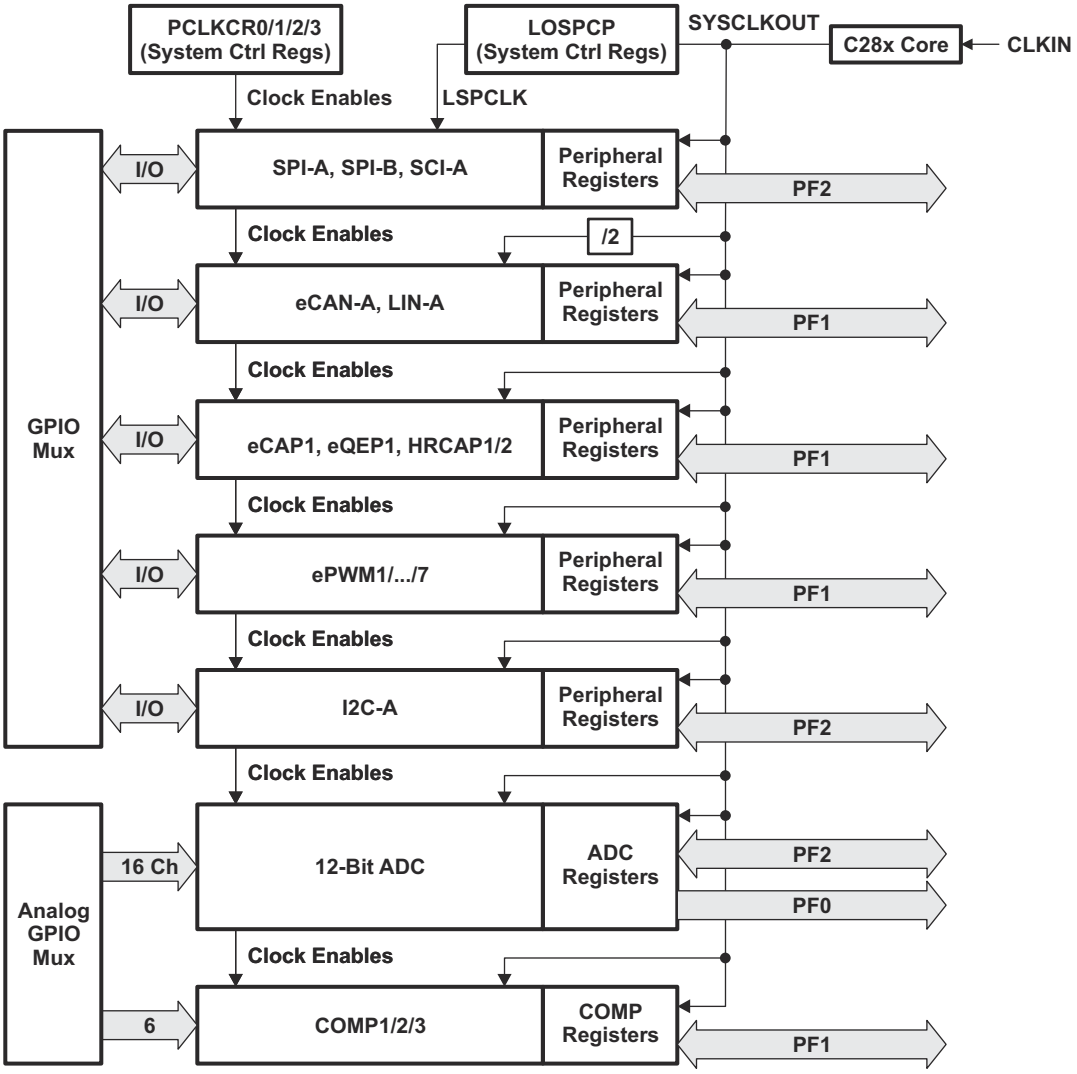


Figure 1-13. Clock and Reset Domains

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

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

Name	Address	Size (x16)	Description	Bit Description
XCLK	0x0000-7010	1	XCLKOUT/XCLKIN Control	Section 1.3.2.2
PLLSTS	0x0000-7011	1	PLL Status Register	Section 1.3.2.4.1.2
CLKCTL	0x0000-7012	1	Clock Control Register	Section 1.3.2.3
PLLLOCKPRD	0x0000-7013	1	PLL Lock Period Register	Section 1.3.2.4.1.3
INTOSC1TRIM	0x0000-7014	1	Internal Oscillator 1 Trim Register	Section 1.3.2.1.1
INTOSC2TRIM	0x0000-7016	1	Internal Oscillator 2 Trim Register	Section 1.3.2.1.1
PCLKCR2	0x0000-7019	1	Peripheral Clock Control Register 2	Section 1.3.1.1

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

Name	Address	Size (x16)	Description	Bit Description
LOSPCP	0x0000-701B	1	Low-Speed Peripheral Clock Pre-Scaler Register	Section 1.3.1.2
PCLKCR0	0x0000-701C	1	Peripheral Clock Control Register 0	Section 1.3.1.1
PCLKCR1	0x0000-701D	1	Peripheral Clock Control Register 1	Section 1.3.1.1
LPMCR0	0x0000-701E	1	Low Power Mode Control Register 0	Section 1.3.3.1
PCLKCR3	0x0000-7020	1	Peripheral Clock Control Register 3	Section 1.3.1.1
PLLCR	0x0000-7021	1	PLL Control Register	Section 1.3.2.4.1.1
SCSR	0x0000-7022	1	System Control and Status Register	Section 1.3.4.5.1
WDCNTR	0x0000-7023	1	Watchdog Counter Register	Section 1.3.4.5.2
WDKEY	0x0000-7025	1	Watchdog Reset Key Register	Section 1.3.4.5.3
WDCR	0x0000-7029	1	Watchdog Control Register	Section 1.3.4.5.4
BORCFG	0x000985	1	BOR Configuration Register	Section 1.7.2

1.3.1.1 Enabling/Disabling Clocks to the Peripheral Modules (PCLKCR0/1/2/3)

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

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

15	14	13	12	11	10	9	8
Reserved	ECANAENCLK	Reserved			SCIA ENCLK	SPIB ENCLK	SPIA ENCLK
R-0	R/W-0	R-0			R/W-0	R/W-0	R/W-0
7		5	4	3	2	1	0
Reserved			I2CA ENCLK	ADC ENCLK	TBCLK SYNC	LINA ENCLK	HRPWM ENCLK
R-0			R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

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

Bit	Field	Value	Description
15	Reserved		Reserved
14	ECANAENCLK		ECAN-A clock enable
		0	The eCAN-A module is not clocked. (default) (1)
		1	The eCAN-A module is clocked (SYSCLKOUT/2).
13-11	Reserved		Reserved

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

Bit	Field	Value	Description
10	SCIAENCLK		SCI-A clock enable
		0	The SCI-A module is not clocked. (default) (1)
		1	The SCI-A module is clocked by the low-speed clock (LSPCLK).
9	SPIBENCLK		SPI-B clock enable
		0	The SPI-B module is not clocked. (default) (1)
		1	The SPI-B module is clocked by the low-speed clock (LSPCLK).
8	SPIAENCLK		SPI-A clock enable
		0	The SPI-A module is not clocked. (default) (1)
		1	The SPI-A module is clocked by the low-speed clock (LSPCLK).
7-5	Reserved		Any writes to these bit(s) must always have a value of 0.
4	I2CAENCLK		I^2C clock enable
		0	The I^2C module is not clocked. (default) (1)
		1	The I^2C module is clocked.
3	ADCENCLK		ADC clock enable
		0	The ADC is not clocked. (default) (1)
		1	The ADC module is clocked
2	TBCLKSYNC		ePWM Module Time Base Clock (TBCLK) Sync: Allows the user to globally synchronize all enabled ePWM modules to the time base clock (TBCLK):
		0	The TBCLK (Time Base Clock) within each enabled ePWM module is stopped. (default). If, however, the ePWM clock enable bit is set in the PCLKCR1 register, then the ePWM module will still be clocked by SYSCLKOUT even if TBCLKSYNC is 0.
		1	All enabled ePWM module clocks are started with the first rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescaler bits in the TBCTL register of each ePWM module must be set identically. The proper procedure for enabling ePWM clocks is as follows: • Enable ePWM module clocks in the PCLKCR1 register. • Set TBCLKSYNC to 0. • Configure prescaler values and ePWM modes. • Set TBCLKSYNC to 1.
1	LINAENCLK		LIN-A clock enable.
		0	The LIN-A clock is not enabled.
		1	The LIN-A clock is enabled.
0	HRPWMENCLK		HRPWM clock enable
		0	HRPWM is not enabled.
		1	HRPWM is enabled.

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

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

15	14	13	9	8
Reserved	EQEP1 ENCLK	Reserved		ECAP1 ENCLK
R-0	R/W-0	R-0		R/W-0

7	6	5	4	3	2	1	0
Reserved	EPWM7 ENCLK	EPWM6 ENCLK	EPWM5 ENCLK	EPWM4 ENCLK	EPWM3 ENCLK	EPWM2 ENCLK	EPWM1 ENCLK
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

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

Bits	Field	Value	Description [1]
15	Reserved		Any writes to these bit(s) must always have a value of 0.
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).
13-9	Reserved		Any writes to these bit(s) must always have a value of 0.
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	Reserved		Any writes to these bit(s) must always have a value of 0.
6	EPWM7ENCLK		ePWM7 clock enable. [3]
		0	The ePWM7 module is not clocked. (default) [2]
		1	The ePWM7 module is clocked by the system clock (SYSCLKOUT).
5	EPWM6ENCLK		ePWM6 clock enable. [3]
		0	The ePWM6 module is not clocked. (default) [2]
		1	The ePWM6 module is clocked by the system clock (SYSCLKOUT).
4	EPWM5ENCLK		ePWM5 clock enable [3]
		0	The ePWM5 module is not clocked. (default) [2]
		1	The ePWM5 module is clocked by the system clock (SYSCLKOUT).
3	EPWM4ENCLK		ePWM4 clock enable. [3]
		0	The ePWM4 module is not clocked. (default) [2]
		1	The ePWM4 module is clocked by the system clock (SYSCLKOUT).
2	EPWM3ENCLK		ePWM3 clock enable. [3]
		0	The ePWM3 module is not clocked. (default) [2]
		1	The ePWM3 module is clocked by the system clock (SYSCLKOUT).
1	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).

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

Bits	Field	Value	Description (1)
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).

(1) This register is EALLOW protected. See Section 1.5.2 for more information.
(2) If a peripheral block is not used, the clock to that peripheral can be turned off to minimize power consumption.
(3) To start the ePWM Time-base clock (TBCLK) within the ePWM modules, the TBCLKSYNC bit in PCLKCR0 must also be set.

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

15	10	9	8
Reserved		HRCAP2 ENCLK	HRCAP1 ENCLK
R-0		R-0	R-0

7	0
Reserved	
R-0	

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

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

Bit	Field	Value	Description
15-10	Reserved		Any writes to these bit(s) must always have a value of 0.
9	HRCAP2ENCLK		HRCAP2 Clock Enable
		0	The HRCAP2 module is not clocked. (default)
		1	The HRCAP2 module is clocked.
8	HRCAP1ENCLK		HRCAP1 Clock Enable
		0	The HRCAP1 module is not clocked. (default)
		1	The HRCAP1 module is clocked.
7-0	Reserved	0	Reserved for ETPWM9 to ETPWM16

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

15	14	13	12	11	10	9	8
Reserved	CLA1 ENCLK	Reserved	Reserved		CPUTIMER2 ENCLK	CPUTIMER1 ENCLK	CPUTIMER0 ENCLK
R-0	R/W-0	R-1	R-0		R/W-1	R/W-1	R/W-1
7				3	2	1	0
Reserved					COMP3 ENCLK	COMP2 ENCLK	COMP1 ENCLK
R-0					R/W-0	R/W-0	R/W-0

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

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

Bit	Field	Value	Description
15	Reserved		Any writes to these bit(s) must always have a value of 0.
14	CLA1ENCLK		CLA Module Clock Enable
		0	CLA Module is not clocked.
		1	CLA Module is clocked.
13	Reserved	1	This bit should always be written as 1.
12-11	Reserved		Any writes to these bit(s) must always have a value of 0.
10	CPUTIMER2ENCLK		CPU Timer 2 Clock Enable
		0	CPU Timer 2 is not clocked.
		1	CPU Timer 2 is clocked.
9	CPUTIMER1ENCLK		CPU Timer 1 Clock Enable
		0	CPU Timer 1 is not clocked.
		1	CPU Timer 1 is clocked.
8	CPUTIMER0ENCLK		CPU Timer 0 Clock Enable
		0	CPU Timer 0 is not clocked.
		1	CPU Timer 0 is clocked.
7-3	Reserved		Any writes to these bit(s) must always have a value of 0.
2	COMP3ENCLK		Comparator3 clock enable
		0	Comparator3 is not clocked
		1	Comparator3 is clocked
1	COMP2ENCLK		Comparator2 clock enable
		0	Comparator2 is not clocked
		1	Comparator2 is clocked
0	COMP1ENCLK		Comparator1 clock enable
		0	Comparator1 is not clocked
		1	Comparator1 is clocked

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

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

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

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

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

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

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

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

1.3.2 OSC and PLL Block

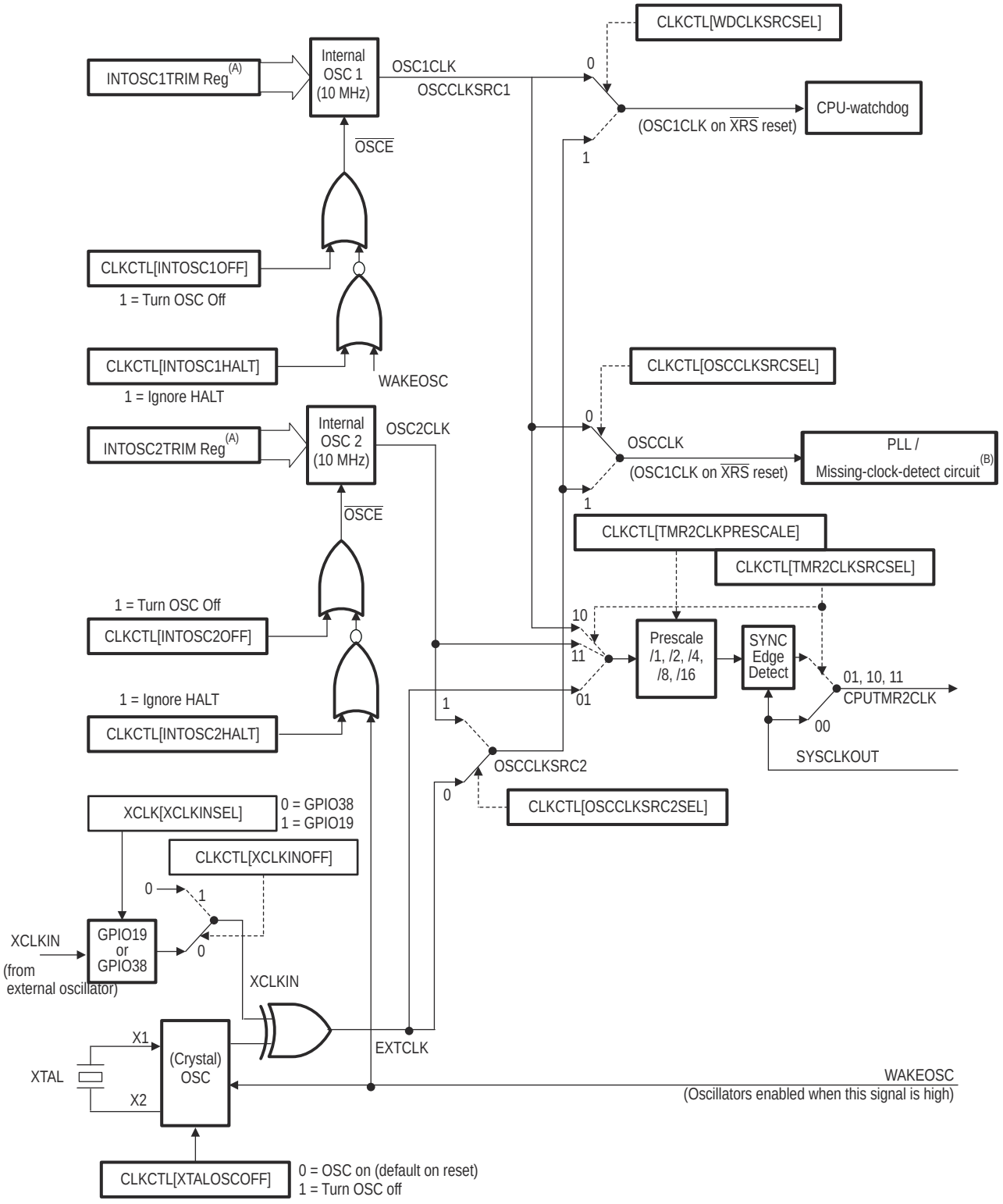
The on-chip oscillator and phase-locked loop (PLL) block provide the clocking signals for the device, as well as control for low-power mode (LPM) entry or exit.

1.3.2.1 Input Clock Options

The device has two internal oscillators (INTOSC1 and INTOSC2) that need no external components. It also has an on-chip, PLL-based clock module. Figure 1-19 shows the different options that are available to clock the device. Following are the input clock options available:

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

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



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

Figure 1-19. Clocking Options

1.3.2.1.1 Trimming INTOSCn

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

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

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

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

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

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

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

1.3.2.1.2 Device_Cal

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

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

Note

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

Step 1: Create a pointer to the Device_cal function as shown in Example 1-1. This #define is included in the Header Files and Peripheral Examples.

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

Example 1-1. Calling the Device_cal() function

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

1.3.2.2 Configuring Input Clock Source and XCLKOUT Options (XCLK)

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

Figure 1-21. Clocking (XCLK) Register

15	8
Reserved	
R-0	

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

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

Table 1-21. Clocking (XCLK) Field Descriptions

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

(1) The XCLKINSEL bit in the XCLK register is reset by $\overline{\text{XRS}}$ input signal.
(2) Refer to the device data sheet for the maximum permissible XCLKOUT frequency.

1.3.2.3 Configuring Device Clock Domains (CLKCTL)

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

Figure 1-22. Clock Control (CLKCTL) Register

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

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

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

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

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

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

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

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

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

1.3.2.3.1 Switching the Input Clock Source

The following procedure may be used to switch clock sources:

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

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

1.3.2.3.2 Switching to INTOSC2 in the Absence of External Clocks

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

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

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

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

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

The system initialization file DSP2803x_SysCtrl.c, provided as part of *C2000Ware* also contain functions to switch to different clock sources. If an attempt is made to switch from INTOSC1 to INTOSC2 without the write to the XTALOSCOFF and XCLKINOFF bits, a missing clock will be detected due to the absence of external clock source (even after the proper source selection). The PLLCR will be zeroed out and the device will automatically clear the MCLKSTS bit and switch back INTOSC1.

1.3.2.4 PLL-based Clock Module

Figure 1-23 shows the OSC and PLL block diagram.

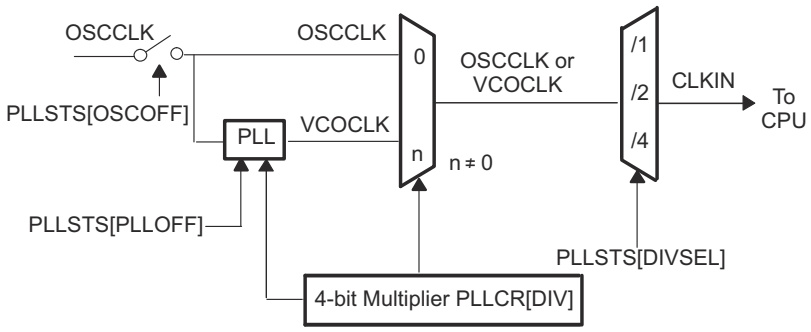


Figure 1-23. OSC and PLL Block

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

Table 1-23. Possible PLL Configuration Modes

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

(1) PLLSTS[DIVSEL] must be 0 before writing to the PLLCR and should be changed only after PLLSTS[PLLLOCKS] = 1. See Figure 1-24.
(2) The input clock and PLLCR[DIV] bits should be chosen in such a way that the output frequency of the PLL (VCOCLK) is a minimum of 50 MHz.

1.3.2.4.1 PLL Control Registers

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

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

Once the PLL is stable and has locked at the new specified frequency, the PLL switches CLKIN to the new value as shown in Table 1-24. When this happens, the PLLLOCKS bit in the PLLSTS register is set, indicating that the PLL has finished locking and the device is now running at the new frequency. User software can monitor the PLLLOCKS bit to determine when the PLL has completed locking. Once PLLSTS[PLLLOCKS] = 1, DIVSEL can be changed.

Follow the procedure in Figure 1-24 any time you are writing to the PLLCR register.

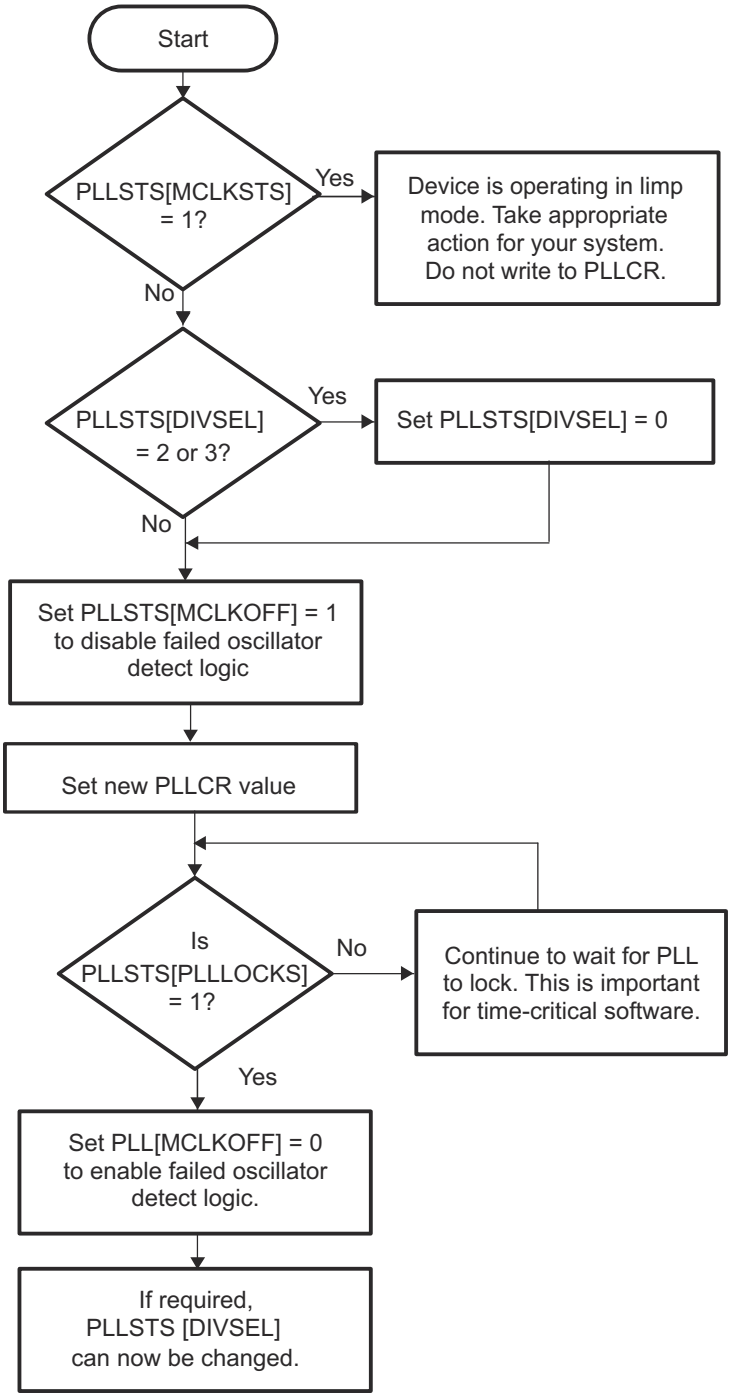


Figure 1-24. PLLCR Change Procedure Flow Chart

1.3.2.4.1.1 PLL Control Register (PLLCR)

The DIV field in the PLLCR register controls whether the PLL is bypassed or not and sets the PLL clocking ratio when it is not bypassed. PLL bypass is the default mode after reset. Do not write to the DIV field if the PLLSTS[DIVSEL] bit is 10 or 11, or if the PLL is operating in limp mode as indicated by the PLLSTS[MCLKSTS] bit being set. See the procedure for changing the PLLCR described in Figure 1-24.

Figure 1-25. PLL Control Register (PLLCR)

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

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

Table 1-24. PLL Settings

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

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

(2) 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.

1.3.2.4.1.2 PLL Status (PLLSTS) Register

Figure 1-26. PLL Status (PLLSTS) Register

15	14	9	8
NORMRDYE	Reserved		DIVSEL
R/W-0	R-0		R/W-0

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

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

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

Bits	Field	Value	Description (1) (2)
15	NORMRDYE		NORMRDY Enable Bit: This bit selects if NORMRDY signal from VREG gates the PLL from turning on when the VREG is out of regulation. It may be required to keep the PLL off while coming in and out of HALT mode and this signal can be used for that purpose:
		0	NORMRDY signal from VREG does not gate PLL (PLL ignores NORMRDY)
		1	NORMRDY signal from VREG will gate PLL (PLL off when NORMRDY low)
			The NORMRDY signal from the VREG is low when the VREG is out of regulation and this signal will go high if the VREG is within regulation.
14-9	Reserved		Any writes to these bit(s) must always have a value of 0.
8:7	DIVSEL		Divide Select: This bit selects between /4, /2, and /1 for CLKIN to the CPU. The configuration of the DIVSEL bit is as follows:
		00, 01	Select Divide By 4 for CLKIN
		10	Select Divide By 2 for CLKIN
		11	Select Divide By 1 for CLKIN
6	MCLKOFF		Missing clock-detect off bit
		0	Main oscillator fail-detect logic is enabled. (default)
		1	Main oscillator fail-detect logic is disabled and the PLL will not issue a limp-mode clock. Use this mode when code must not be affected by the detection circuit. For example, if external clocks are turned off.
5	OSCOFF		Oscillator Clock Off Bit
		0	The OSCCLK signal from X1, 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.

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

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

1.3.2.4.1.3 PLL Lock Period (PLLLOCKPRD) Register

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

15	0
PLLLOCKPRD	
R/W-FFFFh	

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

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

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

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

1.3.2.5 Input Clock Fail Detection

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

Two counters are used to monitor the presence of the OSCCLK signal as shown in Figure 1-28. The first counter is incremented by the OSCCLK signal itself. When the PLL is not turned off, the second counter is incremented by the VCOCLK coming out of the PLL block. These counters are configured such that when the 7-bit OSCCLK counter overflows, it clears the 13-bit VCOCLK counter. In normal operating mode, as long as OSCCLK is present, the VCOCLK counter will never overflow.

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

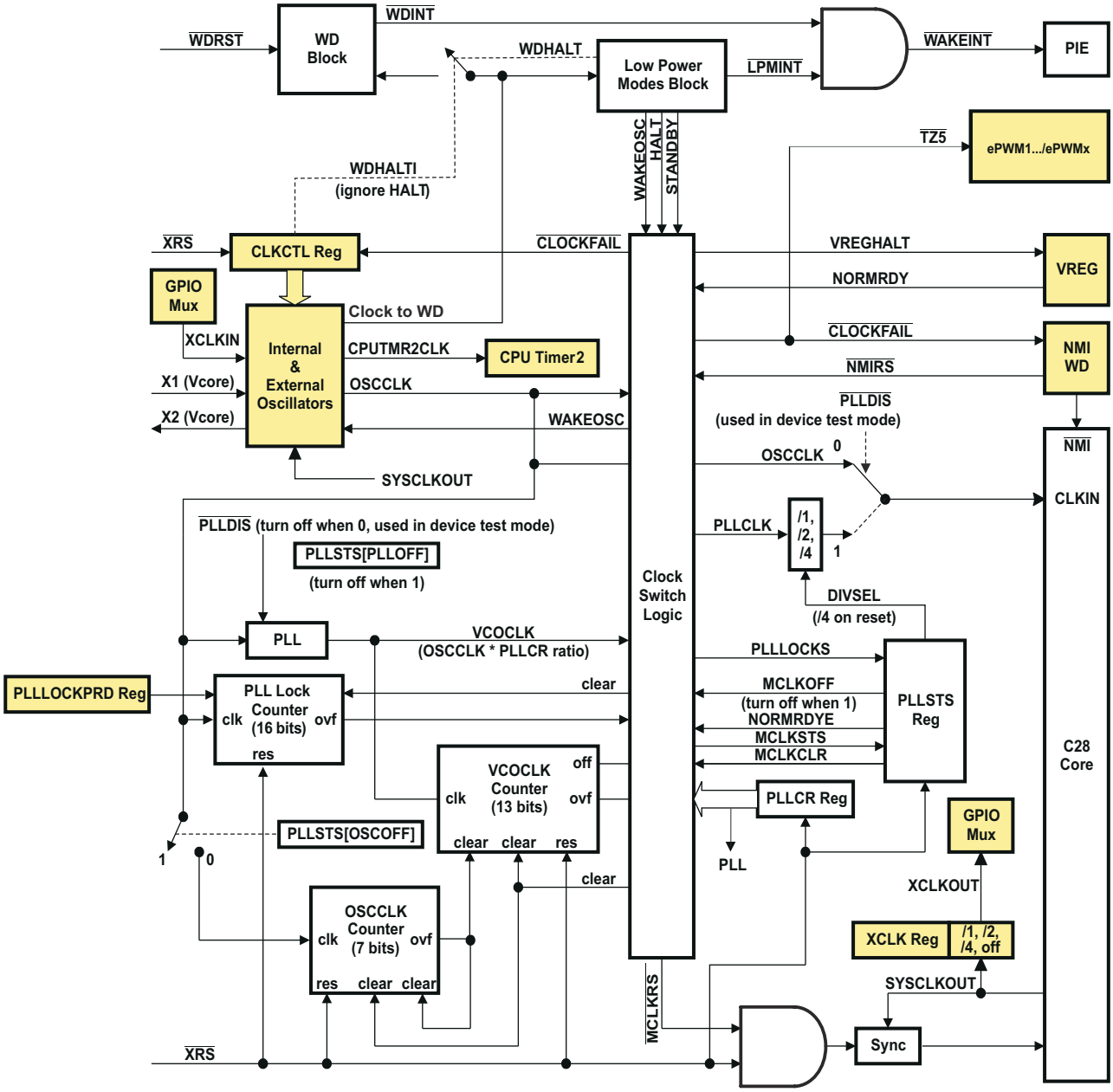


Figure 1-28. Clocking and Reset Logic

Case A:

INTOSC1 is used as the clock source. NMIWD is disabled (NMIRESETSEL = 0)

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

Case B:

INTOSC1 is used as the clock source. NMIWD is enabled (NMIRESETSEL = 1)

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

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

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

Case C:

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

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

Case D:

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

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

1.3.2.6 Missing Clock Reset and Missing Clock Status

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

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

Note

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

The following precautions and limitations should be kept in mind:

- **Use the proper procedure when changing the PLL Control Register.** Always follow the procedure outlined in Figure 1-24 when modifying the PLLCR register.
- **Do not write to the PLLCR register when the device is operating in limp mode.** When writing to the PLLCR register, the device switches to the CPU's CLKIN input to OSCCLK/2. When operating after limp mode has been detected, OSCCLK may not be present and the clocks to the system will stop. Always check that the PLLSTS[MCLKSTS] bit = 0 before writing to the PLLCR register as described in Figure 1-24.
- **Do not enter HALT low power mode when the device is operating in limp mode.** If you try to enter HALT mode when the device is already operating in limp mode then the device may not properly enter HALT. The device may instead enter STANDBY mode or may hang and you may not be able to exit HALT mode. For this reason, always check that the PLLSTS[MCLKSTS] bit = 0 before entering HALT mode.

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

- **PLL by-pass mode**

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

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

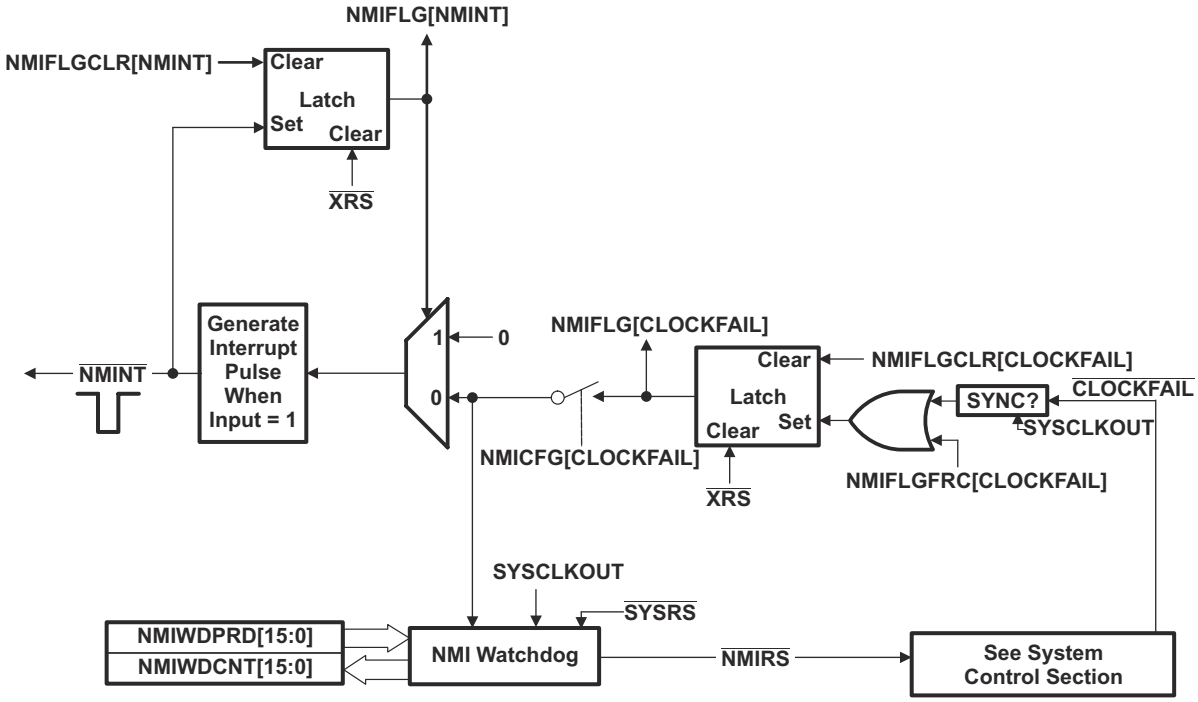
  In HALT low power mode, all of the clocks to the device are turned off. When the device comes out of HALT mode, the oscillator and PLL will power up. The counters that are used to detect a missing input clock (VCOCLK and OSCCLK) will be enabled only after this power-up has completed. If VCOCLK counter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

1.3.2.7 NMI Interrupt and Watchdog

The NMI watchdog (NMIWD) is used to detect and aid in recovery from a clock failure condition. The NMI interrupt enables the monitoring of a clock failure. In 280x/2833x/2823x devices, when the VCOCLK counter overflows (due to loss of input clock), a missing clock condition is detected and a missing-clock-reset ($\overline{\text{MCLKRS}}$) is generated immediately. In this device, a $\overline{\text{CLOCKFAIL}}$ signal is generated first, which is then applied to the NMI watchdog circuit and a reset can be generated after a preprogrammed delay. Alternatively, an interrupt can be asserted. This feature is not enabled upon power-up, however. That is, when this device first powers up, the $\overline{\text{MCLKRS}}$ signal is generated immediately upon a clock failure like on 280x/2833x/2823x devices. The user must enable the generation of the NMI signal with the CLKCTL[NMIRESETSEL] bit. Note that the NMI watchdog is different from the watchdog described in Section 1.3.4.

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

The $\overline{\text{CLOCKFAIL}}$ signal could also be used to activate the TZ5 signal to drive the PWM pins into a high impedance state. This allows the PWM outputs to be tripped in case of clock failure. Figure 1-29 shows the CLOCKFAIL interrupt mechanism. Likewise, TZ6 is connected to EMUSTOP output from the CPU. This allows the user to configure trip action during a CPU halt, such as during emulation or debug sessions.



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

Figure 1-29. Clock Fail Interrupt

1.3.2.7.1 NMI Interrupt Registers

The NMI Interrupt support registers are listed in Table 1-27.

Table 1-27. NMI Interrupt Registers

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

1.3.2.7.1.1 NMI Configuration (NMICFG) Register

Figure 1-30. NMI Configuration (NMICFG) Register Bit Definitions (EALLOW)

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

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

Table 1-28. NMI Configuration (NMICFG) Register Bit Definitions (EALLOW)

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

1.3.2.7.1.2 NMI Flag (NMIFLG) Register

Figure 1-31. NMI Flag (NMIFLG) Register Bit Definitions (EALLOW Protected)

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

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

Table 1-29. NMI Flag (NMIFLG) Register Bit Definitions (EALLOW Protected)

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

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

1.3.2.7.1.3 NMI Flag Clear (NMIFLGCLR) Register

Figure 1-32. NMI Flag Clear (NMIFLGCLR) Register Bit Definitions (EALLOW Protected)

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

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

Table 1-30. NMI Flag Clear (NMIFLGCLR) Register Bit Definitions (EALLOW Protected)

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

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

1.3.2.7.1.4 NMI Flag Force (NMIFLGFRC) Register

Figure 1-33. NMI Flag Force (NMIFLGFRC) Register Bit Definitions (EALLOW Protected)

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

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

Table 1-31. NMI Flag Force (NMIFLGFRC) Register Bit Definitions (EALLOW Protected)

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

1.3.2.7.1.5 NMI Watchdog Counter (NMIWDCNT) Register

Figure 1-34. NMI Watchdog Counter (NMIWDCNT) Register Bit Definitions

15	0
NMIWDCNT	
R-0	

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

Table 1-32. NMI Watchdog Counter (NMIWDCNT) Register Bit Definitions

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

1.3.2.7.1.6 NMI Watchdog Period (NMIWDPRD) Register

Figure 1-35. NMI Watchdog Period (NMIWDPRD) Register Bit Definitions (EALLOW Protected)

15	0
NMIWDPRD	
R/W-0xFFFF	

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

Table 1-33. NMI Watchdog Period (NMIWDPRD) Register Bit Definitions (EALLOW Protected)

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

1.3.2.7.2 NMI Watchdog Emulation Considerations

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

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

1.3.2.8 XCLKOUT Generation

The XCLKOUT signal is directly derived from the system clock SYSCLKOUT as shown in Figure 1-36. XCLKOUT can be either equal to, one-half, or one-fourth of SYSCLKOUT. By default, at power-up, XCLKOUT = SYSCLKOUT/4 or XCLKOUT = OSCCLK/16.

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

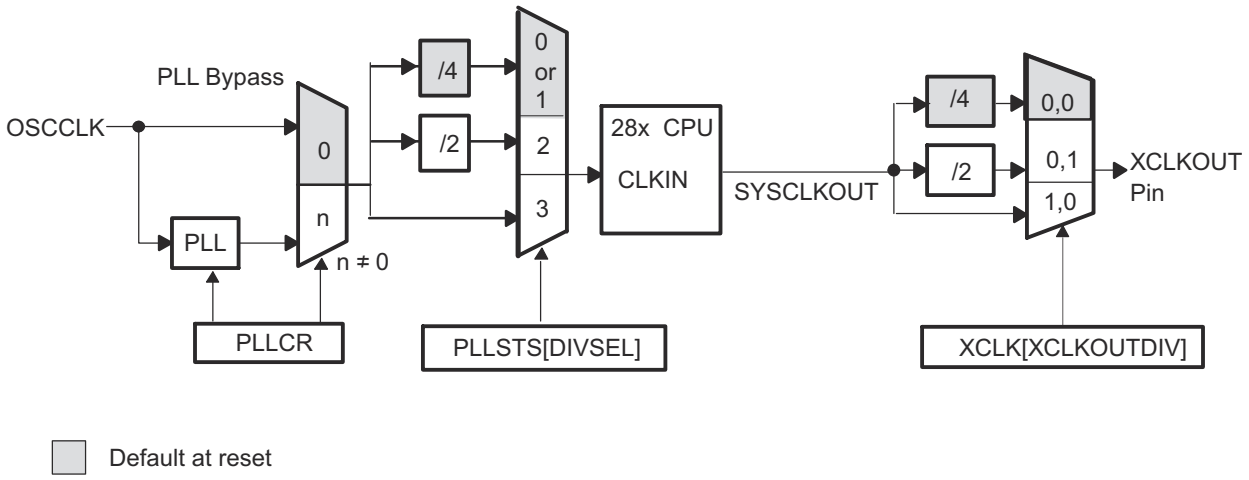


Figure 1-36. XCLKOUT Generation

1.3.2.9 External Reference Oscillator Clock Option

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

1.3.3 Low-Power Modes Block

Table 1-34 summarizes the various modes. The various low-power modes operate as shown in Table 1-35.

See the *TMS320F2803x Real-Time Microcontrollers Data Manual* for exact timing for entering and exiting the low power modes.

The low-power modes are controlled by the LPMCR0 register described in Section 1.3.3.1.

Table 1-34. Low-Power Mode Summary

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

(1) The Exit column lists which signals or under what conditions the low power mode is exited. This signal must be kept low long enough for an interrupt to be recognized by the device. Otherwise the IDLE mode is not exited and the device goes back into the indicated low power mode.
(2) On the 28x, the JTAG port can still function even if the clock to the CPU (CLKIN) is turned off.

Table 1-35. Low Power Modes

Mode	Description
IDLE Mode:	This mode is exited by any enabled interrupt. The LPM block itself performs no tasks during this mode.
STANDBY Mode:	If the LPM bits in the LPMCR0 register are set to 01, the device enters STANDBY mode when the IDLE instruction is executed. In STANDBY mode the clock input to the CPU (CLKIN) is disabled, which disables all clocks derived from SYSCLKOUT. The oscillator and PLL and watchdog will still function. Before entering the STANDBY mode, you should perform the following tasks: • Enable the WAKEINT interrupt in the PIE module. This interrupt is connected to both the watchdog and the low power mode module interrupt. • If desired, specify one of the GPIO port A signals to wake the device in the GPIOLPMSEL register. The GPIOLPMSEL register is part of the GPIO module. In addition to the selected GPIO signal, the $\overline{\text{XRS}}$ input and the watchdog interrupt, if enabled in the LPMCR0 register, can wake the device from the STANDBY mode. • Select the input qualification in the LPMCR0 register for the signal that will wake the device. When the selected external signal goes low, it must remain low a number of OSCCLK cycles as specified by the qualification period in the LPMCR0 register. If the signal should be sampled high during this time, the qualification will restart. At the end of the qualification period, the PLL enables the CLKIN to the CPU and the WAKEINT interrupt is latched in the PIE block. The CPU then responds to the WAKEINT interrupt if it is enabled.
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. 1. For device to exit HALT mode properly, the following conditions must be met: Bit 7 (INT1.8) of PIEIER1 register should be 1. Bit 0 (INT1) of IER register must be 1. 2. If the above conditions are met, a. WAKE_INT ISR will be executed first, followed by the instructions after IDLE, if INTM = 0. b. WAKE_INT ISR will not be executed and instructions after IDLE will be executed, if INTM = 1. **Do not enter HALT low power mode when the device is operating in limp mode (PLLSTS[MCLKSTS] = 1).** If you try to enter HALT mode when the device is already operating in limp mode then the device may not properly enter HALT. The device may instead enter STANDBY mode or may hang and you may not be able to exit HALT mode. For this reason, always check that the PLLSTS[MCLKSTS] bit = 0 before entering HALT mode. When the selected external signal goes low, it is fed asynchronously to the LPM block. The oscillator is turned on and begins to power up. You must hold the signal low long enough for the oscillator to complete power up. When the signal is held low for enough time and driven high, this will asynchronously release the PLL and it will begin to lock. Once the PLL has locked, it feeds the CLKIN to the CPU at which time the CPU responds to the WAKEINT interrupt if enabled.

1.3.3.1 Low Power Mode Control 0 Register (LPMCR0)

Figure 1-37. Low Power Mode Control 0 Register (LPMCR0)

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

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

Table 1-36. Low Power Mode Control 0 Register (LPMCR0) Field Descriptions

Bits	Field	Value	Description (1)
15	WDINTE		Watchdog interrupt enable
		0	The watchdog interrupt is not allowed to wake the device from STANDBY. (default)
		1	The watchdog is allowed to wake the device from STANDBY. The watchdog interrupt must also be enabled in the SCSR Register.
14-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7-2	QUALSTDBY		Select number of OSCCLK clock cycles to qualify the selected GPIO inputs that wake the device from STANDBY mode. This qualification is only used when in STANDBY mode. The GPIO signals that can wake the device from STANDBY are specified in the GPIOLPMSEL register.
		000000	2 OSCCLKs
		000001	3 OSCCLKs
		. . .	. . .
		111111	65 OSCCLKs (default)
1-0	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.

1.3.3.2 Options for Automatic Wakeup in Low-power Modes

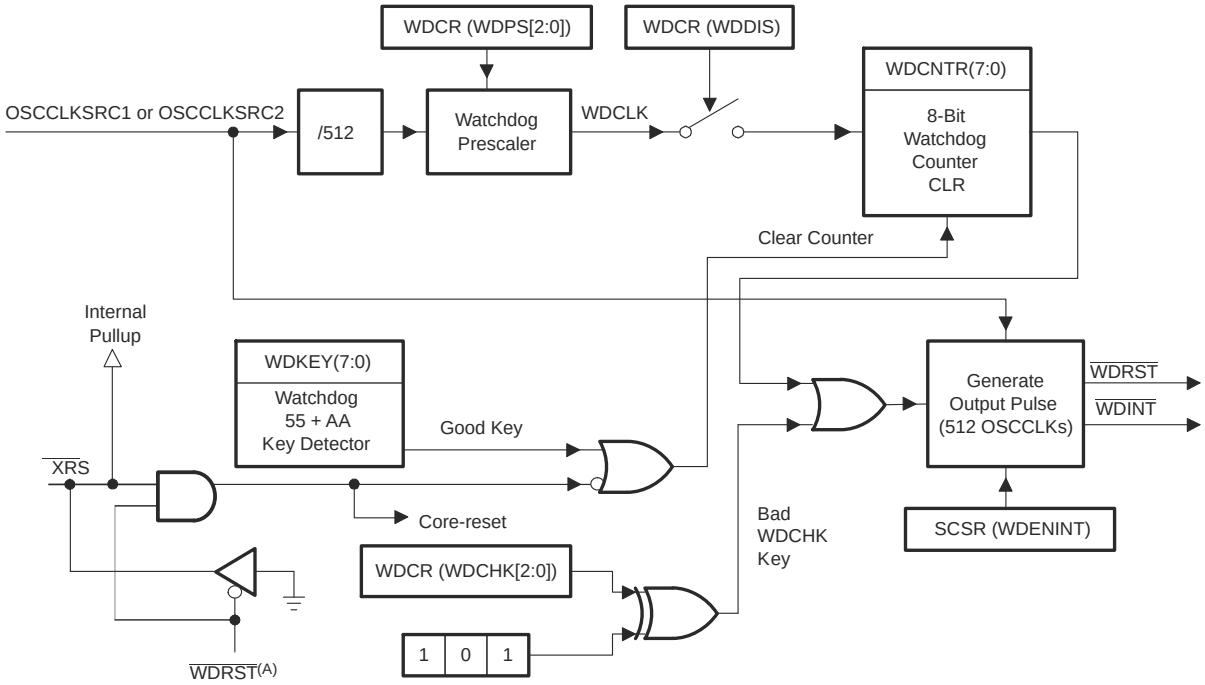
The device provides two options to automatically wake up from HALT and STANDBY modes, without the need for an external stimulus:

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

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

1.3.4 CPU Watchdog Block

The watchdog module generates an output pulse, 512 oscillator-clocks (OSCCLK) wide whenever the 8-bit watchdog up counter has reached its maximum value. To prevent this, either disable the counter or the software must periodically write a 0x55 + 0xAA sequence into the watchdog key register, which resets the watchdog counter. Figure 1-38 shows the various functional blocks within the 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.

Figure 1-38. CPU Watchdog Module

1.3.4.1 Servicing The Watchdog Timer

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

Table 1-37. 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 1-37 is the first action that enables the WDCNTR to be reset. The WDCNTR is not actually reset until step 6. Step 8 again re-enables the WDCNTR to be reset and step 9 resets the WDCNTR. Step 10 again re-enables the WDCNTR to be reset. Writing the wrong key value to the WDKEY in step 11 causes no action, however the WDCNTR is no longer enabled to be reset and the 0xAA in step 12 now has no effect.

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

1.3.4.2 Watchdog Reset or Watchdog Interrupt Mode

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

- **Reset mode:**

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

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

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

1.3.4.3 Watchdog Operation in Low Power Modes

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

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

Note

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

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

1.3.4.4 Emulation Considerations

The watchdog module behaves as follows under various debug conditions:

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

1.3.4.5 Watchdog Registers

1.3.4.5.1 System Control and Status Register (SCSR)

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

Figure 1-39. 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 1-38. System Control and Status Register (SCSR) Field Descriptions

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

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

1.3.4.5.2 Watchdog Counter Register (WDCNTR)

Figure 1-40. 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 1-39. Watchdog Counter Register (WDCNTR) Field Descriptions

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

1.3.4.5.3 Watchdog Reset Key Register (WDKEY)

Figure 1-41. 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 1-40. Watchdog Reset Key Register (WDKEY) Field Descriptions

Bits	Field	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7-0	WDKEY	0x55 + 0xAA	Refer to Table 1-37 for examples of different WDKEY write sequences.
		Other value	Writing 0x55 followed by 0xAA to WDKEY causes the WDCNTR bits to be cleared.
			Writing any value other than 0x55 or 0xAA causes no action to be generated. If any value other than 0xAA is written after 0x55, then the sequence must restart with 0x55.
			Reads from WDKEY return the value of the WDCR register.

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

1.3.4.5.4 Watchdog Control Register (WDCR)

Figure 1-42. Watchdog Control Register (WDCR)

15							8
Reserved							
R-0							

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

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

Table 1-41. Watchdog Control Register (WDCR) Field Descriptions

Bits	Field	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	WDFLAG		Watchdog reset status flag bit.
		0	The reset was caused either by the $\overline{XRS}$ 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 ($\overline{WDRST}$) 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.
5-3	WDCHK		Watchdog check.
		0,0,0	You must ALWAYS write 1,0,1 to these bits whenever a write to this register is performed unless the intent is to reset the device via software.
		other	Writing any other value causes an immediate device reset or watchdog interrupt to be taken. Note that this happens even when watchdog module is disabled. Do not write to WDCHK bits when the watchdog module is disabled. These bits can be used to generate a software reset of the device. These three bits always read back as zero (0, 0, 0).
2-0	WDPS		Watchdog pre-scale. These bits configure the watchdog counter clock (WDCLK) rate relative to (OSCCLKSRC1 or 2)/512:
		000	WDCLK = (OSCCLKSRC1 or 2)/512/1 (default)
		001	WDCLK = (OSCCLKSRC1 or 2)/512/1
		010	WDCLK = (OSCCLKSRC1 or 2)/512/2
		011	WDCLK = (OSCCLKSRC1 or 2)/512/4
		100	WDCLK = (OSCCLKSRC1 or 2)/512/8
		101	WDCLK = (OSCCLKSRC1 or 2)/512/16
		110	WDCLK = (OSCCLKSRC1 or 2)/512/32
		111	WDCLK = (OSCCLKSRC1 or 2)/512/64

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

When the $\overline{XRS}$ line is low, the WDFLAG bit is forced low. The WDFLAG bit is only set if a rising edge on $\overline{WDRST}$ signal is detected (after synch and an 8192 SYSCLKOUT cycle delay) and the $\overline{XRS}$ signal is high. If the $\overline{XRS}$ signal is low when $\overline{WDRST}$ goes high, then the WDFLAG bit remains at 0. In a typical application, the $\overline{WDRST}$

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

1.3.5 32-Bit CPU Timers 0/1/2

This section describes the three 32-bit CPU-timers (TIMER0/1/2) shown in Figure 1-43.

The CPU Timer-0 and CPU-Timer 1 can be used in user applications. Timer 2 is reserved for device/BIOS. If the application is not using device/BIOS, then Timer 2 can be used in the application. The CPU-timer interrupt signals (TINT0, TINT1, TINT2) are connected as shown in Figure 1-44.

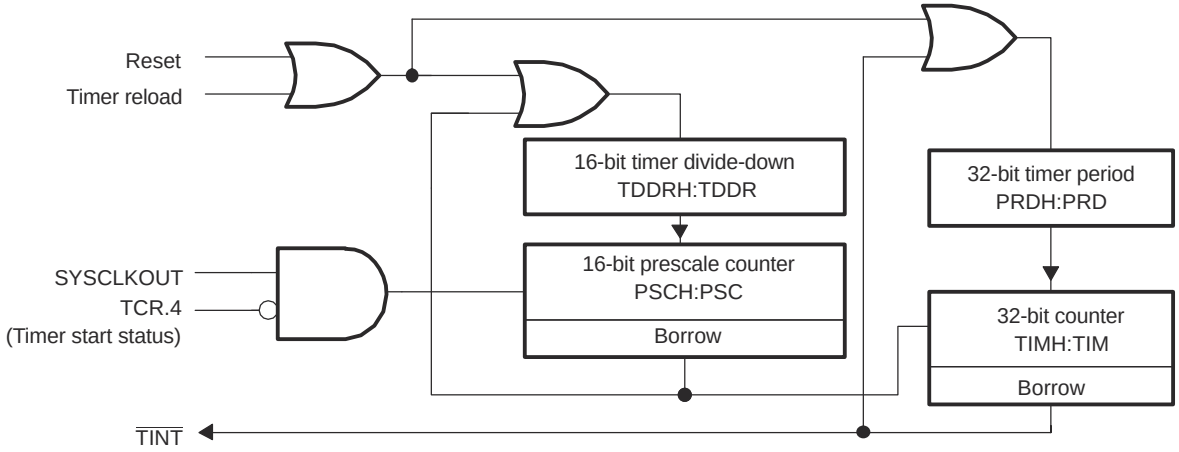
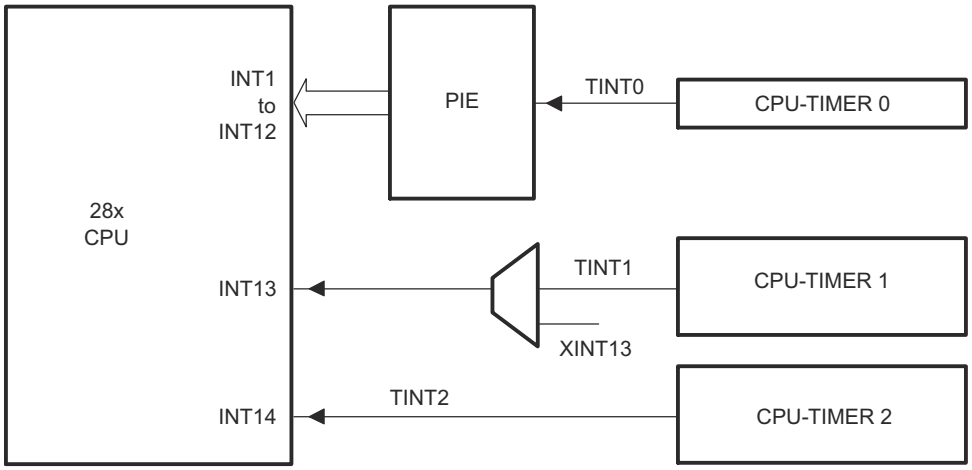


Figure 1-43. CPU-Timers



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

Figure 1-44. CPU-Timer Interrupts Signals and Output Signal

The general operation of the CPU-timer is as follows: The 32-bit counter register TIMH:TIM is loaded with the value in the period register PRDH:PRD. The counter decrements once every (TPR[TDDRH:TDDR]+1) 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 1-42 are used to configure the timers.

Table 1-42. CPU-Timers 0, 1, 2 Configuration and Control Registers

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

1.3.5.1 CPU-Timer x Counter (TIMERxTIM) Register

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

15	0
TIM	
R/W-0	

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

Table 1-43. CPU-Timer x Counter (TIMERxTIM) Register Field Descriptions

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

1.3.5.2 CPU-Timer x Counter (TIMERxTIMH) Register High

Figure 1-46. CPU-Timer x Counter (TIMERxTIMH) Register High (x = 0, 1, 2)

15	0
TIMH	
R/W-0	

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

Table 1-44. CPU-Timer x Counter (TIMERxTIMH) Register High Field Descriptions

Bits	Field	Description
15-0	TIMH	See description for TIMERxTIM (see Section 1.3.5.1).

1.3.5.3 CPU-Timer x Period (TIMERxPRD) Register

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

15	0
PRD	
R/W-1	

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

Table 1-45. CPU-Timer x Period (TIMERxPRD) Register Field Descriptions

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

1.3.5.4 CPU-Timer x Period (TIMERxPRDH) Register High

Figure 1-48. CPU-Timer x Period (TIMERxPRDH) Register High (x = 0, 1, 2)

15	0
PRDH	
R/W-0	

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

Table 1-46. CPU-Timer x Period (TIMERxPRDH) Register High Field Descriptions

Bits	Field	Description
15-0	PRDH	See description for TIMERxPRD (see Section 1.3.5.3).

1.3.5.5 CPU-Timer x Control (TIMERxTCR) Register

Figure 1-49. CPU-Timer x Control (TIMERxTCR) Register (x = 0, 1, 2)

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

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

Table 1-47. CPU-Timer x Control (TIMERxTCR) Register Field Descriptions

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

Table 1-47. CPU-Timer x Control (TIMERxTCR) Register Field Descriptions (continued)

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

1.3.5.6 CPU-Timer x Prescale (TIMERxTPR) Register

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

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

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

Table 1-48. CPU-Timer x Prescale (TIMERxTPR) Register Field Descriptions

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

1.3.5.7 CPU-Timer x Prescale (TIMERxTPRH) Register High

Figure 1-51. CPU-Timer x Prescale (TIMERxTPRH) Register High (x = 0, 1, 2)

15	8	7	0
PSCH		TDDRH	
R-0		R/W-0	

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

Table 1-49. CPU-Timer x Prescale (TIMERxTPRH) Register High Field Descriptions

Bits	Field	Description
15-8	PSCH	See description of TIMERxTPR (see Section 1.3.5.6).
7-0	TDDRH	See description of TIMERxTPR (see Section 1.3.5.6).

1.4 General-Purpose Input/Output (GPIO)

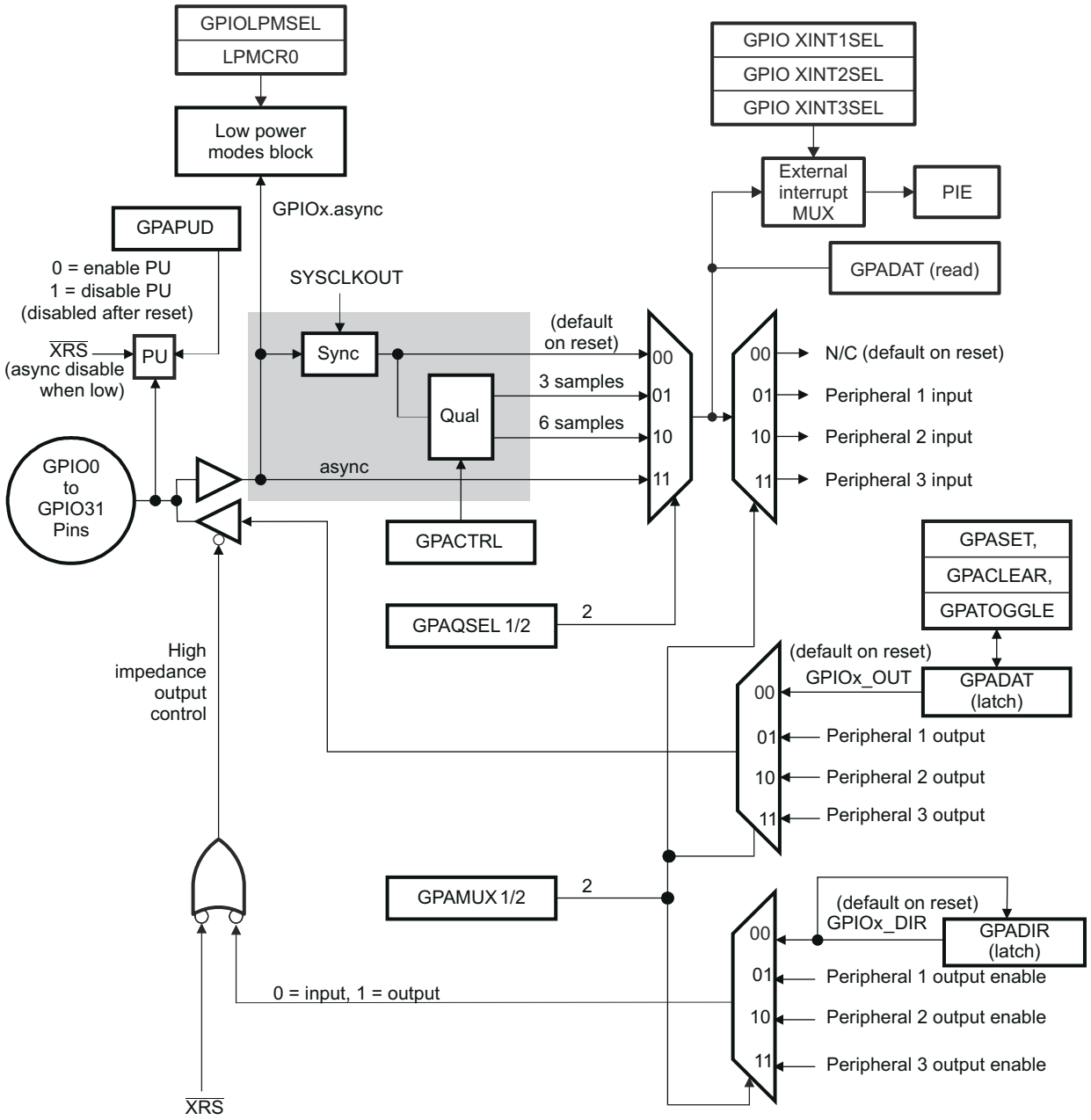
The GPIO multiplexing (MUX) registers are used to select the operation of shared pins. The pins are named by their general-purpose I/O name (that is, GPIO0 - GPIO44). These pins can be individually selected to operate as digital I/O, referred to as GPIO, or connected to one of up to three peripheral I/O signals (by the GPxMUXn registers). If selected for digital I/O mode, registers are provided to configure the pin direction (by the GPxDIR registers). You can also qualify the input signals to remove unwanted noise (by the GPxQSELn, GPACTRL, and GPBCTRL registers).

1.4.1 GPIO Module Overview

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

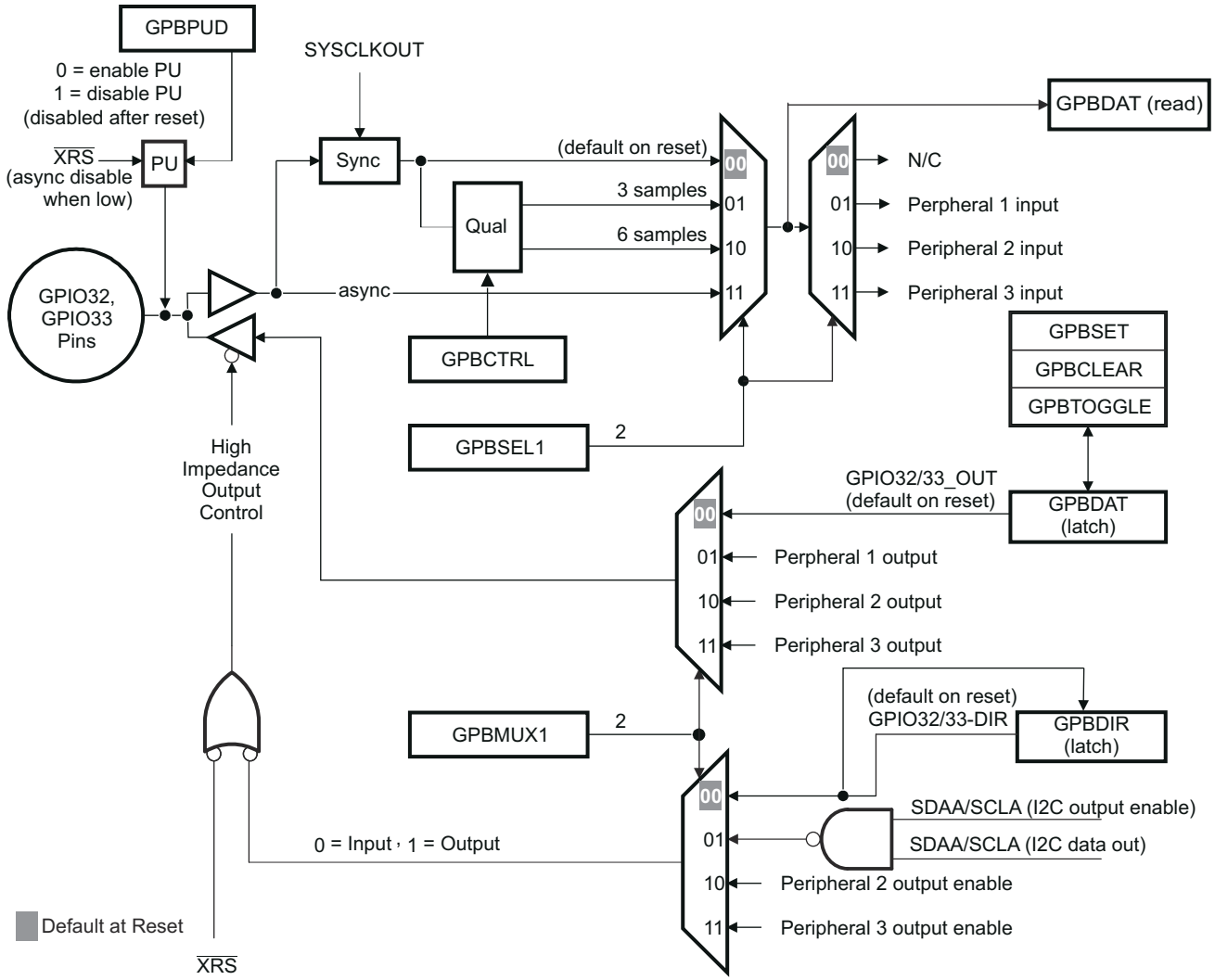
- Port A consists of GPIO0-GPIO31
- Port B consists of GPIO32-GPIO44
- Analog port consists of AIO0-AIO15

Figure 1-52 shows the basic modes of operation for the GPIO module. Note that GPIO functionality is provided on JTAG pins as well.



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

Figure 1-52. GPIO0 to GPIO31 Multiplexing Diagram



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

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

Figure 1-53. GPIO32, GPIO33 Multiplexing Diagram

1.4.1.1 JTAG Port

On this device, the JTAG port is reduced to five pins ($\overline{\text{TRST}}$, TCK, TDI, TMS, and TDO). The TCK, TDI, TMS, and TDO pins are also GPIO pins. The $\overline{\text{TRST}}$ signal selects either the JTAG or GPIO operating mode for the pins in Figure 1-54.

Note

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

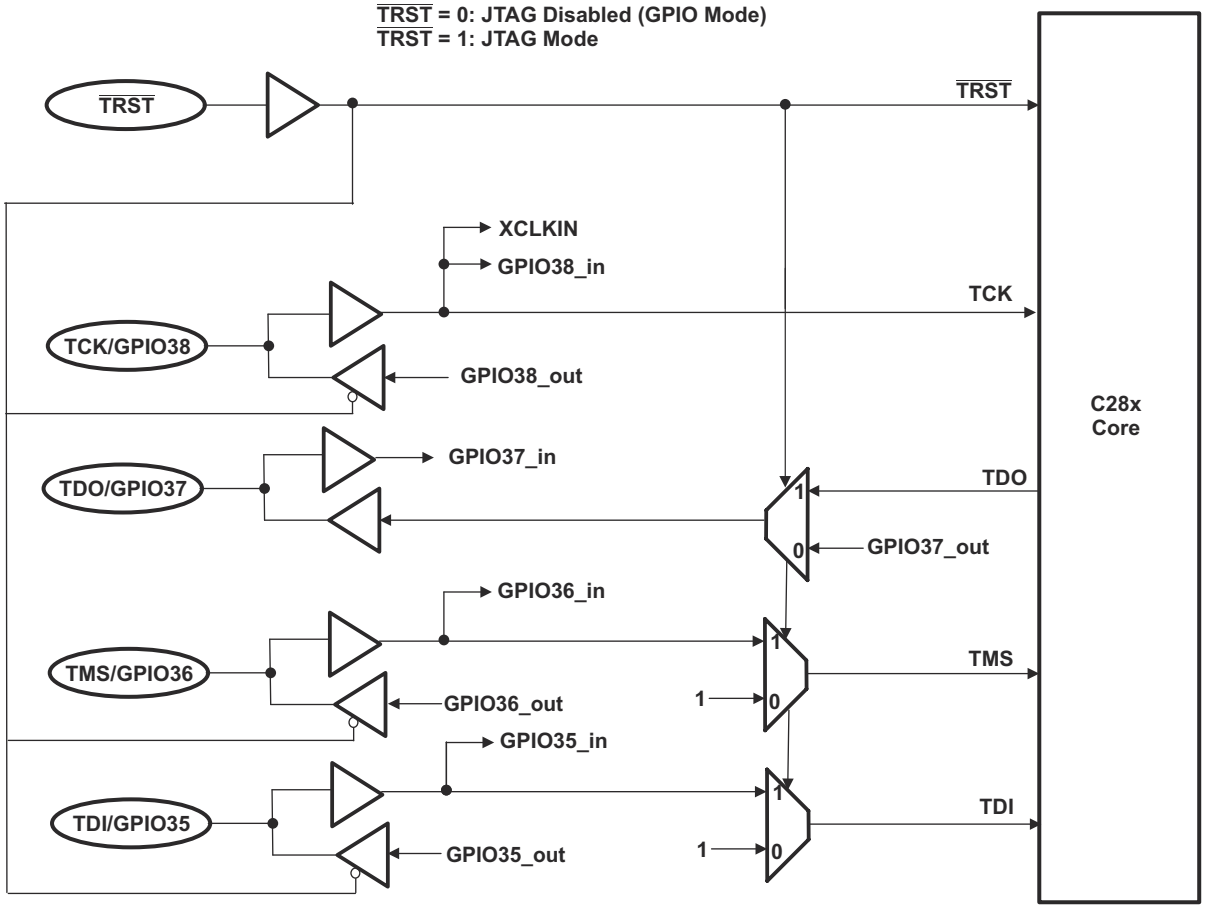


Figure 1-54. JTAG Port/GPIO Multiplexing

1.4.1.2 Choosing JTAG or GPIO Functionality

The $\overline{\text{TRST}}$ signal selects the functionality of the JTAG signals, in combination with the JTAGDIS bit in the JTAGDEBUG register as shown in Table 1-50.

Table 1-50. JTAG Port Mode

$\overline{\text{TRST}}$	JTAGDIS bit	JTAG Port Mode
0	X	GPIO mode enabled, JTAG port disabled
1	0	JTAG port enabled (GPIOs should be configured as inputs)
1	1	GPIO mode enabled, JTAG port disabled

The JTAGDEBUG register is shown in Figure 1-55 and described in Table 1-51.

Figure 1-55. JTAGDEBUG Register (Address 0x702A, EALLOW protected)

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

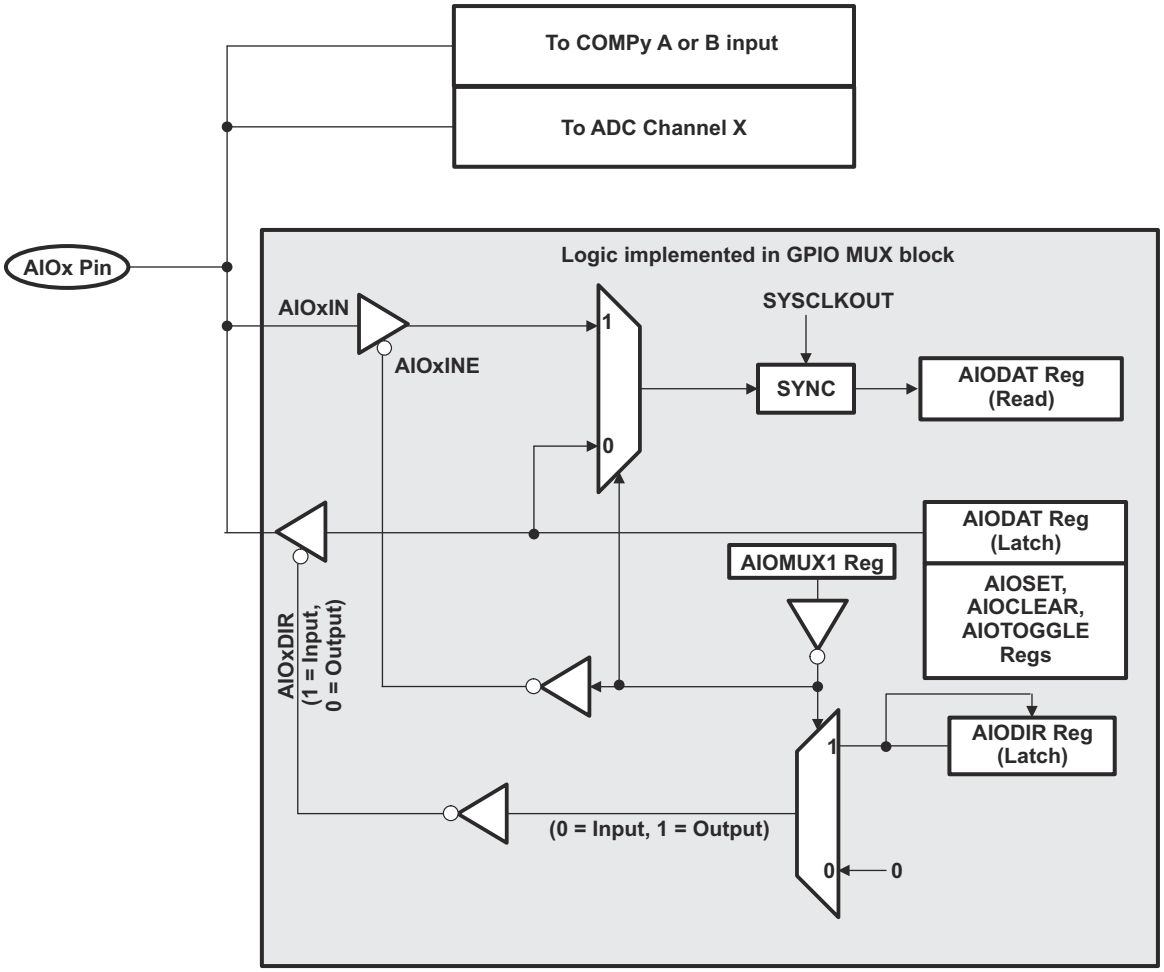
LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 1-51. JTAGDEBUG Register Field Descriptions

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

1.4.1.3 Analog IO

Specific analog pins also have the ability to function as basic IO pins controlled through software registers. See the device data sheet for the exact analog pins that have this functionality. Figure 1-56 shows how the AIO and analog function co-exist on the device.



A. The ADC/Comparator path is always enabled, irrespective of the AIOMUX1 value.
B. The AIO section is blocked off when the corresponding AIOMUX1 bit is 1.

Figure 1-56. Analog/GPIO Multiplexing

1.4.2 Configuration Overview

The pin function assignments, input qualification, and the external interrupt sources are all controlled by the GPIO configuration control registers. In addition, you can assign pins to wake the device from the HALT and STANDBY low power modes and enable/disable internal pullup resistors. Table 1-52 and Table 1-53 list the registers that are used to configure the GPIO pins to match the system requirements.

Table 1-52. GPIO Control Registers

Name (1)	Address	Size (x16)	Register Description	Bit Description
GPACTRL	0x6F80	2	GPIO A Control Register (GPIO0-GPIO31)	Section 1.4.6.1
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register (GPIO0-GPIO15)	Section 1.4.6.2
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register (GPIO16-GPIO31)	Section 1.4.6.3
GPAMUX1	0x6F86	2	GPIO A MUX 1 Register (GPIO0-GPIO15)	Section 1.4.6.4
GPAMUX2	0x6F88	2	GPIO A MUX 2 Register (GPIO16-GPIO31)	Section 1.4.6.5
GPADIR	0x6F8A	2	GPIO A Direction Register (GPIO0-GPIO31)	Section 1.4.6.6
GPAPUD	0x6F8C	2	GPIO A Pullup Disable Register (GPIO0-GPIO31)	Section 1.4.6.7
GPBCTRL	0x6F90	2	GPIO B Control Register (GPIO32-GPIO 44)	Section 1.4.6.8
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register (GPIO32-GPIO 44)	Section 1.4.6.9
GPBMUX1	0x6F96	2	GPIO B MUX 1 Register (GPIO32-GPIO 44)	Section 1.4.6.10
GPBDIR	0x6F9A	2	GPIO B Direction Register (GPIO32-GPIO 44)	Section 1.4.6.11
GPBPUD	0x6F9C	2	GPIO B Pullup Disable Register (GPIO32-GPIO44)	Section 1.4.6.12
AIOMUX1	0x6FB6	2	Analog, I/O MUX 1 register (AIO0-AIO15)	Section 1.4.6.13
AIODIR	0x6FBA	2	Analog, I/O Direction Register (AIO0-AIO15)	Section 1.4.6.14

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

Table 1-53. 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)	Section 1.4.6.15
GPIOXINT2SEL	0x6FE1	1	XINT2 Source Select Register (GPIO0-GPIO31)	Section 1.4.6.15
GPIOXINT3SEL	0x6FE2	1	XINT3 Source Select Register (GPIO0 - GPIO31)	Section 1.4.6.15
GPIOLPMSEL	0x6FE8	1	Low Power Mode Wakeup Source Select Register (GPIO0-GPIO31)	Section 1.4.6.16

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

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

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

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

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

3. **Select input qualification:**

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

   Configure the GPxMUXn or AIOMUXn registers such that the pin is a GPIO or one of three available peripheral functions. By default, all GPIO-capable pins are configured at reset as general purpose input pins.
5. **For digital general purpose I/O, select the direction of the pin:**

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

   Specify which pins, if any, will be able to wake the device from HALT and STANDBY low power modes. The pins are specified in the GPIOLPMSEL register.
7. **Select external interrupt sources:**

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

Note

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

1.4.3 Digital General-Purpose I/O Control

For pins that are configured as GPIO, the values on the pins are changed by using the registers in Table 1-54.

Table 1-54. GPIO Data Registers

Name [1]	Address	Size (x16)	Register Description	Bit Description
GPADAT	0x6FC0	2	GPIO A Data Register (GPIO0-GPIO31)	Section 1.4.6.17
GPASET	0x6FC2	2	GPIO A Set Register (GPIO0-GPIO31)	Section 1.4.6.18
GPACLEAR	0x6FC4	2	GPIO A Clear Register (GPIO0-GPIO31)	Section 1.4.6.18
GPATOGGLE	0x6FC6	2	GPIO A Toggle Register (GPIO0-GPIO31)	Section 1.4.6.18
GPBDAT	0x6FC8	2	GPIO B Data Register (GPIO32-GPIO 44)	Section 1.4.6.19
GPBSET	0x6FCA	2	GPIO B Set Register (GPIO32-GPIO 44)	Section 1.4.6.20
GPBCLEAR	0x6FCC	2	GPIO B Clear Register (GPIO32-GPIO 44)	Section 1.4.6.20
GPBTOGGLE	0x6FCE	2	GPIO B Toggle Register (GPIO32-GPIO 44)	Section 1.4.6.20
AIODAT	0x6FD8	2	Analog I/O Data Register (AIO0-AIO15)	Section 1.4.6.21
AIOSET	0x6FDA	2	Analog I/O Data Set Register (AIO0-AIO15)	Section 1.4.6.22
AIOCLEAR	0x6FDC	2	Analog I/O Clear Register (AIO0-AIO15)	Section 1.4.6.22
AIOTOGGLE	0x6FDE	2	Analog I/O Toggle Register (AIO0-AIO15)	Section 1.4.6.22

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

- **GPxDAT/AIODAT Registers**

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

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

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

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

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

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

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

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

- **GPxSET/AIOSET Registers**

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

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

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

1.4.4 Input Qualification

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

1.4.4.1 No Synchronization (asynchronous input)

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

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

1.4.4.2 Synchronization to SYSCLKOUT Only

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

1.4.4.3 Qualification Using a Sampling Window

In this mode, the signal is first synchronized to the system clock (SYSCLKOUT) and then qualified by a specified number of cycles before the input is allowed to change. Figure 1-57 and Figure 1-58 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.

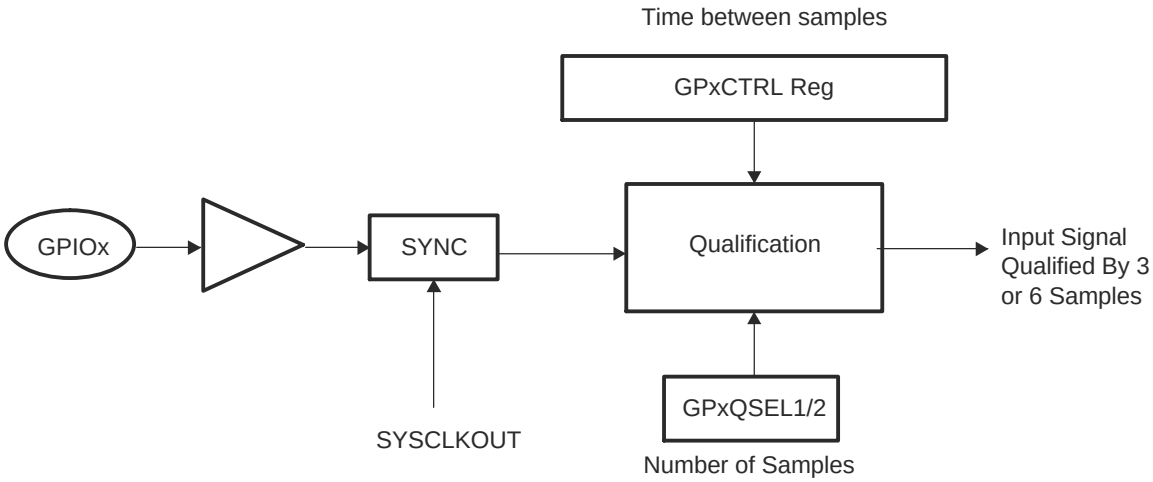


Figure 1-57. Input Qualification Using a Sampling Window

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 1-55 and Table 1-56 show the relationship between the sampling period or sampling frequency and the GPxCTRL[QUALPRDn] setting.

Table 1-55. 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 1-56. Sampling Frequency

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

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

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

Example: Minimum Sampling Frequency	
If GPxCTRL[QUALPRDn] = 0xFF (that is, 255)	then the sampling frequency is $f_{SYSCLKOUT} \times 1 \div (2 \times$ GPxCTRL[QUALPRDn])
If, for example, $f_{SYSCLKOUT}$ = 60 MHz	then the signal will be sampled at 60 MHz × 1 ÷ (2 × 255) or one sample every 8.5 µs.

Number of samples:

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

Total Sampling Window Width:

The sampling window is the time during which the input signal will be sampled as shown in Figure 1-58. 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 1-55. Likewise, for a six-sample window, the sampling window width is 5 sampling periods wide. Table 1-57 and Table 1-58 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 1-57. Case 1: Three-Sample Sampling Window Width

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

Table 1-58. Case 2: Six-Sample Sampling Window Width

	Total Sampling Window Width
If GPxCTRL[QUALPRDn] = 0	5 × $T_{SYSCLKOUT}$
If GPxCTRL[QUALPRDn] ≠ 0	5 × 2 × GPxCTRL[QUALPRDn] × $T_{SYSCLKOUT}$
	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 1-58, the input qualification has been configured as follows:

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

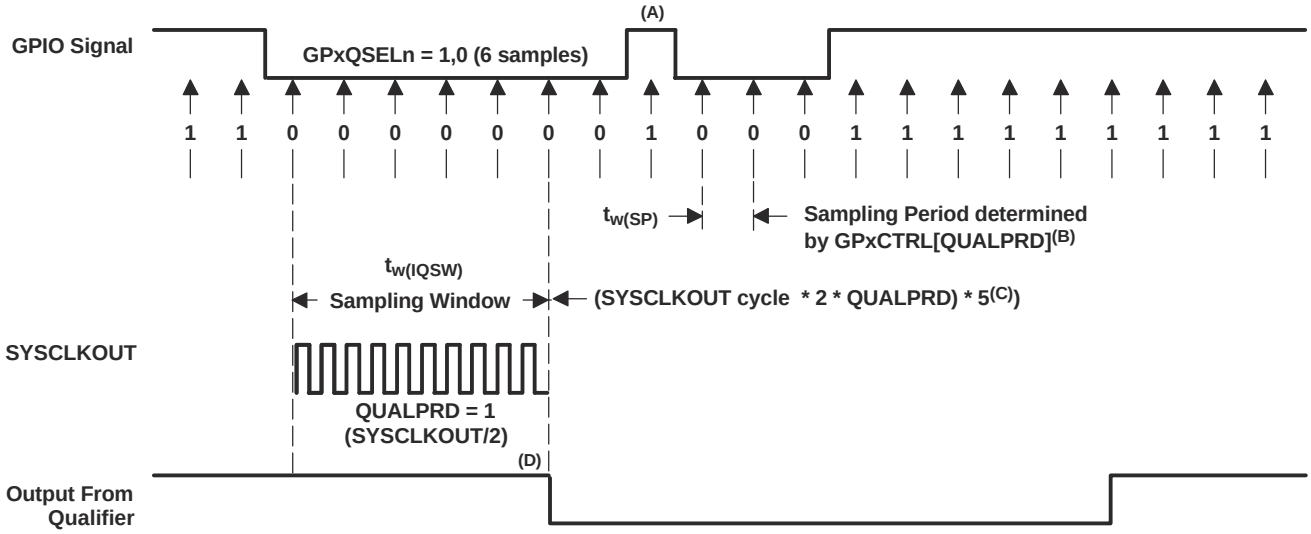
This configuration results in the following:

- The width of the sampling window is: .

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

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

  $t_w(SP) + T_{SYSCLKOUT}$ = 333.4 ns + 166.67 ns = 500.1 ns
- In Figure 1-58, the glitch (A) is shorter then the qualification window and will be ignored by the input qualifier.



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

Figure 1-58. Input Qualifier Clock Cycles

1.4.5 GPIO and Peripheral Multiplexing (MUX)

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

Table 1-60 and Table 1-61 show an overview of the possible multiplexing combinations sorted by GPIO pin. The second column indicates the I/O name of the pin on the device. Since the I/O name is unique, it is the best way to identify a particular pin. Therefore, the register descriptions in this section refer only to the GPIO name of a particular pin. The MUX register and particular bits that control the selection for each pin are indicated in the first column.

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

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

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

Note

If you should select a reserved GPIO MUX configuration that is not mapped to a peripheral, the state of the pin will be undefined and the pin may be driven. Reserved configurations are for future expansion and should not be selected. In the device MUX tables (Table 1-60 and Table 1-61), these options are indicated as Reserved.

Some peripherals can be assigned to more than one pin with the MUX registers. For example, in the 2803x device, the SPISIMOB can be assigned to either the GPIO12 or GPIO24 pin, depending on individual system requirements as:

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

If no pin is configured as an input to a peripheral or if more than one pin is configured as an input for the same peripheral, then the input to the peripheral will either default to a 0 or a 1 as shown in Table 1-59. For example, if SPISIMOB were assigned to both GPIO12 and GPIO24, the input to the SPI peripheral would default to a high state as shown in Table 1-59 and the input would not be connected to GPIO12 or GPIO24.

Table 1-59. Default State of Peripheral Input

Peripheral Input	Description	Default Input (1)
$\overline{TZ1}$- $\overline{TZ3}$	Trip zone 1-3	1
EPWMSYNCI	ePWM Synch Input	0
ECAP1	eCAP1 input	1
EQEP1A	eQEP input	1
EQEP1I	eQEP index	1
EQEP1S	eQEP strobe	1
SPICLKA/SPICLKB	SPI-A clock	1
$\overline{SPISTEA}$ / $\overline{SPISTEB}$	SPI-A transmit enable	0
SPISIMOA/SPISIMOB	SPI-A Slave-in, master-out	1
SPISOMIA/SPISOMIB	SPI-A Slave-out, master-in	1
SCIRXDA - SCIRXDB	SCI-A - SCI-B receive	1
CANRXA	eCAN-A receive	1
SDAA	I^2C data	1
SCLA1	I^2C 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 1-60. 2803x GPIOA MUX

	Default at Reset Primary I/O Function	Peripheral Selection 1	Peripheral Selection 2	Peripheral Selection 3
GPAMUX1 Register Bits	**(GPAMUX1 bits = 00)**	**(GPAMUX1 bits = 01)**	**(GPAMUX1 bits = 10)**	**(GPAMUX1 bits = 11)**
1-0	GPIO0	EPWM1A (O)	Reserved(1)	Reserved(1)
3-2	GPIO1	EPWM1B (O)	Reserved	COMP1OUT (O)
5-4	GPIO2	EPWM2A (O)	Reserved(1)	Reserved(1)
7-6	GPIO3	EPWM2B (O)	SPISOMIA (I/O)	COMP2OUT (O)
9-8	GPIO4	EPWM3A (O)	Reserved(1)	Reserved(1)
11-10	GPIO5	EPWM3B (O)	SPISIMOA (I/O)	ECAP1 (I/O)
13-12	GPIO6	EPWM4A (O)	EPWMSYNCI (I)	EPWMSYNCO (O)
15-14	GPIO7	EPWM4B (O)	SCIRXDA (I)	Reserved
17-16	GPIO8	EPWM5A (O)	Reserved	$\overline{\text{ADCSOCAO}}$ (O)
19-18	GPIO9	EPWM5B (O)	LINTXA (O)	HRCAP1 (I)
21-20	GPIO10	EPWM6A (O)	Reserved	$\overline{\text{ADCSOCBO}}$ (O)
23-22	GPIO11	EPWM6B (O)	LINRXA (I)	HRCAP2 (I)
25-24	GPIO12	$\overline{\text{TZ1}}$ (I)	SCITXDA (O)	SPISIMOB (I/O)
27-26	GPIO13(2)	$\overline{\text{TZ2}}$(I)	Reserved	SPISOMIB (I/O)
29-28	GPIO14(2)	$\overline{\text{TZ3}}$ (I)	LINTXA (O)	SPICLKB (I/O)
31-30	GPIO15(2)	$\overline{\text{TZ1}}$ (I)	LINRXA (I)	$\overline{\text{SPISTEB}}$ (I/O)
GPAMUX2 Register Bits	**(GPAMUX2 bits = 00)**	**(GPAMUX2 bits = 01)**	**(GPAMUX2 bits = 10)**	**(GPAMUX2 bits = 11)**
1-0	GPIO16	SPISIMOA (I/O)	Reserved	$\overline{\text{TZ2}}$ (I)
3-2	GPIO17	SPISOMIA (I/O)	Reserved	$\overline{\text{TZ3}}$ (I)
5-4	GPIO18	SPICLKA (I/O)	LINTXA (O)	XCLKOUT (O)
7-6	GPIO19/XCLKIN	$\overline{\text{SPISTEA}}$ (I/O)	LINRXA (I)	ECAP1 (I/O)
9-8	GPIO20	EQEP1A (I)	Reserved	COMP1OUT (O)
11-10	GPIO21	EQEP1B (I)	Reserved	COMP2OUT (O)
13-12	GPIO22	EQEP1S (I/O)	Reserved	LINTXA (O)
15-14	GPIO23	EQEP1I (I/O)	Reserved	LINRXA (I)
17-16	GPIO24	ECAP1 (I/O)	Reserved	SPISIMOB (I/O)
19-18	GPIO25(2)	Reserved	Reserved	SPISOMIB (I/O)
21-20	GPIO26(2)	HRCAP1 (I)	Reserved	SPICLKB (I/O)
23-22	GPIO27(2)	HRCAP2 (I)	Reserved	$\overline{\text{SPISTEB}}$ (I/O)
25-24	GPIO28	SCIRXDA (I)	SDAA (I/OC)	$\overline{\text{TZ2}}$ (I)
27-26	GPIO29	SCITXDA (O)	SCLA (I/OC)	$\overline{\text{TZ3}}$ (I)
29-28	GPIO30	CANRXA (I)	Reserved	Reserved
31-30	GPIO31	CANTXA (O)	Reserved	Reserved

(1) The word Reserved means that there is no peripheral assigned to this GPxMUX1/2 register setting. Should it be selected, the state of the pin will be undefined and the pin may be driven. This selection is a reserved configuration for future expansion.
(2) These pins are not available in the 64-pin package.

Table 1-61. 2803x GPIOB MUX

GPBMUX1 Register Bits	Default at Reset Primary I/O Function (GPBMUX1 bits = 00)	Peripheral Selection 1 (GPBMUX1 bits = 01)	Peripheral Selection 2 (GPBMUX1 bits = 10)	Peripheral Selection 3 (GPBMUX1 bits = 11)
1-0	GPIO32	SDAA (I/OC)	EPWMSYNCI (I)	$\overline{\text{ADCSOCAO}}$ (O)
3-2	GPIO33	SCLA (I/OC)	EPWMSYNCO (O)	$\overline{\text{ADCSOCBO}}$ (O)
5-4	GPIO34	COMP2OUT (O)	Reserved	COMP3OUT (O)
7-6	GPIO35 (TDI)	Reserved	Reserved	Reserved
9-8	GPIO36 (TMS)	Reserved	Reserved	Reserved
11-10	GPIO37 (TDO)	Reserved	Reserved	Reserved
13-12	GPIO38/XCLKIN (TCK)	Reserved	Reserved	Reserved
15-14	GPIO39	Reserved	Reserved	Reserved
17-16	GPIO40	EPWM7A (O)	Reserved	Reserved
19-18	GPIO41	EPWM7B (O)	Reserved	Reserved
21-20	GPIO42	Reserved	Reserved	COMP1OUT (O)
23-22	GPIO43	Reserved	Reserved	COMP2OUT (O)
25-24	GPIO44	Reserved	Reserved	Reserved
27-26	Reserved	Reserved	Reserved	Reserved
29-28	Reserved	Reserved	Reserved	Reserved
31-30	Reserved	Reserved	Reserved	Reserved

Table 1-62. Analog MUX

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

1.4.6 GPIO Registers

1.4.6.1 GPIO Port A Qualification Control (GPACTRL) Register

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

Figure 1-59. 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

Table 1-63. 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.

1.4.6.2 GPIO Port A Qualification Select 1 (GPAQSEL1) Register

Figure 1-60. 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 1-64. 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 1-60.
		00	Synchronize to SYSCLKOUT only. Valid for both peripheral and GPIO pins.
		01	Qualification using 3 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		10	Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		11	Asynchronous. (no synchronization or qualification). This option applies to pins configured as peripherals only. If the pin is configured as a GPIO input, then this option is the same as 0,0 or synchronize to SYSCLKOUT.

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

1.4.6.3 GPIO Port A Qualification Select 2 (GPAQSEL2) Register

Figure 1-61. 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 1-65. 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 1-61.
		00	Synchronize to SYSCLKOUT only. Valid for both peripheral and GPIO pins.
		01	Qualification using 3 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		10	Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		11	Asynchronous. (no synchronization or qualification). This option applies to pins configured as peripherals only. If the pin is configured as a GPIO input, then this option is the same as 0,0 or synchronize to SYSCLKOUT.

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

1.4.6.4 GPIO Port A MUX 1 (GPAMUX1) Register

Figure 1-62. GPIO Port A MUX 1 (GPAMUX1) Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GPIO15		GPIO14		GPIO13		GPIO12		GPIO11		GPIO10		GPIO9		GPIO8	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO7		GPIO6		GPIO5		GPIO4		GPIO3		GPIO2		GPIO1		GPIO0	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

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

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

Bits	Field	Value	Description (1)
31-30	GPIO15		Configure the GPIO15 pin as:
		00	GPIO15 - General purpose input/output 15 (default) (I/O)
		01	$\overline{TZ1}$ - Trip Zone 1(I) . 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 $\overline{TZ1}$ function is chosen.
		10	LINRXA - LIN A receive (I)
		11	$\overline{SPISTEB}$ - SPI-B Slave transmit enable
29-28	GPIO14		Configure the GPIO14 pin as:
		00	GPIO14 - General purpose I/O 14 (default) (I/O)
		01	$\overline{TZ3}$ - Trip zone 3,
		10	LINTXA - LIN A Transmit (O)
		11	SPICLKB - SPI-B clock This option is reserved on devices that do not have an SPI-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	Reserved
		11	SPISOMIB (I/O) - SPI-B Slave Output/Master input This option is reserved on devices that do not have an SPI-B port. (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	SCITXDA - SCI-A Transmit (O)
		11	SPISIMOB (I/O) - SPI-B Slave input/Master output This option is reserved on devices that do not have an SPI-B port. (2)
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	LINRXA - LIN A receive (I)
		11	HRCAP2 (I)
21-20	GPIO10		Configure the GPIO10 pin as:
		00	GPIO10 - General purpose I/O 10 (default) (I/O)
		01	EPWM6A - ePWM6 output A (O)
		10	Reserved
		11	$\overline{ADCSOCBO}$ - 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	LINTXA - LIN-A Transmit (O)
		11	HRCAP1
17-16	GPIO8		Configure the GPIO8 pin as:
		00	GPIO8 - General purpose I/O 8 (default) (I/O)
		01	EPWM5A - ePWM5 output A (O)
		10	Reserved
		11	$\overline{ADCSOCAO}$ - ADC Start of conversion A

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

Bits	Field	Value	Description [1]
15-14	GPIO7		Configure the GPIO7 pin as:
		00	GPIO7 - General purpose I/O 7 (default) (I/O)
		01	EPWM4B - ePWM4 output B (O)
		10	SCIRXDA (I) - SCI-A receive (I)
		11	Reserved
13-12	GPIO6		Configure the GPIO6 pin as:
		00	GPIO6 - General purpose I/O 6 (default)
		01	EPWM4A - ePWM4 output A (O)
		10	EPWMSYNCI - ePWM Synch-in (I)
		11	EPWMSYNCO - ePWM Synch-out (O)
11-10	GPIO5		Configure the GPIO5 pin as:
		00	GPIO5 - General purpose I/O 5 (default) (I/O)
		01	EPWM3B - ePWM3 output B
		10	SPISIMOA (I/O) - SPI-A Slave input/Master output
		11	ECAP1 - eCAP1 (I/O)
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	SPISOMIA (I/O) - SPI-A Slave output/Master input
		11	COMP2OUT (O) - Comparator 2 output
5-4	GPIO2		Configure the GPIO2 pin as:
		00	GPIO2 (I/O) General purpose I/O 2 (default) (I/O)
		01	EPWM2A - ePWM2 output A (O)
		10	Reserved. [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	Reserved
		11	COMP1OUT (O) - Comparator 1 output
1-0	GPIO0		Configure the GPIO0 pin as:
		00	GPIO0 - General purpose I/O 0 (default) (I/O)
		01	EPWM1A - ePWM1 output A (O)
		10	Reserved. [2]
		11	Reserved. [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.

1.4.6.5 GPIO Port A MUX 2 (GPAMUX2) Register

Figure 1-63. 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 1-67. GPIO Port A MUX 2 (GPAMUX2) Register Field Descriptions

Bits	Field	Value	Description [(1)]
31-30	GPIO31		Configure the GPIO31 pin as:
		00	GPIO31 - General purpose I/O 31 (default) (I/O)
		01	CANTXA - eCAN-A transmit (O)
		10 or 11	Reserved
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	Reserved
27-26	GPIO29		Configure the GPIO29 pin as:
		00	GPIO29 (I/O) General purpose I/O 29 (default) (I/O)
		01	SCITXDA - SCI-A transmit. (O)
		10	SCLA - I^2C clock open drain bidirectional port (I/O)
		11	$\overline{TZ3}$ - Trip zone 3(I)
25-24	GPIO28		Configure the GPIO28 pin as:
		00	GPIO28 (I/O) General purpose I/O 28 (default) (I/O)
		01	SCIRXDA - SCI-A receive (I)
		10	SDAA - I^2C data open drain bidirectional port (I/O)
		11	$\overline{TZ2}$ - Trip zone 2 (I)
23-22	GPIO27		Configure the GPIO27 pin as:
		00	GPIO27 - General purpose I/O 27 (default) (I/O)
		01	HRCAP2
		10	Reserved
		11	$\overline{SPISTEB}$ (I/O) - SPI-B Slave transmit enable
21-20	GPIO26		Configure the GPIO26 pin as:
		00	GPIO26 - General purpose I/O 26 (default) (I/O)
		01	HRCAP1
		10	Reserved
		11	SPICLKB (I/O) - SPI-B clock

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

Bits	Field	Value	Description (1)
19-18	GPIO25		Configure the GPIO25 pin as:
		00	GPIO25 - General purpose I/O 25 (default) (I/O)
		01	Reserved
		10	Reserved
		11	SPISOMIB (I/O) - SPI-B Slave Output/Master input
17-16	GPIO24		Configure the GPIO24 pin as:
		00	GPIO24 - General purpose I/O 24 (default) (I/O)
		01	ECAP1 - eCAP1 (I/O)
		10	Reserved
		11	SPISIMOB (I/O) - SPI-B Slave input/Master output
15-14	GPIO23		Configure the GPIO23 pin as:
		00	GPIO23 - General purpose I/O 23 (default) (I/O)
		01	EQEP1I - eQEP1 index (I/O)
		10	Reserved
		11	LINRXA - LIN A receive (I)
13-12	GPIO22		Configure the GPIO22 pin as:
		00	GPIO22 - General purpose I/O 22 (default) (I/O)
		01	EQEP1S - eQEP1 strobe (I/O)
		10	Reserved
		11	LINTXA - LIN A 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	Reserved
		11	COMP2OUT (O) - Comparator 2 output
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	Reserved
		11	COMP1OUT (O) - Comparator 1 output
7-6	GPIO19/XCLKIN		Configure the GPIO19 pin as:
		00	GPIO19 - General purpose I/O 19 (default) (I/O)
		01	$\overline{\text{SPISTEA}}$ - SPI-A slave transmit enable (I/O)
		10	LINRXA (I) - LIN A receive (I)
		11	ECAP1 - eCAP1 (I/O)
5-4	GPIO18		Configure the GPIO18 pin as:
		00	GPIO18 - General purpose I/O 18 (default) (I/O)
		01	SPICLKA - SPI-A clock (I/O)
		10	LINTXA (O) - LIN A Transmit (O)
		11	XCLKOUT (O) - External clock output
3-2	GPIO17		Configure the GPIO17 pin as:
		00	GPIO17 - General purpose I/O 17 (default) (I/O)
		01	SPISOMIA - SPI-A Slave output/Master input (I/O)
		10	Reserved
		11	$\overline{\text{TZ3}}$ - Trip zone 3 (I)

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

Bits	Field	Value	Description [1]
1-0	GPIO16		Configure the GPIO16 pin as:
		00	GPIO16 - General purpose I/O 16 (default) (I/O)
		01	SPISIMOA - SPI-A slave-in, master-out (I/O)
		10	Reserved
		11	$\overline{TZ2}$ - Trip zone 2 (I)

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

1.4.6.6 GPIO Port A Direction (GPADIR) Register

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

Figure 1-64. GPIO Port A Direction (GPADIR) Register

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

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

Table 1-68. 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.

1.4.6.7 GPIO Port A Pullup Disable (GPAPUD) Register

The pullup disable (GPAPUD) register allows you to specify which pins should have an internal pullup 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 1-65. GPIO Port A Pullup Disable (GPAPUD) Register

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

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

Table 1-69. 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.

1.4.6.8 GPIO Port B Qualification Control (GPBCTRL) Register

Figure 1-66. GPIO Port B Qualification Control (GPBCTRL) Register

31	16
Reserved	
R-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 1-70. GPIO Port B Qualification Control (GPBCTRL) Register Field Descriptions

Bits	Field	Value	Description (1)
31-16	Reserved		Reserved
15-8	QUALPRD1		Specifies the sampling period for pins GPIO40 to GPIO44
		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 GPIO 39
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$
		0x00	Sampling Period = $T_{SYSCLKOUT}$ (2)
		0x01	Sampling Period = 2 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		. . .	. . .
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$

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

1.4.6.9 GPIO Port B Qualification Select 1 (GPBQSEL1) Register

Figure 1-67. GPIO Port B Qualification Select 1 (GPBQSEL1) Register

31	26	25	24	23	22	21	20	19	18	17	16
Reserved		GPIO44		GPIO43		GPIO42		GPIO41		GPIO40	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO39		GPIO38		GPIO37		GPIO36		GPIO35		GPIO34		GPIO33		GPIO32	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

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

Table 1-71. GPIO Port B Qualification Select 1 (GPBQSEL1) Register Field Descriptions

Bits	Field	Value	Description (1)
31- 26	Reserved		
25-0	GPIO 44-GPIO32		Select input qualification type for GPIO32 to GPIO44. The input qualification of each GPIO input is controlled by two bits as shown in Figure 1-67.
		00	Synchronize to SYSCLKOUT only. Valid for both peripheral and GPIO pins.
		01	Qualification using 3 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		10	Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register.
		11	Asynchronous. (no synchronization or qualification). This option applies to pins configured as peripherals only. If the pin is configured as a GPIO input, then this option is the same as 0,0 or synchronize to SYSCLKOUT.

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

1.4.6.10 GPIO Port B MUX 1 (GPBMUX1) Register

Figure 1-68. GPIO Port B MUX 1 (GPBMUX1) Register

31	26	25	24	23	22	21	20	19	18	17	16
Reserved		GPIO44		GPIO43		GPIO42		GPIO41		GPIO40	
R-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO39		GPIO38/ XCLKIN (TCK)		GPIO37(TDO)		GPIO36(TMS)		GPIO35(TDI)		GPIO34		GPIO33		GPIO32	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

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

Table 1-72. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions

Bit	Field	Value	Description
31-26	Reserved		Reserved
25-24	GPIO44		Configure this pin as:
		00	GPIO 44 - general purpose I/O 44 (default)
		01	Reserved
		10 or 11	Reserved
23-22	GPIO43		Configure this pin as:
		00	GPIO 43 - general purpose I/O 43 (default)
		01 or 10	Reserved
		11	COMP2OUT (O) - Comparator 2 output
21-20	GPIO42		Configure this pin as:
		00	GPIO 42 - general purpose I/O 42 (default)
		01 or 10	Reserved
		11	COMP1OUT (O) - Comparator 1 output
19-18	GPIO41		Configure this pin as:
		00	GPIO 41 - general purpose I/O 41 (default)
		01	EPWM7B (O) ePWM7 output B (O)
		10 or 11	Reserved
17-16	GPIO40		Configure this pin as:
		00	GPIO 40 - general purpose I/O 40 (default)
		01	EPWM7A (O) - ePWM7 output A (O)
		10 or 11	Reserved
15-14	GPIO39		Configure this pin as:
		00	GPIO 39 - general purpose I/O 39 (default)
		01	Reserved
		10 or 11	Reserved
13-12	GPIO38/XCLKIN (TCK)		Configure this pin as:
		00	GPIO 38 - general purpose I/O 38 (default) If $\overline{\text{TRST}}$ = 1, JTAG TCK function is chosen for this pin. This pin can also be used to provide a clock from an external oscillator to the core.
		01	Reserved
		10 or 11	Reserved
11-10	GPIO37(TDO)		Configure this pin as:
		00	GPIO 37 - general purpose I/O 37 (default) If $\overline{\text{TRST}}$ = 1, JTAG TDO function is chosen for this pin.
		01	Reserved
		10 or 11	Reserved
9-8	GPIO36(TMS)		Configure this pin as:
		00	GPIO 36 - general purpose I/O 36 (default) If $\overline{\text{TRST}}$ = 1, JTAG TMS function is chosen for this pin.
		01	Reserved
		10 or 11	Reserved
7-6	GPIO35(TDI)		Configure this pin as:
		00	GPIO 35 - general purpose I/O 35 (default) If $\overline{\text{TRST}}$ = 1, JTAG TDI function is chosen for this pin.
		01	Reserved
		10 or 11	Reserved

Table 1-72. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions (continued)

Bit	Field	Value	Description
5-4	GPIO34		Configure this pin as:
		00	GPIO 34 - general purpose I/O 34 (default)
		01	COMP2OUT (O) - Comparator 2 output
		10	Reserved
		11	COMP3OUT (O) - Comparator 3 output
3-2	GPIO33		Configure this pin as:
		00	GPIO 33 - general purpose I/O 33 (default)
		01	SCLA - I^2C 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 - I^2C 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)

1.4.6.11 GPIO Port B Direction (GPBDIR) Register

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

Figure 1-69. GPIO Port B Direction (GPBDIR) Register

31	13	12	11	10	9	8
Reserved		GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
GPIO39	GPIO38	GPIO37	GPIO36	GPIO35	GPIO34	GPIO33	GPIO32
R-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 1-73. GPIO Port B Direction (GPBDIR) Register Field Descriptions

Bits	Field	Value	Description (1)
31-13	Reserved		
12-0	GPIO44-GPIO32		Controls direction of GPIO pin when GPIO mode is selected. Reading the register returns the current value of the register setting
		0	Configures the GPIO pin as an input. (default)
		1	Configures the GPIO pin as an output.

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

1.4.6.12 GPIO Port B Pullup Disable (GPBPUD) Register

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

Figure 1-70. GPIO Port B Pullup Disable (GPBPUD) Register

31							16
Reserved							
R-0							
15		13	12	11	10	9	8
Reserved			GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
			R/W-0	R/W-0	R/W-0	R/W-1	R/W-1
7	6	5	4	3	2	1	0
GPIO39	GPIO38	GPIO37	GPIO36	GPIO35	GPIO34	GPIO33	GPIO32
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 1-74. GPIO Port B Internal Pullup Disable (GPBPUD) Register Field Descriptions

Bits	Field	Value	Description (1)
31-13	Reserved		Reserved
12-0	GPIO 44-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.

1.4.6.13 Analog I/O MUX (AIOMUX1) Register

Figure 1-71. Analog I/O MUX (AIOMUX1) Register

31	30	29	28	27	26	25	24	23	22	21	20	19	16
Reserved		AIO14		Reserved		AIO12		Reserved		AIO10		Reserved	
R-0		R/W-1,x		R-0		R/W-1,x		R-0		R/W-1,x		R-0	

15	14	13	12	11	10	9	8	7	6	5	4	3	0
Reserved		AIO6		Reserved		AIO4		Reserved		AIO2		Reserved	
R-0		R/W-1,x		R-0		R/W-1,x		R-0		R/W-1,x		R-0	

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

Table 1-75. Analog I/O MUX (AIOMUX1) Register Field Descriptions

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

1.4.6.14 Analog I/O Direction (AIODIR) Register

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

31							16
Reserved							
R-0							

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

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

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

Table 1-76. Analog I/O Direction (AIODIR) Register Field Descriptions

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

1.4.6.15 GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register

Figure 1-73. GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register

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

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

Table 1-77. GPIO XINTn Interrupt Select (GPIOXINTnSEL) Register Field Descriptions

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

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

Table 1-78. XINT1/XINT2/XINT3 Interrupt Select and Configuration Registers

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

1.4.6.16 GPIO Low Power Mode Wakeup Select (GPIOLPMSEL) Register

Figure 1-74. GPIO Low Power Mode Wakeup Select (GPIOLPMSEL) Register

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

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

Table 1-79. 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 1-74.
		0	If the bit is cleared, the signal on the corresponding pin has no effect on the HALT and STANDBY low power modes.
		1	If the respective bit is set to 1, the signal on the corresponding pin is able to wake the device from both HALT and STANDBY low power modes.

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

1.4.6.17 GPIO Port A Data (GPADAT) Register

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

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

Figure 1-75. GPIO Port A Data (GPADAT) Register

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

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

Table 1-80. GPIO Port A Data (GPADAT) Register Field Descriptions

Bits	Field	Value	Description
31-0	GPIO31-GPIO0		Each bit corresponds to one GPIO port A pin (GPIO0-GPIO31) as shown in Figure 1-75.
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured.
			Writing a 0 forces an output of 0, if the pin is configured as a GPIO output in the appropriate GPAMUX1/2 and GPADIR registers; otherwise, the value is latched but not used to drive the pin.
		1	Reading a 1 indicates that the state of the pin is currently high irrespective of the mode the pin is configured.
			Writing a 1 forces an output of 1, if the pin is configured as a GPIO output in the appropriate GPAMUX1/2 and GPADIR registers; otherwise, the value is latched but not used to drive the pin.

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

Figure 1-76. 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 1-81. 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 1-76.
		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 1-82. 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 1-76.
		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 1-83. 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 1-76.
		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.

1.4.6.19 GPIO Port B Data (GPBDAT) Register

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

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

Figure 1-77. GPIO Port B Data (GPBDAT) Register

31							16
Reserved							
R-0							
15		13	12	11	10	9	8
Reserved			GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
R-0			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; x = state of the GPADAT register is unknown after reset (depends on the level of the pin after reset).

Table 1-84. GPIO Port B Data (GPBDAT) Register Field Descriptions

Bit	Field	Value	Description
31- 13	Reserved		Reserved

Table 1-84. GPIO Port B Data (GPBDAT) Register Field Descriptions (continued)

Bit	Field	Value	Description
12-0	GPIO 44-GPIO32		Each bit corresponds to one GPIO port B pin (GPIO32-GPIO 44) as shown in Figure 1-77.
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured. Writing a 0 forces an output of 0, if the pin is configured as a GPIO output in the appropriate GPBMUX1 and GPBDIR registers; otherwise, the value is latched but not used to drive the pin.
		1	Reading a 1 indicates that the state of the pin is currently high irrespective of the mode the pin is configured. Writing a 1 forces an output of 1, if the pin is configured as a GPIO output in the GPBMUX1 and GPBDIR registers; otherwise, the value is latched but not used to drive the pin.

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

Figure 1-78. GPIO Port B Set, Clear, and Toggle (GPBSET, GPBCLEAR, GPBTOGGLE) Registers

31						16
Reserved						
R-0						
15		13	12	11	10	9	8
Reserved			GPIO44	GPIO43	GPIO42	GPIO41	GPIO40
			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 1-85. GPIO Port B Set (GPBSET) Register Field Descriptions

Bits	Field	Value	Description
31- 13	Reserved		Reserved
12-0	GPIO 44-GPIO32		Each GPIO port B pin (GPIO32-GPIO 44) corresponds to one bit in this register as shown in Figure 1-78.
		0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to high. If the pin is configured as a GPIO output, then it will be driven high. If the pin is not configured as a GPIO output, then the latch is set but the pin is not driven.

Table 1-86. GPIO Port B Clear (GPBCLEAR) Register Field Descriptions

Bits	Field	Value	Description
31- 13	Reserved		Reserved
12-0	GPIO 44-GPIO32		Each GPIO port B pin (GPIO32-GPIO 44) corresponds to one bit in this register as shown in Figure 1-78.
		0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to low. If the pin is configured as a GPIO output, then it will be driven low. If the pin is not configured as a GPIO output, then the latch is cleared but the pin is not driven.

Table 1-87. GPIO Port B Toggle (GPBTOGGLE) Register Field Descriptions

Bits	Field	Value	Description
31- 13	Reserved		Reserved
12-0	GPIO 44-GPIO32		Each GPIO port B pin (GPIO32-GPIO 44) corresponds to one bit in this register as shown in Figure 1-78.
		0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to toggle from its current state. If the pin is configured as a GPIO output, then it will be driven in the opposite direction of its current state. If the pin is not configured as a GPIO output, then the latch is cleared but the pin is not driven.

1.4.6.21 Analog I/O Data (AIODAT) Register

Figure 1-79. Analog I/O Data (AIODAT) 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
Reserved	AIO14	Reserved	AIO12	Reserved	AIO10	Reserved	
R-0	R/W-x	R-0	R/W-x	R-0	R/W-x	R-0	

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

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

Table 1-88. Analog I/O Data (AIODAT) Register Field Descriptions

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

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

Figure 1-80. Analog I/O Set, Clear, and Toggle (AIOSET, AIOCLEAR, AIOTOGGLE) Registers

31							16
Reserved							
R-0							

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

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

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

Table 1-89. Analog I/O Set (AIOSET) Register Field Descriptions

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

Table 1-90. Analog I/O Clear (AIOCLEAR) Register Field Descriptions

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

Table 1-91. Analog I/O Toggle (AIOTOGGLE) Register Field Descriptions

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

1.5 Peripheral Frames

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

1.5.1 Peripheral Frame Registers

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

- Peripheral Frame 0: These are peripherals that are mapped directly to the CPU memory bus. See Table 1-92.
- Peripheral Frame 1: These are peripherals that are mapped to the 32-bit peripheral bus. See Table 1-93.
- Peripheral Frame 2: These are peripherals that are mapped to the 16-bit peripheral bus. See Table 1-94.

Table 1-92. Peripheral Frame 0 Registers

Name (1)	Address Range	Size (x16)	Access Type(2)
Device Emulation Registers	0x00 0880 - 0x00 0984	261	EALLOW protected
System Power Control Registers	0x00 0985 - 0x00 0987	3	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, CPU)	0x00 0B00 - 0x00 0B1F	32	Not EALLOW protected
CPU–TIMER0/1/2 Registers	0x00 0C00 - 0x00 0C3F	64	Not EALLOW protected
PIE Registers	0x00 0CE0 - 0x00 0CFF	32	Not EALLOW protected
PIE Vector Table	0x00 0D00 - 0x00 0DFF	256	EALLOW protected
CLA Registers	0x00 1400 - 0x00 147F	128	Yes
CLA to CPU Message RAM (CPU writes ignored)	0x00 1480 - 0x00 14FF	128	N/A
CPU to CLA Message RAM (CLA writes ignored)	0x00 1500 - 0x00 157F	128	N/A

(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 1-93. Peripheral Frame 1 Registers

Name (1)	Address Range	Size (x16)	Access Type (2)
eCANA Registers	0x6000 - 0x60FF	256	(3)
eCANA Mailbox RAM	0x6100 - 0x61FF	256	Not EALLOW protected
Comparator1 Registers	0x6400 - 0x641F	32	(3)
Comparator2 Registers	0x6420 - 0x643F	32	(3)
Comparator3 Registers	0x6440 - 0x645F	32	(3)
ePWM1 + HRPWM1 Registers	0x6800 - 0x683F	64	(3)
ePWM2 + HRPWM2 Registers	0x6840 - 0x687F	64	(3)
ePWM3 + HRPWM3 Registers	0x6880 - 0x68BF	64	(3)
ePWM4 + HRPWM4 Registers	0x68C0 - 0x68FF	64	(3)
ePWM5 + HRPWM5 Registers	0x6900 - 0x693F	64	(3)
ePWM6 + HRPWM6 Registers	0x6940 - 0x697F	64	(3)
ePWM7 + HRPWM7 Registers	0x6980 - 0x69BF	64	(3)
eCAP1 Registers	0x6A00 - 0x6A1F	32	Not EALLOW-protected
eQEP1 Registers	0x6B00 - 0x6B3F	64	Not EALLOW-protected
LIN-A Registers	0x6C00 - 0x6C7F	128	(3)
GPIO Control Registers	0x6F80 - 0x6FBF	128	EALLOW-protected
GPIO Data Registers	0x6FC0 - 0x6FDF	32	Not EALLOW-protected
GPIO Interrupt and LPM Select Registers	0x6FE0 - 0x6FFF	32	EALLOW-protected

(1) Back-to-back write operations to Peripheral Frame 1 registers will incur a 1-cycle stall (1 cycle delay).
(2) Peripheral Frame 1 allows 16-bit and 32-bit accesses. All 32-bit accesses are aligned to even address boundaries.
(3) Some Registers/Bits are EALLOW protected. See the module reference guide for more information.

Table 1-94. 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
SPI-B Registers	0x7740 - 0x7747	16	Not EALLOW protected
SCI-A Registers	0x7050 - 0x705F	16	Not EALLOW protected
NMI Watchdog Interrupt Registers	0x7060 - 0x706F	16	
External Interrupt Registers	0x7070 - 0x707F	16	Not EALLOW protected
ADC Registers	0x7100 - 0x711F	32	Not EALLOW protected
I^2C Registers	0x7900 - 0x793F	64	Not EALLOW protected

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

1.5.2 EALLOW-Protected Registers

Several control registers are protected from spurious CPU writes by the EALLOW protection mechanism. The EALLOW bit in status register 1 (ST1) indicates if the state of protection as shown in Table 1-95.

Table 1-95. 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™ IDE.

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

The following registers are EALLOW-protected:

- Device Emulation Registers (Table 1-96)
- Flash Registers (Table 1-97)
- CSM Registers (Table 1-98)
- System Control Registers (Table 1-99)
- GPIO MUX Registers (Table 1-100)
- PIE Vector Table (Table 1-101)

Table 1-96. EALLOW-Protected Device Emulation Registers

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

Table 1-97. 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 1-98. EALLOW-Protected Code Security Module (CSM) Registers

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

Table 1-99. EALLOW-Protected PLL, Clocking, Watchdog, and Low-Power Mode Registers

Name	Address	Size (x16)	Description
BORCFG	0x0000-0985	1	BOR Configuration Register
XCLK	0x0000-7010	1	XCLKOUT/XCLKIN Control
PLLSTS	0x0000-7011	1	PLL Status Register
CLKCTL	0x0000-7012	1	Clock Control Register
PLLLOCKPRD	0x0000-7013	1	PLL Lock Period Register
INTOSC1TRIM	0x0000-7014	1	Internal Oscillator 1 Trim Register
INTOSC2TRIM	0x0000-7016	1	Internal Oscillator 2 Trim Register
PCLKCR2	0x0000-7019	1	Peripheral Clock Control Register 2
LOSPCP	0x0000-701B	1	Low-Speed Peripheral Clock Pre-Scaler Register
PCLKCR0	0x0000-701C	1	Peripheral Clock Control Register 0
PCLKCR1	0x0000-701D	1	Peripheral Clock Control Register 1
LPMCR0	0x0000-701E	1	Low Power Mode Control Register 0
PCLKCR3	0x0000-7020	1	Peripheral Clock Control Register 3
PLLCR	0x0000-7021	1	PLL Control Register
SCSR	0x0000-7022	1	System Control and Status Register
WDCNTR	0x0000-7023	1	Watchdog Counter Register
WDKEY	0x0000-7025	1	Watchdog Reset Key Register
WDCR	0x0000-7029	1	Watchdog Control Register

Table 1-100. EALLOW-Protected GPIO Registers

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

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

Table 1-101. EALLOW-Protected PIE Vector Table

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

Table 1-102 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 1-102. EALLOW-Protected ePWM1-ePWM7 Registers

ePWMn	TZSEL	TZCTL	TZEINT	TZCLR	TZFRC	HRCNFG	Size x16
ePWM1	0x6812	0x6814	0x6815	0x6817	0x6818	0x6820	1
ePWM2	0x6852	0x6854	0x6855	0x6857	0x6858	0x6860	1
ePWM3	0x6892	0x6894	0x6895	0x6897	0x6898	0x68A0	1
ePWM4	0x68D2	0x68D4	0x68D5	0x68D7	0x68D8	0x68E0	1
ePWM5	0x6912	0x6914	0x6915	0x6917	0x6918	0x6920	1
ePWM6	0x6952	0x6954	0x6955	0x6957	0x6958	0x6960	1
ePWM7	0x6992	0x6994	0x6995	0x6997	0x6998	0x69A0	1

1.5.3 Device Emulation Registers

The registers are listed in Table 1-103 are used to control the protection mode of the C28x CPU and to monitor some critical device signals.

Table 1-103. Device Emulation Registers

Name	Address	Size (x16)	Description	Bit Description
DEVICECNF	0x0880 0x0881	2	Device Configuration Register	Section 1.5.3.1
PARTID	0x3D7E80	1	Part ID Register	Section 1.5.3.2
CLASSID	0x0882	1	Class ID Register	Section 1.5.3.3
REVID	0x0883	1	Revision ID Register	Section 1.5.3.4

1.5.3.1 Device Configuration (DEVICECNF) Register

Figure 1-81. Device Configuration (DEVICECNF) Register

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

15	6	5	4	3	2	0
Reserved		XRS	Rsvd	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 1-104. 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 status of the $\overline{\text{TRST}}$ signal.
		0	No JTAG debug probe is connected.
		1	JTAG debug probe is connected.
26:20	Reserved		
19	ENPROT		Enable Write-Read Protection Mode Bit.
		0	Disables write-read protection mode
		1	Enables write-read protection for the address range 0x4000-0x7FFF
18-6	Reserved		Reserved
5	XRS		Reset Input Signal Status. This is connected directly to the $\overline{\text{XRS}}$ input pin.
4	Reserved		Reserved
3	VMAPS		VMAP Configure Status. This indicates the status of VMAP.
2-0	Reserved		Reserved

1.5.3.2 Part ID (PARTID) Register

Figure 1-82. Part ID (PARTID) Register

15	8	7	0
PARTTYPE		PARTNO	
R		R	

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

Table 1-105. Part ID (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:
			All other values are reserved.
		0xA9	TMS320F28030RSH
		0xAA	TMS320F28030PAG
		0xAB	TMS320F28030PN
		0xAD	TMS320F28031RSH
		0xAE	TMS320F28031PAG
		0xAF	TMS320F28031PN
		0xB1	TMS320F28032RSH
		0xB2	TMS320F28032PAG
		0xB3	TMS320F28032PN
		0xB5	TMS320F28033RSH
		0xB6	TMS320F28033PAG
		0xB7	TMS320F28033PN
		0xB9	TMS320F28034RSH
		0xBA	TMS320F28034PAG
		0xBB	TMS320F28034PN
		0xBD	TMS320F28035RSH
		0xBE	TMS320F28035PAG
		0xBF	TMS320F28035PN

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

1.5.3.3 Class ID (CLASSID) Register

Figure 1-83. Class ID (CLASSID) Register

15	0
CLASSID	
R	

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

Table 1-106. Class ID (CLASSID) Register Field Descriptions

Bit	Field	Value (1)	Description
15-0	CLASSID		These 16 bits specify the feature set of the device as follows:
			All other values are reserved.
		0x00AB	TMS320F28030
		0x00AF	TMS320F28031
		0x00B3	TMS320F28032
		0x00B7	TMS320F28033
		0x00BB	TMS320F28034
		0x00BF	TMS320F28035

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

1.5.3.4 Revision ID (REVID) Register

Figure 1-84. Revision ID (REVID) Register

15	0
REVID	
R	

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

Table 1-107. Revision ID (REVID) Register Field Descriptions

Bits	Field	Value (1)	Description
15-0	REVID		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 - This silicon revision is available as TMX and TMS.
		0x0001	Silicon Revision A - TMS

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

1.5.4 Write-Followed-by-Read Protection

The memory address range for which CPU write followed by read operations are protected is 0x4000 - 0x7FFF (operations occur in sequence rather than in their natural pipeline order). This is necessary protection for certain peripheral operations.

Example: The following lines of code perform a write to register 1 (REG1) location and the next instruction performs a read from Register 2 (REG2) location. On the processor memory bus, with block protection disabled, the read operation is issued before the write as shown.

```
MOV   @REG1,AL      ----------+
TBIT  @REG2,#BIT_X  ----------|-------> Read
                              +-------> Write
If block protection is enabled, then the read is stalled until the write occurs as shown:
MOV       @REG1,AL        ----------+
TBIT      @REG2,#BIT_X    ----------|------+
                                    +------|---> Write
                                           +---> Read
```

1.6 Peripheral Interrupt Expansion (PIE)

The peripheral interrupt expansion (PIE) block multiplexes numerous interrupt sources into a smaller set of interrupt inputs. The PIE block can support 96 individual interrupts that are grouped into blocks of eight. Each group is fed into one of 12 core interrupt lines (INT1 to INT12). Each of the 96 interrupts is supported by its own vector stored in a dedicated RAM block that you can modify. The CPU, upon servicing the interrupt, automatically fetches the appropriate interrupt vector. It takes nine CPU clock cycles to fetch the vector and save critical CPU registers. Therefore, the CPU can respond quickly to interrupt events. Prioritization of interrupts is controlled in hardware and software. Each individual interrupt can be enabled/disabled within the PIE block.

1.6.1 Overview of the PIE Controller

The 28x CPU supports one nonmaskable interrupt (NMI) and 16 maskable prioritized interrupt requests (INT1-INT14, RTOSINT, and DLOGINT) at the CPU level. The 28x devices have many peripherals and each peripheral is capable of generating one or more interrupts in response to many events at the peripheral level. Because the CPU does not have sufficient capacity to handle all peripheral interrupt requests at the CPU level, a centralized peripheral interrupt expansion (PIE) controller is required to arbitrate the interrupt requests from various sources such as peripherals and other external pins.

The PIE vector table is used to store the address (vector) of each interrupt service routine (ISR) within the system. There is one vector per interrupt source including all MUXed and nonMUXed interrupts. You populate the vector table during device initialization and you can update it during operation.

1.6.1.1 Interrupt Operation Sequence

Figure 1-85 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.

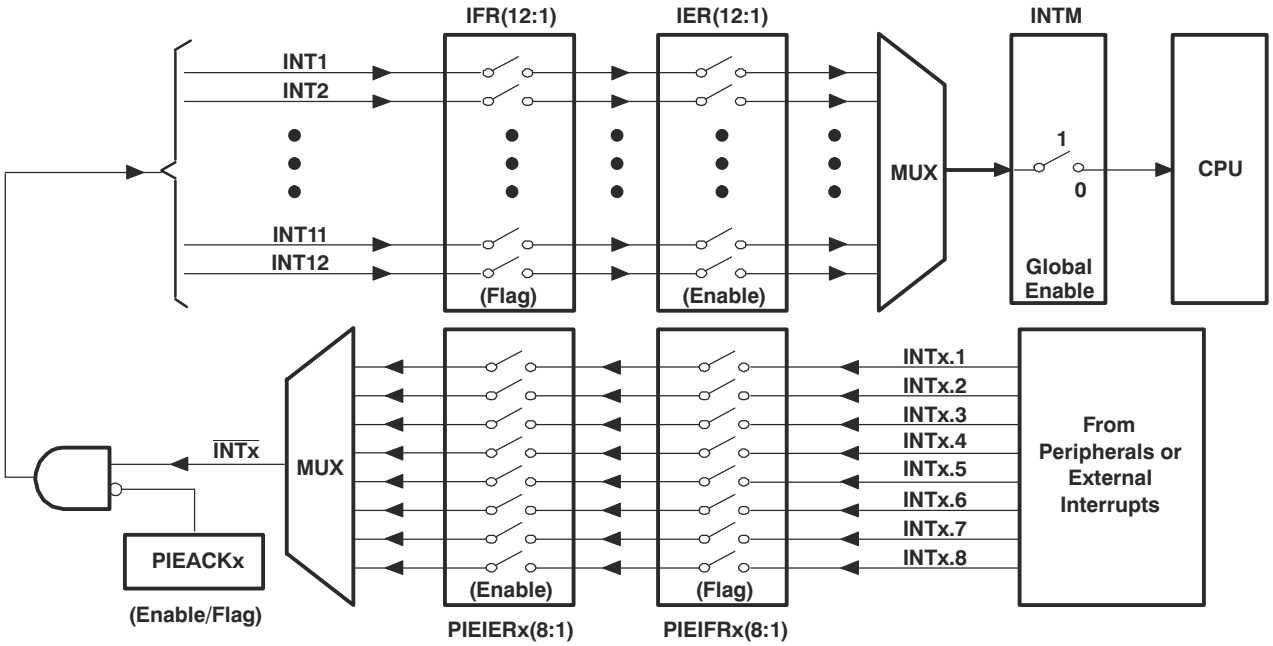


Figure 1-85. Overview: Multiplexing of Interrupts Using the PIE Block

- **Peripheral Level**

  An interrupt-generating event occurs in a peripheral. The interrupt flag (IF) bit corresponding to that event is set in a register for that particular peripheral.

  If the corresponding interrupt enable (IE) bit is set, the peripheral generates an interrupt request to the PIE controller. If the interrupt is not enabled at the peripheral level, the IF remains set until cleared by software. If the interrupt is enabled at a later time, and the interrupt flag is still set, the interrupt request is asserted to the PIE.

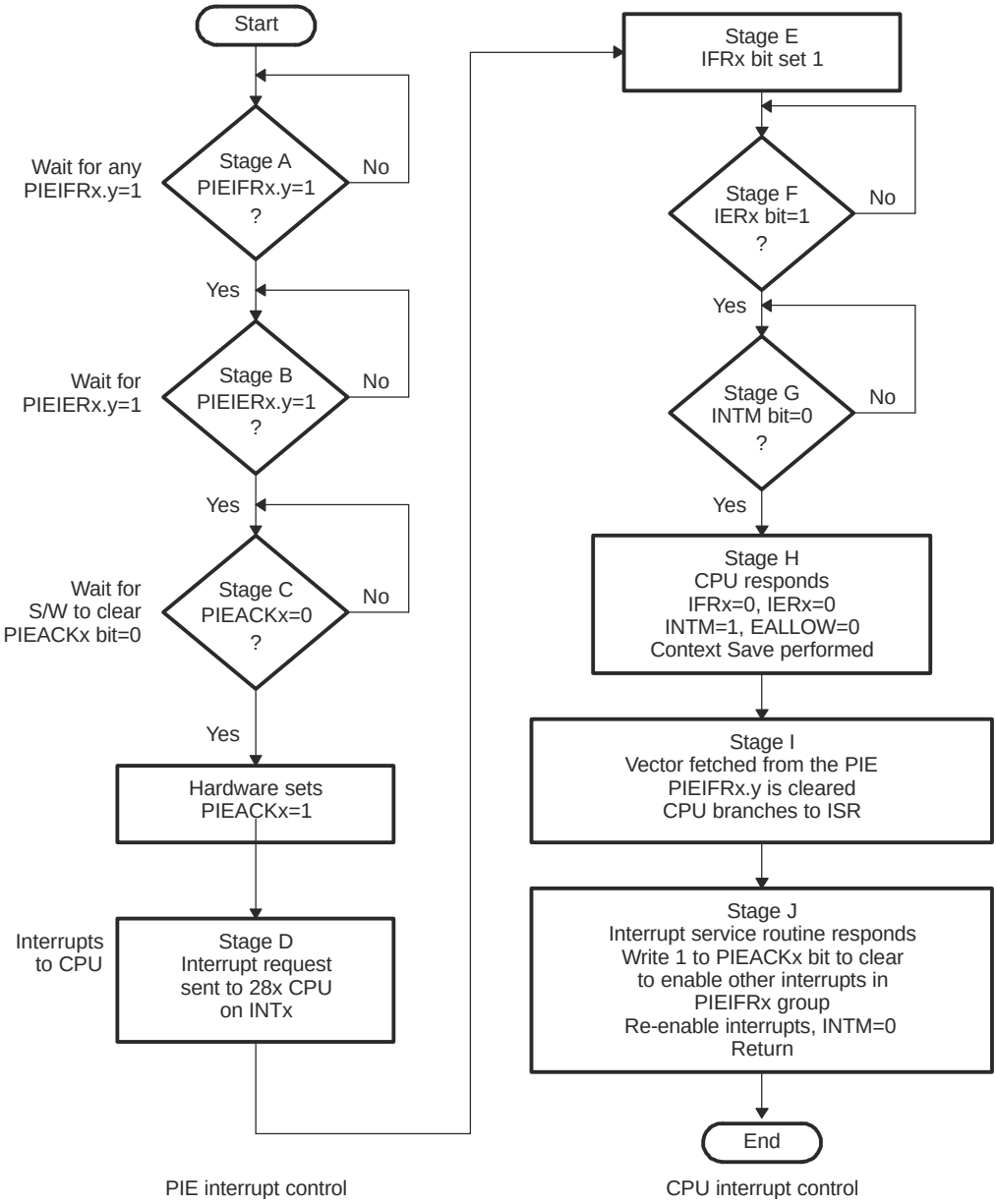
  Interrupt flags within the peripheral registers must be manually cleared. See the peripheral reference guide for a specific peripheral for more information.
- **PIE Level**

  The PIE block multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 - PIE group 12. The interrupts within a group are multiplexed into one CPU interrupt. For example, PIE group 1 is multiplexed into CPU interrupt 1 (INT1) while PIE group 12 is multiplexed into CPU interrupt 12 (INT12). Interrupt sources connected to the remaining CPU interrupts are not multiplexed. For the nonmultiplexed interrupts, the PIE passes the request directly to the CPU.

  For multiplexed interrupt sources, each interrupt group in the PIE block has an associated flag register (PIEIFRx) and enable (PIEIERx) register (x = PIE group 1 - PIE group 12). Each bit, referred to as y, corresponds to one of the 8 MUXed interrupts within the group. Thus PIEIFRx.y and PIEIERx.y correspond to interrupt y (y = 1-8) in PIE group x (x = 1-12). In addition, there is one acknowledge bit (PIEACK) for every PIE interrupt group referred to as PIEACKx (x = 1-12). Figure 1-86 illustrates the behavior of the PIE hardware under various PIEIFR and PIEIER register conditions.

  Once the request is made to the PIE controller, the corresponding PIE interrupt flag (PIEIFRx.y) bit is set. If the PIE interrupt enable (PIEIERx.y) bit is also set for the given interrupt, the PIE checks the corresponding PIEACKx bit to determine if the CPU is ready for an interrupt from that group. If the PIEACKx bit is clear for that group, the PIE sends the interrupt request to the CPU. If PIEACKx is set, the PIE waits until it is cleared to send the request for INTx. See Figure 1-89 for details.
- **CPU Level**

  Once the request is sent to the CPU, the CPU level interrupt flag (IFR) bit corresponding to INTx is set. After a flag has been latched in the IFR, the corresponding interrupt is not serviced until it is appropriately enabled in the CPU interrupt enable (IER) register or the debug interrupt enable register (DBGIER) and the global interrupt mask (INTM) bit.



A. For multiplexed interrupts, the PIE responds with the highest priority interrupt that is both flagged and enabled. If there is no interrupt both flagged and enabled, then the highest priority interrupt within the group (INTx.1 where x is the PIE group) is used. See Section 1.6.3.3 for details.

Figure 1-86. Typical PIE/CPU Interrupt Response - INTx.y

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 device 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
Device in real-time mode and halted	Bit in IER is 1 and DBGIER is 1

The CPU then prepares to service the interrupt. This preparation process is described in the *TMS320C28x DSP CPU and Instruction Set Reference Guide*. In preparation, the corresponding CPU IFR and IER bits are cleared, EALLOW and LOOP are cleared, INTM and DBGM are set, the pipeline is flushed and the return address is stored, and the automatic context save is performed. The vector of the ISR is then fetched from the PIE module. If the interrupt request comes from a multiplexed interrupt, the PIE module uses the group PIEIERx and PIEIFRx registers to decode which interrupt needs to be serviced. This decode process is described in Section 1.6.3.3.

The address for the interrupt service routine that is executed is fetched directly from the PIE interrupt vector table. There is one 32-bit vector for each of the possible 96 interrupts within the PIE. Interrupt flags within the PIE module (PIEIFRx.y) are automatically cleared when the interrupt vector is fetched. The PIE acknowledge bit for a given interrupt group, however, must be cleared manually when ready to receive more interrupts from the PIE group.

1.6.2 Vector Table Mapping

On 28xx devices, the interrupt vector table can be mapped to four distinct locations in memory. In practice only the PIE vector table mapping is used.

This vector mapping is controlled by the following mode bits/signals:

VMAP: VMAP is found in Status Register 1 ST1 (bit 3). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC VMAP instructions. For normal operation leave this bit set.

M0M1MAP: M0M1MAP is found in Status Register 1 ST1 (bit 11). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC M0M1MAP instructions. For normal 28xx device operation, this bit should remain set. M0M1MAP = 0 is reserved for TI testing only.

ENPIE: ENPIE is found in PIECTRL Register (bit 0). The default value of this bit, on reset, is set to 0 (PIE disabled). The state of this bit can be modified after reset by writing to the PIECTRL register (address 0x0000 0CE0).

Using these bits and signals the possible vector table mappings are shown in Table 1-109.

Table 1-109. Interrupt Vector Table Mapping

Vector MAPS	Vectors Fetched From	Address Range	VMAP (1)	M0M1MAP (1)	ENPIE (1)
M1 Vector(2)	M1 SARAM Block	0x000000 - 0x00003F	0	0	X
M0 Vector(2)	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) On the 280x devices, the VMAP and M0M1MAP modes are set to 1 on reset. The ENPIE mode is forced to 0 on reset.
(2) Vector map M0 and M1 Vector is a reserved mode only. On the 28x devices these are used as SARAM.

The M1 and M0 vector table mapping are reserved for TI testing only. When using other vector mappings, the M0 and M1 memory blocks are treated as SARAM blocks and can be used freely without any restrictions.

After a device reset operation, the vector table is mapped as shown in Table 1-110.

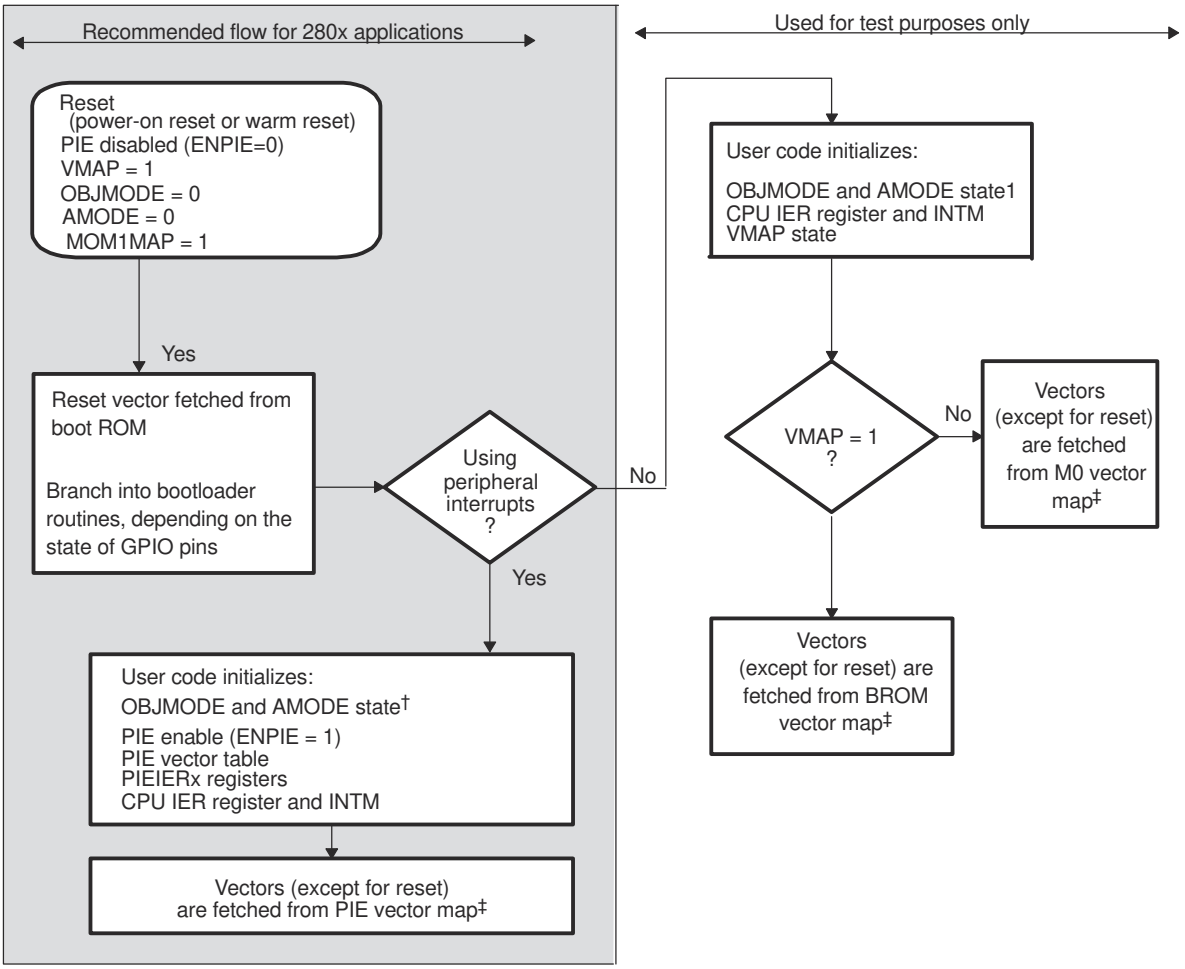
Table 1-110. Vector Table Mapping After Reset Operation

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

Figure 1-87 illustrates the process by which the vector table mapping is selected.



A. The compatibility operating mode of the 28x CPU is determined by a combination of the OBJMODE and AMODE bits in Status Register 1 (ST1):

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

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

Figure 1-87. Reset Flow Diagram

1.6.3 Interrupt Sources

Figure 1-88 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.

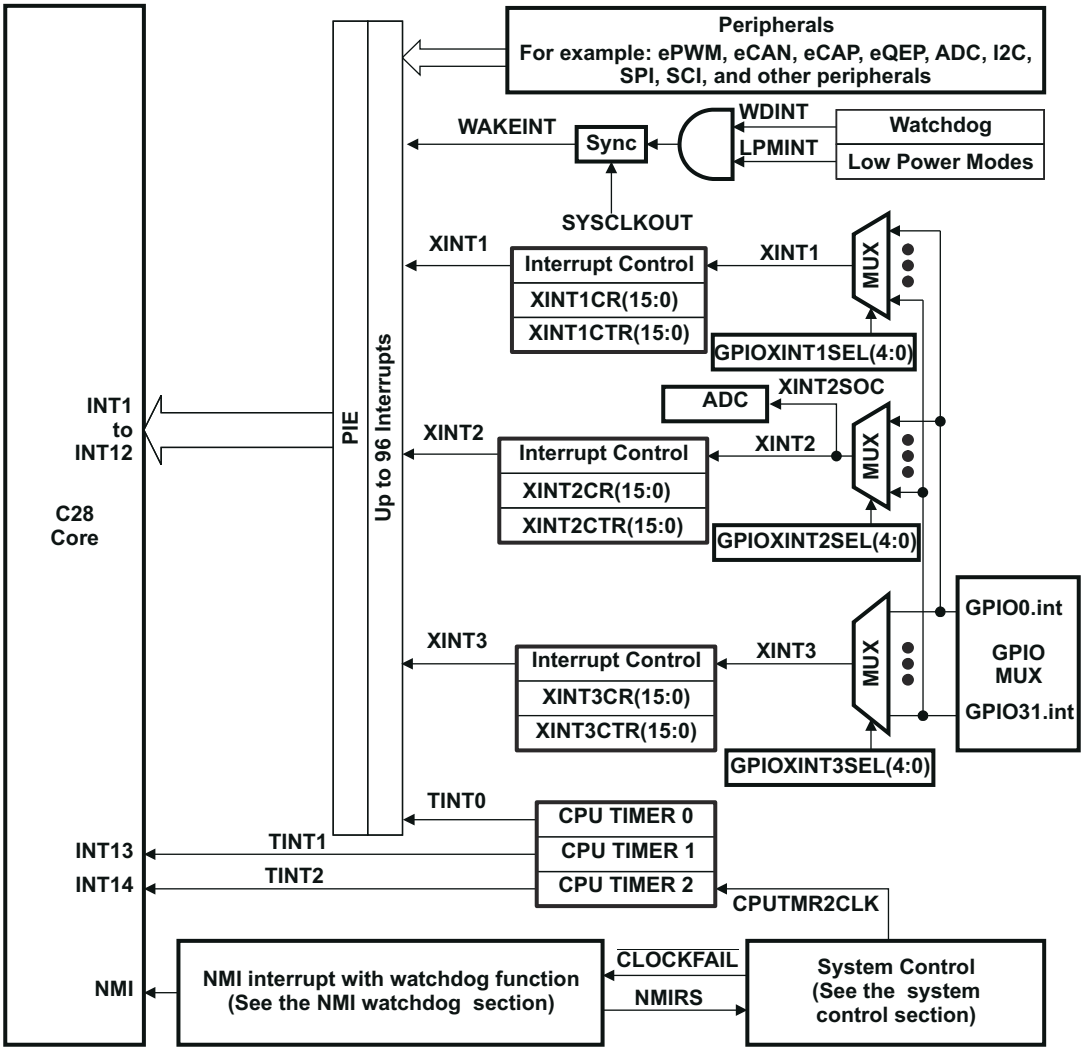


Figure 1-88. PIE Interrupt Sources and External Interrupts XINT1/XINT2/XINT3

1.6.3.1 Procedure for Handling Multiplexed Interrupts

The PIE module multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 - PIE group 12. Each group has an associated enable PIEIER and flag PIEIFR register. These registers are used to control the flow of interrupts to the CPU. The PIE module also uses the PIEIER and PIEIFR registers to decode to which interrupt service routine the CPU should branch.

There are three main rules that should be followed when clearing bits within the PIEIFR and the PIEIER registers:

Rule 1: Never clear a PIEIFR bit by software

An incoming interrupt may be lost while a write or a read-modify-write operation to the PIEIFR register takes place. To clear a PIEIFR bit, the pending interrupt must be serviced. If you want to clear the PIEIFR bit without executing the normal service routine, use the following procedure:

1. Set the EALLOW bit to allow modification to the PIE vector table.
2. Modify the PIE vector table so that the vector for the peripheral's service routine points to a temporary ISR. This temporary ISR will only perform a return from interrupt (IRET) operation.
3. Enable the interrupt so that the interrupt will be serviced by the temporary ISR.
4. After the temporary interrupt routine is serviced, the PIEIFR bit will be clear
5. Modify the PIE vector table to re-map the peripheral's service routine to the proper service routine.
6. Clear the EALLOW bit.

Rule 2: Procedure for software-prioritizing interrupts

Use the method found in the C2000Ware example for software prioritization of interrupts.SPRC530).

1. Use the CPU IER register as a global priority and the individual PIEIER registers for group priorities. In this case the PIEIER register is only modified within an interrupt. In addition, only the PIEIER for the same group as the interrupt being serviced is modified. This modification is done while the PIEACK bit holds additional interrupts back from the CPU.
2. Never disable a PIEIER bit for a group when servicing an interrupt from an unrelated group.

Rule 3: Disabling interrupts using PIEIER

If the PIEIER registers are used to enable and then later disable an interrupt, the procedure described in Section 1.6.3.2 must be followed.

1.6.3.2 Procedures for Enabling And Disabling Multiplexed Peripheral Interrupts

The proper procedure for enabling or disabling an interrupt is by using the peripheral interrupt enable/disable flags. The primary purpose of the PIEIER and CPU IER registers is for software prioritization of interrupts within the same PIE interrupt group. Use the method found in the C2000Ware example for software prioritization of interrupts.SPRC530).

Should bits within the PIEIER registers need to be cleared outside of this context, one of the following two procedures should be followed. The first method preserves the associated PIE flag register so that interrupts are not lost. The second method clears the associated PIE flag register.

Method 1: Use the PIEIERx register to disable the interrupt and preserve the associated PIEIFRx flags.

To clear bits within a PIEIERx register while preserving the associated flags in the PIEIFRx register, the following procedure should be followed:

1. Disable global interrupts (INTM = 1).
2. Clear the PIEIERx.y bit to disable the interrupt for a given peripheral. This can be done for one or more peripherals within the same group.
3. Wait 5 cycles. This delay is required to be sure that any interrupt that was incoming to the CPU has been flagged within the CPU IFR register.
4. Clear the CPU IFRx bit for the peripheral group. This is a safe operation on the CPU IFR register.
5. Clear the PIEACKx bit for the peripheral group.
6. Enable global interrupts (INTM = 0).

Method 2: Use the PIEIERx register to disable the interrupt and clear the associated PIEIFRx flags.

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

1. Disable global interrupts (INTM = 1).
2. Set the EALLOW bit.
3. Modify the PIE vector table to temporarily map the vector of the specific peripheral interrupt to a empty interrupt service routine (ISR). This empty ISR will only perform a return from interrupt (IRET) instruction. This is the safe way to clear a single PIEIFRx.y bit without losing any interrupts from other peripherals within the group.
4. Disable the peripheral interrupt at the peripheral register.
5. Enable global interrupts (INTM = 0).
6. Wait for any pending interrupt from the peripheral to be serviced by the empty ISR routine.
7. Disable global interrupts (INTM = 1).
8. Modify the PIE vector table to map the peripheral vector back to its original ISR.
9. Clear the EALLOW bit.
10. Disable the PIEIER bit for given peripheral.
11. Clear the IFR bit for given peripheral group (this is safe operation on CPU IFR register).
12. Clear the PIEACK bit for the PIE group.
13. Enable global interrupts.

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

Figure 1-89 shows the flow with the steps shown in circled numbers. Following the diagram, the steps are described.

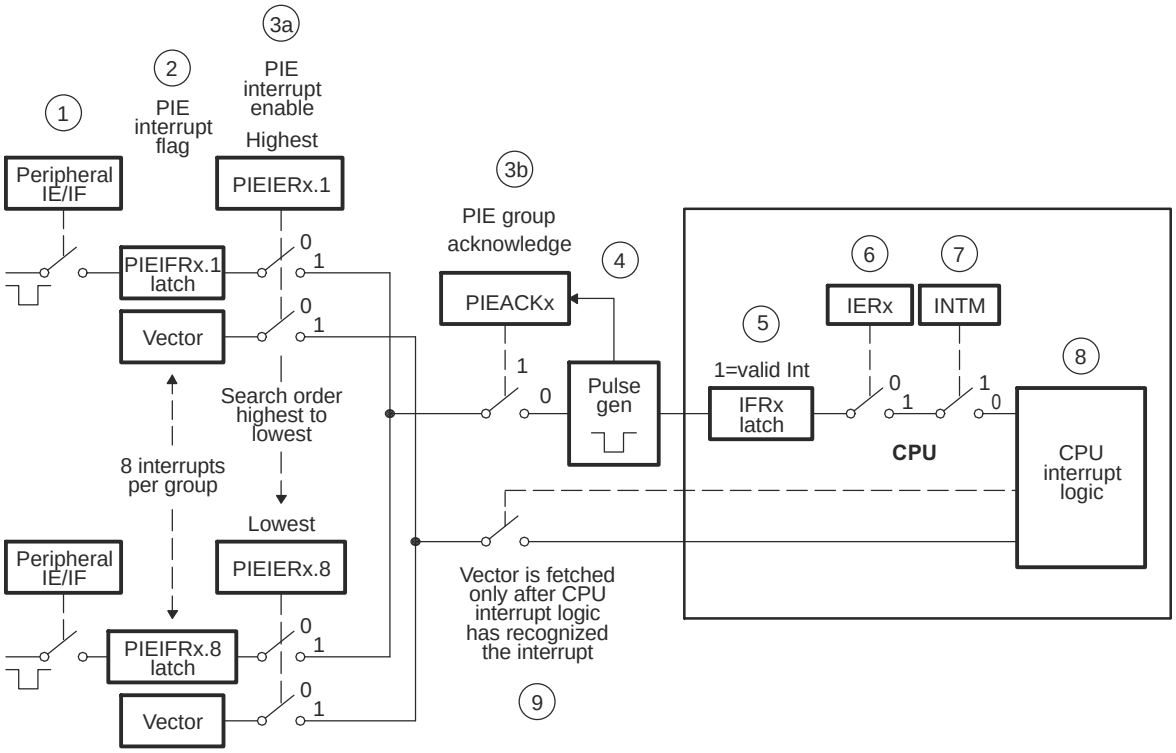


Figure 1-89. Multiplexed Interrupt Request Flow Diagram

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.
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.
3. For the interrupt request to be sent from the PIE to the CPU, both of the following conditions must be true:
   a. The proper enable bit must be set (PIEIERx.y = 1) and
   b. The PIEACKx bit for the group must be clear.
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.
5. The CPU interrupt flag bit is set (CPU IFRx = 1) to indicate a pending interrupt x at the CPU level.
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.
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 described in the *TMS320C28x CPU and Instruction Set Reference Guide*.
8. The CPU will then request the appropriate vector from the PIE.
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, the PIE will respond with the vector for the highest priority interrupt within that group. That is the branch address used for INTx.1. This behavior corresponds to the 28x TRAP or INT instructions.

Note

Because the PIEIERx register is used to determine which vector will be used for the branch, you must take care when clearing bits within the PIEIERx register. The proper procedure for clearing bits within a PIEIERx register is described in Section 1.6.3.2. Failure to follow these steps can result in changes occurring to the PIEIERx register after an interrupt has been passed to the CPU at Step 5 in Figure 1-89. In this case, the PIE will respond as if a TRAP or INT instruction was executed unless there are other interrupts both pending and enabled.

At this point, the PIEIFRx.y bit is cleared and the CPU branches to the vector of the interrupt fetched from the PIE.

1.6.3.4 PIE Vector Table

The PIE vector table (see Table 1-112) 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 INT1.8, both interrupts are presented to the CPU simultaneously by the PIE block, and the CPU services INT1.1 first. If INT1.1 should occur simultaneously with INT1.8, then INT1.1 is sent to the CPU first and then INT1.8 follows. Interrupt prioritization is performed during the vector fetch portion of the interrupt processing.

When the PIE is enabled, a TRAP #1 through TRAP #12 or an INTR INT1 to INTR INT12 instruction transfers program control to the interrupt service routine corresponding to the first vector within the PIE group. For example: TRAP #1 fetches the vector from INT1.1, TRAP #2 fetches the vector from INT2.1 and so forth. Similarly an OR IFR, #16-bit operation causes the vector to be fetched from INTR1.1 to INTR12.1 locations, if the respective interrupt flag is set. All other TRAP, INTR, OR IFR,#16-bit operations fetch the vector from the respective table location. The vector table is EALLOW protected.

Out of the 96 possible MUXed interrupts in Table 1-111, 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	ADCINT9	XINT2	XINT1	Reserved	ADCINT2	ADCINT1
	(LPM/WD)	(TIMER 0)	(ADC)	Ext. int. 2	Ext. int. 1	-	(ADC)	(ADC)
	0xD4E	0xD4C	0xD4A	0xD48	0xD46	0xD44	0xD42	0xD40
INT2.y	Reserved	EPWM7_TZINT	EPWM6_TZINT	EPWM5_TZINT	EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
	–	(ePWM7)	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD5E	0xD5C	0xD5A	0xD58	0xD56	0xD54	0xD52	0xD50
INT3.y	Reserved	EPWM7_INT	EPWM6_ INT	EPWM5_INT	EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
	-	(EPWM7)	(ePWM6)	(ePWM5)	(ePWM4)	(ePWM3)	(ePWM2)	(ePWM1)
	0xD6E	0xD6C	0xD6A	0xD68	0xD66	0xD64	0xD62	0xD60
INT4.y	HRCAP2_INT	HRCAP1_INT	Reserved	Reserved	Reserved	Reserved	Reserved	ECAP1_INT
	(HRCAP2)	(HRCAP1)	–	–	–	–	–	(eCAP1)
	0xD7E	0xD7C	0xD7A	0xD78	0xD76	0xD74	0xD72	0xD70
INT5.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	EQEP1_INT
	-	-	-	-	-	-	-	(eQEP1)
	0xD8E	0xD8C	0xD8A	0xD88	0xD86	0xD84	0xD82	0xD80
INT6.y	Reserved	Reserved	Reserved	Reserved	SPITXINTB	SPIRXINTB	SPITXINTA	SPIRXINTA
	-	-	-	-	(SPI-B)	(SPI-B)	(SPI-A)	(SPI-A)
	0xD9E	0xD9C	0xD9A	0xD98	0xD96	0xD94	0xD92	0xD90
INT7.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	-	-	-	-	-	-	-	-
	0xDAE	0xDAC	0xDAA	0xDA8	0xDA6	0xDA4	0xDA2	0xDA0
INT8.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	I2CINT2A	I2CINT1A
	–	–	–	–	–	–	(I2C-A)	(I2C-A)
	0xDBE	0xDBC	0xDBA	0xDB8	0xDB6	0xDB4	0xDB2	0xDB0
INT9.y	Reserved	Reserved	ECANA_INT1	ECANA_INT0	LINA_INT1	LINA_INT0	SCITXINTA	SCIRXINTA
	–	–	(CAN-A)	(CAN-A)	(LIN-A)	(LIN-A)	(SCI-A)	(SCI-A)
	0xDCE	0xDCC	0xDCA	0xDC8	0xDC6	0xDC4	0xDC2	0xDC0
INT10.y	ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1
	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)	(ADC)
	0xDDE	0xDDC	0xDDA	0xDD8	0xDD6	0xDD4	0xDD2	0xDD0
INT11.y	CLA1_INT8	CLA1_INT7	CLA1_INT6	CLA1_INT5	CLA1_INT4	CLA1_INT3	CLA1_INT2	CLA1_INT1
	(CLA)	(CLA)	(CLA)	(CLA)	(CLA)	(CLA)	(CLA)	(CLA)
	0xDEE	0xDEC	0xDEA	0xDE8	0xDE6	0xDE4	0xDE2	0xDE0
INT12.y	LUF	LVF	Reserved	Reserved	Reserved	Reserved	Reserved	XINT3
	(CLA)	(CLA)	–	–	–	–	–	Ext. Int. 3
	0xDFE	0xDFC	0xDFA	0xDF8	0xDF6	0xDF4	0xDF2	0xDF0

Table 1-112. PIE Vector Table

Name	VECTOR ID	Address [1]	Size (x16)	Description [2]		CPU Priority	PIE Group Priority
Reset	0	0x0000 0D00	2	Reset is always fetched from location 0x003F FFC0 in Boot ROM.		1 (highest)	-
INT1	1	0x0000 0D02	2	Not used. See PIE Group 1		5	-
INT2	2	0x0000 0D04	2	Not used. See PIE Group 2		6	-
INT3	3	0x0000 0D06	2	Not used. See PIE Group 3		7	-
INT4	4	0x0000 0D08	2	Not used. See PIE Group 4		8	-
INT5	5	0x0000 0D0A	2	Not used. See PIE Group 5		9	-
INT6	6	0x0000 0D0C	2	Not used. See PIE Group 6		10	-
INT7	7	0x0000 0D0E	2	Not used. See PIE Group 7		11	-
INT8	8	0x0000 0D10	2	Not used. See PIE Group 8		12	-
INT9	9	0x0000 0D12	2	Not used. See PIE Group 9		13	-
INT10	10	0x0000 0D14	2	Not used. See PIE Group 10		14	-
INT11	11	0x0000 0D16	2	Not used. See PIE Group 11		15	-
INT12	12	0x0000 0D18	2	Not used. See PIE Group 12		16	-
INT13	13	0x0000 0D1A	2	CPU-Timer1		17	-
INT14	14	0x0000 0D1C	2	CPU-Timer2		18	-
DATALOG	15	0x0000 0D1E	2	CPU Data Logging Interrupt		19 (lowest)	-
RTOSINT	16	0x0000 0D20	2	CPU Real-Time OS Interrupt		4	-
EMUINT	17	0x0000 0D22	2	CPU Emulation Interrupt		2	-
NMI	18	0x0000 0D24	2	External Non-Maskable Interrupt		3	-
ILLEGAL	19	0x0000 0D26	2	Illegal Operation		-	-
USER1	20	0x0000 0D28	2	User-Defined Trap		-	-
USER2	21	0x0000 0D2A	2	User Defined Trap		-	-
USER3	22	0x0000 0D2C	2	User Defined Trap		-	-
USER4	23	0x0000 0D2E	2	User Defined Trap		-	-
USER5	24	0x0000 0D30	2	User Defined Trap		-	-
USER6	25	0x0000 0D32	2	User Defined Trap		-	-
USER7	26	0x0000 0D34	2	User Defined Trap		-	-
USER8	27	0x0000 0D36	2	User Defined Trap		-	-
USER9	28	0x0000 0D38	2	User Defined Trap		-	-
USER10	29	0x0000 0D3A	2	User Defined Trap		-	-
USER11	30	0x0000 0D3C	2	User Defined Trap		-	-
USER12	31	0x0000 0D3E	2	User Defined Trap		-	-
PIE Group 1 Vectors - MUXed into CPU INT1							
INT1.1	32	0x0000 0D40	2	ADCINT1	(ADC)	5	1 (highest)
INT1.2	33	0x0000 0D42	2	ADCINT2	(ADC)	5	2
INT1.3	34	0x0000 0D44	2	Reserved	-	5	3
INT1.4	35	0x0000 0D46	2	XINT1		5	4
INT1.5	36	0x0000 0D48	2	XINT2		5	5
INT1.6	37	0x0000 0D4A	2	ADCINT9	(ADC)	5	6
INT1.7	38	0x0000 0D4C	2	TINT0	(CPU-Timer0)	5	7
INT1.8	39	0x0000 0D4E	2	WAKEINT	(LPM/WD)	5	8 (lowest)

Table 1-112. PIE Vector Table (continued)

Name	VECTOR ID	Address [1]	Size (x16)	Description [2]		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	EPWM7_TZINT	(EPWM7)	6	7
INT2.8	47	0x0000 0D5E	2	Reserved	-	6	8 (lowest)
PIE Group 3 Vectors - MUXed into CPU INT3							
INT3.1	48	0x0000 0D60	2	EPWM1_INT	(EPWM1)	7	1 (highest)
INT3.2	49	0x0000 0D62	2	EPWM2_INT	(EPWM2)	7	2
INT3.3	50	0x0000 0D64	2	EPWM3_INT	(EPWM3)	7	3
INT3.4	51	0x0000 0D66	2	EPWM4_INT	(EPWM4)	7	4
INT3.5	52	0x0000 0D68	2	EPWM5_INT	(EPWM5)	7	5
INT3.6	53	0x0000 0D6A	2	EPWM6_INT	(EPWM6)	7	6
INT3.7	54	0x0000 0D6C	2	EPWM7_INT	(EPWM7)	7	7
INT3.8	55	0x0000 0D6E	2	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	Reserved	-	8	2
INT4.3	58	0x0000 0D74	2	Reserved	-	8	3
INT4.4	59	0x0000 0D76	2	Reserved	-	8	4
INT4.5	60	0x0000 0D78	2	Reserved	-	8	5
INT4.6	61	0x0000 0D7A	2	Reserved	-	8	6
INT4.7	62	0x0000 0D7C	2	HRCAP1_INT	-	8	7
INT4.8	63	0x0000 0D7E	2	HRCAP2_INT	-	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	Reserved	(EQEP2)	9	2
INT5.3	66	0x0000 0D84	2	Reserved	-	9	3
INT5.4	67	0x0000 0D86	2	Reserved	-	9	4
INT5.5	68	0x0000 0D88	2	Reserved	-	9	5
INT5.6	69	0x0000 0D8A	2	Reserved	-	9	6
INT5.7	70	0x0000 0D8C	2	Reserved	-	9	7
INT5.8	71	0x0000 0D8E	2	Reserved	-	9	8 (lowest)
PIE Group 6 Vectors - MUXed into CPU INT6							
INT6.1	72	0x0000 0D90	2	SPIRXINTA	(SPI-A)	10	1 (highest)
INT6.2	73	0x0000 0D92	2	SPITXINTA	(SPI-A)	10	2
INT6.3	74	0x0000 0D94	2	SPIRXINTB	(SPI-B)	10	3
INT6.4	75	0x0000 0D96	2	SPITXINTB	(SPI-B)	10	4
INT6.5	76	0x0000 0D98	2	Reserved	-	10	5
INT6.6	77	0x0000 0D9A	2	Reserved	-	10	6
INT6.7	78	0x0000 0D9C	2	Reserved	-	10	7
INT6.8	79	0x0000 0D9E	2	Reserved	-	10	8 (lowest)

Table 1-112. PIE Vector Table (continued)

Name	VECTOR ID	Address [1]	Size (x16)	Description [2]		CPU Priority	PIE Group Priority
PIE Group 7 Vectors - MUXed into CPU INT7							
INT7.1	80	0x0000 0DA0	2	Reserved	-	11	1 (highest)
INT7.2	81	0x0000 0DA2	2	Reserved	-	11	2
INT7.3	82	0x0000 0DA4	2	Reserved	-	11	3
INT7.4	83	0x0000 0DA6	2	Reserved	-	11	4
INT7.5	84	0x0000 0DA8	2	Reserved	-	11	5
INT7.6	85	0x0000 0DAA	2	Reserved	-	11	6
INT7.7	86	0x0000 0DAC	2	Reserved	-	11	7
INT7.8	87	0x0000 0DAE	2	Reserved	-	11	8 (lowest)
PIE Group 8 Vectors - MUXed into CPU INT8							
INT8.1	88	0x0000 0DB0	2	I2CINT1A	(I^2C-A)	12	1 (highest)
INT8.2	89	0x0000 0DB2	2	I2CINT2A	(I^2C-A)	12	2
INT8.3	90	0x0000 0DB4	2	Reserved	-	12	3
INT8.4	91	0x0000 0DB6	2	Reserved	-	12	4
INT8.5	92	0x0000 0DB8	2	Reserved	-	12	5
INT8.6	93	0x0000 0DBA	2	Reserved	-	12	6
INT8.7	94	0x0000 0DBC	2	Reserved	-	12	7
INT8.8	95	0x0000 0DBE	2	Reserved	-	12	8 (lowest)
PIE Group 9 Vectors - MUXed into CPU INT9							
INT9.1	96	0x0000 0DC0	2	SCIRXINTA	(SCI-A)	13	1 (highest)
INT9.2	97	0x0000 0DC2	2	SCITXINTA	(SCI-A)	13	2
INT9.3	98	0x0000 0DC4	2	LINAINT0	(LIN-A)	13	3
INT9.4	99	0x0000 0DC6	2	LINAINT1	(LIN-A)	13	4
INT9.5	100	0x0000 0DC8	2	ECANAINT0	(CAN-A)	13	5
INT9.6	101	0x0000 0DCA	2	ECANAINT1	(CAN-A)	13	6
INT9.7	102	0x0000 0DCC	2	Reserved	-	13	7
INT9.8	103	0x0000 0DCE	2	Reserved	-	13	8 (lowest)
PIE Group 10 Vectors - MUXed into CPU INT10							
INT10.1	104	0x0000 0DD0	2	ADCINT1	(ADC)	14	1 (highest)
INT10.2	105	0x0000 0DD2	2	ADCINT2	(ADC)	14	2
INT10.3	106	0x0000 0DD4	2	ADCINT3	(ADC)	14	3
INT10.4	107	0x0000 0DD6	2	ADCINT4	(ADC)	14	4
INT10.5	108	0x0000 0DD8	2	ADCINT5	(ADC)	14	5
INT10.6	109	0x0000 0DDA	2	ADCINT6	(ADC)	14	6
INT10.7	110	0x0000 0DDC	2	ADCINT7	(ADC)	14	7
INT10.8	111	0x0000 0DDE	2	ADCINT8	(ADC)	14	8 (lowest)
PIE Group 11 Vectors - MUXed into CPU INT11							
INT11.1	112	0x0000 0DE0	2	CLA1_INT1	(CLA)	15	1 (highest)
INT11.2	113	0x0000 0DE2	2	CLA1_INT2	(CLA)	15	2
INT11.3	114	0x0000 0DE4	2	CLA1_INT3	(CLA)	15	3
INT11.4	115	0x0000 0DE6	2	CLA1_INT4	(CLA)	15	4
INT11.5	116	0x0000 0DE8	2	CLA1_INT5	(CLA)	15	5
INT11.6	117	0x0000 0DEA	2	CLA1_INT6	(CLA)	15	6
INT11.7	118	0x0000 0DEC	2	CLA1_INT7	(CLA)	15	7
INT11.8	119	0x0000 0DEE	2	CLA1_INT8	(CLA)	15	8 (lowest)

Table 1-112. PIE Vector Table (continued)

Name	VECTOR ID	Address [1]	Size (x16)	Description [2]		CPU Priority	PIE Group Priority
PIE Group 12 Vectors - MUXed into CPU INT12							
INT12.1	120	0x0000 0DF0	2	XINT3	-	16	1 (highest)
INT12.2	121	0x0000 0DF2	2	Reserved	-	16	2
INT12.3	122	0x0000 0DF4	2	Reserved	-	16	3
INT12.4	123	0x0000 0DF6	2	Reserved	-	16	4
INT12.5	124	0x0000 0DF8	2	Reserved	-	16	5
INT12.6	125	0x0000 0DFA	2	Reserved	-	16	6
INT12.7	126	0x0000 0DFC	2	LVF	(CLA)	16	7
INT12.8	127	0x0000 0DFE	2	LUF	(CLA)	16	8 (lowest)

(1) Reset is always fetched from location 0x003F FFC0 in Boot ROM.
(2) All the locations within the PIE vector table are EALLOW protected.

1.6.4 PIE Configuration Registers

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

Table 1-113. PIE Configuration and Control Registers

Name	Address	Size (x16)	Description	Bit Description
PIECTRL	0x0000 - 0CE0	1	PIE, Control Register	Section 1.6.4.1
PIEACK	0x0000 - 0CE1	1	PIE, Acknowledge Register	Section 1.6.4.2
PIEIER1	0x0000 - 0CE2	1	PIE, INT1 Group Enable Register	Section 1.6.4.3
PIEIFR1	0x0000 - 0CE3	1	PIE, INT1 Group Flag Register	Section 1.6.4.4
PIEIER2	0x0000 - 0CE4	1	PIE, INT2 Group Enable Register	Section 1.6.4.3
PIEIFR2	0x0000 - 0CE5	1	PIE, INT2 Group Flag Register	Section 1.6.4.4
PIEIER3	0x0000 - 0CE6	1	PIE, INT3 Group Enable Register	Section 1.6.4.3
PIEIFR3	0x0000 - 0CE7	1	PIE, INT3 Group Flag Register	Section 1.6.4.4
PIEIER4	0x0000 - 0CE8	1	PIE, INT4 Group Enable Register	Section 1.6.4.3
PIEIFR4	0x0000 - 0CE9	1	PIE, INT4 Group Flag Register	Section 1.6.4.4
PIEIER5	0x0000 - 0CEA	1	PIE, INT5 Group Enable Register	Section 1.6.4.3
PIEIFR5	0x0000 - 0CEB	1	PIE, INT5 Group Flag Register	Section 1.6.4.4
PIEIER6	0x0000 - 0CEC	1	PIE, INT6 Group Enable Register	Section 1.6.4.3
PIEIFR6	0x0000 - 0CED	1	PIE, INT6 Group Flag Register	Section 1.6.4.4
PIEIER7	0x0000 - 0CEE	1	PIE, INT7 Group Enable Register	Section 1.6.4.3
PIEIFR7	0x0000 - 0CEF	1	PIE, INT7 Group Flag Register	Section 1.6.4.4
PIEIER8	0x0000 - 0CF0	1	PIE, INT8 Group Enable Register	Section 1.6.4.3
PIEIFR8	0x0000 - 0CF1	1	PIE, INT8 Group Flag Register	Section 1.6.4.4
PIEIER9	0x0000 - 0CF2	1	PIE, INT9 Group Enable Register	Section 1.6.4.3
PIEIFR9	0x0000 - 0CF3	1	PIE, INT9 Group Flag Register	Section 1.6.4.4
PIEIER10	0x0000 - 0CF4	1	PIE, INT10 Group Enable Register	Section 1.6.4.3
PIEIFR10	0x0000 - 0CF5	1	PIE, INT10 Group Flag Register	Section 1.6.4.4
PIEIER11	0x0000 - 0CF6	1	PIE, INT11 Group Enable Register	Section 1.6.4.3
PIEIFR11	0x0000 - 0CF7	1	PIE, INT11 Group Flag Register	Section 1.6.4.4
PIEIER12	0x0000 - 0CF8	1	PIE, INT12 Group Enable Register	Section 1.6.4.3
PIEIFR12	0x0000 - 0CF9	1	PIE, INT12 Group Flag Register	Section 1.6.4.4

1.6.4.1 PIE Control (PIECTRL) Register

Figure 1-90. PIE Control (PIECTRL) Register

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 (PIECTRL) Register Field Descriptions

Bits	Field	Value	Description
15-1	PIEVECT		These bits indicate the address within the PIE vector table from which the vector was fetched. The least significant bit of the address is ignored and only bits 1 to 15 of the address is shown. You can read the vector value to determine which interrupt generated the vector fetch. **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 (PIEACK) Register

Figure 1-91. PIE Interrupt Acknowledge (PIEACK) Register

15	12	11	0
Reserved		PIEACK	
R-0		R/W1C-0	

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

Table 1-115. PIE Interrupt Acknowledge (PIEACK) Register Field Descriptions

Bits	Field	Value	Description
15-12	Reserved		Reserved
11-0	PIEACK		Each bit in PIEACK refers to a specific PIE group. Bit 0 refers to interrupts in PIE group 1 that are MUXed into $\overline{\text{INT1}}$ up to Bit 11, which refers to PIE group 12 which is MUXed into CPU $\overline{\text{IN T12}}$
		bit x = 0 (1)	If a bit reads as a 0, it indicates that the PIE can send an interrupt from the respective group to the CPU. Writes of 0 are ignored.
		bit x = 1	Reading a 1 indicates if an interrupt from the respective group has been sent to the CPU and all other interrupts from the group are currently blocked. Writing a 1 to the respective interrupt bit clears the bit and enables the PIE block to drive a pulse into the CPU interrupt input if an interrupt is pending for that group.

(1) bit x = PIEACK bit 0 to PIEACK bit 11. Bit 0 refers to CPU $\overline{\text{INT1}}$ up to bit 11 that refers to CPU $\overline{\text{INT12}}$.

1.6.4.3 PIE Interrupt Enable (PIEIERn) Registers

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

Note

Take care when clearing the PIEIERn bits during normal operation. See Section 1.6.3.2 for the proper procedure for handling these bits.

Figure 1-92. PIE Interrupt Enable (PIEIERn) Registers (n = 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 (PIEIERn) Registers Field Descriptions

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

1.6.4.4 PIE Interrupt Flag (PIEIFRn) Registers

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

CAUTION

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

Figure 1-93. PIE Interrupt Flag (PIEIFRn) Registers (n = 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 (PIEIFRn) Registers Field Descriptions

Bits	Field	Description
15-8	Reserved	Reserved
7	INTx.8	These register bits indicate whether an interrupt is currently active. They behave very much like the CPU interrupt flag register. When an interrupt is active, the respective register bit is set. The bit is cleared when the interrupt is serviced or by writing a 0 to the register bit. This register can also be read to determine which interrupts are active or pending. x = 1 to 12. INTx means CPU INT1 to INT12.
6	INTx.7	The PIEIFR register bit is cleared during the interrupt vector fetch portion of the interrupt processing.
5	INTx.6	Hardware has priority over CPU accesses to the PIEIFR registers.
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

1.6.4.5 CPU Interrupt Flag Register (IFR)

The CPU interrupt flag register (IFR), is a 16-bit, CPU register and is used to identify and clear pending interrupts. The IFR contains flag bits for all the maskable interrupts at the CPU level (INT1-INT14, DLOGINT and RTOSINT). When the PIE is enabled, the PIE module multiplexes interrupt sources for INT1-INT12.

When a maskable interrupt is requested, the flag bit in the corresponding peripheral control register is set to 1. If the corresponding mask bit is also 1, the interrupt request is sent to the CPU, setting the corresponding flag in the IFR. This indicates that the interrupt is pending or waiting for acknowledgment.

To identify pending interrupts, use the PUSH IFR instruction and then test the value on the stack. Use the OR IFR instruction to set IFR bits and use the AND IFR instruction to manually clear pending interrupts. All pending interrupts are cleared with the AND IFR #0 instruction or by a hardware reset.

The following events also clear an IFR flag:

- The CPU acknowledges the interrupt.
- The 28x device is reset.

Note

1. To clear a CPU IFR bit, you must write a zero to it, not a one.
2. When a maskable interrupt is acknowledged, only the IFR bit is cleared automatically. The flag bit in the corresponding peripheral control register is not cleared. If an application requires that the control register flag be cleared, the bit must be cleared by software.
3. When an interrupt is requested by an INTR instruction and the corresponding IFR bit is set, the CPU does not clear the bit automatically. If an application requires that the IFR bit be cleared, the bit must be cleared by software.
4. IMR and IFR registers pertain to core-level interrupts. All peripherals have their own interrupt mask and flag bits in their respective control/configuration registers. Note that several peripheral interrupts are grouped under one core-level interrupt.

Figure 1-94. Interrupt Flag Register (IFR) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 1-118. 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

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

Bits	Field	Value	Description
14	DLOGINT		Data logging interrupt fag. DLOGINT is the flag for data logging interrupts.
		0	No DLOGINT is pending
		1	At least one DLOGINT interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
13	INT14		Interrupt 14 flag. INT14 is the flag for interrupts connected to CPU interrupt level INT14.
		0	No INT14 interrupt is pending
		1	At least one INT14 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
12	INT13		Interrupt 13 flag. INT13 is the flag for interrupts connected to CPU interrupt level INT13I.
		0	No INT13 interrupt is pending
		1	At least one INT13 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
11	INT12		Interrupt 12 flag. INT12 is the flag for interrupts connected to CPU interrupt level INT12.
		0	No INT12 interrupt is pending
		1	At least one INT12 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
10	INT11		Interrupt 11 flag. INT11 is the flag for interrupts connected to CPU interrupt level INT11.
		0	No INT11 interrupt is pending
		1	At least one INT11 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
9	INT10		Interrupt 10 flag. INT10 is the flag for interrupts connected to CPU interrupt level INT10.
		0	No INT10 interrupt is pending
		1	At least one INT6 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
8	INT9		Interrupt 9 flag. INT9 is the flag for interrupts connected to CPU interrupt level INT6.
		0	No INT9 interrupt is pending
		1	At least one INT9 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
7	INT8		Interrupt 8 flag. INT8 is the flag for interrupts connected to CPU interrupt level INT6.
		0	No INT8 interrupt is pending
		1	At least one INT8 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
6	INT7		Interrupt 7 flag. INT7 is the flag for interrupts connected to CPU interrupt level INT7.
		0	No INT7 interrupt is pending
		1	At least one INT7 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
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

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

Bits	Field	Value	Description
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)

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

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

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

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

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

Figure 1-95. Interrupt Enable Register (IER) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

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

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

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

Bits	Field	Value	Description
11	INT12		Interrupt 12 enable. INT12 enables or disables CPU interrupt level INT12.
		0	Level INT12 is disabled
		1	Level INT12 is enabled
10	INT11		Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11.
		0	Level INT11 is disabled
		1	Level INT11 is enabled
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
		1	Level INT10 is enabled
8	INT9		Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0	Level INT9 is disabled
		1	Level INT9 is enabled
7	INT8		Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0	Level INT8 is disabled
		1	Level INT8 is enabled
6	INT7		Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT7.
		0	Level INT7 is disabled
		1	Level INT7 is enabled
5	INT6		Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
4	INT5		Interrupt 5 enable.INT5 enables or disables CPU interrupt level INT5.
		0	Level INT5 is disabled
		1	Level INT5 is enabled
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)

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 1-96. Debug Interrupt Enable Register (DBGIER) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

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

Bits	Field	Value	Description
15	RTOSINT		Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt.
		0	RTOSINT is disabled
		1	RTOSINT is enabled
14	DLOGINT		Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt.
		0	DLOGINT is disabled
		1	DLOGINT 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 1-120. Debug Interrupt Enable Register (DBGIER) Field Descriptions (continued)

Bits	Field	Value	Description
9	INT10		Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0	Level INT10 is disabled
		1	Level INT10 is enabled
8	INT9		Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0	Level INT9 is disabled
		1	Level INT9 is enabled
7	INT8		Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0	Level INT8 is disabled
		1	Level INT8 is enabled
6	INT7		Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT77.
		0	Level INT7 is disabled
		1	Level INT7 is enabled
5	INT6		Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0	Level INT6 is disabled
		1	Level INT6 is enabled
4	INT5		Interrupt 5 enable.INT5 enables or disables CPU interrupt level INT5.
		0	Level INT5 is disabled
		1	Level INT5 is enabled
3	INT4		Interrupt 4 enable.INT4 enables or disables CPU interrupt level INT4.
		0	Level INT4 is disabled
		1	Level INT4 is enabled
2	INT3		Interrupt 3 enable.INT3 enables or disables CPU interrupt level INT3.
		0	Level INT3 is disabled
		1	Level INT3 is enabled
1	INT2		Interrupt 2 enable.INT2 enables or disables CPU interrupt level INT2.
		0	Level INT2 is disabled
		1	Level INT2 is enabled
0	INT1		Interrupt 1 enable.INT1 enables or disables CPU interrupt level INT1.
		0	Level INT1 is disabled
		1	Level INT1 is enabled

1.6.5 External Interrupt Configuration Registers

Three external interrupts, XINT1 – XINT3, are supported. Each of these external interrupts can be selected for negative or positive edge triggered and can also be enabled or disabled. The masked interrupts also contain a 16-bit free running up counter that is reset to zero when a valid interrupt edge is detected. This counter can be used to accurately time stamp the interrupt.

Table 1-121. Interrupt Control and Counter Registers (not EALLOW Protected)

Name	Address Range	Size (x16)	Description	Bit Description
XINT1CR	0x0000 7070	1	XINT1 control register	Section 1.6.5.1
XINT2CR	0x0000 7071	1	XINT2 control register	Section 1.6.5.1
XINT3CR	0x0000 7072	1	XINT3 control register	Section 1.6.5.1
reserved	0x0000 7073 - 0x0000 7077	5		
XINT1CTR	0x0000 7078	1	XINT1 counter register	Section 1.6.5.2
XINT2CTR	0x0000 7079	1	XINT2 counter register	Section 1.6.5.2
XINT3CTR	0x0000 707A	1	XINT3 counter register	Section 1.6.5.2
reserved	0x0000 707B - 0x0000 707E	5		

1.6.5.1 External Interrupt Control Registers (XINTnCR)

XINT1CR through XINT3CR are identical except for the interrupt number; therefore, Figure 1-97 and Table 1-122 represent registers for external interrupts 1 through 3 as XINT*n*CR, where *n* = the interrupt number.

Figure 1-97. External Interrupt Control Registers (XINTnCR) (n = 1 to 3)

15	4	3	2	1	0
Reserved		POLARITY		Reserved	ENABLE
R-0		R/W-0		R-0	R/W-0

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

Table 1-122. External Interrupt Control Registers (XINTnCR) Field Descriptions

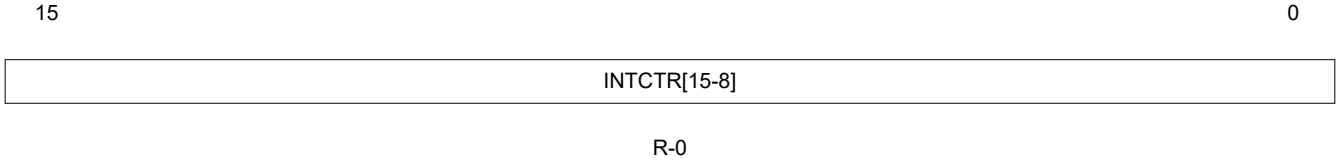
Bits	Field	Value	Description
15-4	Reserved		Any writes to these (bits) must always have a value of 0.
3-2	POLARITY		This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin.
		00	Interrupt generated on a falling edge (high-to-low transition)
		01	Interrupt generated on a rising edge (low-to-high transition)
		10	Interrupt generated on a falling edge (high-to-low transition)
		11	Interrupt generated on both a falling edge and a rising edge (high-to-low and low-to-high transition)
1	Reserved		Any writes to these (bits) must always have a value of 0.
0	ENABLE		This read/write bit enables or disables external interrupt XINTn.
		0	Disable interrupt
		1	Enable interrupt

1.6.5.2 External Interrupt n Counter Registers (XINTnCTR)

For XINT1/XINT2/XINT3, there is also a 16-bit counter that is reset to 0x000 whenever an interrupt edge is detected. These counters can be used to accurately time stamp an occurrence of the interrupt.

XINT1CTR through XINT3CTR are identical except for the interrupt number; therefore, Figure 1-98 and Table 1-123 represent registers for the external interrupts as XINTnCTR, where n = the interrupt number.

Figure 1-98. External Interrupt n Counter Registers (XINTnCTR)



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

Table 1-123. External Interrupt n Counter Registers (XINTnCTR)Field Descriptions

Bits	Field	Description
15-0	INTCTR	This is a free running 16-bit up-counter that is clocked at the SYSCLKOUT rate. The counter value is reset to 0x0000 when a valid interrupt edge is detected and then continues counting until the next valid interrupt edge is detected. When the interrupt is disabled, the counter stops. The counter is a free-running counter and wraps around to zero when the maximum value is reached. The counter is a read only register and can only be reset to zero by a valid interrupt edge or by reset.

1.7 VREG/BOR/POR

Although the core and I/O circuitry operate on two different voltages, these devices have an on-chip voltage regulator (VREG) to generate the V_{DD} voltage from the V_{DDIO} supply. This eliminates the cost and area of a second external regulator on an application board. Additionally, internal power-on reset (POR) and brown-out reset (BOR) circuits monitor both the V_{DD} and V_{DDIO} rails during power-up and run mode, eliminating a need for any external voltage supervisory circuits.

The V_{DD} BOR is only valid when the VREG is enabled. If VREG is disabled, and external LDO is used for 1.8V, then there is no BOR functIon on V_{DD}.

1.7.1 On-Chip Voltage Regulator (VREG)

An on-chip voltage regulator facilitates the powering of the device without adding the cost or board space of a second external regulator. This linear regulator generates the core V_{DD} voltage from the V_{DDIO} supply. Therefore, although capacitors are required on each V_{DD} pin to stabilize the generated voltage, power need not be supplied to these pins to operate the device. Conversely, the VREG can be bypassed or overdriven, should power or redundancy be the primary concern of the application.

1.7.1.1 Using the On-Chip VREG

To utilize the on-chip VREG, the VREGENZ pin should be pulled low and the appropriate recommended operating voltage should be supplied to the V_{DDIO} and V_{DDA} pins. In this case, the V_{DD} voltage needed by the core logic will be generated by the VREG. Each V_{DD} pin requires on the order of 1.2 µF capacitance for proper regulation of the VREG. These capacitors should be located as close as possible to the device pins. See the *TMS320F2803x Real-Time Microcontrollers Data Manual* for the acceptable range of capacitance.

1.7.1.2 Bypassing the On-Chip VREG

To conserve power, it is also possible to bypass the on-chip VREG and supply the core logic voltage to the V_{DD} pins with an external regulator. To enable this option, the VREGENZ pin must be pulled high. Refer to the *TMS320F2803x Real-Time Microcontrollers Data Manual* for the acceptable range of voltage that must be supplied to the V_{DD} pins.

1.7.2 On-chip Power-On Reset (POR) and Brown-Out Reset (BOR) Circuit

Two on-chip supervisory circuits, the power-on reset (POR) and the brown-out reset (BOR) remove the burden of monitoring the V_{DD} and V_{DDIO} supply rails from the application board. The purpose of the POR is to create a clean reset throughout the device during the entire power-up procedure. The trip point is a looser, lower trip point than the BOR, which watches for dips in the V_{DD} or V_{DDIO} rail during device operation. The POR function is present on both V_{DD} and V_{DDIO} rails at all times. After initial device power-up, the BOR function is present on V_{DDIO} at all times, and on V_{DD} when the internal VREG is enabled (VREGENZ pin is pulled low). Both functions pull the $\overline{XRS}$ pin low when one of the voltages is below their respective trip point. Additionally, when monitoring the V_{DD} rail, the BOR pulls $\overline{XRS}$ low when V_{DD} is above its over-voltage trip point. See the device data sheet for the various trip points as well as the delay time from the removal of the fault condition to the release of the $\overline{XRS}$ pin.

A bit is provided in the BORCFG register (address 0x985) to disable both the VDD and VDDIO BOR functions. The default state of this bit is to enable the BOR function. When the BOR functions are disabled, the POR functions will remain enabled. The BORCFG register is only reset by the XRS signal.

Figure 1-99. BOR Configuration (BORCFG) Register

15	3	2	1	0
Reserved		Reserved	Reserved	BORENZ
R-0		R-1	R/W-0	R/W-0

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

Table 1-124. BOR Configuration (BORCFG) Field Descriptions

Bits	Field	Value	Description
15-3	Reserved	0	Any writes to these bits must always have a value of 0.
2	Reserved	1	Reads always return a one. Writes have no effect.
1	Reserved	0	Any writes to these bits must always have a value of 0.
0	BORENZ		BOR enable active low bit.
		0	BOR functions are enabled.
		1	BOR functions are disabled.

This page intentionally left blank.

Chapter 2

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.

2.1 Boot ROM Memory Map

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

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

This document describes the following items:

- Bootloader functions
- Version number, release date and checksum
- Reset vector
- Illegal trap vector (ITRAP)
- CPU vector table (used for test purposes only)
- IQmath Tables
- Selected IQmath functions
- Flash API library

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

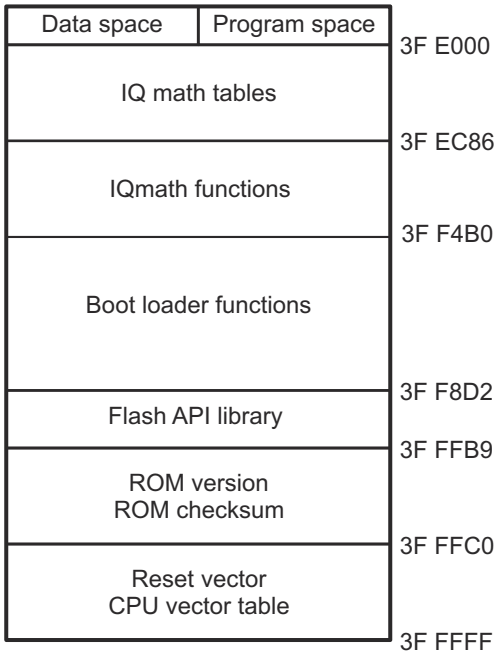


Figure 2-1. Memory Map of On-Chip ROM

2.1.1 On-Chip Boot ROM IQmath Tables

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

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

The IQmath library accesses the tables through the IQmathTables and the IQmathTablesRam linker sections. Both of these sections are completely included in the boot ROM.

If you do not wish to load a copy of these tables already included in the ROM into the device, use the boot ROM memory addresses and label the sections as “NOLOAD” as shown in Example 2-1. This facilitates referencing the lookup tables without actually loading the section to the target.

The preferred alternative to using the linker command file is to use the IQmath boot ROM symbol library. If this library is linked in the project before the IQmath library, and the linker-priority option is used, then any math tables and IQmath functions within the boot ROM will be used first. Refer to the IQMath library documentation for more information.

The following math tables are included in the boot ROM:

- **Sine/Cosine table, IQ Math Table**
  - Table size: 1282 words
  - Q format: Q30
  - Contents: 32-bit samples for one and a quarter period sine wave

  This is useful for accurate sine wave generation and 32-bit FFTs. This can also be used for 16-bit math; just skip over every second value
- **Normalized Inverse Table, IQ Math Table**
  - Table size: 528 words
  - Q format: Q29
  - Contents: 32-bit normalized inverse samples plus saturation limits

  This table is used as an initial estimate in the Newton-Raphson inverse algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.
- **Normalized Square Root Table, IQ Math Table**
  - Table size: 274 words
  - Q format: Q30
  - Contents: 32-bit normalized inverse square root samples plus saturation

  This table is used as an initial estimate in the Newton-Raphson square-root algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.
- **Normalized Arctan Table, IQ Math Table**
  - Table size: 452 words
  - Q format: Q30
  - Contents 32-bit second order coefficients for line of best fit plus normalization table

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

- **Exp Min/Max Table, IQMath Table**
  - Table size: 120 words
  - Q format: Q1 - Q30
  - Contents: 32-bit Min and Max values for each Q value
- **Exp Coefficient Table, IQMath Table**
  - Table size: 20 words
  - Q format: Q31
  - Contents: 32-bit coefficients for calculating exp (X) using a Taylor series
- **Inverse Sin/Cos Table, IQ Math Table**
  - Table size: 85 x 16
  - Q format: Q29
  - Contents: Coefficient table to calculate the formula f(x) = c4*x^4 + c3*x^3 + c2*x^2 + c1*x + c0.

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

```
MEMORY
{
    PAGE 0 :
    ...
    IQTABLES  : origin = 0x3FE000, length = 0x000b50
    IQTABLES2 : origin = 0x3FEB50, length = 0x00008c
    IQTABLES3 : origin = 0x3FEBDC, length = 0x0000AA
    ...
}
SECTIONS
{
    ...
    IQmathTables : load = IQTABLES, type = NOLOAD, PAGE = 0
    IQmathTables2 > IQTABLES2, type = NOLOAD, PAGE = 0
    {
         IQmath.lib<IQNexpTable.obj> (IQmathTablesRam)
    }
         IQmathTables3 : load = IQTABLES3, PAGE = 0
    {
         IQNasinTable.obj (IQmathTablesRam)
    }
    ...
}
```

2.1.2 On-Chip Boot ROM IQmath Functions

The following IQmath functions are included in the Boot ROM:

- IQNatan2 N= 15, 20, 24, 29
- IQNcos N= 15, 20, 24, 29
- IQNdiv N= 15, 20, 24, 29
- IQisqrt N= 15, 20, 24, 29
- IQNmag N= 15, 20, 24, 29
- IQNsin N= 15, 20, 24, 29
- IQNsqrt N= 15, 20, 24, 29

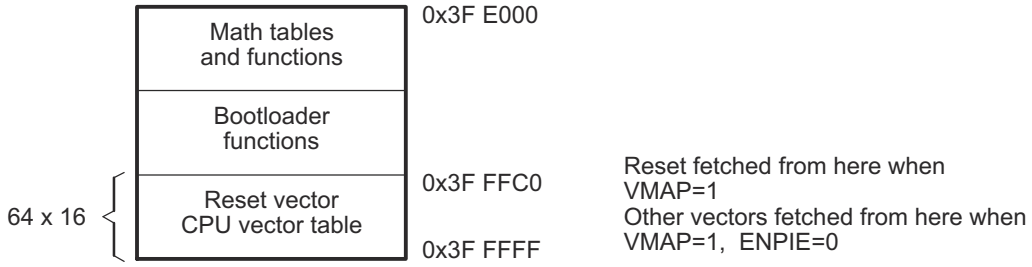
These functions can be accessed using the IQmath boot ROM symbol library included with the boot ROM source. If this library is linked in the project before the IQmath library, and the linker-priority option is used, then any math tables and IQmath functions within the boot ROM will be used first. Refer to the IQMath Library documentation for more information.

2.1.3 On-Chip Flash API

The boot ROM contains the API to program and erase the flash. This flash API can be accessed using the boot ROM flash API symbol library released with the boot ROM source. Refer to the Flash API Library documentation in C2000Ware for information on how to use the symbol library.

2.1.4 CPU Vector Table

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



A. 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.

B. The ENPIE bit is located in the PIECTRL register. The default state of this bit at reset is 0, which disables the Peripheral Interrupt Expansion block (PIE).

Figure 2-2. Vector Table Map

The only vector that will normally be handled from the internal boot ROM memory is the reset vector located at 0x3F FFC0. The reset vector is factory programmed to point to the InitBoot function stored in the boot ROM. This function starts the bootload process. A series of checking operations is performed on $\overline{\text{TRST}}$ and select general-purpose I/O (GPIO) pins to determine which boot mode to use. This boot mode selection is described in Section 2.2.9 of this document.

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

For TI silicon debug and test purposes the vectors located in the boot ROM memory point to locations in the M0 SARAM block as shown in Table 2-1. During silicon debug, you can program the specified locations in M0 with branch instructions to catch any vectors fetched from boot ROM. This is not required for normal device operation.

Table 2-1. Vector Locations

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

2.2 Bootloader Features

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

2.2.1 Bootloader Functional Operation

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

The bootloader provides a variety of different ways to download code to accommodate different system requirements. The bootloader uses the state of $\overline{\text{TRST}}$ and two GPIO signals to determine which boot mode to use. The boot mode selection process and the specifics of each bootloader are described in the remainder of this document. Figure 2-3 shows the basic bootloader flow:

The reset vector in boot ROM redirects program execution to the InitBoot function. After performing device initialization the bootloader will check the state of the $\overline{\text{TRST}}$ pin to determine if an emulation pod is connected.

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

In debugger boot, the boot ROM will check two SARAM locations called EMU_KEY and EMU_BMODE for a boot mode. If the contents of either location are invalid, then the "wait" boot mode is used. All boot mode options can be accessed by modifying the value of EMU_BMODE through the debugger when performing an debugger boot.

- **Standalone Boot ($\overline{\text{TRST}}$ = 0)**

If the device is in standalone boot mode, then the state of two GPIO pins is used to determine which boot mode will execute. Options include: GetMode, wait, SCI, and parallel I/O. Each of the modes is described in Section 2.2.9. The GetMode option by default boots to flash but can be customized by programming two values into OTP to select another boot loader.

After the selection process and if the required bootloading is complete, the processor will continue execution at an entry point determined by the boot mode selected. If a bootloader was called, then the input stream loaded by the peripheral determines this entry address. This data stream is described in Section 2.2.11. If, instead, you choose to boot directly to Flash, OTP, or SARAM, the entry address is predefined for each of these memory blocks.

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

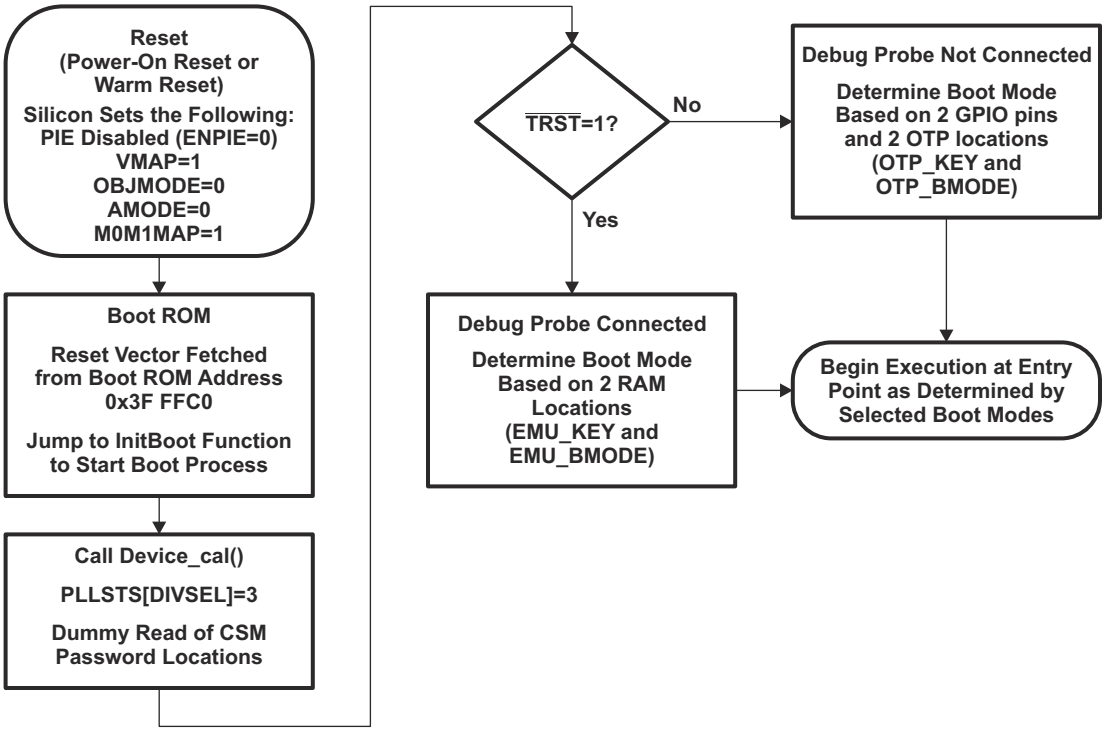


Figure 2-3. Bootloader Flow Diagram

2.2.2 Bootloader Device Configuration

At reset, the device is in C27x object-compatible mode. It is up to the application to place the device in the proper operating mode before execution proceeds.

When booting from the internal boot ROM, the device is configured for C28x operating mode by the boot ROM software. You are responsible for any additional configuration required.

For example, if your application includes C2xLP device source, then you are responsible for configuring the device for C2xLP source compatibility prior to execution of code generated from C2xLP source.

The configuration required for each operating mode is summarized in Table 2-2.

Table 2-2. Configuration for Device Modes

	C27x Mode (Reset)(2)	C28x Mode	C2xLP Source Compatible Mode(2)
OBJMODE	0	1	1
AMODE	0	0	1
PAGE0	0	0	0
M0M1MAP(1)	1	1	1
Other Settings			SXM = 1, C = 1, SPM = 0

(1) Normally for C27x compatibility, the M0M1MAP would be 0. On these devices, however, it is tied off high internally; therefore, at reset, M0M1MAP is always configured for C28x mode.

(2) C27x refers to the TMS320C27x family of processors. C2xLP refers to the TMS320F24x/ TMS320LF240xA family of devices that incorporate the C2xLP core. The information in the table above is for reference only and is not applicable for the typical user development. For more information on the C2xLP core, refer to the *TMS320C28x DSP CPU and Instruction Set Reference Guide*.

2.2.3 PLL Multiplier and DIVSEL Selection

The Boot ROM changes the PLL multiplier (PLLCR) and divider (PLLSTS[DIVSEL]) bits as follows:

- **All boot modes:**
  - PLLCR is not modified. PLLSTS[DIVSEL] is set to 3 for SYSCLKOUT = CLKIN/1. This increases the speed of the loaders.

Note

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

The reset value of PLLSTS[DIVSEL] is 0. This configures the device for SYSCLKOUT = CLKIN/4 . The boot ROM will change this to SYSCLKOUT = CLKIN/1 to improve performance of the loaders. PLLSTS[DIVSEL] is left in this state when the boot ROM exits and it is up to the application to change it before configuring the PLLCR register.

2.2.4 Watchdog Module

When branching directly to Flash, OTP, or M0 single-access RAM (SARAM) the watchdog is not touched. In the other boot modes, the watchdog is disabled before booting and then re-enabled and cleared before branching to the final destination address. In the case of an incorrect key value passed to the loader, the watchdog will be enabled and the device will boot to flash.

2.2.5 Taking an ITRAP Interrupt

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

2.2.6 Internal Pullup Circuit

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

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

2.2.7 PIE Configuration

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

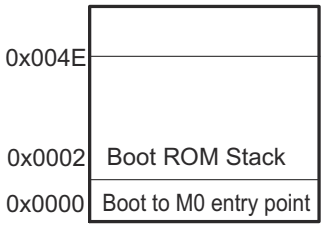
The boot ROM does, however, use the first six locations within the PIE vector table for emulation boot mode information and Flash API variables. These locations are not used by the PIE itself and not used by typical applications.

Note

If you are porting code from another 28x processor, check to see if the code initializes the first 6 locations in the PIE vector table to some default value. If it does, consider modifying the code to not write to these locations so the EMU boot mode will not be overwritten during debug.

2.2.8 Reserved Memory

The M0 memory block address range 0x0002 - 0x004E is reserved for the stack and .ebss code sections during the boot-load process. If code is bootloaded into this region there is no error checking to prevent it from corrupting the boot ROM stack. Address 0x0000-0x0001 is the boot to M0 entry point. This should be loaded with a branch instruction to the start of the main application when using "boot to SARAM" mode.



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

Figure 2-4. Boot ROM Stack

Note

If code or data is bootloaded into the address range address range 0x0002 - 0x004E, there is no error checking to prevent it from corrupting the boot ROM stack.

In addition, the first six locations of the PIE vector table are used by the boot ROM. These locations are not used by the PIE itself and not used by typical applications. These locations are used as SARAM by the boot ROM and will not affect the behavior of the PIE.

Note

Some example code from previous devices may initialize these locations. This will overwrite any boot mode you have populated. These locations are listed in Table 2-3.

Table 2-3. PIE Vector SARAM Locations Used by the Boot ROM

Location	Name	Note
0x0D00 x 16	EMU_KEY	Used for emulation boot
0x0D01 x 16	EMU_BMODE	Used for emulation boot
0x0D02 x 32	Flash_CallbackPtr	Used by the flash API
0x0D04 x 32	Flash_CPUScaleFactor	Used by the flash API

2.2.9 Bootloader Modes

To accommodate different system requirements, the boot ROM offers a variety of boot modes. This section describes the different boot modes and gives a brief summary of their functional operation. The states of $\overline{\text{TRST}}$ and two GPIO pins are used to determine the desired boot mode as shown in Table 2-4 and Figure 2-5.

Table 2-4. Boot Mode Selection

	GPIO37/TDO	GPIO34/ CMP2OUT	$\overline{\text{TRST}}$	
Mode EMU	x	x	1	Emulation Boot
Mode 0	0	0	0	Parallel I/O
Mode 1	0	1	0	SCI
Mode 2	1	0	0	Wait
Mode 3	1	1	0	GetMode

Note

The default behavior of the GetMode option on unprogrammed devices is to boot to flash. This behavior can be changed by programming two locations in the OTP as shown in Table 2-6. In addition, if these locations are used by an application, then GetMode will jump to flash as long as OTP_KEY != 0x55AA and/or OTP_BMODE is not a valid value.

This device does not support the hardware wait-in-reset mode that is available on other C2000 parts. The "wait" boot mode can be used to emulate a wait-in-reset mode. The "wait" mode is very important for debugging devices with the CSM password programmed (that is, secured). When the device is powered up, the CPU will start running and may execute an instruction that performs an access to an emulation code security logic (ECSL) protected area. If this happens, the ECSL will trip and cause the JTAG debug probe connection to be broken. The "wait" mode keeps this from happening by looping within the boot ROM until a JTAG debug probe is connected.

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

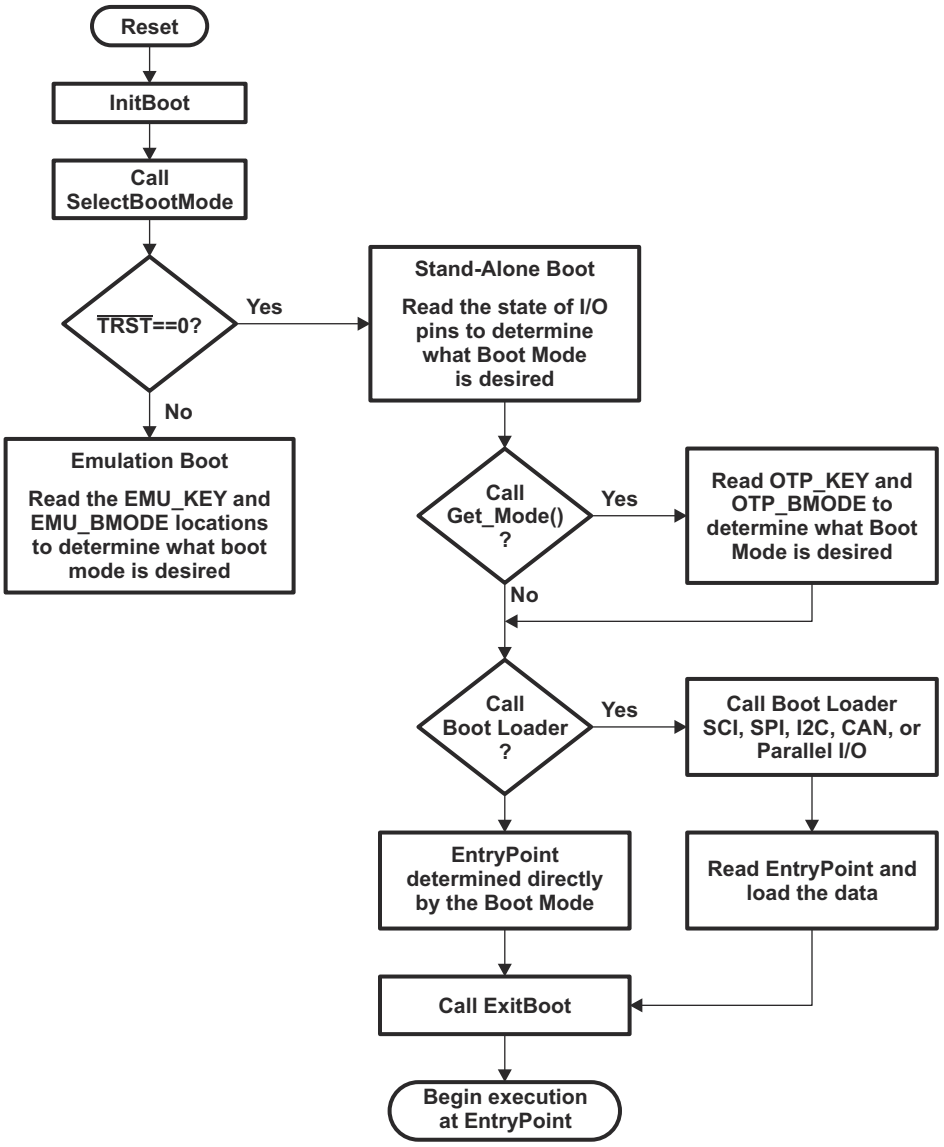


Figure 2-5. Boot ROM Function Overview

The following boot mode is used when a JTAG debug probe is connected:

- **Emulation Boot**

  In this case an emulation pod is connected to the device ($\overline{TRST}$ = 1) and the boot ROM derives the boot mode from the first two locations in the PIE vector table. These locations, called EMU_KEY and EMU_BMODE, are not used by the PIE module and not typically used by applications. Valid values for EMU_KEY and EMU_BMODE are shown in Table 2-5.

  An EMU_KEY value of 0x55AA indicates the EMU_BMODE is valid. An invalid key or invalid mode will result in calling the wait boot mode. EMU_BMODE and EMU_KEY are automatically populated by the boot ROM when powering up with $\overline{TRST}$ = 0. EMU_BMODE can also be initialized manually through the debugger.

Table 2-5. Valid EMU_KEY and EMU_BMODE Values

Address	Name	Value	
0x0D00	EMU_KEY	if TRST == 1 and EMU_KEY == 0x55AA, then check EMU_BMODE for the boot mode, else { Invalid EMU_KEY Boot mode = WAIT_BOOT }	
0x0D01	EMU_BMODE	0x0000	Boot mode = PARALLEL_BOOT
		0x0001	Boot mode = SCI_BOOT
		0x0002	Boot mode = WAIT_BOOT
		0x0003	Boot mode = GET_BOOT (GetMode from OTP_KEY/OTP_BMODE)
		0x0004	Boot mode = SPI_BOOT
		0x0005	Boot mode = I2C_BOOT (1)
		0x0006	Boot mode = OTP_BOOT
		0x0007	Boot mode = CAN_BOOT
		0x000A	Boot mode = RAM_BOOT
		0x000B	Boot mode = FLASH_BOOT
		Other	Boot mode = WAIT_BOOT

(1) I2C boot uses GPIO32 and GPIO33, which are not available on all packages.

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

Following are two examples of an emulation boot.

2.2.9.1 Example 1: Debug an application that loads through the SCI at boot.

To debug an application that loads through the SCI at boot, follow these steps:

- Configure the pins for mode 1, SCI, and initiate a power-on-reset.
- The boot ROM will detect $\overline{TRST}$ = 0 and will use the two pins to determine SCI boot.
- The boot ROM populates EMU_KEY with 0x55AA and EMU_BMODE with SCI_BOOT.
- The boot ROM waits in the SCI loader for data.
- Connect the debugger. $\overline{TRST}$ will go high.
- Perform a debugger reset and run. The boot loader will use the EMU_BMODE and boot to SCI.

2.2.9.2 Example 2: You want to connect your JTAG debug probe, but do not want application code to start executing before the JTAG debug probe connects.

To connect your JTAG debug probe, but keep application code from executing before the JTAG debug probe connects:

- Configure GPIO37 and GPIO34 pins for mode 2, WAIT, and initiate a power-on-reset.
- The boot ROM will detect $\overline{\text{TRST}}$ = 0 and will use the two pins to determine wait boot.
- The boot ROM populates EMU_KEY with 0x55AA and EMU_BMODE with WAIT_BOOT.
- The boot ROM waits in the wait routine.
- Connect the debugger; $\overline{\text{TRST}}$ will go high.
- Modify the EMU_BMODE via the debugger to boot to FLASH or other desired boot mode.
- Perform a debugger reset and run. The boot loader will use the EMU_BMODE and boot to the desired loader or location.

Note

The behavior of JTAG debug probes with regards to $\overline{\text{TRST}}$ differs. Some JTAG debug probes pull $\overline{\text{TRST}}$ high only when Code Composer Studio is in a connected state. For these JTAG debug probes, if CCS is disconnected $\overline{\text{TRST}}$ will return to a low state. With CCS disconnected, GPIO34 and GPIO37 will be used to determine the boot mode. For these JTAG debug probes, this is true even if the JTAG debug probe pod is physically connected.

Some JTAG debug probes pull $\overline{\text{TRST}}$ high when CCS connects and leave it high as long as the power sense pin is active. $\overline{\text{TRST}}$ will remain high even after CCS disconnects. For these JTAG debug probes, the EMU mode stored in RAM will be used unless the target is power cycled, causing the state of $\overline{\text{TRST}}$ to reset back to a low state.

The following boot modes are invoked by the state of the boot mode pins if an JTAG debug probe is not connected:

- **Wait**

  This device does not support the hardware wait-in-reset mode that is available on other C2000 parts. The "wait" boot mode can be used to emulate a wait-in-reset mode. The "wait" mode is very important for debugging devices with the CSM password programmed (that is, secured). When the device is powered up, the CPU will start running and may execute an instruction that performs an access to a emulation code security logic (ECSL) protected area. If this happens, the ECSL will trip and cause the JTAG debug probe connection to be broken. The "wait" mode keeps this from happening by looping within the boot ROM until a JTAG debug probe is connected

  This mode writes WAIT_BOOT to EMU_BMODE. Once the JTAG debug probe is connected you can then manually populate the EMU_BMODE with the appropriate boot mode for the debug session.
- **SCI**

  In this mode, the boot ROM will load code to be executed into on-chip memory via the SCI-A port. When invoked as a stand-alone mode, the boot ROM writes SCI_BOOT to EMU_BMODE.
- **Parallel I/O 8-bit**

  The parallel I/O boot mode is typically used only by production flash programmers.
- **GetMode**

  The GetMode option uses two locations within the OTP to determine the boot mode. On an unprogrammed device this mode will always boot to flash. On a programmed device, you can choose to program these locations to change the behavior. If either of these locations is not an expected value, then boot to flash will be used.

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

Table 2-6. OTP Values for GetMode

Address	Name	Value
0x3D 7BFE	OTP_KEY	GetMode will be entered if one of the two conditions is true: Case 1: $\overline{TRST}$ == 0, GPIO34 == 1 and GPIO37 == 1 Case 2: $\overline{TRST}$ == 1, EMU_KEY == 0x55AA and EMU_BMODE == GET_BOOT GetMode first checks the value of OTP_KEY: if OTP_KEY == 0x55AA, then check OTP_BMODE for the boot mode else { Invalid key: Boot mode = FLASH_BOOT }
0x3D 7BFF	OTP_BMODE	0x0001 Boot mode = SCI_BOOT 0x0004 Boot mode = SPI_BOOT 0x0005 Boot mode = I2C_BOOT (1) 0x0006 Boot mode = OTP_BOOT 0x0007 Boot mode = CAN_BOOT Other Boot mode = FLASH_BOOT

(1) The I2C boot loader uses GPIO32 and GPIO33, which are not available on all packages.

The following boot modes are available through the emulation boot option. Some are also available as a programmed get mode option.

- **Jump to M0 SARAM**

  This mode is only available in emulation boot. The boot ROM software configures the device for 28x operation and branches directly to address 0x000000. This is the first address in the M0 memory block.

- **Jump to branch instruction in Flash memory.**

  Jump to Flash is the default behavior of the Get Mode boot option. Jump to flash is also available as an emulation boot option.

  In this mode, the boot ROM software configures the device for 28x operation and branches directly to location 0x3F 7FF6. This location is just before the 128-bit code security module (CSM) password locations. You are required to have previously programmed a branch instruction at location 0x3F 7FF6 that will redirect code execution to either a custom boot-loader or the application code.

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

  Jump to SPI is available in stand-alone mode as a programmed Get Mode option. That is, to configure a device for SPI boot in stand-alone mode, the OTP_KEY and OTP_BMODE locations must be programmed for SPI_BOOT and the boot mode pins configured for the Get Mode boot option.

  SPI boot is also available as an emulation boot option.

  In this mode, the boot ROM will load code and data into on-chip memory from an external SPI EEPROM or SPI flash via the SPI-A port.

- **I2C-A boot mode (I2C-A)**

  Jump to I2C is available in stand-alone mode as a programmed Get mode option. That is, to configure a device for I2C boot in stand-alone mode, the OTP_KEY and OTP_BMODE locations must be programmed for I2C_BOOT and the boot mode pins configured for the Get Mode boot option.

  I2C boot is also available as an emulation boot option.

  In this mode, the boot ROM will load code and data into on-chip memory from an external serial EEPROM or flash at address 0x50 on the I2C-A bus.

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

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

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

$\overline{\text{TRST}}$	GPIO37/ TDO	GPIO34	EMU KEY	EMU BMODE	OTP KEY	OTP BMODE	Boot Mode Selected (1)	EMU KEY	EMU BMODE
			Read from 0x0D00	Read from 0x0D01	Read from 0x3D7BFE	Read from 0x3D7BFF		Written to 0x0D00	Written to 0x0D01
1	x (2)	x	!=0x55AA	x	x	x	Wait	-	-
			0x55AA	0x0000	x	x	Parallel I/O	-	-
				0x0001	x	x	SCI	-	-
				0x0002	x	x	Wait	-	-
				0x0003	!= 0x55AA	x	GetMode: Flash	-	-
					0x55AA	0x0001	GetMode: SCI	-	-
						0x0003	GetMode: Flash	-	-
						0x0004	GetMode: SPI	-	-
						0x0005	GetMode: I2C (3)	-	-
						0x0006	GetMode: OTP	-	-
						0x0007	GetMode: CAN	-	
						Other	GetMode: Flash	-	-
				0x0004	x	x	SPI	-	-
				0x0005	x	x	I2C (3)	-	-
				0x0006	x	x	OTP	-	-
				0x0007	x	x	CAN	-	-
				0x000A	x	x	Boot to RAM	-	-
				0x000B	x	x	Boot to FLASH	-	-
				Other	x	x	Wait	-	-

1. Get Mode indicated the boot mode was derived from the values programmed in the OTP_KEY and OTP_BMODE locations.
2. x = don't care.
3. I2C uses GPIO32 and GPIO33 which are not available on all packages.

Table 2-8. Standalone Boot Modes (TRST = 0)

$\overline{\text{TRST}}$	GPIO37/ TDO	GPIO34	EMU KEY	EMU BMODE	OTP KEY	OTP BMODE	Boot Mode Selected[1]	EMU KEY[2]	EMU BMODE[2]
			Read from 0x0D00	Read from 0x0D01	Read from 0x3D7BFE	Read from 0x3D7BFF		Written to 0x0D00	Written to 0x0D01
0	0	0	x [3]	x	x	x	Parallel I/O	0x55AA	0x0000
0	0	1	x	x	x	x	SCI	0x55AA	0x0001
0	1	0	x	x	x	x	Wait	0x55AA	0x0002
0	1	1	x	x	!=0x55AA	x	GetMode: Flash	0x55AA	0x0003
					0x55AA	0x0001	GetMode: SCI		
						0x0003	GetMode: Flash		
						0x0004	GetMode: SPI		
						0x0005	GetMode: I2C		
						0x0006	GetMode: OTP		
						0x0007	GetMode: CAN		
						Other	GetMode: Flash		

(1) Get Mode indicates the boot mode was derived from the values programmed in the OTP_KEY and OTP_BMODE locations.
(2) The boot ROM will write this value to EMU_KEY and EMU_BMODE. This value can be used or overwritten by the user if a debugger is connected.
(3) x = don't care.

2.2.10 Device_Cal

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

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

Note

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

Step 1: Create a pointer to the *Device_cal()* function as shown in Example 2-2. This #define is included in C2000Ware.

Step 2: Call the function pointed to by *Device_cal* as shown in Example 2-2. The ADC clocks must be enabled before making this call.

Example 2-2. Calling the Device_cal() function

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

2.2.11 Bootloader Data Stream Structure

The basic structure is the same for all the bootloaders and is based on the C54x source data stream generated by the C54x hex utility. The C28x hex utility (hex2000.exe) has been updated to support this structure. The hex2000.exe utility is included with the C2000 code generation tools. All values in the data stream structure are in hex.

The first 16-bit word in the data stream is known as the key value, which is 0x08AA. If a bootloader receives an invalid key value, then the load is aborted.

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

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

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

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

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

In 8-bit mode, the least significant byte (LSB) of the word is sent first followed by the most significant byte (MSB). For 32-bit values, such as a destination address, the most significant word (MSW) is loaded first, followed by the least significant word (LSW). The bootloaders take this into account when loading an 8-bit data stream. Table 2-9 and Example 2-3 show the structure of the incoming data stream to the bootloader.

Table 2-9. LSB/MSB Loading Sequence in 8-bit Data Stream

Byte		Contents	
		LSB (First Byte of 2)	MSB (Second Byte of 2)
1	2	LSB: AA (KeyValue for memory width = 8 bits)	MSB: 08h (KeyValue for memory width = 8 bits)
3	4	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
5	6	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
7	8	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
...	...	...	...
...	...	...	...
17	18	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
19	20	LSB: Upper half of Entry point PC[23:16]	MSB: Upper half of entry point PC[31:24] (Always 0x00)
21	22	LSB: Lower half of Entry point PC[7:0]	MSB: Lower half of Entry point PC[15:8]
23	24	LSB: Block size in words of the first block to load. If the block size is 0, this indicates the end of the source program. Otherwise another block follows. For example, a block size of 0x000A would indicate 10 words or 20 bytes in the block.	MSB: block size
25	26	LSB: MSW destination address, first block Addr[23:16]	MSB: MSW destination address, first block Addr[31:24]
27	28	LSB: LSW destination address, first block Addr[7:0]	MSB: LSW destination address, first block Addr[15:8]
29	30	LSB: First word of the first block being loaded	MSB: First word of the first block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the first block to load	MSB: Last word of the first block to load
.	.	LSB: Block size of the second block	MSB: Block size of the second block
.	.	LSB: MSW destination address, second block Addr[23:16]	MSB: MSW destination address, second block Addr[31:24]
.	.	LSB: LSW destination address, second block Addr[7:0]	MSB: LSW destination address, second block Addr[15:8]
.	.	LSB: First word of the second block being loaded	MSB: First word of the second block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the second block	MSB: Last word of the second block
.	.	LSB: Block size of the last block	MSB: Block size of the last block
.	.	LSB: MSW of destination address of last block Addr[23:16]	MSB: MSW destination address, last block Addr[31:24]
.	.	LSB: LSW destination address, last block Addr[7:0]	MSB: LSW destination address, last block Addr[15:8]
.	.	LSB: First word of the last block being loaded	MSB: First word of the last block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the last block	MSB: Last word of the last block
n	n+1	LSB: 00h	MSB: 00h - indicates the end of the source

Example 2-3. Data Stream Structure 8-bit

```
AA 08          ; 0x08AA 8-bit key value
00 00 00 00    ; 8 reserved words
00 00 00 00
00 00 00 00
00 00 00 00
3F 00 00 80    ; 0x003F8000 EntryAddr, starting point after boot load completes
05 00          ; 0x0005 - First block consists of 5 16-bit words
3F 00 10 90    ; 0x003F9010 - First block will be loaded starting at 0x3F9010
01 00          ; Data loaded = 0x0001 0x0002 0x0003 0x0004 0x0005
02 00
03 00
04 00
05 00
02 00          ; 0x0002 - 2nd block consists of 2 16-bit words
3F 00 00 80    ; 0x003F8000 - 2nd block will be loaded starting at 0x3F8000
00 77          ; Data loaded = 0x7700 0x7625
25 76
00 00          ; 0x0000 - Size of 0 indicates end of data stream
After load has completed the following memory values will have been initialized as follows:
Location   Value
0x3F9010   0x0001
0x3F9011   0x0002
0x3F9012   0x0003
0x3F9013   0x0004
0x3F9014   0x0005
0x3F8000   0x7700
0x3F8001   0x7625
PC Begins execution at 0x3F8000
```

2.2.12 Basic Transfer Procedure

Figure 2-6 illustrates the basic process a bootloader uses to transfer data and start program execution. This process occurs after the bootloader determines the valid boot mode selected by the state of the $\overline{\text{TRST}}$ and GPIO pins.

The bootloader compares the first word sent by the host against the key value of 0x08AA. If the first word matches the key value, the loader continuously fetches data until it transfers all the hex words and then starts program execution. But, if the key does not match the key value, then the loader aborts and the CPU jumps to the default flash entry point.

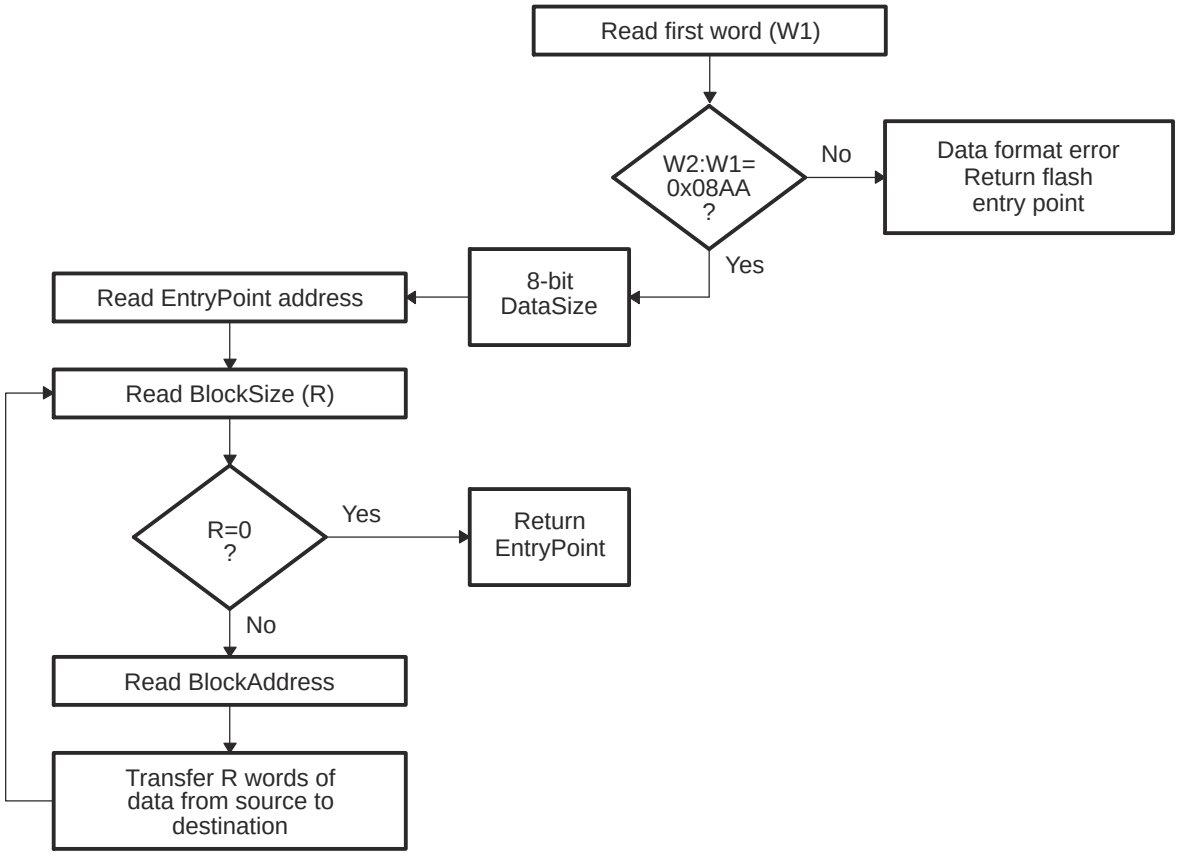


Figure 2-6. Bootloader Basic Transfer Procedure

Note

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

2.2.13 InitBoot Assembly Routine

The first routine called after reset is the *InitBoot* assembly routine. This routine initializes the device for operation in C28x object mode. *InitBoot* also performs a dummy read of the Code Security Module (CSM) password locations. If the CSM passwords are erased (all 0xFFFFs) then this has the effect of unlocking the device. Otherwise the device will remain locked and this dummy read of the password locations will have no effect. This can be useful if you have a new device that you want to boot load.

After the dummy read of the CSM password locations, the *InitBoot* routine calls the *SelectBootMode* function. This function determines the type of boot mode desired by the state of $\overline{\text{TRST}}$ and certain GPIO pins. This process is described in Section 2.2.14. Once the boot is complete, the *SelectBootMode* function passes back the entry point address (*EntryAddr*) to the *InitBoot* function. The *EntryAddr* is the location where code execution will begin after the bootloader exits. *InitBoot* then calls the *ExitBoot* routine that then restores CPU registers to their reset state and exits to the *EntryAddr* that was determined by the boot mode.

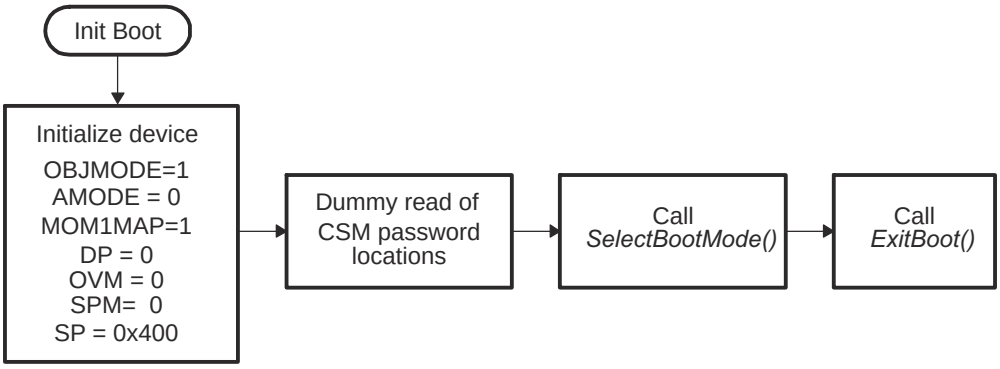


Figure 2-7. Overview of InitBoot Assembly Function

2.2.14 SelectBootMode Function

To determine the desired boot mode, the *SelectBootMode* function examines the state of $\overline{\text{TRST}}$ and 2 GPIO pins as shown in Table 2-4.

For a boot mode to be selected, the pins corresponding to the desired boot mode have to be pulled low or high until the selection process completes. Note that the state of the selection pins is not latched at reset; they are sampled some cycles later in the *SelectBootMode* function. The internal pullup resistors are enabled at reset for the boot mode selection pins. It is still suggested that the boot mode configuration be made externally to avoid the effect of any noise on these pins.

Note

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

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

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

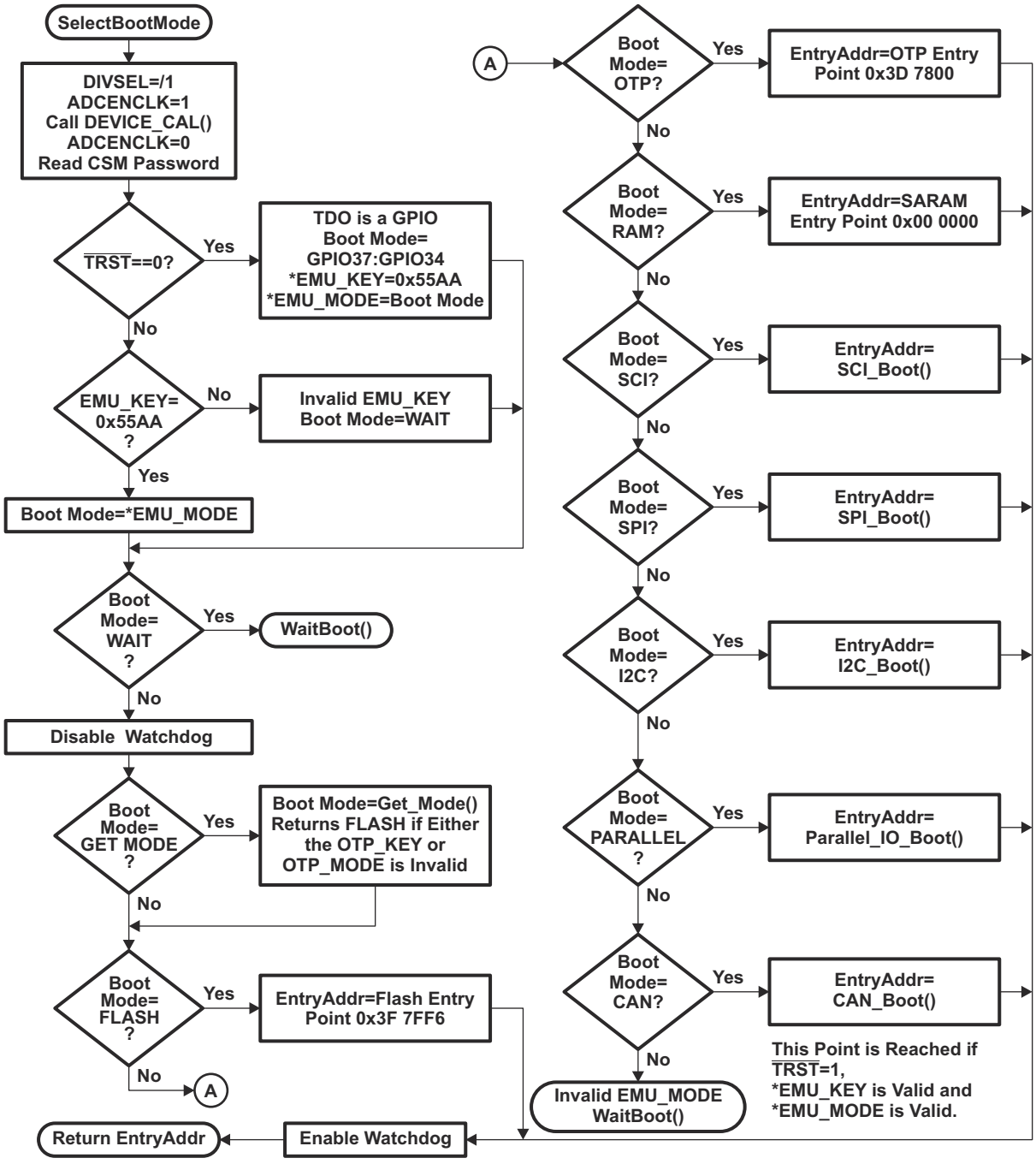


Figure 2-8. Overview of the SelectBootMode Function

Note

Throughout the SelectBootMode function code there are various references to a 32-pin package. This is a TI test package and can be ignored when considering the 2803x device boot mode selection.

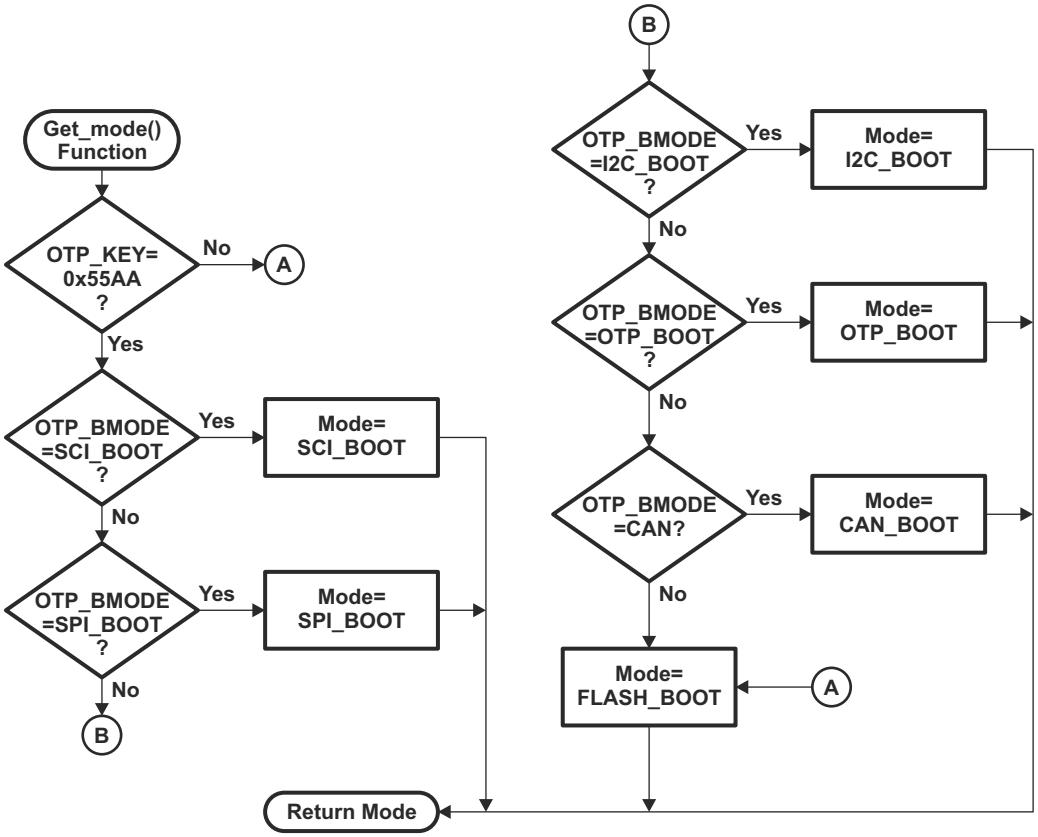


Figure 2-9. Overview of Get_mode() Function

2.2.15 CopyData Function

All bootloaders use the same function to copy data from the port to the device's SARAM. This function is the *CopyData()* function. This function uses a pointer to a *GetWordData* function that is initialized by each of the loaders to properly read data from that port. For example, when the SPI loader is evoked, the *GetWordData* function pointer is initialized to point to the *SPI-specific SPI_GetWordData* function. Thus when the *CopyData()* function is called, the correct port is accessed. The flow of the *CopyData* function is shown in Figure 2-10.

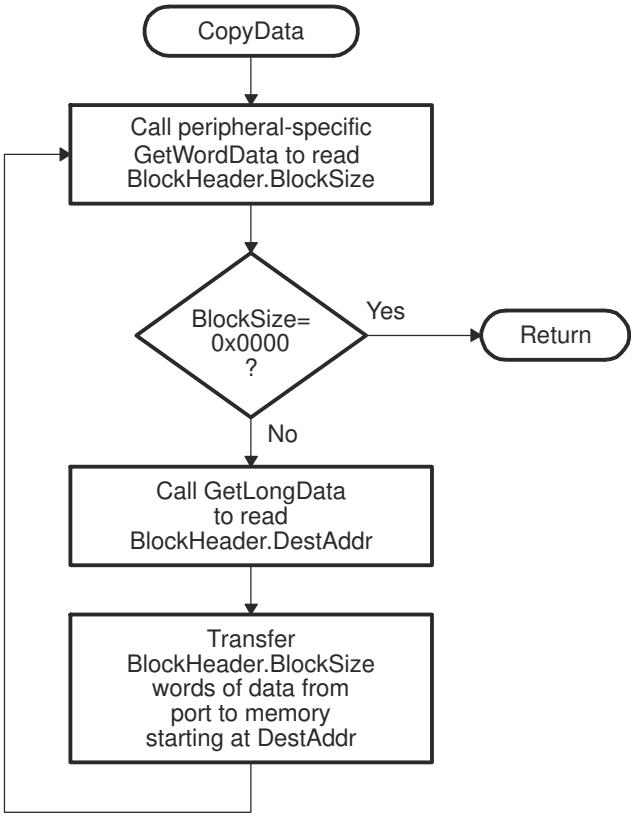


Figure 2-10. Overview of CopyData Function

2.2.16 SCI_Boot Function

The SCI boot mode asynchronously transfers code from SCI-A to internal memory. This boot mode only supports an incoming 8-bit data stream and follows the same data flow as outlined in Example 2-3.

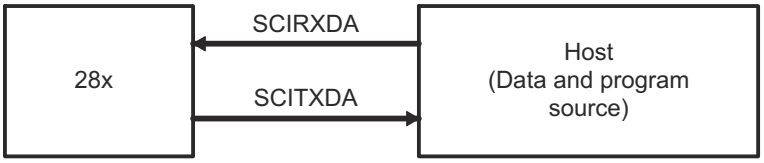


Figure 2-11. Overview of SCI Bootloader Operation

The SCI-A loader uses following pins:

- SCIRXDA on GPIO28
- SCITXDA on GPIO29

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

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

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

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

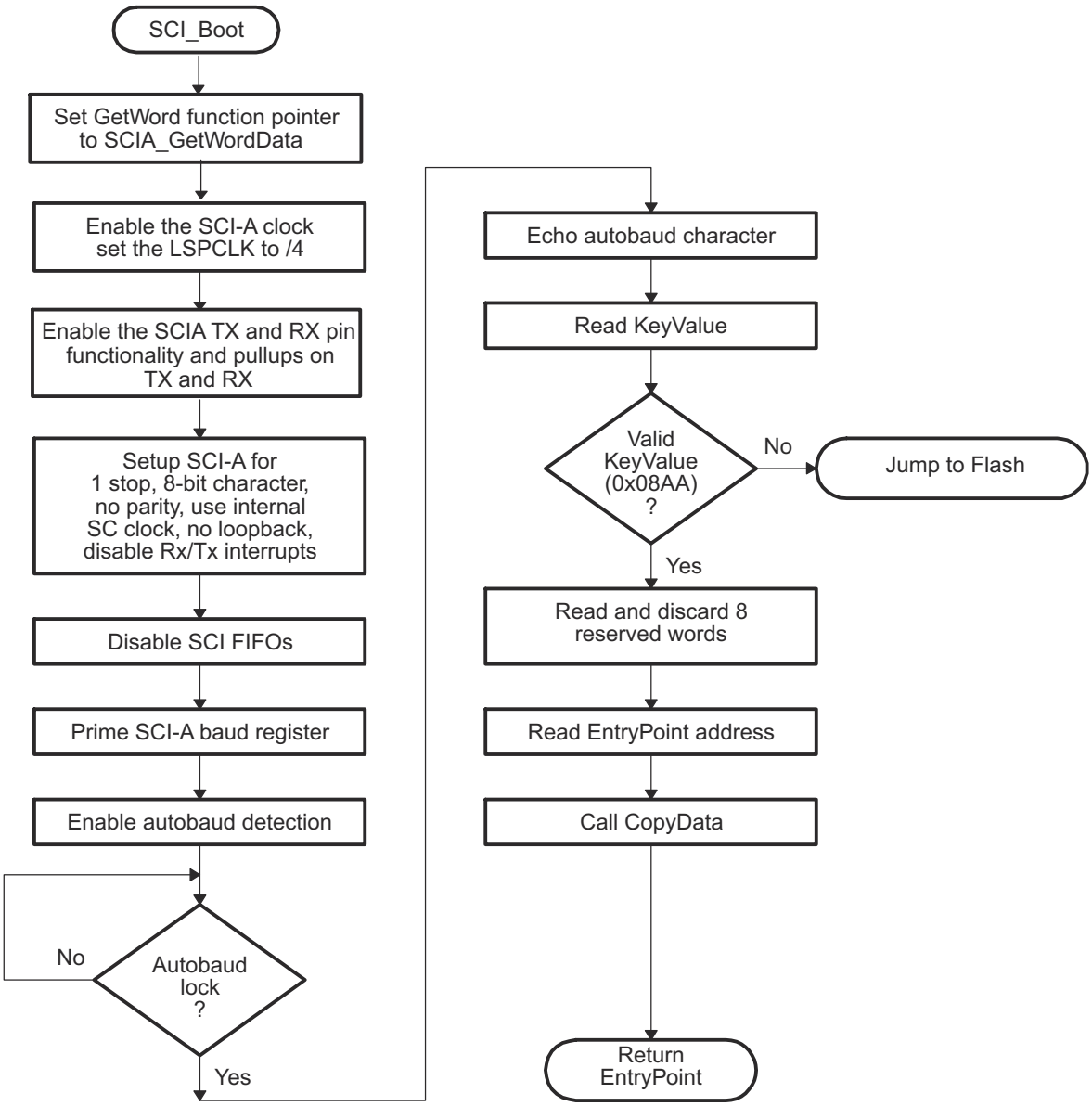


Figure 2-12. Overview of SCI_Boot Function

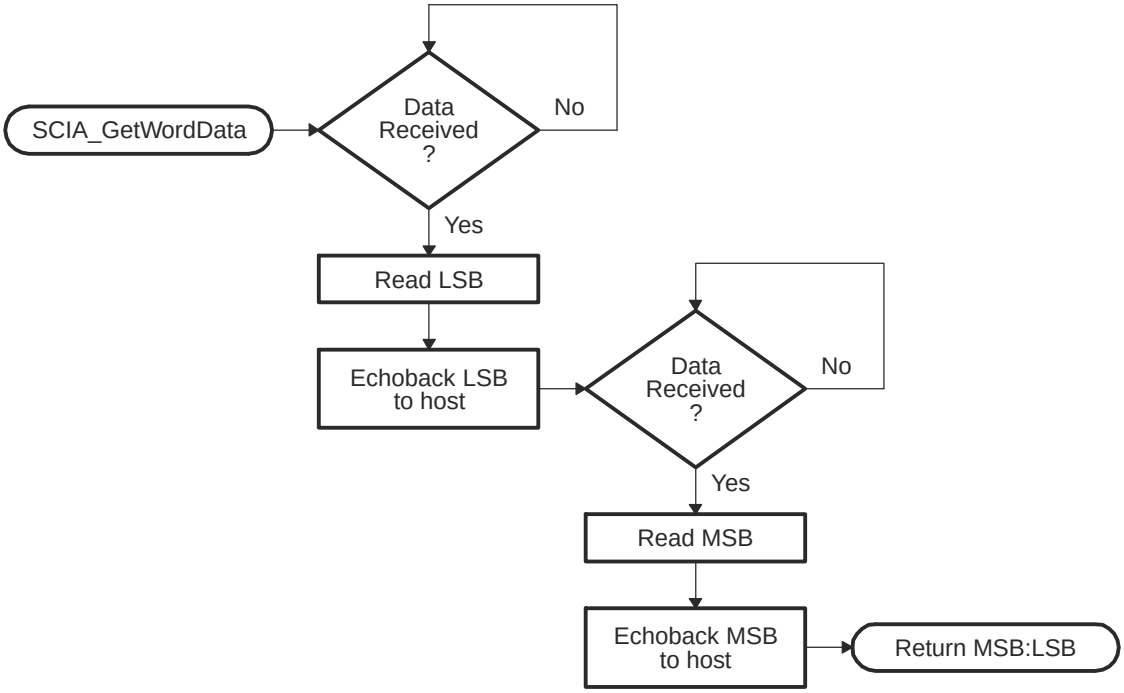


Figure 2-13. Overview of SCI_GetWordData Function

2.2.17 Parallel_Boot Function (GPIO)

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from GPIO0 -GPIO5, GPIO30-GPIO31 to internal memory. Each value is 8 bits long and follows the same data flow as outlined in Section 2.2.11.

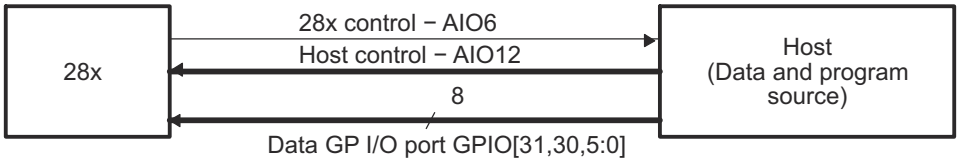


Figure 2-14. Overview of Parallel GPIO Bootloader Operation

The parallel GPIO loader uses following pins:

- Data on GPIO[31,30,5:0]
- 28x Control on AIO6 (external pull-up resistor may be required)
- Host Control on AIO12 (external pull-up resistor required)

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

Two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from GPIO[31,30,5:0].

The 8-bit data stream is shown in Table 2-10.

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

Bytes		GPIO[31,30,5:0] (Byte 1 of 2)	GPIO[31,30,5:0] (Byte 2 of 2)	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 8bits)
3	4	00	00	8 reserved words (words 2 - 9)
...	...	...	...	...
17	18	00	00	Last reserved word
19	20	BB	00	Entry point PC[22:16]
21	22	DD	CC	Entry point PC[15:0] (PC = 0x00BBCCDD)
23	24	NN	MM	Block size of the first block of data to load = 0xMMNN words
25	26	BB	AA	Destination address of first block Addr[31:16]
27	28	DD	CC	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
29	30	BB	AA	First word of the first block in the source being loaded = 0xAABB
... ...				... Data for this section. ...
.		BB	AA	Last word of the first block of the source being loaded = 0xAABB
.		NN	MM	Block size of the 2nd block to load = 0xMMNN words
.		BB	AA	Destination address of second block Addr[31:16]
.		DD	CC	Destination address of second block Addr[15:0]
.		BB	AA	First word of the second block in the source being loaded
.				…
n	n+1	BB	AA	Last word of the last block of the source being loaded (More sections if required)
n+2	n+3	00	00	Block size of 0000h - indicates end of the source program

The 28x device first signals the host that it is ready to begin data transfer by pulling the AIO6 pin low. The host load then initiates the data transfer by pulling the AIO12 pin low. The complete protocol is shown in Figure 2-15.

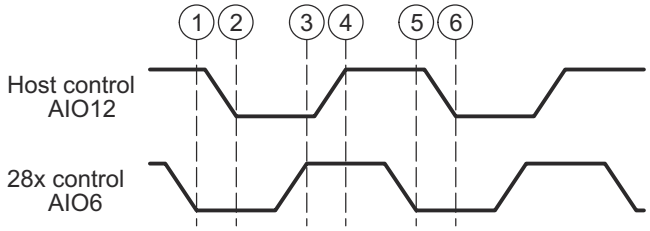


Figure 2-15. Parallel GPIO Boot Loader Handshake Protocol

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

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

Figure 2-16 shows an overview of the Parallel GPIO bootloader flow.

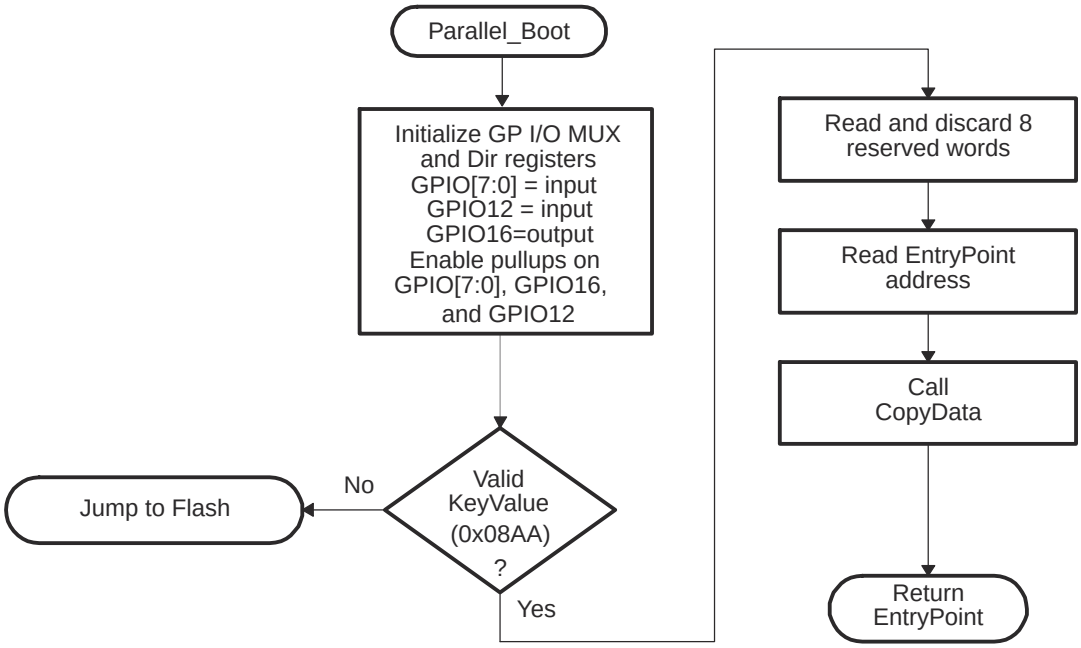


Figure 2-16. Parallel GPIO Mode Overview

Figure 2-17 shows the transfer flow from the host side. The operating speed of the CPU and host are not critical in this mode as the host will wait for the 28x device, and the 28x device will in turn wait for the host. In this manner the protocol will work with both a host running faster and a host running slower than the 28x device.

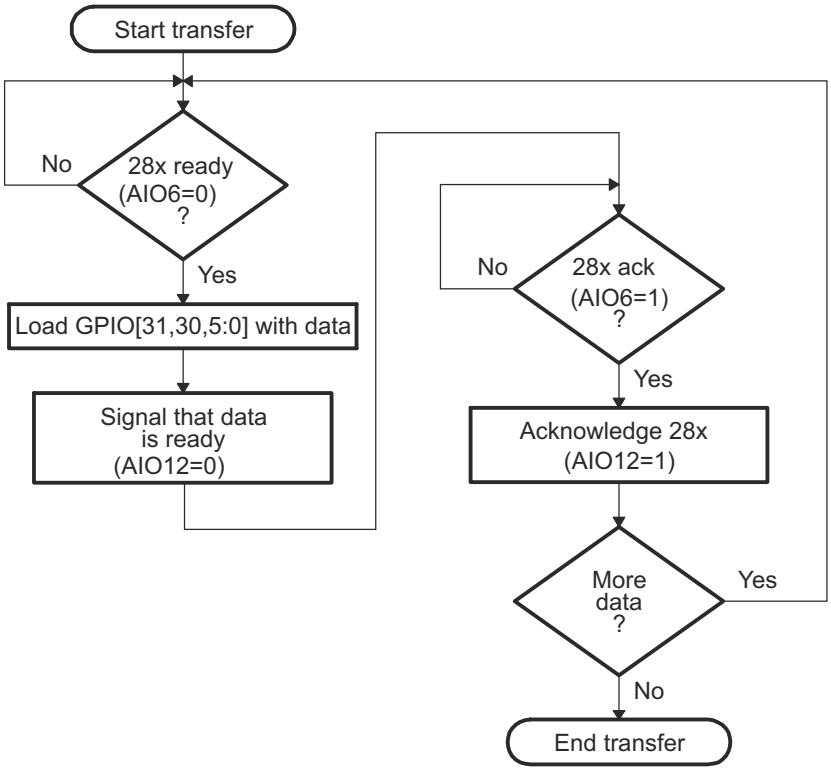


Figure 2-17. Parallel GPIO Mode - Host Transfer Flow

Figure 2-18 show the flow used to read a single word of data from the parallel port.

- **8-bit data stream**

  The 8-bit routine, shown in Figure 2-18, discards the upper 8 bits of the first read from the port and treats the lower 8 bits masked with GPIO31 in bit position 7 and GPIO30 in bit position 6 as the least significant byte (LSB) of the word to be fetched. The routine will then perform a second read to fetch the most significant byte (MSB). It then combines the MSB and LSB into a single 16-bit value to be passed back to the calling routine.

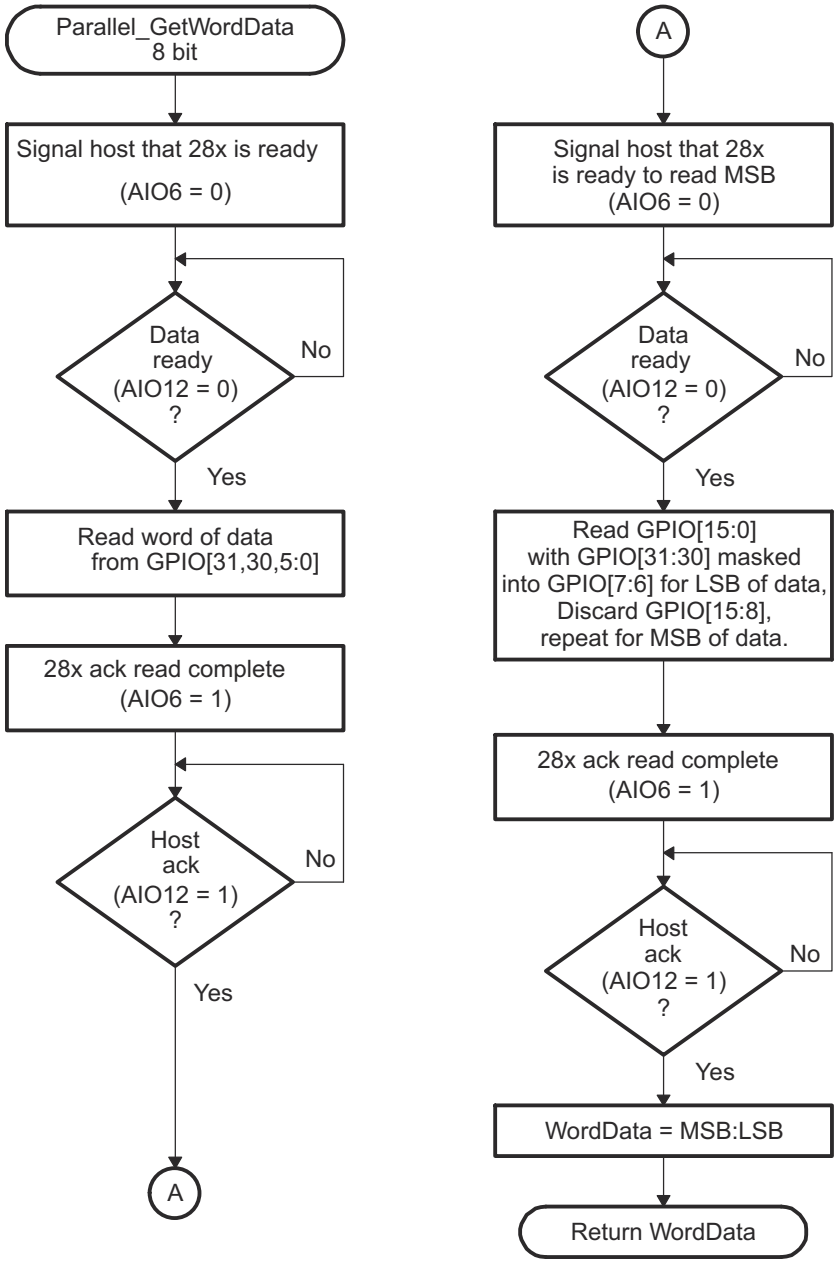


Figure 2-18. 8-Bit Parallel GetWord Function

2.2.18 SPI_Boot Function

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

It expects such a device to be present on the SPI-A pins as indicated in Figure 2-19. The SPI bootloader supports an 8-bit data stream. It does not support a 16-bit data stream.

Figure 2-19. SPI Loader

The SPI-A loader uses following pins:

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

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

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

Table 2-11. SPI 8-Bit Data Stream

Byte	Contents
1	LSB: AA (KeyValue for memory width = 8-bits)
2	MSB: 08h (KeyValue for memory width = 8-bits)
3	LSB: LOSPCP
4	MSB: SPIBRR
5	LSB: reserved for future use
6	MSB: reserved for future use
...	...
...	Data for this section.
	...
17	LSB: reserved for future use
18	MSB: reserved for future use
19	LSB: Upper half (MSW) of Entry point PC[23:16]
20	MSB: Upper half (MSW) of Entry point PC[31:24] (Note: Always 0x00)
21	LSB: Lower half (LSW) of Entry point PC[7:0]
22	MSB: Lower half (LSW) of Entry point PC[15:8]
...	....
...	Data for this section.
	...
...	Blocks of data in the format size/destination address/data as shown in the generic data stream description

Table 2-11. SPI 8-Bit Data Stream (continued)

Byte	Contents
...	...
...	Data for this section. ...
n	LSB: 00h
n+1	MSB: 00h - indicates the end of the source

The data transfer is done in "burst" mode from the serial SPI EEPROM. The transfer is carried out entirely in byte mode (SPI at 8 bits/character). A step-by-step description of the sequence follows:

1. The SPI-A port is initialized
2. The GPIO19 (SPISTE) pin is used as a chip-select for the serial SPI EEPROM or flash
3. The SPI-A outputs a read command for the serial SPI EEPROM or flash
4. The SPI-A sends the serial SPI EEPROM an address 0x0000; that is, the host requires that the EEPROM or flash must have the downloadable packet starting at address 0x0000 in the EEPROM or flash. The loader is compatible with both 16-bit addresses and 24-bit addresses.
5. The next word fetched must match the key value for an 8-bit data stream (0x08AA). The least significant byte of this word is the byte read first and the most significant byte is the next byte fetched. This is true of all word transfers on the SPI. If the key value does not match, then the load is aborted and the device will branch to the flash entry point address.
6. The next two bytes fetched can be used to change the value of the low speed peripheral clock register (LOSPCP) and the SPI baud rate register (SPIBRR). The first byte read is the LOSPCP value and the second byte read is the SPIBRR value. The next 7 words are reserved for future enhancements. The SPI bootloader reads these 7 words and discards them.
7. The next two words makeup the 32-bit entry point address where execution will continue after the boot load process is complete. This is typically the entry point for the program being downloaded through the SPI port.
8. Multiple blocks of code and data are then copied into memory from the external serial SPI EEPROM through the SPI port. The blocks of code are organized in the standard data stream structure presented earlier. This is done until a block size of 0x0000 is encountered. At that point in time the entry point address is returned to the calling routine that then exits the bootloader and resumes execution at the address specified.

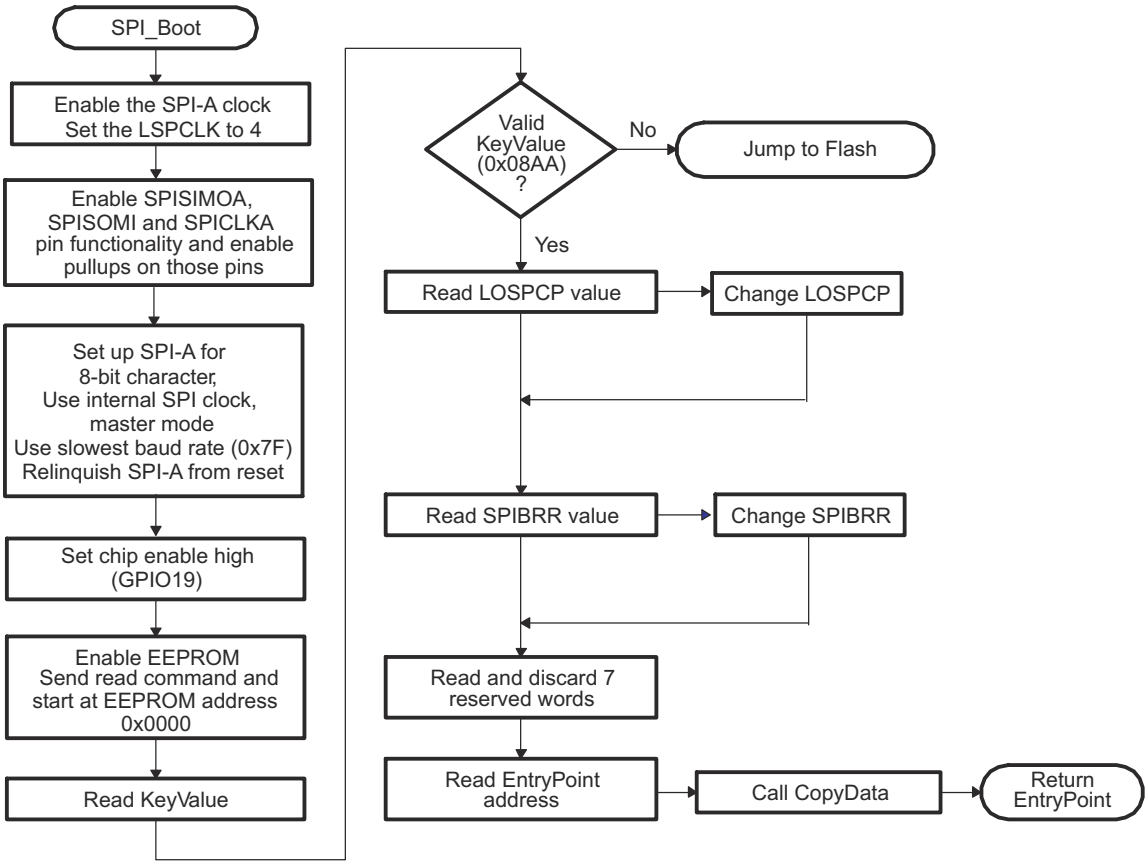


Figure 2-20. Data Transfer From EEPROM Flow

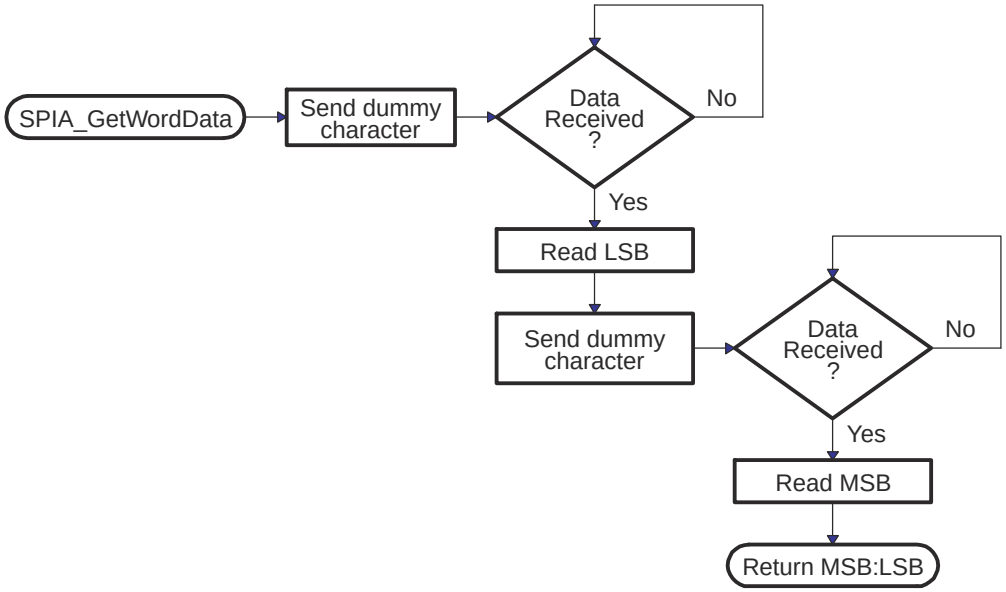


Figure 2-21. Overview of SPIA_GetWordData Function

2.2.19 I2C Boot Function

The I2C bootloader expects an 8-bit wide I2C-compatible EEPROM device to be present at address 0x50 on the I2C-A bus as indicated in Figure 2-22. The EEPROM must adhere to conventional I2C EEPROM protocol, as described in this section, with a 16-bit base address architecture.

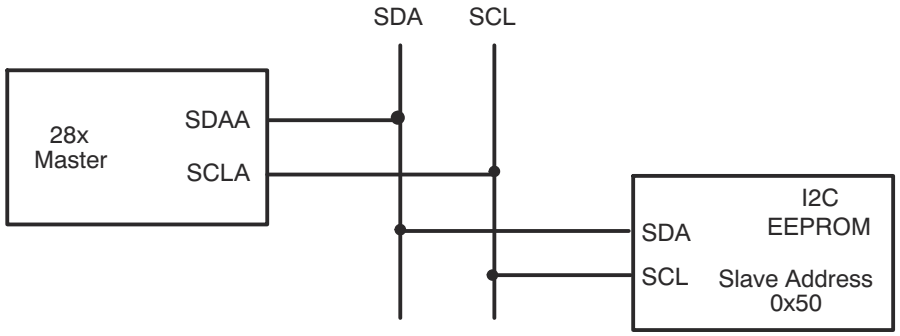


Figure 2-22. EEPROM Device at Address 0x50

The I2C loader uses following pins:

- SDAA on GPIO 28
- SCLA on GPIO 29

If the download is to be performed from a device other than an EEPROM, then that device must be set up to operate in the slave mode and mimic the I2C EEPROM. Immediately after entering the I2C boot function, the GPIO pins are configured for I2C-A operation and the I2C is initialized. The following requirements must be met when booting from the I2C module:

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

The bit-period prescalers (I2CCLKH and I2CCLKL) are configured by the bootloader to run the I2C at a 50 percent duty cycle at 100-kHz bit rate (standard I2C mode) when the system clock is 10 MHz. These registers can be modified after receiving the first few bytes from the EEPROM. This allows the communication to be increased up to a 400-kHz bit rate (fast I2C mode) during the remaining data reads.

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

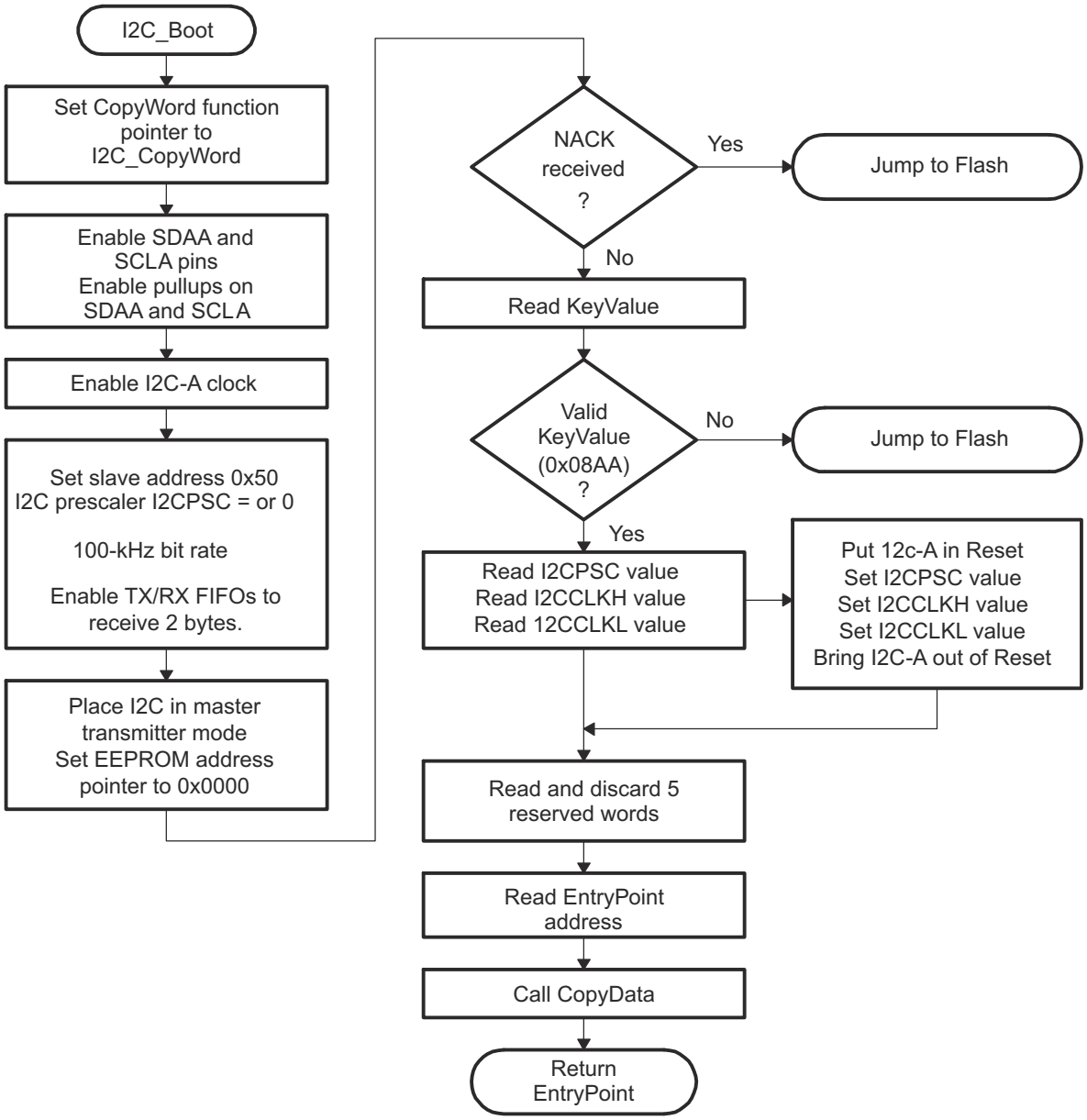


Figure 2-23. Overview of I2C_Boot Function

The nonacknowledgment bit is checked only during the first message sent to initialize the EEPROM base address. This is to make sure that an EEPROM is present at address 0x50 before continuing. If an EEPROM is not present, code will The nonacknowledgment bit is not checked during the address phase of the data read messages (I2C_Get Word). If a non acknowledgment is received during the data read messages, the I2C bus will hang. Table 2-12 shows the 8-bit data stream used by the I2C.

Table 2-12. I2C 8-Bit Data Stream

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

The I2C EEPROM protocol required by the I2C bootloader is shown in Figure 2-24 and Figure 2-25. The first communication, which sets the EEPROM address pointer to 0x0000 and reads the KeyValue (0x08AA) from it, is shown in Figure 2-24. All subsequent reads are shown in Figure 2-25 and are read two bytes at a time.

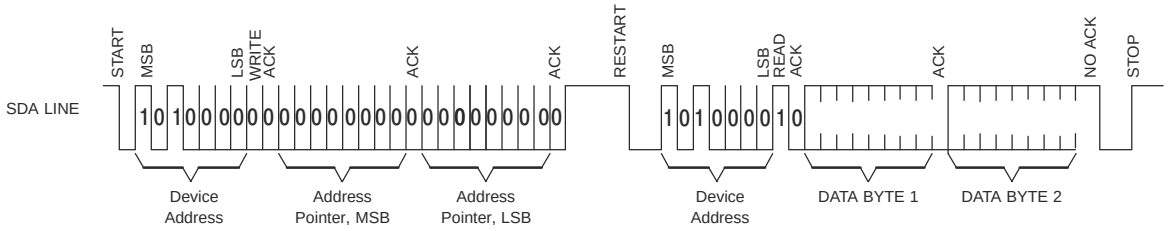


Figure 2-24. Random Read

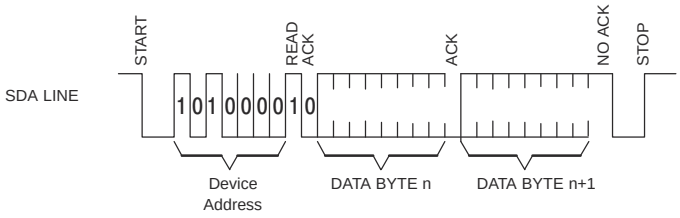


Figure 2-25. Sequential Read

2.2.20 eCAN Boot Function

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

The eCAN-A loader uses following pins:

- CANRXA on GPIO30
- CANTXA on GPIO31

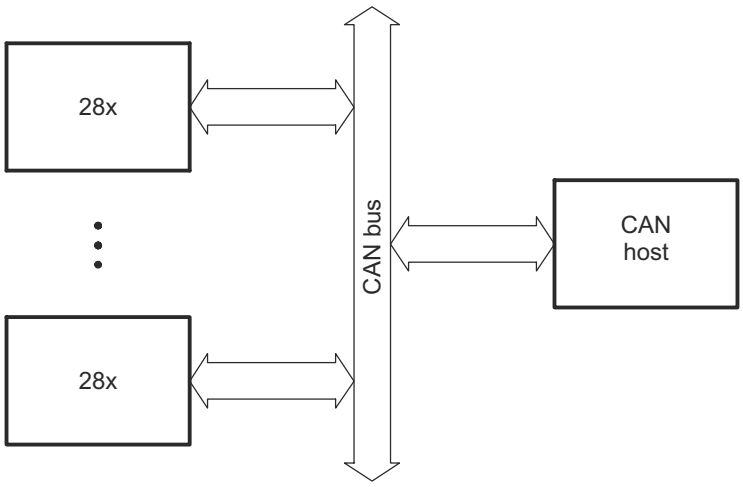


Figure 2-26. Overview of eCAN-A Bootloader Operation

The bit-timing registers are programmed in such a way that a valid bit-rate is achieved for a 10 MHz internal oscillator frequency as shown in Table 2-13.

Table 2-13. Bit-Rate Value for Internal Oscillators

OSCCLK	SYSCLKOUT	Bit Rate
10 MHz	10 MHz	100 kbps

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

Mailbox 1 is programmed with a standard MSGID of 0x1 for boot-loader communication. The CAN host should transmit only 2 bytes at a time, LSB first and MSB next. For example, to transmit the word 0x08AA to the device, transmit AA first, followed by 08. The program flow of the CAN bootloader is identical to the SCI bootloader. The data sequence for the CAN bootloader is shown in Table 2-14.

Table 2-14. 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

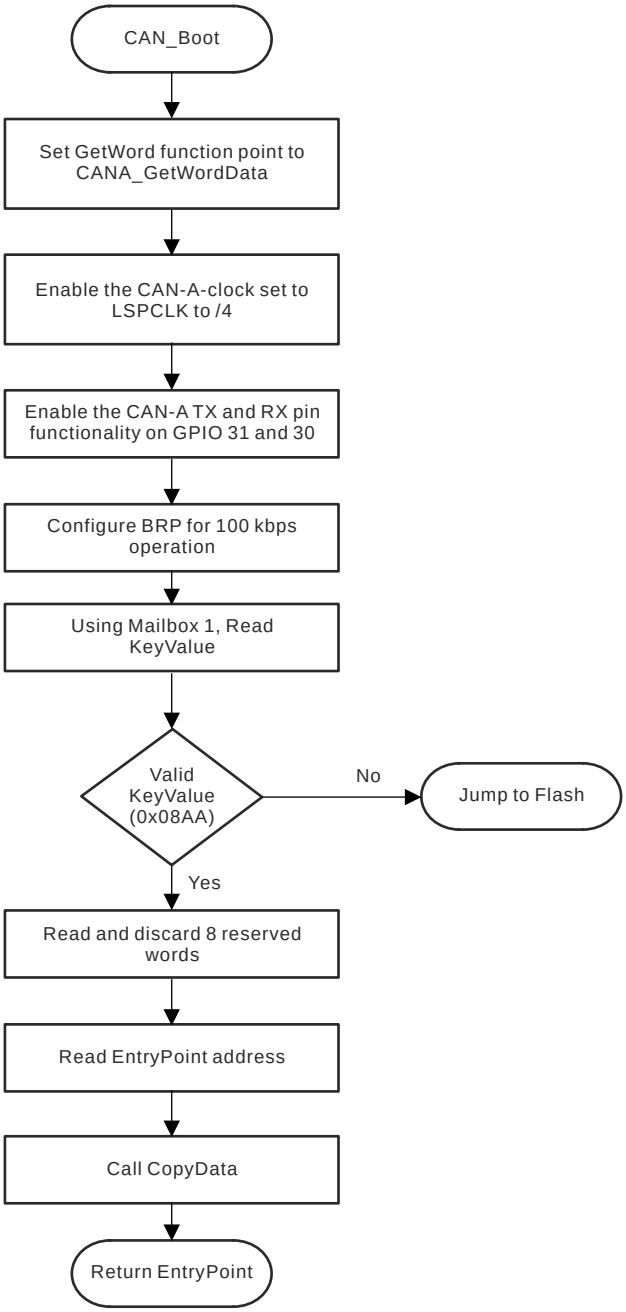


Figure 2-27. CAN_Boot Function

2.2.21 ExitBoot Assembly Routine

The Boot ROM includes an ExitBoot routine that restores the CPU registers to their default state at reset. This is performed on all registers with one exception. The OBJMODE bit in ST1 is left set so that the device remains configured for C28x operation. This flow is detailed in Figure 2-28:

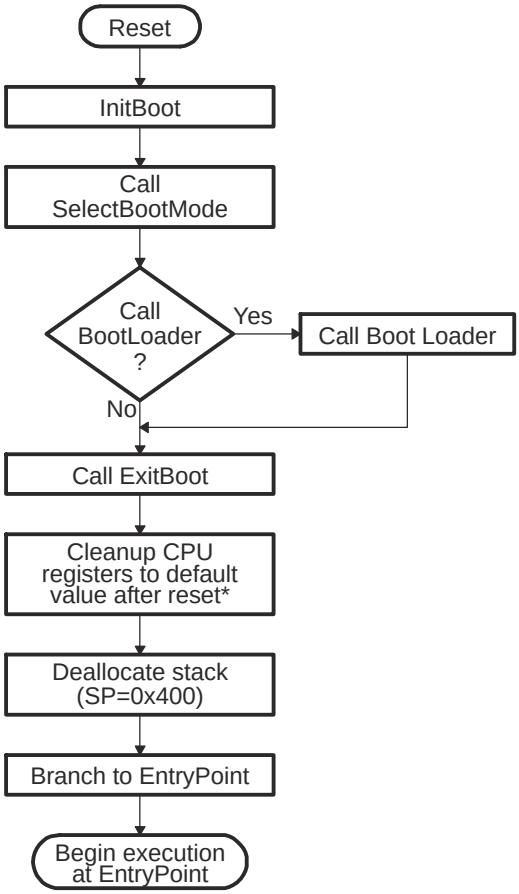


Figure 2-28. ExitBoot Procedure Flow

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 2-15.

Table 2-15. CPU Register Restored Values

Register	Value			Register	Value		
ACC	0x0000 0000			P	0x0000 0000		
XT	0x0000 0000			RPC	0x00 0000		
XAR0-XAR7	0x0000 0000			DP	0x0000		
ST0	0x0000	15:10	OVC = 0	ST1	0x0A0B	15:13	ARP = 0
		9:7	PM = 0			12	XF = 0
		6	V = 0			11	M0M1MAP = 1
		5	N = 0			10	reserved
		4	Z = 0			9	OBJMODE = 1
		3	C = 0			8	AMODE = 0
		2	TC = 0			7	IDLESTAT = 0
		1	OVM = 0			6	EALLOW = 0
		0	SXM = 0			5	LOOP = 0
						4	SPA = 0
						3	VMAP = 1
						2	PAGE0 = 0
						1	DBGM = 1
						0	INTM = 1

2.3 Building the Boot Table

This chapter explains how to generate the data stream and boot table required for the bootloader.

2.3.1 The C2000 Hex Utility

To use the features of the bootloader, you must generate a data stream and boot table as described in Section 2.2.11. The hex conversion utility tool, included with the 28x code generation tools, can generate the required data stream including the required boot table. This section describes the hex2000 utility. An example of a file conversion performed by hex2000 is described in Section 2.3.3.

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, or hex) will differ from one specific application to another and some additional conversion may be required.

To build the boot table, follow these steps:

1. **Assemble or compile the code.**

   This creates the object files that will then be used by the linker to create a single output file.
2. **Link the file.**

   The linker combines all of the object files into a single output file in common object file format (COFF). The specified linker command file is used by the linker to allocate the code sections to different memory blocks. Each block of the boot table data corresponds to an initialized section in the COFF file. Uninitialized sections are not converted by the hex conversion utility. The following options may be useful:

   The linker -m option can be used to generate a map file. This map file will show all of the sections that were created, their location in memory and their length. It can be useful to check this file to make sure that the initialized sections are where you expect them to be.

   The linker -w option is also very useful. This option will tell you if the linker has assigned a section to a memory region on its own. For example, if you have a section in your code called ramfuncs.

3. **Run the hex conversion utility.**

   Choose the appropriate options for the desired boot mode and run the hex conversion utility to convert the COFF file produced by the linker to a boot table.

Table 2-16 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 2-16. Bootloader Options

Option	Description
-boot	Convert all sections into bootable form (use instead of a SECTIONS directive)
-sci8	Specify the source of the bootloader table as the SCI-A port, 8-bit mode
-spi8	Specify the source of the bootloader table as the SPI-A port, 8-bit mode
-gpio8	Specify the source of the bootloader table as the GPIO port, 8-bit mode
-bootorg value	Specify the source address of the bootloader table
-lospcp value	Specify the initial value for the LOSPCP register. This value is used only for the spi8 boot table format and ignored for all other formats. If the value is greater than 0x7F, the value is truncated to 0x7F.
-spibrr value	Specify the initial value for the SPIBRR register. This value is used only for the spi8 boot table format and ignored for all other formats. If the value is greater than 0x7F, the value is truncated to 0x7F.
-e value	Specify the entry point at which to begin execution after boot loading. The value can be an address or a global symbol. This value is optional. The entry point can be defined at compile time using the linker -e option to assign the entry point to a global symbol. The entry point for a C program is normally _c_int00 unless defined otherwise by the -e linker option.
-i2c8	Specify the source of the bootloader table as the I2C-A port, 8-bit
-i2cpsc value	Specify the value for the I2CPSC register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM. This value will be truncated to the least significant eight bits and should be set to maintain an I2C module clock of 7-12 MHz.
-i2cclkh value	Specify the value for the I2CCLKH register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM.
-i2cclkl value	Specify the value for the I2CCLKL register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM.

2.3.2 Example: Preparing a COFF File for eCAN Bootloading

This section shows how to convert a COFF file into a format suitable for CAN based bootloading. This example assumes that the host sending the data stream is capable of reading an ASCII hex format file. An example COFF file named GPIO34TOG.out has been used for the conversion.

Build the project and link using the -m linker option to generate a map file. Examine the .map file produced by the linker. The information shown in Example 2-4 has been copied from the example map file (GPIO34TOG.map). This shows the section allocation map for the code. The map file includes the following information:

- **Output Section**

  This is the name of the output section specified with the SECTIONS directive in the linker command file.
- **Origin**

  The first origin listed for each output section is the starting address of that entire output section. The following origin values are the starting address of that portion of the output section.
- **Length**

  The first length listed for each output section is the length for that entire output section. The following length values are the lengths associated with that portion of the output section.
- **Attributes/input sections**

  This lists the input files that are part of the section or any value associated with an output section.

See the *TMS320C28x Assembly Language Tools User's Guide* for detailed information on generating a linker command file and a memory map.

All sections shown in Example 2-4 that are initialized need to be loaded into the device in order for the code to execute properly. In this case, the codestart, ramfuncs, .cinit, myreset and .text sections need to be loaded. The other sections are uninitialized and will not be included in the loading process. The map file also indicates the size of each section and the starting address. For example, the .text section has 0x155 words and starts at 0x3FA000.

Example 2-4. GPIO34TOG Map File

```
output                                    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 2-5. HEX2000.exe Command Syntax

```
C: HEX2000 GPIO34TOG.OUT -boot -gpio8 -a
Where:
- boot   Convert all sections into bootable form.
- gpio8  Use the GPIO in 8-bit mode data format. The 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 2-5 will generate an ASCII-Hex output file called GPIO34TOG.a00, whose contents are explained in Example 2-6. This example assumes that the host will be able to read an ASCII hex format file. The format may differ for your application. Each section of data loaded can be tied back to the map file described in Example 2-4. After the data stream is loaded, the boot ROM will jump to the Entry point address that was read as part of the data stream. In this case, execution will begin at 0x3FA0000.

Example 2-6. GPIO34TOG Data Stream

```
AA 08                                       ;Keyvalue
00 00 00 00 00 00 00 00                     ;8 reserved words
00 00 00 00 00 00 00 00
3F 00 00 A0                                 ;Entrypoint 0x003FA000
02 00                                       ;Load 2 words - codestart section
00 00 00 00                                 ;Load block starting at 0x000000
7F 00 9A A0                                 ;Data block 0x007F, 0xA09A
16 00                                       ;Load 0x0016 words - ramfuncs section
00 00 02 00                                 ;Load block starting at 0x000002
22 76 1F 76 2A 00 00 1A 01 00 06 CC F0      ;Data = 0x7522, 0x761F etc...
FF 05 50 06 96 06 CC FF F0 A9 1A 00 05
06 96 04 1A FF 00 05 1A FF 00 1A 76 07
F6 00 77 06 00
55 01                                       ;Load 0x0155 words - .text section
3F 00 00 A0                                 ;Load block starting at 0x003FA000
AD 28 00 04 69 FF 1F 56 16 56 1A 56 40      ;Data = 0x28AD, 0x4000 etc...
29 1F 76 00 00 02 29 1B 76 22 76 A9 28
18 00 A8 28 00 00 01 09 1D 61 C0 76 18
00 04 29 0F 6F 00 9B A9 24 01 DF 04 6C
04 29 A8 24 01 DF A6 1E A1 F7 86 24 A7
06 .. ..
.. .. ..
.. .. ..
FC 63 E6 6F
19 00 ;Load 0x0019 words - .cinit section
00 00 18 00                                 ;Load block starting at 0x000018
FF FF 00 B0 3F 00 00 00 FE FF 02 B0 3F      ;Data = 0xFFFF, 0xB000 etc...
00 00 00 00 00 FE FF 04 B0 3F 00 00 00
00 00 FE FF .. .. ..
.. .. ..
3F 00 00 00
02 00                                       ;Load 0x0002 words - myreset section
00 00 32 00                                 ;Load block starting at 0x000032
00 00 00 00                                 ;Data = 0x0000, 0x0000
00 00                                       ;Block size of 0 - end of data
```

2.3.3 Example: Preparing a COFF File for SCI Bootloading

This section shows how to convert a COFF file into a format suitable for SCI- based bootloading. This example assumes that the host sending the data stream is capable of reading an ASCII hex format file. An example COFF file named GPIO34TOG.out has been used for the conversion.

Build the project and link using the -m linker option to generate a map file. Examine the .map file produced by the linker. The information shown in Example 2-7 has been copied from the example map file (GPIO34TOG.map). This shows the section allocation map for the code. The map file includes the following information:

- **Output Section**

  This is the name of the output section specified with the SECTIONS directive in the linker command file.
- **Origin**

  The first origin listed for each output section is the starting address of that entire output section. The following origin values are the starting address of that portion of the output section.
- **Length**

  The first length listed for each output section is the length for that entire output section. The following length values are the lengths associated with that portion of the output section.
- **Attributes/input sections**

  This lists the input files that are part of the section or any value associated with an output section.

See the *TMS320C28x Assembly Language Tools v21.6.0.LTS User's Guide* for detailed information on generating a linker command file and a memory map.

All sections shown in Example 2-7 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 2-7. GPIO34TOG Map File

```
output                                    attributes/
section         page     origin      length       input sections
--------        ----   ----------  ----------  ----------------
codestart
                  0     00000000     00000002
                        00000000     00000002   device280x_CodeStartBranch.obj (codestart)
.pinit            0     00000002     00000000
.switch           0     00000002     00000000   UNINITIALIZED
ramfuncs          0     00000002     00000016
                        00000002     00000016   device280x_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   device280x_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 SCI bootloader, the host must send the data in the format that the bootloader understands. That is, the data must be sent as blocks of data with a size, starting address followed by the data. A block size of 0 indicates the end of the data. The HEX2000.exe utility can be used to convert the COFF file into a format that includes this boot information. The following command syntax has been used to convert the application into an ASCII hex format file that includes all of the required information for the bootloader:

Example 2-8. HEX2000.exe Command Syntax

```
C: HEX2000 GPIO34TOG.OUT -boot -gpio8 -a
Where:
- boot   Convert all sections into bootable form.
- gpio8  Use the GPIO in 8-bit mode data format. The SCI
         uses the same data format as the GPIO in 8-bit mode.
- a      Select ASCII-Hex as the output format.
```

The command line shown in Example 2-8 will generate an ASCII-Hex output file called GPIO34TOG.a00, whose contents are explained in Example 2-9. 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 2-7. After the data stream is loaded, the boot ROM will jump to the Entry point address that was read as part of the data stream. In this case, execution will begin at 0x3FA0000.

Example 2-9. 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 2-17. Bootloader Revision and Checksum Information

Address	Contents
0x3F FFB9	
0x3F FFBA	Boot ROM Version Number
0x3F FFBB	MM/YY of release (in decimal)
0x3F FFBC	Least significant word of checksum
0x3F FFBD	. . .
0x3F FFBE	. . .
0x3F FFBF	Most significant word of checksum

Table 2-18 shows the boot ROM revision details.

Table 2-18. Bootloader Revision Per Device

Device(s)	Silicon REVID (Address 0x883)	Boot ROM Revision
2803x	0 (First silicon)	Version 1
2803x	1	Version 1

2.4.2 Bootloader Code Revision History

The associated boot ROM source code can be found under the \libraries\boot-rom directory in C2000Ware.

This page intentionally left blank.

Chapter 3

Enhanced Pulse Width Modulator (ePWM) Module

The enhanced pulse width modulator (ePWM) module is a key element in controlling many of the power electronic systems found in both commercial and industrial equipments. These systems include digital motor control, switch mode power supply control, uninterruptible power supplies (UPS), and other forms of power conversion. The ePWM module performs a digital-to-analog (DAC) function, where the duty cycle is equivalent to a DAC analog value; it is sometimes referred to as a Power DAC.

This chapter is applicable for ePWM type 1. See the *C2000 Real-Time Control Peripheral Reference Guide* for a list of all devices with an ePWM module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

This chapter includes an overview of the module and information about each of the submodules:

- Time-Base Module
- Counter-Compare Module
- Action-Qualifier Module
- Dead-Band Generator Module
- PWM-Chopper (PC) Module
- Trip-Zone Module
- Event-Trigger Module

ePWM Type 1 is fully compatible to the Type 0 module. Type 1 has the following enhancements in addition to the Type 0 module:

- **Increased Dead-Band Resolution:** The dead-band clocking has been enhanced to allow half-cycle clocking to double resolution.
- **Enhanced Interrupt and SOC Generation:** Interrupts and ADC start-of-conversion can now be generated on both the TBCTR == zero and TBCTR == period events. This feature enables dual edge PWM control. Additionally, the ADC start-of-conversion can be generated from an event defined in the digital compare sub-module.
- **High-Resolution Period Capability:** Provides the ability to enable high-resolution period. This is discussed in more detail in Chapter 4.
- **Digital Compare Sub-module:** The digital compare sub-module enhances the event triggering and trip zone sub-modules by providing filtering, blanking and improved trip functionality to digital compare signals. Such features are essential for peak current mode control and for support of analog comparators.

3.1 Introduction

An effective PWM peripheral must be able to generate complex pulse width waveforms with minimal CPU overhead or intervention. It needs to be highly programmable and very flexible while being easy to understand and use. The ePWM unit described here addresses these requirements by allocating all needed timing and control resources on a per PWM channel basis. Cross coupling or sharing of resources has been avoided; instead, the ePWM is built up from smaller single channel modules with separate resources that can operate together as required to form a system. This modular approach results in an orthogonal architecture and provides a more transparent view of the peripheral structure, helping users to understand its operation quickly.

In this document the letter x within a signal or module name is used to indicate a generic ePWM instance on a device. For example output signals EPWMxA and EPWMxB refer to the output signals from the ePWMx instance. Thus, EPWM1A and EPWM1B belong to ePWM1 and likewise EPWM4A and EPWM4B belong to ePWM4.

3.1.1 EPWM Related Collateral

Foundational Materials

- *Real-Time Control Reference Guide*
  - Refer to the EPWM section

Getting Started Materials

- *Flexible PWMs Enable Multi-Axis Drives, Multi-Level Inverters Application Report*
- Getting Started with the C2000 ePWM Module (Video)
- *Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Control Application Report*
  - Chapters 1 to 6 are Fundamental material, derivations, and explanations that are useful for learning about how PWM can be used to implement a DAC. Subsequent chapters are Getting Started and Expert material for implementing in a system.
- *Using the Enhanced Pulse Width Modulator (ePWM) Module Application Report*

Expert Materials

- C2000 real-time microcontrollers - Reference designs
  - See TI designs related to specific end applications used.

3.1.2 Submodule Overview

The ePWM module represents one complete PWM channel composed of two PWM outputs: EPWMxA and EPWMxB. Multiple ePWM modules are instanced within a device as shown in Figure 3-1. Each ePWM instance is identical with one exception. Some instances include a hardware extension that allows more precise control of the PWM outputs. This extension is the high-resolution pulse width modulator (HRPWM) and is described in Chapter 4. See your device-specific data sheet to determine which ePWM instances include this feature. Each ePWM module is indicated by a numerical value starting with 1. For example, ePWM1 is the first instance and ePWM3 is the third instance in the system and ePWMx indicates any instance.

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

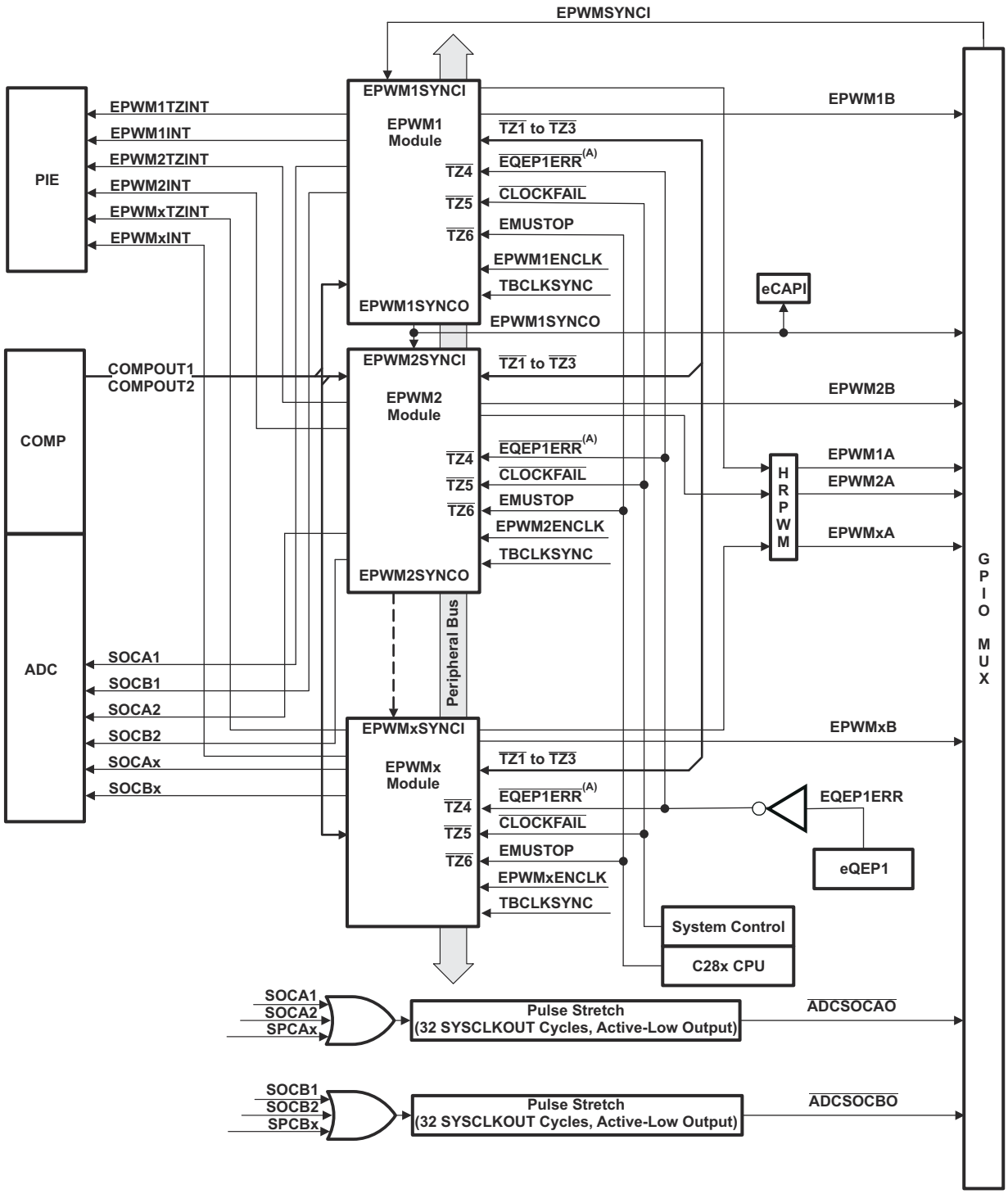
Each ePWM module supports the following features:

- Dedicated 16-bit time-base counter with period and frequency control.
- Two PWM outputs (EPWMxA and EPWMxB) that can be used in the following configurations:
  - Two independent PWM outputs with single-edge operation
  - Two independent PWM outputs with dual-edge symmetric operation
  - One independent PWM output with dual-edge asymmetric operation

- Asynchronous override control of PWM signals through software.
- Programmable phase-control support for lag or lead operation relative to other ePWM modules.
- Hardware-locked (synchronized) phase relationship on a cycle-by-cycle basis.
- Dead-band generation with independent rising and falling edge delay control.
- Programmable trip zone allocation of both cycle-by-cycle trip and one-shot trip on fault conditions.
- A trip condition can force either high, low, or high-impedance state logic levels at PWM outputs.
- Comparator module outputs and trip zone inputs can generate events, filtered events, or trip conditions.
- All events can trigger both CPU interrupts and ADC start of conversion (SOC).
- Programmable event prescaling minimizes CPU overhead on interrupts.
- PWM chopping by high-frequency carrier signal, useful for pulse transformer gate drives.

Each ePWM module is connected to the input/output signals shown in Figure 3-1. The signals are described in detail in subsequent sections.

The order in which the ePWM modules are connected may differ from what is shown in Figure 3-1. See Section 3.2.2.3.3 for the synchronization scheme for a particular device. Each ePWM module consists of eight submodules and is connected within a system with the signals shown in Figure 3-2.



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

Figure 3-1. Multiple ePWM Modules

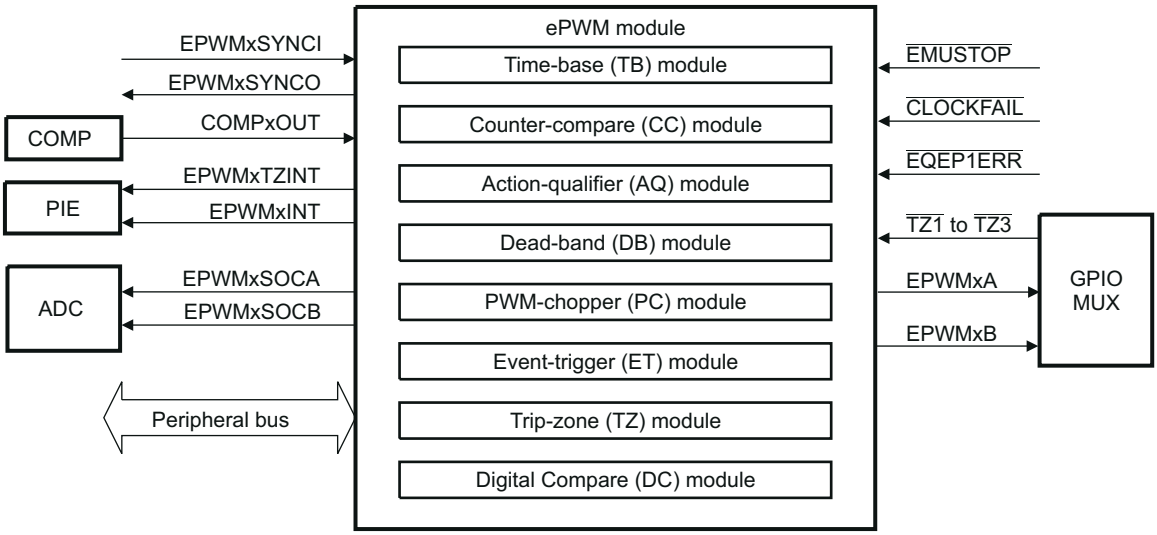
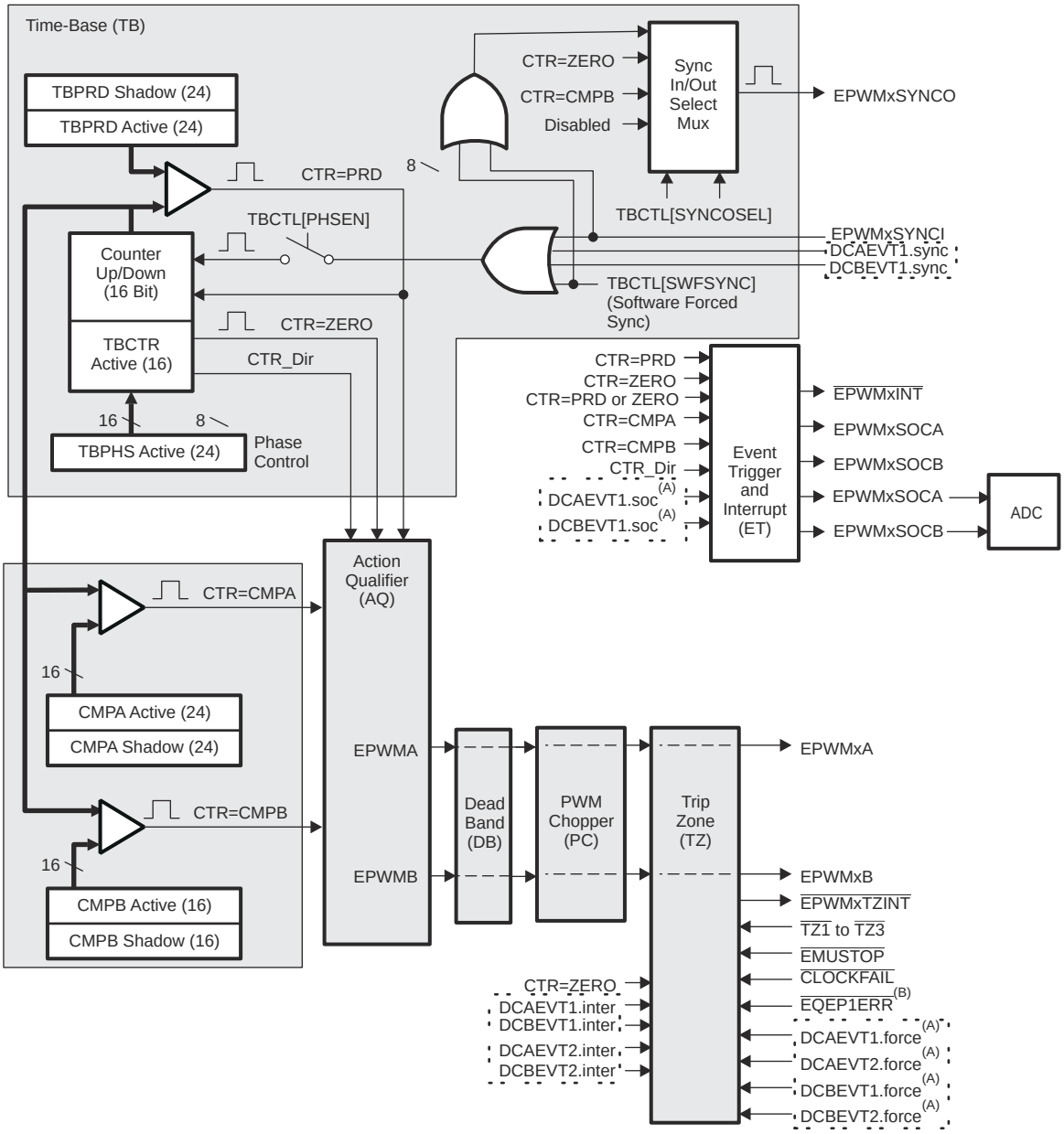


Figure 3-2. Submodules and Signal Connections for an ePWM Module

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

- **PWM output signals (EPWMxA and EPWMxB):** The PWM output signals are made available external to the device through the GPIO peripheral described in the system control and interrupts guide for your device.
- **Trip-zone signals ($\overline{TZ1}$ 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 $\overline{TZ3}$ trip-zone signals can be configured as asynchronous inputs through the GPIO peripheral. $\overline{TZ4}$ is connected to an inverted EQEP1 error signal (EQEP1ERR) from the EQEP1 module (for those devices with an EQEP1 module). $\overline{TZ5}$ is connected to the system clock fail logic, and $\overline{TZ6}$ is connected to the $\overline{EMUSTOP}$ output from the CPU. This allows you to configure a trip action when the clock fails or the CPU halts.
- **Time-base synchronization input (EPWMxSYNCI) and output (EPWMxSYNCO) signals:** The synchronization signals daisy chain the ePWM modules together. Each module can be configured to either use or ignore its synchronization input. The clock synchronization input and output signal are brought out to pins only for ePWM1 (ePWM module #1). The synchronization output for ePWM1 (EPWM1SYNCO) is also connected to the SYNCI of the first enhanced capture module (eCAP1).
- **ADC start-of-conversion signals (EPWMxSOCA and EPWMxSOCB):** Each ePWM module has two ADC start of conversion signals . Any ePWM module can trigger a start of conversion. Whichever event triggers the start of conversion is configured in the Event-Trigger submodule of the ePWM.
- **Comparator output signals (COMPxOUT).:** Output signals from the comparator module in conjunction with the trip zone signals can generate digital compare events.
- **Peripheral Bus:** The peripheral bus is 32-bits wide and allows both 16-bit and 32-bit writes to the ePWM register file.

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



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

B. This signal exists only on devices with in eQEP1 module.

Figure 3-3. ePWM Submodules and Critical Internal Signal Interconnects

3.1.3 Register Mapping

The complete ePWM module control and status register set is grouped by submodule as shown in Table 3-1. Each register set is duplicated for each instance of the ePWM module. The start address for each ePWM register file instance on a device is specified in the appropriate data manual.

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

Name	Offset [1]	Size (x16)	Shadow	EALLOW	Description
Time-Base Submodule Registers					
TBCTL	0x0000	1			Time-Base Control Register
TBSTS	0x0001	1			Time-Base Status Register
TBPHSHR	0x0002	1			Extension for HRPWM Phase Register [2]
TBPHS	0x0003	1			Time-Base Phase Register
TBCTR	0x0004	1			Time-Base Counter Register
TBPRD	0x0005	1	Yes		Time-Base Period Register
TBPRDHR	0x0006	1	Yes		Time Base Period High Resolution Register [3]
Counter-Compare Submodule Registers					
CMPCTL	0x0007	1			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			Action-Qualifier Control Register for Output A (EPWMxA)
AQCTLB	0x000C	1			Action-Qualifier Control Register for Output B (EPWMxB)
AQSFRC	0x000D	1			Action-Qualifier Software Force Register
AQCSFRC	0x000E	1	Yes		Action-Qualifier Continuous S/W Force Register Set
Dead-Band Generator Submodule Registers					
DBCTL	0x000F	1			Dead-Band Generator Control Register
DBRED	0x0010	1			Dead-Band Generator Rising Edge Delay Count Register
DBFED	0x0011	1			Dead-Band Generator Falling Edge Delay Count Register
Trip-Zone Submodule Registers					
TZSEL	0x0012	1		Yes	Trip-Zone Select Register
TZDCSEL	0x0013	1		Yes	Trip Zone Digital Compare Select Register
TZCTL	0x0014	1		Yes	Trip-Zone Control Register [3]
TZEINT	0x0015	1		Yes	Trip-Zone Enable Interrupt Register [3]
TZFLG	0x0016	1			Trip-Zone Flag Register [3]
TZCLR	0x0017	1		Yes	Trip-Zone Clear Register [3]
TZFRC	0x0018	1		Yes	Trip-Zone Force Register [3]
Event-Trigger Submodule Registers					
ETSEL	0x0019	1			Event-Trigger Selection Register
ETPS	0x001A	1			Event-Trigger Pre-Scale Register
ETFLG	0x001B	1			Event-Trigger Flag Register
ETCLR	0x001C	1			Event-Trigger Clear Register
ETFRC	0x001D	1			Event-Trigger Force Register
PWM-Chopper Submodule Registers					
PCCTL	0x001E	1			PWM-Chopper Control Register
High-Resolution Pulse Width Modulator (HRPWM) Extension Registers					
HRCNFG	0x0020	1		Yes	HRPWM Configuration Register [2] [3]
HRPWR	0x0021	1		Yes	HRPWM Power Register [3] [4]

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

Name	Offset [1]	Size (x16)	Shadow	EALLOW	Description
HRMSTEP	0x0026	1		Yes	HRPWM MEP Step Register[3] [4]
HRPCTL	0x0028	1		Yes	High Resolution Period Control Register[3]
TBPRDHRM	0x002A	1	Writes		Time Base Period High Resolution Register Mirror[3]
TBPRDM	0x002B	1	Writes		Time Base Period Register Mirror
CMPAHRM	0x002C	1	Writes		Compare A High Resolution Register Mirror[3]
CMPAM	0x002D	1	Writes		Compare A Register Mirror
Digital Compare Event Registers					
DCTRIPSEL	0x0030	1		Yes	Digital Compare Trip Select Register
DCACTL	0x0031	1		Yes	Digital Compare A Control Register
DCBCTL	0x0032	1		Yes	Digital Compare B Control Register
DCFCTL	0x0033	1		Yes	Digital Compare Filter Control Register
DCCAPCTL	0x0034	1		Yes	Digital Compare Capture Control Register
DCFOFFSET	0x0035	1	Writes		Digital Compare Filter Offset Register
DCFOFFSETCNT	0x0036	1			Digital Compare Filter Offset Counter Register
DCFWINDOW	0x0037	1			Digital Compare Filter Window Register
DCFWINDOWCNT	0x0038	1			Digital Compare Filter Window Counter Register
DCCAP	0x0039	1	Yes		Digital Compare Counter Capture Register

(1) Locations not shown are reserved.
(2) These registers are only available on ePWM instances that include the high-resolution PWM extension. Otherwise these locations are reserved. These registers are also described in Chapter 4. See your device-specific data sheet to determine which instances include the HRPWM.
(3) EALLOW protected registers as described in the *System Control and Interrupts* chapter.
(4) These registers only exist in the ePWM1 register space. They cannot be accessed from any other ePWM module's register space.

The CMPA , CMPAHR, TBPRD, and TBPRDHR registers are mirrored in the register map (Mirror registers include an "-M" suffix - CMPAM, CMPAHRM, TBPRDM, and TBPRDHRM). Note in the tables below, that in both Immediate mode and Shadow mode, reads from these mirror registers result in the active value of the register or a TI internal test value.

In Immediate Mode:

Register	Offset	Write	Read	Register	Offset	Write	Read
TBPRDHR	0x06	Active	Active	TBPRDHRM	0x2A	Active	TI_Internal
TBPRD	0x05	Active	Active	TBPRDM	0x2B	Active	Active
CMPAHR	0x08	Active	Active	CMPAHRM	0x2C	Active	TI_Internal
CMPA	0x09	Active	Active	CMPAM	0x2D	Active	Active

In Shadow Mode:

Register	Offset	Write	Read	Register	Offset	Write	Read
TBPRDHR	0x06	Shadow	Shadow	TBPRDHRM	0x2A	Shadow	TI_Internal
TBPRD	0x05	Shadow	Shadow	TBPRDM	0x2B	Shadow	Active
CMPAHR	0x08	Shadow	Shadow	CMPAHRM	0x2C	Shadow	TI_Internal
CMPA	0x09	Shadow	Shadow	CMPAM	0x2D	Shadow	Active

3.2 ePWM Submodules

Eight submodules are included in every ePWM peripheral. Each of these submodules performs specific tasks that can be configured by software.

3.2.1 Overview

Table 3-2 lists the eight key submodules together with a list of their main configuration parameters. For example, if you need to adjust or control the duty cycle of a PWM waveform, then you should see the counter-compare submodule in Section 3.2.3 for relevant details.

Table 3-2. Submodule Configuration Parameters

Submodule	Configuration Parameter or Option
Time-base (TB)	• 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 a debug probe. • Specify the source for the synchronization output of the ePWM module: – Synchronization input signal – Time-base counter equal to zero – Time-base counter equal to counter-compare B (CMPB) – No output synchronization signal generated.
Counter-compare (CC)	• Specify the PWM duty cycle for output EPWMxA and/or output EPWMxB • Specify the time at which switching events occur on the EPWMxA or EPWMxB output
Action-qualifier (AQ)	• Specify the type of action taken when a time-base or counter-compare submodule event occurs: – No action taken – Output EPWMxA and/or EPWMxB switched high – Output EPWMxA and/or EPWMxB switched low – Output EPWMxA and/or EPWMxB toggled • Force the PWM output state through software control • Configure and control the PWM dead-band through software
Dead-band (DB)	• Control of traditional complementary dead-band relationship between upper and lower switches • Specify the output rising-edge-delay value • Specify the output falling-edge delay value • Bypass the dead-band module entirely. In this case the PWM waveform is passed through without modification. • Option to enable half-cycle clocking for double resolution.
PWM-chopper (PC)	• Create a chopping (carrier) frequency. • Pulse width of the first pulse in the chopped pulse train. • Duty cycle of the second and subsequent pulses. • Bypass the PWM-chopper module entirely. In this case the PWM waveform is passed through without modification.

Table 3-2. Submodule Configuration Parameters (continued)

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

Code examples are provided in the remainder of this document that show how to implement various ePWM module configurations. These examples use the constant definitions in the device *EPwm_defines.h* file in the device-specific header file and peripheral examples software package.

3.2.2 Time-Base (TB) Submodule

Each ePWM module has its own time-base submodule that determines all of the event timing for the ePWM module. Built-in synchronization logic allows the time-base of multiple ePWM modules to work together as a single system. Figure 3-4 illustrates the time-base module's place within the ePWM.

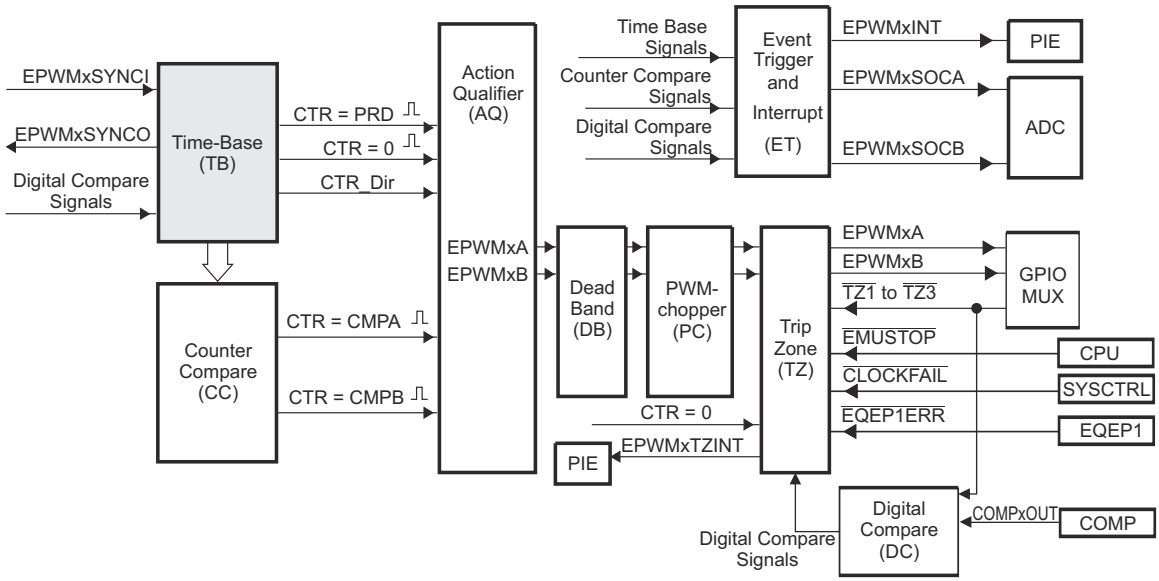


Figure 3-4. Time-Base Submodule Block Diagram

3.2.2.1 Purpose of the Time-Base Submodule

You can configure the time-base submodule for the following:

- Specify the ePWM time-base counter (TBCTR) frequency or period to control how often events occur.
- Manage time-base synchronization with other ePWM modules.
- Maintain a phase relationship with other ePWM modules.
- Set the time-base counter to count-up, count-down, or count-up-and-down mode.
- Generate the following events:
  - CTR = PRD: Time-base counter equal to the specified period (TBCTR = TBPRD) .
  - CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000).
- Configure the rate of the time-base clock; a prescaled version of the CPU system clock (SYSCLKOUT). This allows the time-base counter to increment/decrement at a slower rate.

3.2.2.2 Controlling and Monitoring the Time-base Submodule

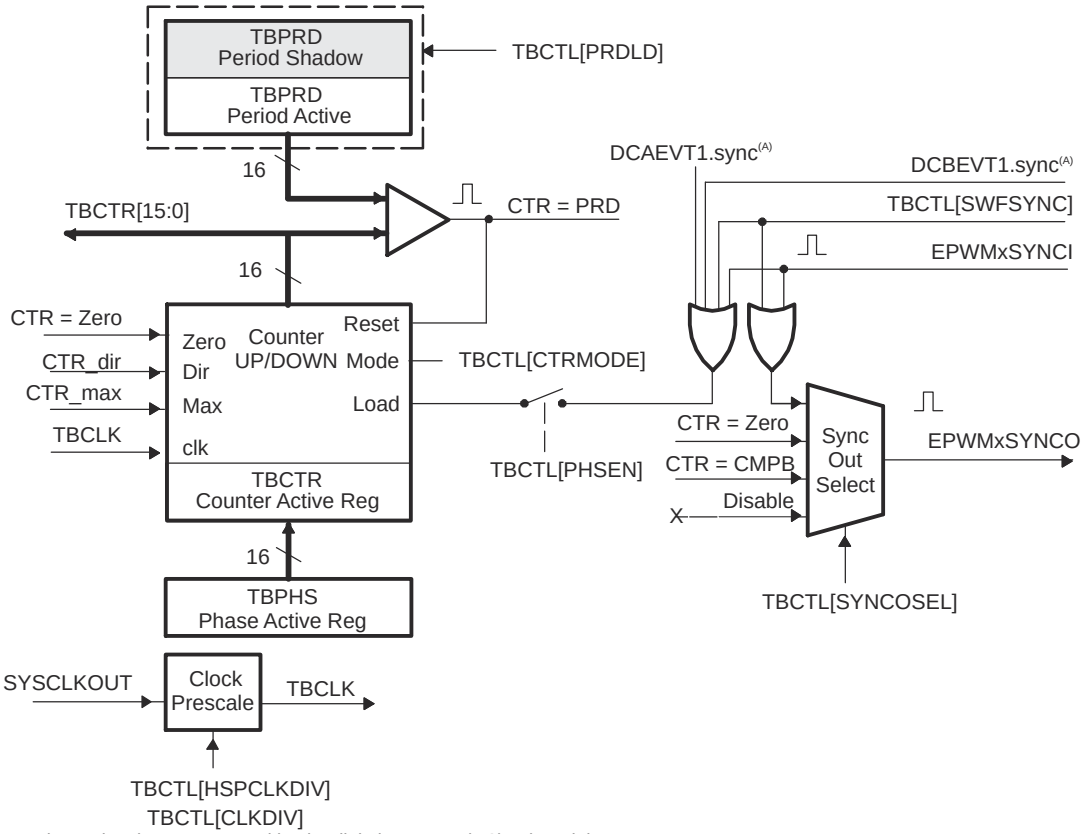
Table 3-3 shows the registers used to control and monitor the time-base submodule.

Table 3-3. Time-Base Submodule Registers

Register	Address Offset	Shadowed	Description	Bit Description
TBCTL	0x0000	No	Time-Base Control Register	Section 3.4.1.1
TBSTS	0x0001	No	Time-Base Status Register	Section 3.4.1.2
TBPHSHR	0x0002	No	HRPWM Extension Phase Register (1)	Section 3.4.1.3
TBPHS	0x0003	No	Time-Base Phase Register	Section 3.4.1.4
TBCTR	0x0004	No	Time-Base Counter Register	Section 3.4.1.5
TBPRD	0x0005	Yes	Time-Base Period Register	Section 3.4.1.6
TBPRDHR	0x0006	Yes	HRPWM Extension Period Register(1)	Section 3.4.1.7
TBPRDHRM	0x002A	Yes	HRPWM Time-Base Period Extension Mirror Register(1)	Section 3.4.1.8
TBPRDM	0x002B	Yes	HRPWM Extension Period Mirror Register(1)	Section 3.4.1.9

(1) This register is available only on ePWM instances that include the high-resolution extension (HRPWM). On ePWM modules that do not include the HRPWM, this location is reserved. This register is also described in Chapter 4. See your device-specific data sheet to determine which ePWM instances include this feature.

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



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

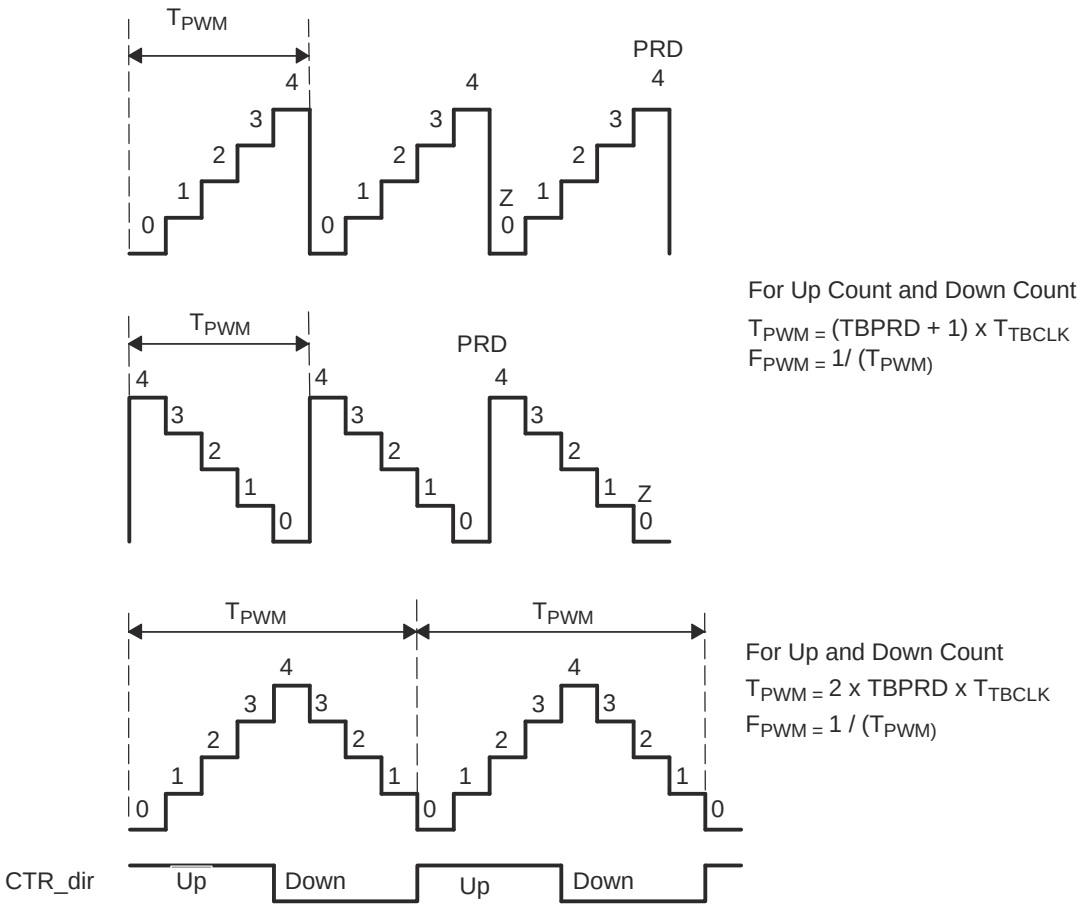


Figure 3-6. Time-Base Frequency and Period

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

Scheme 1 is shown in Figure 3-7.

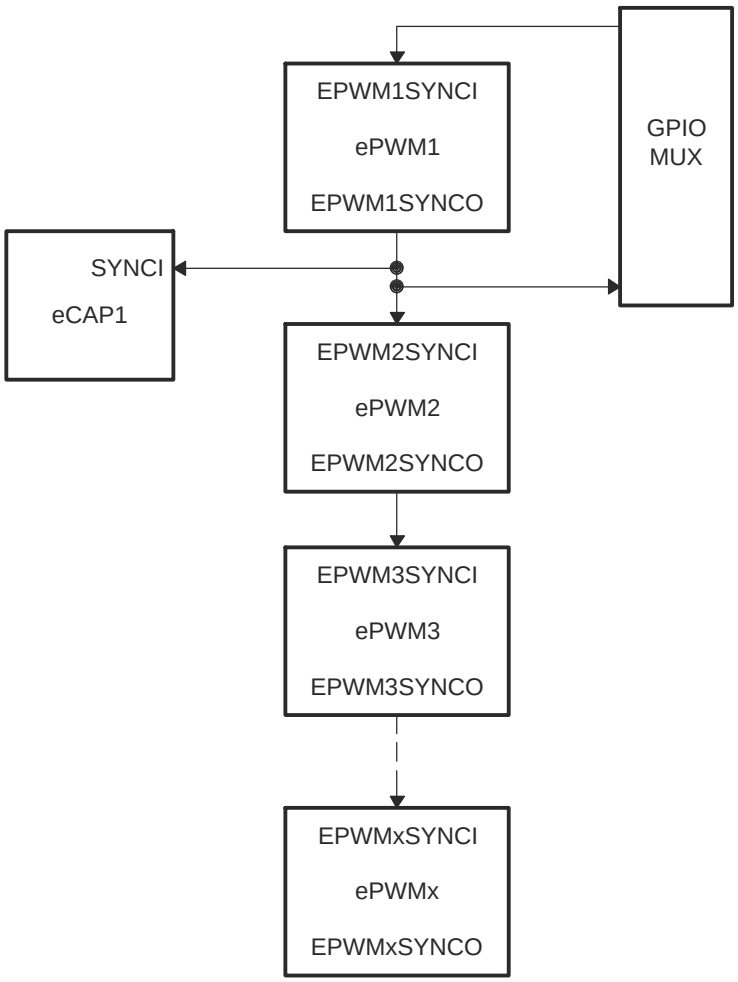


Figure 3-7. Time-Base Counter Synchronization Scheme 1

Each ePWM module can be configured to use or ignore the synchronization input. If the TBCTL[PHSEN] bit is set, then the time-base counter (TBCTR) of the ePWM module will be automatically loaded with the phase register (TBPHS) contents when one of the following conditions occur:

- **EPWMxSYNCI: Synchronization Input Pulse:** The value of the phase register is loaded into the counter register when an input synchronization pulse is detected (TBPHS → TBCTR). This operation occurs on the next valid time-base clock (TBCLK) edge. The source of the EPWMSYNCI pulse is selected by setting a GPIO to the EPWMSYNCI option per the GPIO Mux.

  The delay from internal master module to slave modules is given by:
  – if (TBCLK = SYSCLKOUT): 2 x SYSCLKOUT
  – if (TBCLK != SYSCLKOUT):1 TBCLK
- **Software Forced Synchronization Pulse:** Writing a 1 to the TBCTL[SWFSYNC] control bit invokes a software forced synchronization. This pulse is ORed with the synchronization input signal, and therefore has the same effect as a pulse on EPWMxSYNCI.
- **Digital Compare Event Synchronization Pulse:** DCAEVT1 and DCBEVT1 digital compare events can be configured to generate synchronization pulses which have the same affect as EPWMxSYNCI.

This feature enables the ePWM module to be automatically synchronized to the time base of another ePWM module. Lead or lag phase control can be added to the waveforms generated by different ePWM modules to synchronize them. In up-down-count mode, the TBCTL[PSHDIR] bit configures the direction of the time-base counter immediately after a synchronization event. The new direction is independent of the direction prior to the synchronization event. The 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 Section 3.3 for more details on synchronization strategies.

3.2.2.4 Phase Locking the Time-Base Clocks of Multiple ePWM Modules

The TBCLKSYNC bit can be used to globally synchronize the time-base clocks of all enabled ePWM modules on a device. This bit is part of the device's clock enable registers and is described in the *System Control and Interrupts* section of this manual. When TBCLKSYNC = 0, the time-base clock of all ePWM modules is stopped (default). When TBCLKSYNC = 1, all ePWM time-base clocks are started with the rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescaler bits in the TBCTL register of each ePWM module must be set identically. The proper procedure for enabling the ePWM clocks is as follows:

1. Enable the individual ePWM module clocks. This is described in the device-specific version of the *System Control and Interrupts* section.
2. Set TBCLKSYNC = 0. This will stop the time-base clock within any enabled ePWM module.
3. Configure the prescaler values and desired ePWM modes.
4. Set TBCLKSYNC = 1.

3.2.2.5 Time-base Counter Modes and Timing Waveforms

The time-base counter operates in one of four modes:

- Up-count mode which is asymmetrical
- Down-count mode which is asymmetrical
- Up-down-count which is symmetrical
- Frozen where the time-base counter is held constant at the current value

To illustrate the operation of the first three modes, the following timing diagrams show when events are generated and how the time-base responds to an EPWMxSYNCI signal.

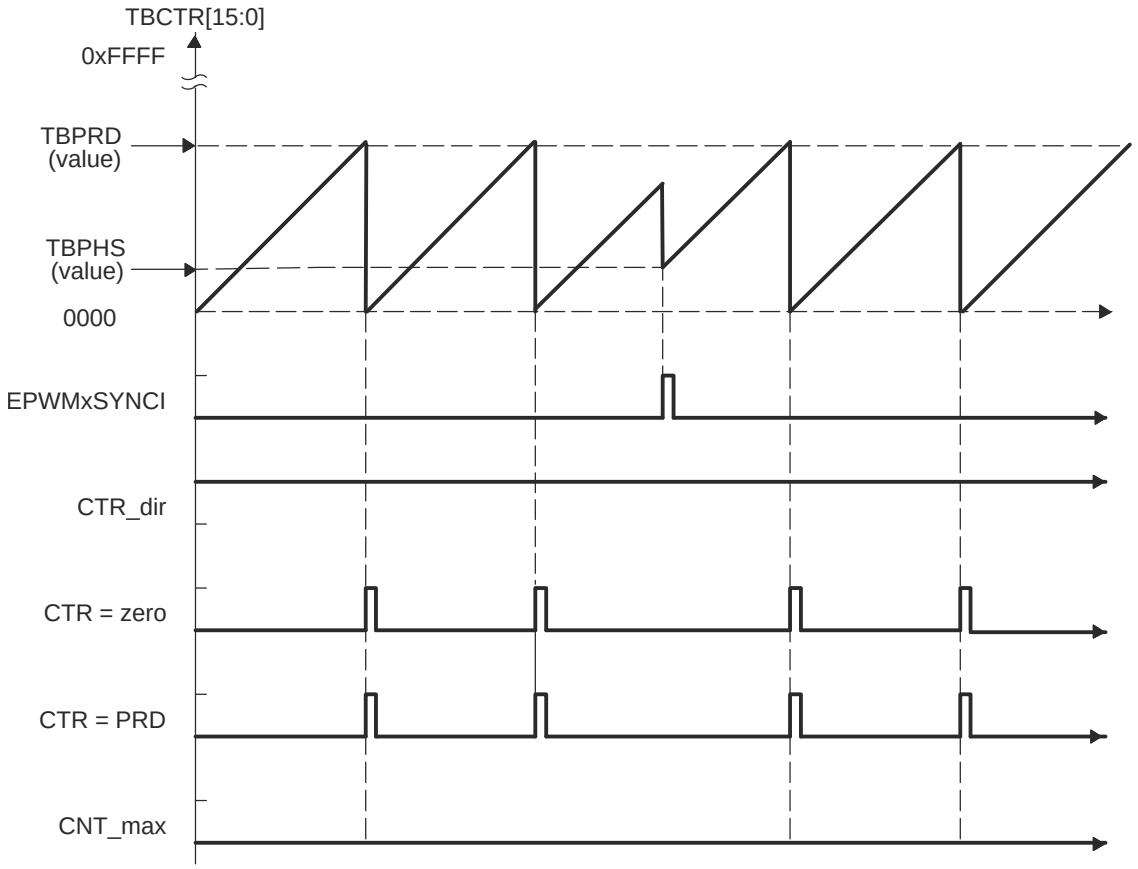


Figure 3-8. Time-Base Up-Count Mode Waveforms

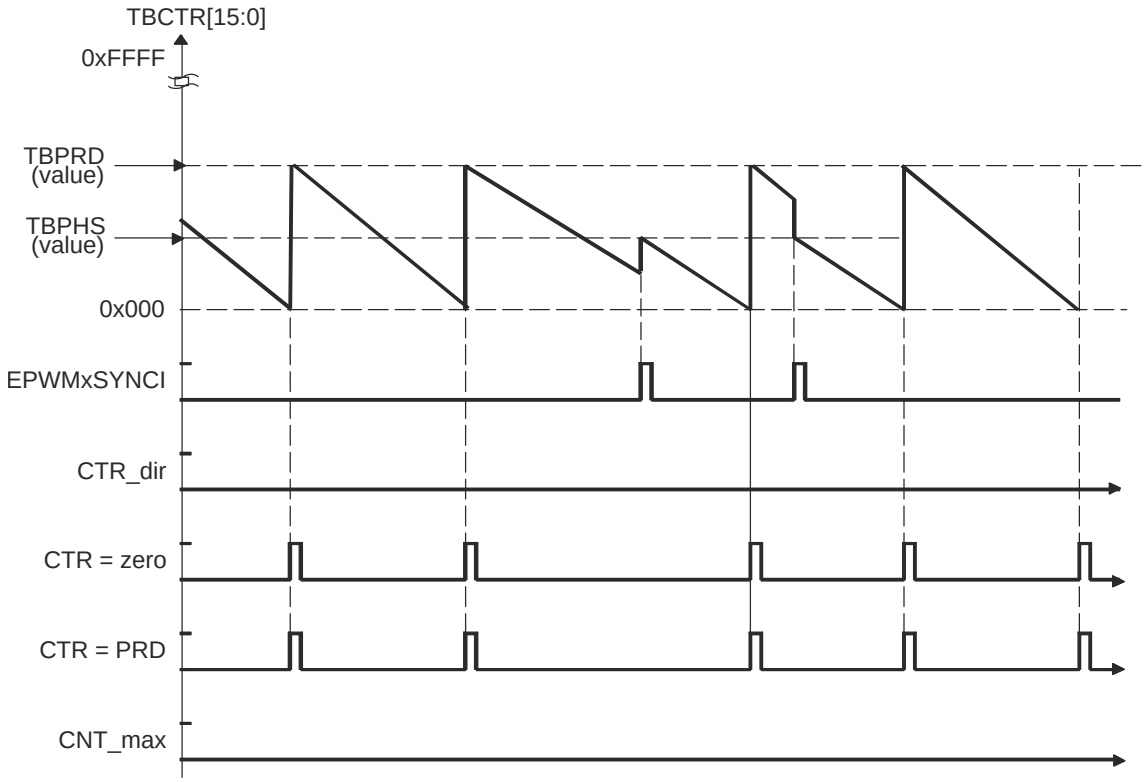


Figure 3-9. Time-Base Down-Count Mode Waveforms

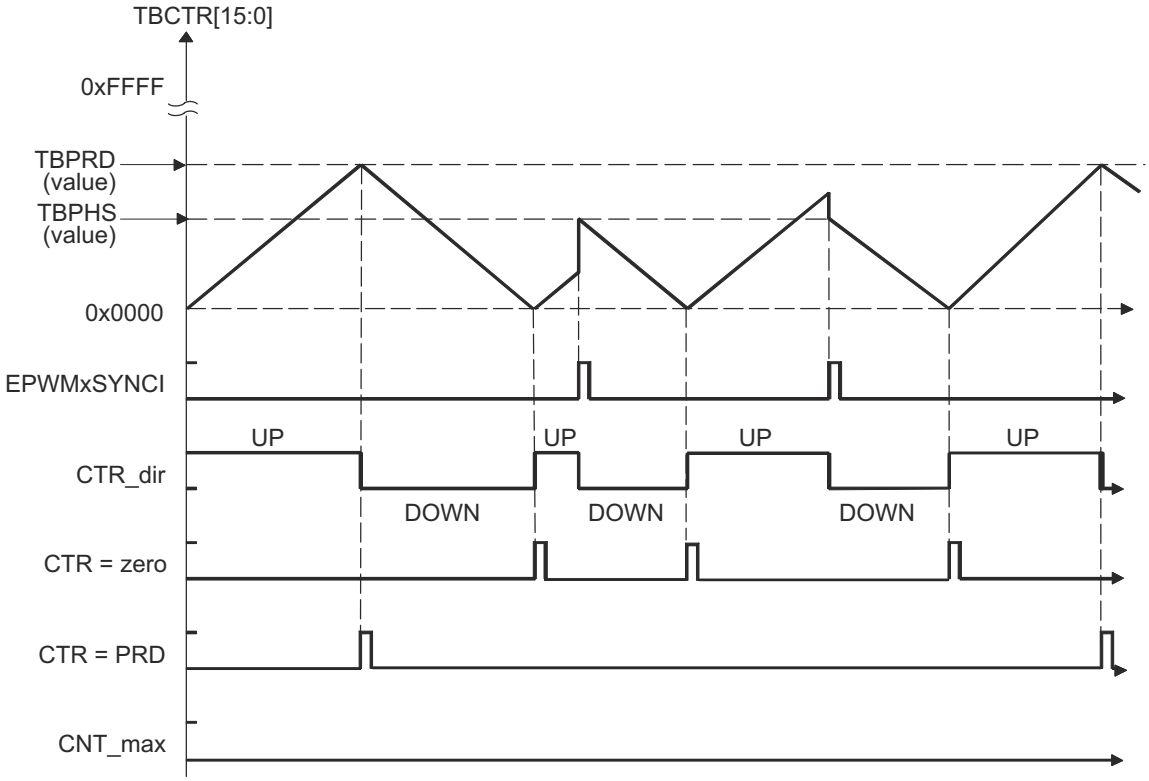


Figure 3-10. Time-Base Up-Down-Count Waveforms, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

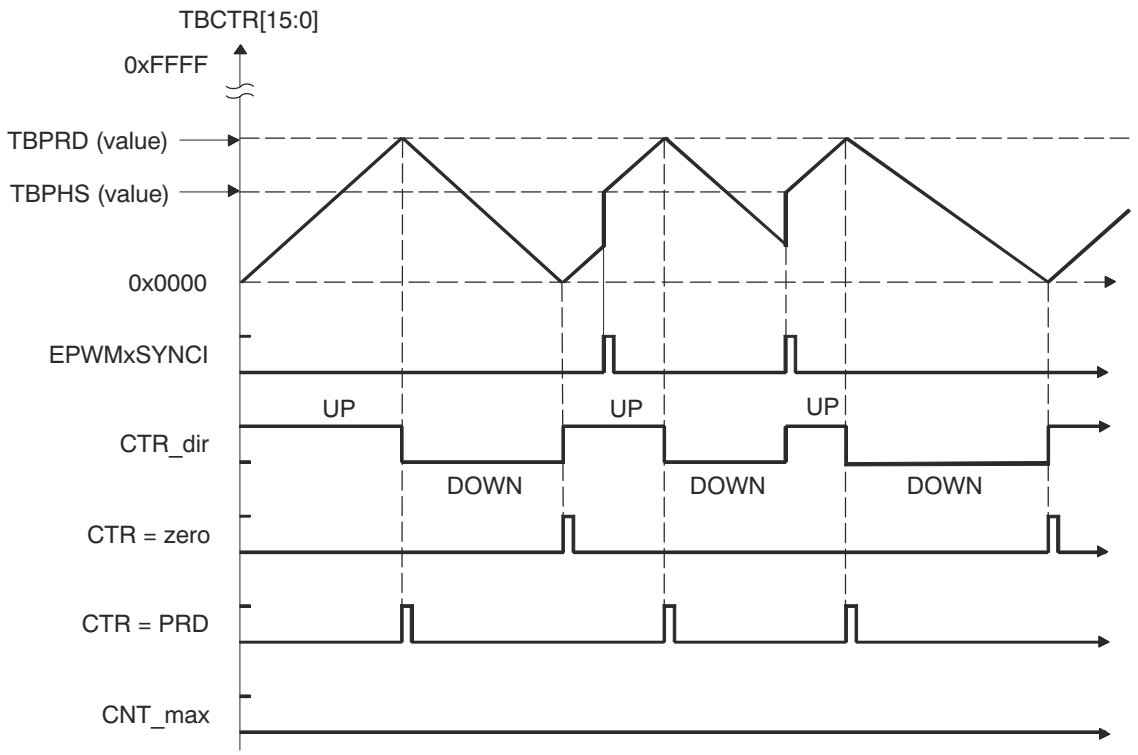


Figure 3-11. Time-Base Up-Down Count Waveforms, TBCTL[PHSDIR = 1] Count Up On Synchronization Event

3.2.3 Counter-Compare (CC) Submodule

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

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

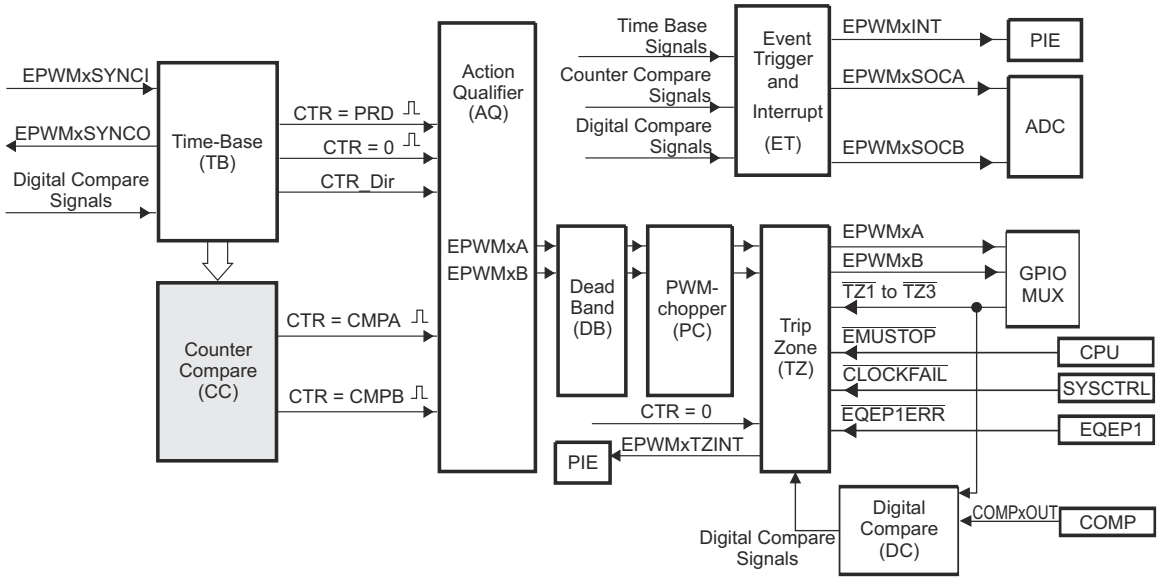


Figure 3-12. Counter-Compare Submodule

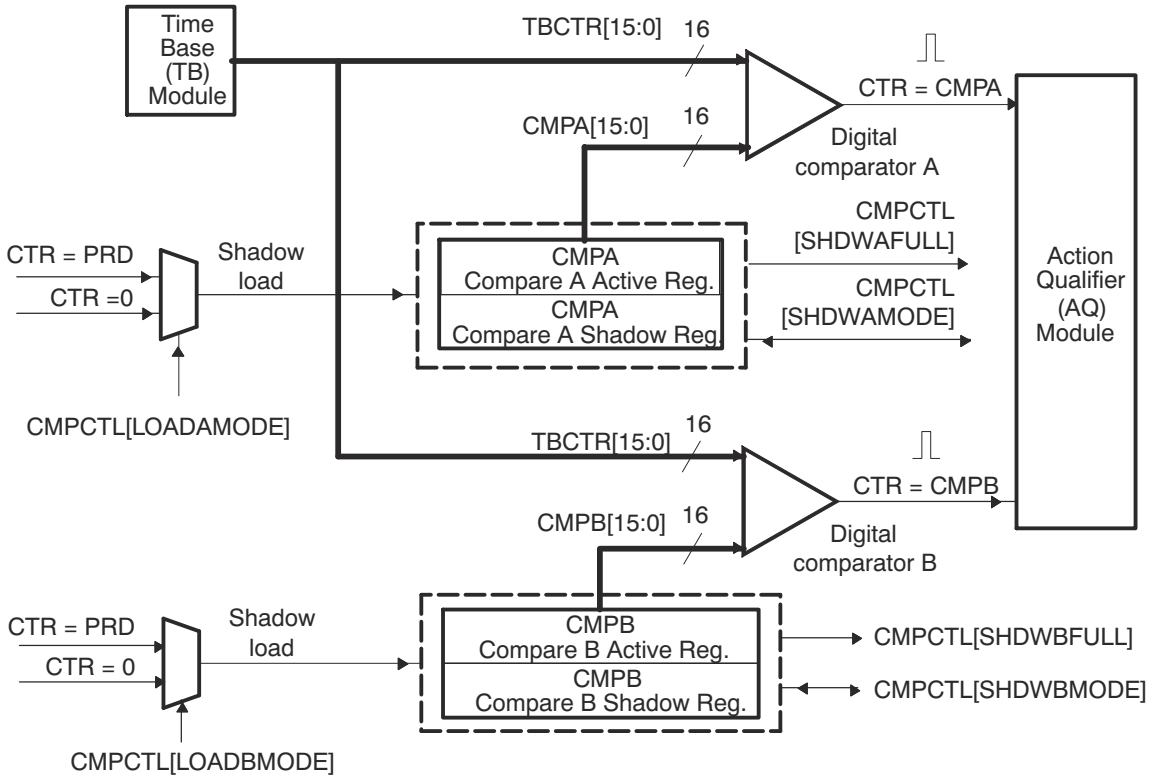


Figure 3-13. Detailed View of the Counter-Compare Submodule

3.2.3.1 Purpose of the Counter-Compare Submodule

The counter-compare submodule takes as input the time-base counter value. This value is continuously compared to the counter-compare A (CMPA) and counter-compare B (CMPB) registers. When the time-base counter is equal to one of the compare registers, the counter-compare unit generates an appropriate event.

The counter-compare:

- Generates events based on programmable time stamps using the CMPA and CMPB registers
  - CTR = CMPA: Time-base counter equals counter-compare A register (TBCTR = CMPA)
  - CTR = CMPB: Time-base counter equals counter-compare B register (TBCTR = CMPB)
- Controls the PWM duty cycle if the action-qualifier submodule is configured appropriately
- Shadows new compare values to prevent corruption or glitches during the active PWM cycle

3.2.3.2 Controlling and Monitoring the Counter-Compare Submodule

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

Table 3-5. Counter-Compare Submodule Registers

Register	Address Offset	Shadowed	Description	Bit Description
CMPCTL	0x0007	No	Counter-Compare Control Register.	Section 3.4.2.1
CMPAHR	0x0008	Yes	HRPWM Counter-Compare A Extension Register (1)	Section 3.4.2.2
CMPA	0x0009	Yes	Counter-Compare A Register	Section 3.4.2.3
CMPB	0x000A	Yes	Counter-Compare B Register	Section 3.4.2.4
CMPAHRM	0x002C	Writes	HRPWM counter-compare A Extension Mirror Register(1)	Section 3.4.2.5
CMPAM	0x002D	Writes	Counter-compare A mirror Register	Section 3.4.2.6

(1) This register is available only on ePWM modules with the high-resolution extension (HRPWM). On ePWM modules that do not include the HRPWM, this location is reserved. This register is also described in Chapter 4. Refer to your device-specific data sheet to determine which ePWM instances include this feature.

The key signals associated with the counter-compare submodule are described in Table 3-6.

Table 3-6. Counter-Compare Submodule Key Signals

Signal	Description of Event	Registers Compared
CTR = CMPA	Time-base counter equal to the active counter-compare A value	TBCTR = CMPA
CTR = CMPB	Time-base counter equal to the active counter-compare B value	TBCTR = CMPB
CTR = PRD	Time-base counter equal to the active period. Used to load active counter-compare A and B registers from the shadow register	TBCTR = TBPRD
CTR = ZERO	Time-base counter equal to zero. Used to load active counter-compare A and B registers from the shadow register	TBCTR = 0x0000

3.2.3.3 Operational Highlights for the Counter-Compare Submodule

The counter-compare submodule is responsible for generating two independent compare events based on two compare registers:

1. CTR = CMPA: Time-base counter equal to counter-compare A register (TBCTR = CMPA)
2. CTR = CMPB: Time-base counter equal to counter-compare B register (TBCTR = CMPB)

For up-count or down-count mode, each event occurs only once per cycle. For up-down-count mode each event occurs twice per cycle if the compare value is between 0x0000-TBPRD and once per cycle if the compare value is equal to 0x0000 or equal to TBPRD. These events are fed into the action-qualifier submodule where they are qualified by the counter direction and converted into actions if enabled. Refer to Section 3.2.4.1 for more details.

The counter-compare registers CMPA and CMPB each have an associated shadow register. Shadowing provides a way to keep updates to the registers synchronized with the hardware. When shadowing is used, updates to the active registers only occur at strategic points. This prevents corruption or spurious operation due to the register being asynchronously modified by software. The memory address of the active register and the shadow register is identical. Which register is written to or read from is determined by the CMPCTL[SHDWAMODE] and CMPCTL[SHDWBMODE] bits. These bits enable and disable the CMPA shadow register and CMPB shadow register respectively. The behavior of the two load modes is:

Shadow Mode: The shadow mode for the CMPA is enabled by clearing the CMPCTL[SHDWAMODE] bit and the shadow register for CMPB is enabled by clearing the CMPCTL[SHDWBMODE] bit. Shadow mode is enabled by default for both CMPA and CMPB.

If the shadow register is enabled then the content of the shadow register is transferred to the active register on one of the following events as specified by the CMPCTL[LOADAMODE] and CMPCTL[LOADBMODE] register bits:

- CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD)
- CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000)
- Both CTR = PRD and CTR = Zero

Only the active register contents are used by the counter-compare submodule to generate events to be sent to the action-qualifier.

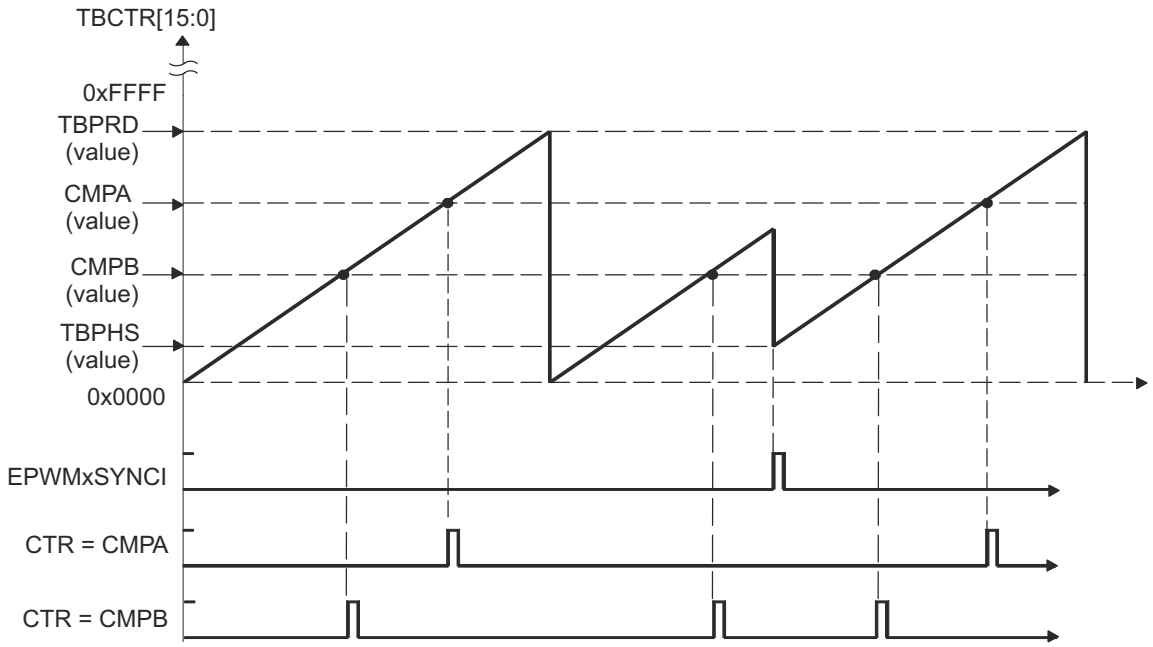
Immediate Load Mode: If immediate load mode is selected (TBCTL[SHADWAMODE] = 1 or TBCTL[SHADWBMODE] = 1), then a read from or a write to the register will go directly to the active register.

3.2.3.4 Count Mode Timing Waveforms

The counter-compare module can generate compare events in all three count modes:

- Up-count mode: used to generate an asymmetrical PWM waveform
- Down-count mode: used to generate an asymmetrical PWM waveform
- Up-down-count mode: used to generate a symmetrical PWM waveform

To best illustrate the operation of the first three modes, the timing diagrams in Figure 3-14 through Figure 3-17 show when events are generated and how the EPWMxSYNCI signal interacts.



An 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 3-14. Counter-Compare Event Waveforms in Up-Count Mode

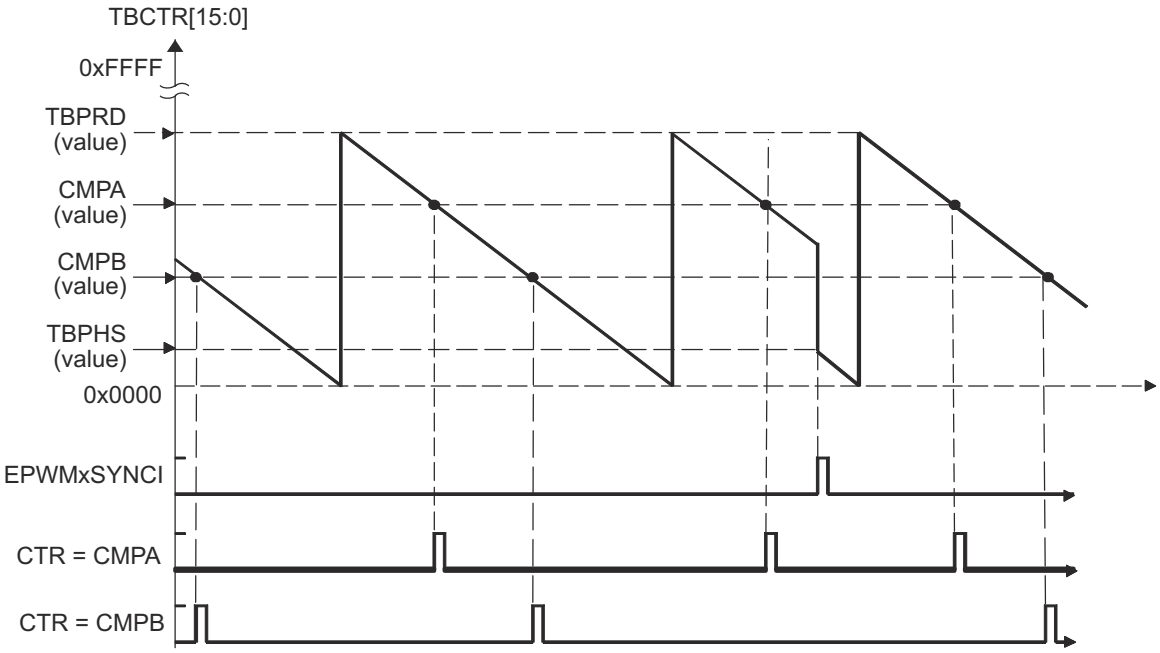


Figure 3-15. Counter-Compare Events in Down-Count Mode

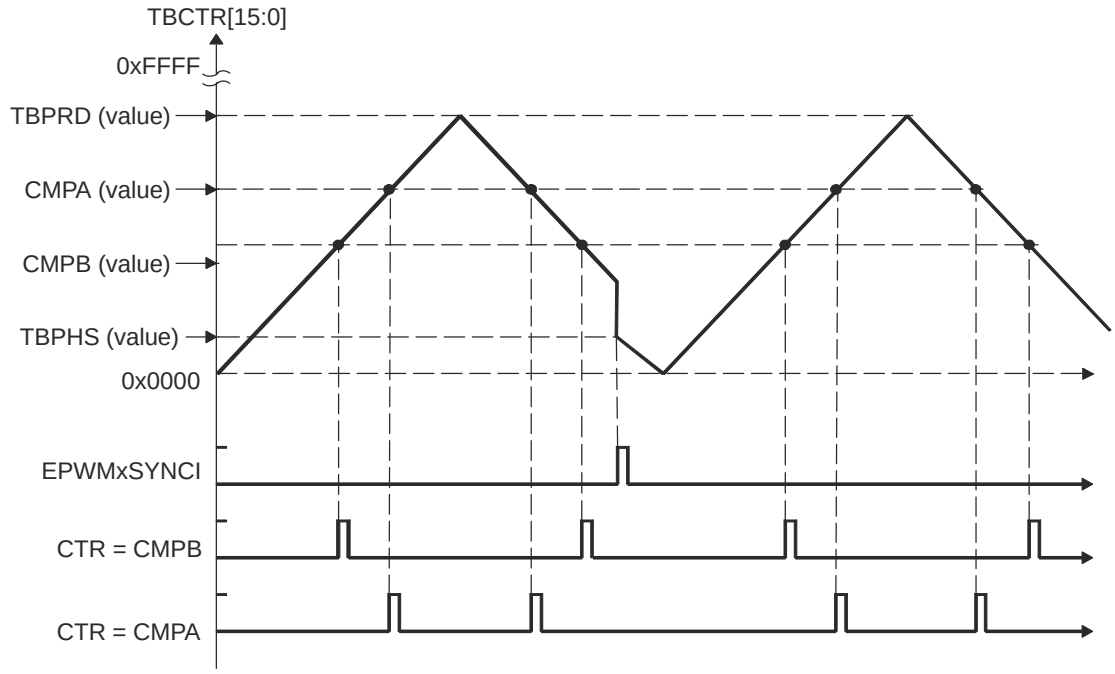


Figure 3-16. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

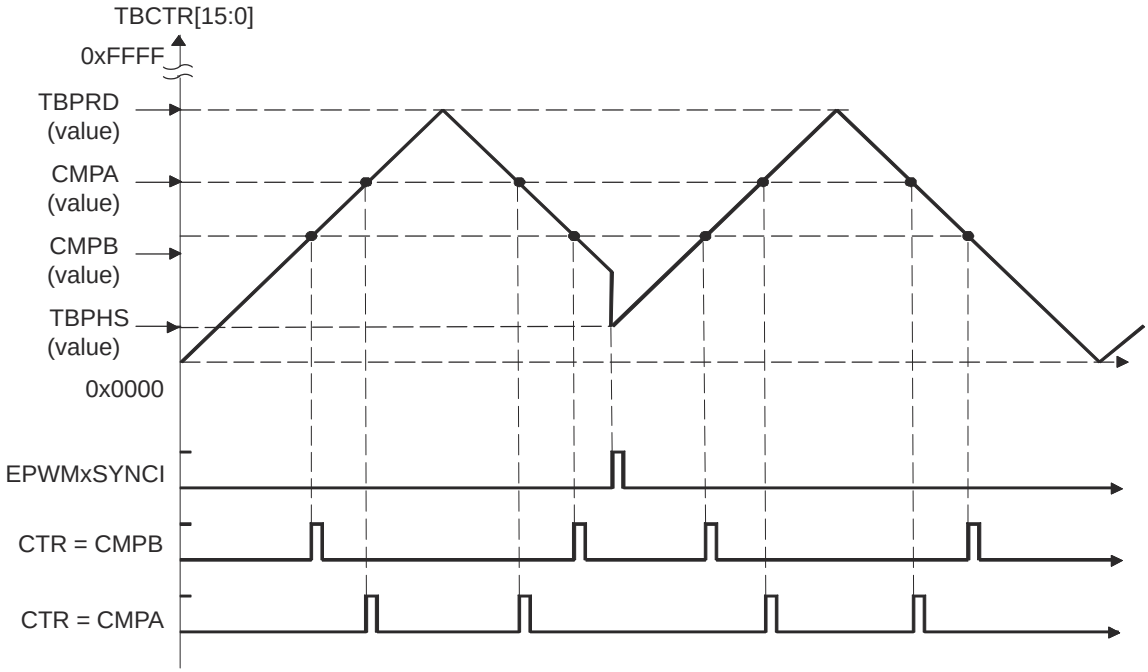


Figure 3-17. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 1] Count Up On Synchronization Event

3.2.4 Action-Qualifier (AQ) Submodule

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

The action-qualifier submodule has the most important role in waveform construction and PWM generation. It decides which events are converted into various action types, thereby producing the required switched waveforms at the EPWMxA and EPWMxB outputs.

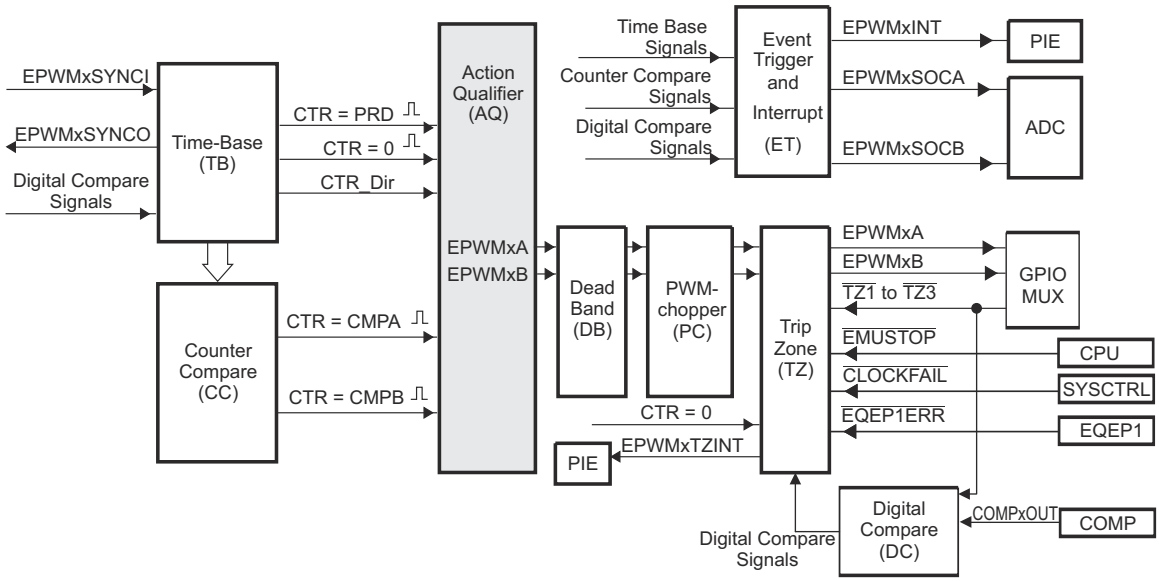


Figure 3-18. Action-Qualifier Submodule

3.2.4.1 Purpose of the Action-Qualifier Submodule

The action-qualifier submodule is responsible for the following:

- Qualifying and generating actions (set, clear, toggle) based on the following events:
  - CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD).
  - CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000)
  - CTR = CMPA: Time-base counter equal to the counter-compare A register (TBCTR = CMPA)
  - CTR = CMPB: Time-base counter equal to the counter-compare B register (TBCTR = CMPB)
- Managing priority when these events occur concurrently
- Providing independent control of events when the time-base counter is increasing and when it is decreasing

3.2.4.2 Action-Qualifier Submodule Control and Status Register Definitions

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

Table 3-7. Action-Qualifier Submodule Registers

Register	Address Offset	Shadowed	Description	Bit Description
AQCTLA	0x000B	No	Action-Qualifier Control Register for Output A (EPWMxA)	Section 3.4.3.1
AQCTLB	0x000C	No	Action-Qualifier Control Register for Output B (EPWMxB)	Section 3.4.3.2
AQSFRC	0x000D	No	Action-Qualifier Software Force Register	Section 3.4.3.3
AQCSFRC	0x000E	Yes	Action-Qualifier Continuous Software Force	Section 3.4.3.4

The action-qualifier submodule is based on event-driven logic. It can be thought of as a programmable cross switch with events at the input and actions at the output, all of which are software controlled via the set of registers shown in Table 3-7.

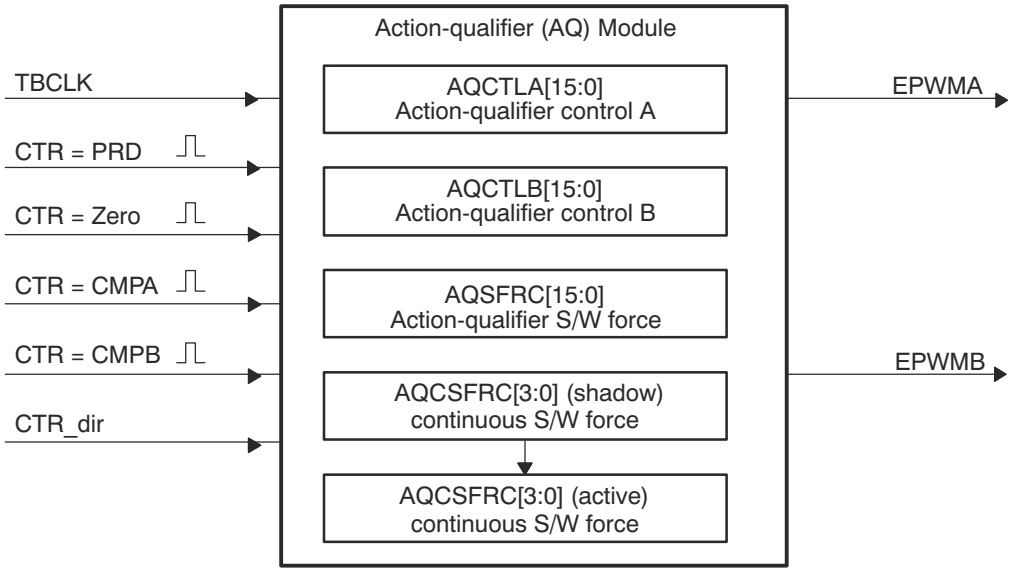


Figure 3-19. Action-Qualifier Submodule Inputs and Outputs

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

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

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

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

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

The possible actions imposed on outputs EPWMxA and EPWMxB are:

- **Set High:** Set output EPWMxA or EPWMxB to a high level.
- **Clear Low:** Set output EPWMxA or EPWMxB to a low level.
- **Toggle:** If EPWMxA or EPWMxB is currently pulled high, then pull the output low. If EPWMxA or EPWMxB is currently pulled low, then pull the output high.
- **Do Nothing:** Keep outputs EPWMxA and EPWMxB at same level as currently set. Although the "Do Nothing" option prevents an event from causing an action on the EPWMxA and EPWMxB outputs, this event can still trigger interrupts and ADC start of conversion. See the Event-trigger Submodule description in Section 3.2.8 for details.

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

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

S/W force	TB Counter equals:				Actions
	Zero	Comp A	Comp B	Period	
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

Figure 3-20. Possible Action-Qualifier Actions for EPWMxA and EPWMxB Outputs

3.2.4.3 Action-Qualifier Event Priority

It is possible for the ePWM action qualifier to receive more than one event at the same time. In this case events are assigned a priority by the hardware. The general rule is events occurring later in time have a higher priority and software forced events always have the highest priority. The event priority levels for up-down-count mode are shown in Table 3-9. A priority level of 1 is the highest priority and level 7 is the lowest. The priority changes slightly depending on the direction of TBCTR.

Table 3-9. Action-Qualifier Event Priority for Up-Down-Count Mode

Priority Level	Event If TBCTR is Incrementing TBCTR = Zero up to TBCTR = TBPRD	Event If TBCTR is Decrementing TBCTR = TBPRD down to TBCTR = 1
1 (Highest)	Software forced event	Software forced event
2	Counter equals CMPB on up-count (CBU)	Counter equals CMPB on down-count (CBD)
3	Counter equals CMPA on up-count (CAU)	Counter equals CMPA on down-count (CAD)
4	Counter equals zero	Counter equals period (TBPRD)
5	Counter equals CMPB on down-count (CBD)	Counter equals CMPB on up-count (CBU)
6 (Lowest)	Counter equals CMPA on down-count (CAD)	Counter equals CMPA on up-count (CBU)

Table 3-10 shows the action-qualifier priority for up-count mode. In this case, the counter direction is always defined as up and thus down-count events will never be taken.

Table 3-10. Action-Qualifier Event Priority for Up-Count Mode

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to period (TBPRD)
3	Counter equal to CMPB on up-count (CBU)
4	Counter equal to CMPA on up-count (CAU)
5 (Lowest)	Counter equal to Zero

Table 3-11 shows the action-qualifier priority for down-count mode. In this case, the counter direction is always defined as down and thus up-count events will never be taken.

Table 3-11. Action-Qualifier Event Priority for Down-Count Mode

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to Zero
3	Counter equal to CMPB on down-count (CBD)
4	Counter equal to CMPA on down-count (CAD)
5 (Lowest)	Counter equal to period (TBPRD)

It is possible to set the compare value greater than the period. In this case the action will take place as shown in Table 3-12.

Table 3-12. Behavior if CMPA/CMPB is Greater than the Period

Counter Mode	Compare on Up-Count Event CAU/CBU	Compare on Down-Count Event CAD/CBD
Up-Count Mode	If CMPA/CMPB ≤ TBPRD period, then the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB > TBPRD, then the event will not occur.	Never occurs.

Table 3-12. Behavior if CMPA/CMPB is Greater than the Period (continued)

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

3.2.4.4 Waveforms for Common Configurations

Note

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

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

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

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

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

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

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

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

See *Using the ePWM Module for 0% - 100% Duty Cycle Control*. The software configurations described in this application report are not applicable when changing compare registers from a non-zero value to zero. However, they still apply when changing from a compare value of 0 to a non-zero value.

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

When using this configuration in practice, if you load CMPA/CMPB on zero, then use CMPA/CMPB values greater than or equal to 1. If you load CMPA/CMPB on period, then use CMPA/CMPB values less than or equal to TBPRD-1. This means there will always be a pulse of at least one TBCLK cycle in a PWM period which, when very short, tend to be ignored by the system.

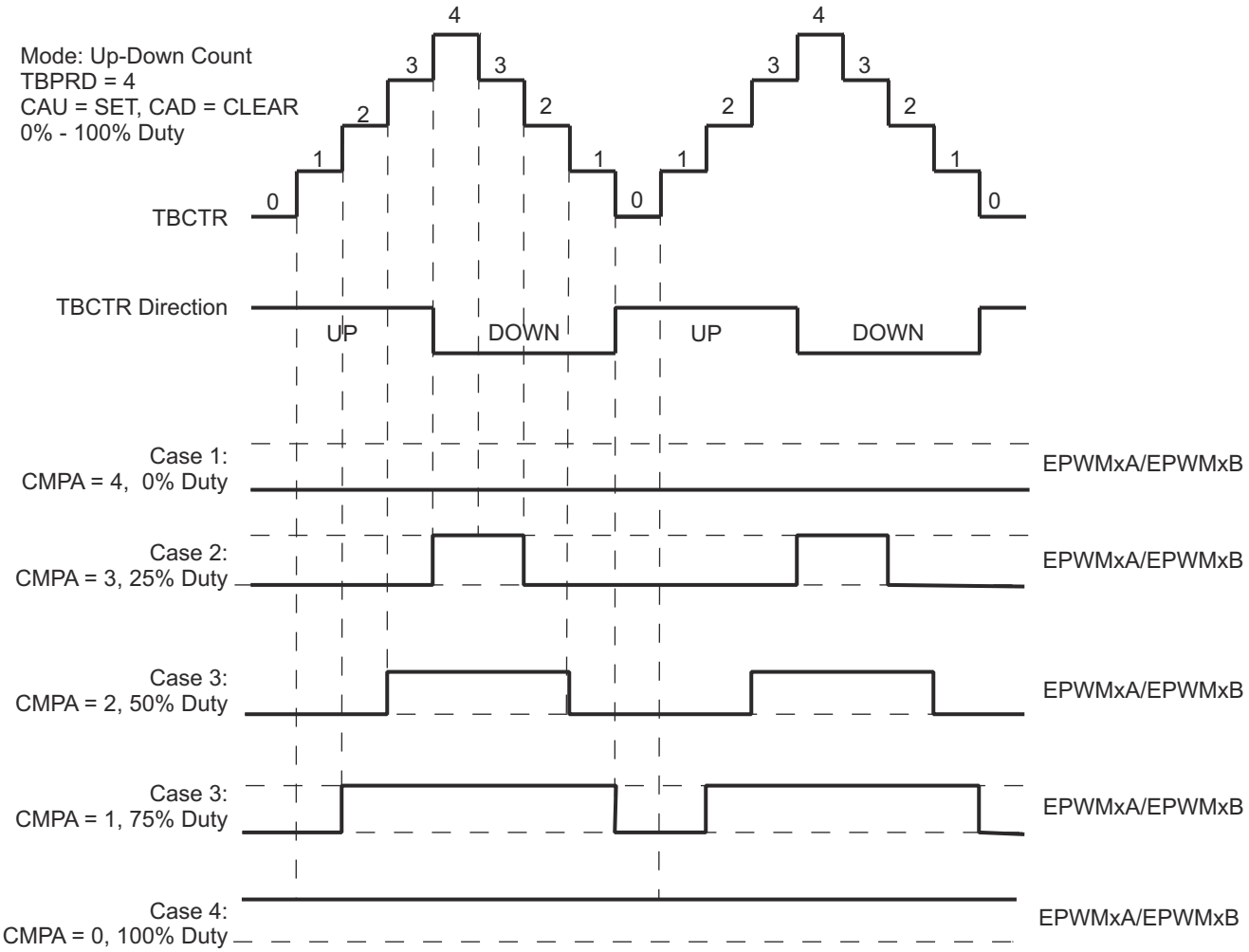
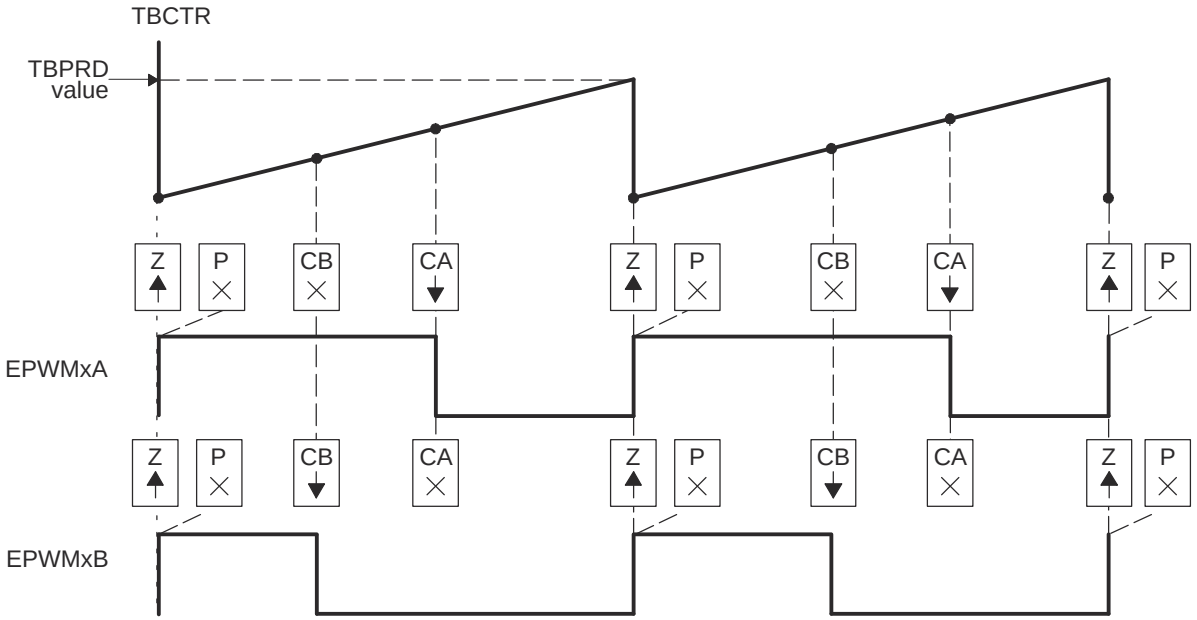


Figure 3-21. Up-Down-Count Mode Symmetrical Waveform

The PWM waveforms in Figure 3-22 through Figure 3-27 show some common action-qualifier configurations. The C-code samples in Example 3-1 through Example 3-6 shows how to configure an ePWM module for each case. Some conventions used in the figures and examples are as follows:

- TBPRD, CMPA, and CMPB refer to the value written in their respective registers. The active register, not the shadow register, is used by the hardware.
- CMPx, refers to either CMPA or CMPB.
- EPWMxA and EPWMxB refer to the output signals from ePWMx
- Up-Down means Count-up-and-down mode, Up means up-count mode and Dwn means down-count mode
- Sym = Symmetric, Asym = Asymmetric

Example 3-1 contains a code sample showing initialization and run time for the waveforms in Figure 3-22.



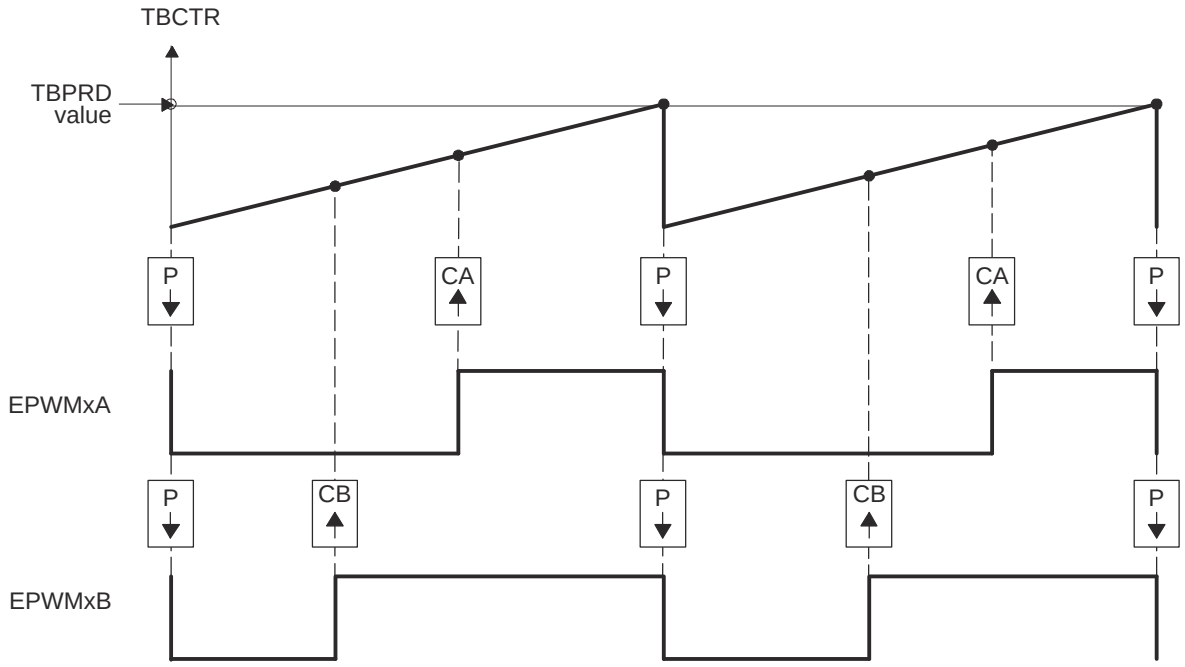
A. PWM period = (TBPRD + 1) × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active high (that is, high time duty proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active high (that is, high time duty proportional to CMPB).
D. The "Do Nothing" actions (X) are 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.

Figure 3-22. Up, Single Edge Asymmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB—Active High

Example 3-1. Code Sample for Figure 3-22

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                        // Period = 601 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350;               // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 200;                         // Compare B = 200 TBCLK counts
EPwm1Regs.TBPHS = 0;                          // Set Phase register to zero
EPwm1Regs.TBCTR = 0;                          // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Phase loading disabled
EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // TBCLK = SYSCLK
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR = Zero
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;
EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
//
// Run Time
// = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.CMPA.half.CMPA = Duty1A;            // adjust duty for output EPWM1A
EPwm1Regs.CMPB = Duty1B;                      // adjust duty for output EPWM1B
```

Example 3-2 contains a code sample showing initialization and run time for the waveforms in Figure 3-23.



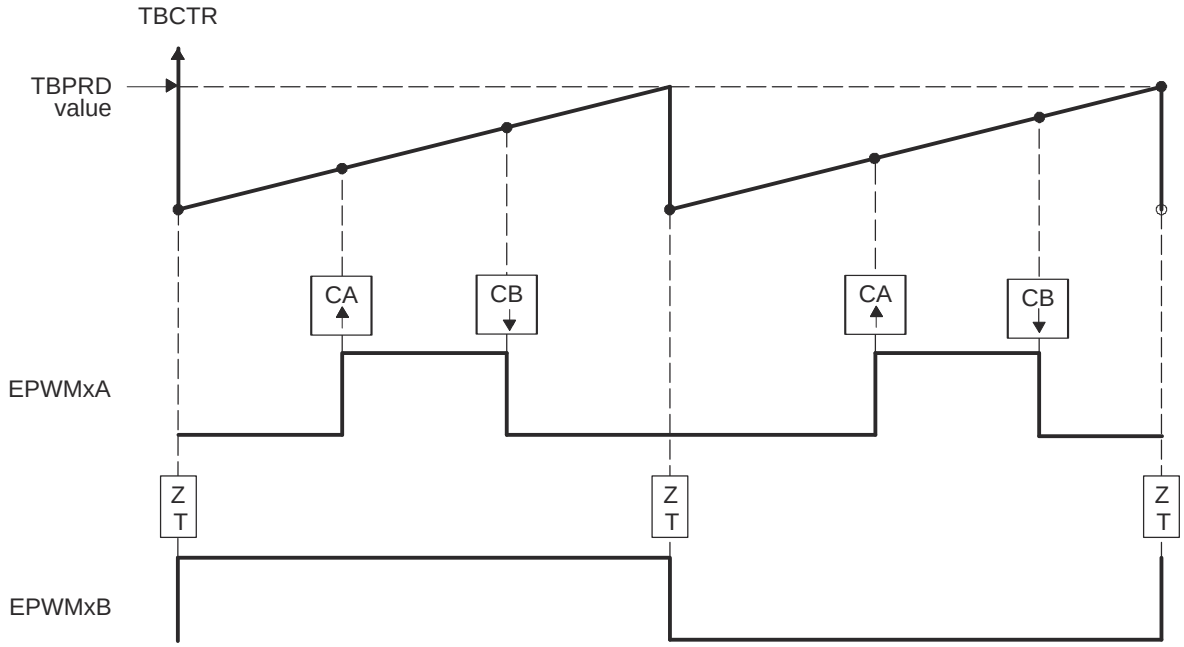
A. PWM period = (TBPRD + 1) × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
D. Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

Figure 3-23. Up, Single Edge Asymmetric Waveform With Independent Modulation on EPWMxA and EPWMxB—Active Low

Example 3-2. Code Sample for Figure 3-23

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

Example 3-3 contains a code sample showing initialization and run time for the waveforms Figure 3-24. Use the constant definitions in the device-specific header file.



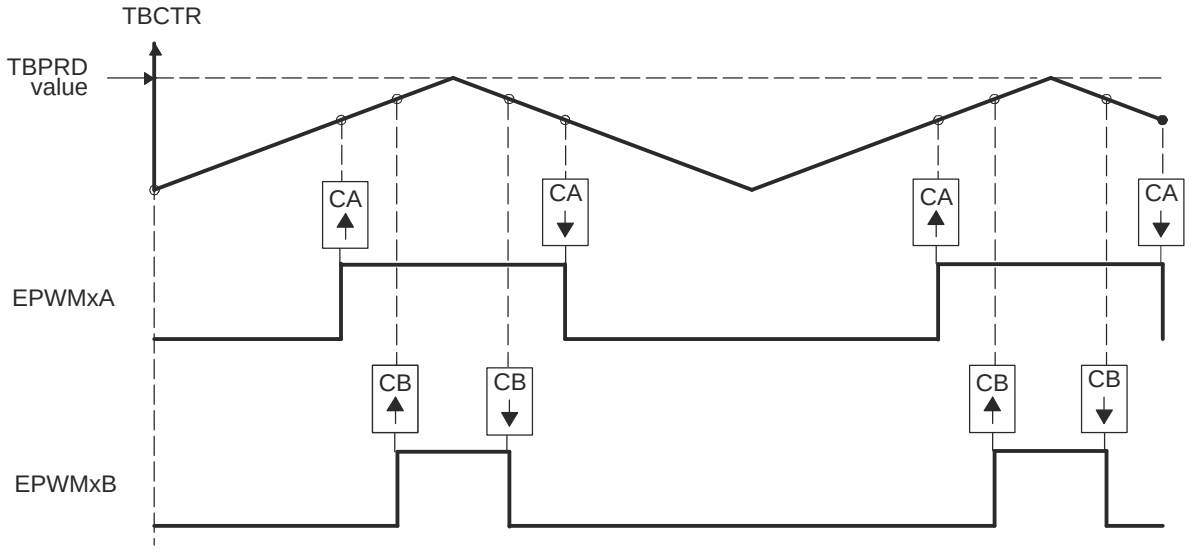
A. PWM frequency = 1/((TBPRD + 1) × T_{TBCLK})
B. Pulse can be placed anywhere within the PWM cycle (0000 - TBPRD)
C. High time duty proportional to (CMPB - CMPA)
D. EPWMxB can be used to generate a 50% duty square wave with frequency = **?** × ((TBPRD + 1) × TBCLK)

Figure 3-24. Up-Count, Pulse Placement Asymmetric Waveform With Independent Modulation on EPWMxA

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

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

Example 3-4 contains a code sample showing initialization and run time for the waveforms in Figure 3-25. Use the constant definitions in the device-specific header file.



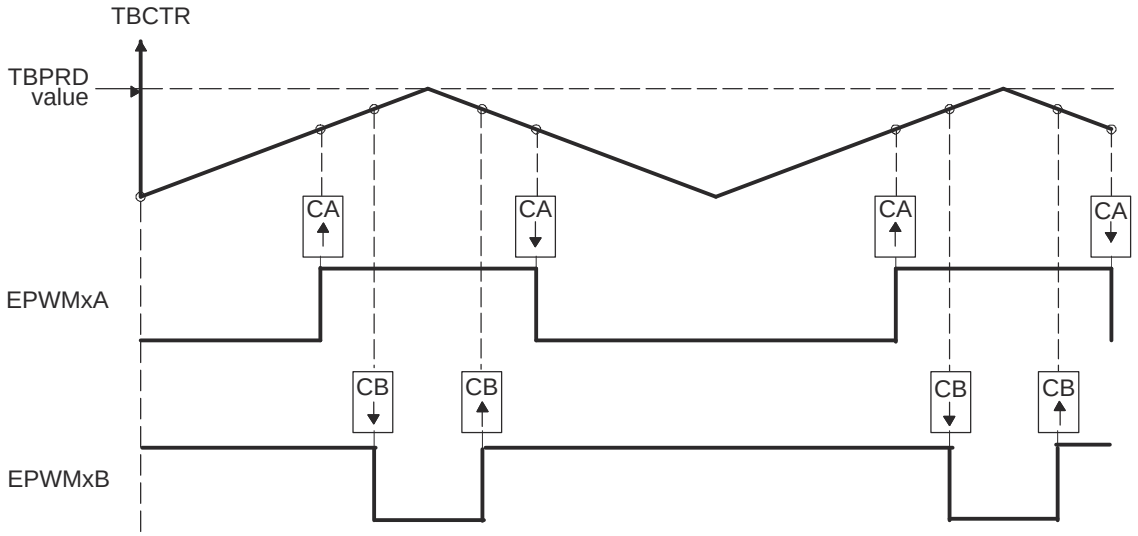
A. PWM period = 2 x TBPRD × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
D. Outputs EPWMxA and EPWMxB can drive independent power switches

Figure 3-25. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Active Low

Example 3-4. Code Sample for Figure 3-25

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

Example 3-5 contains a code sample showing initialization and run time for the waveforms in Figure 3-26. Use the constant definitions in the device-specific header file.



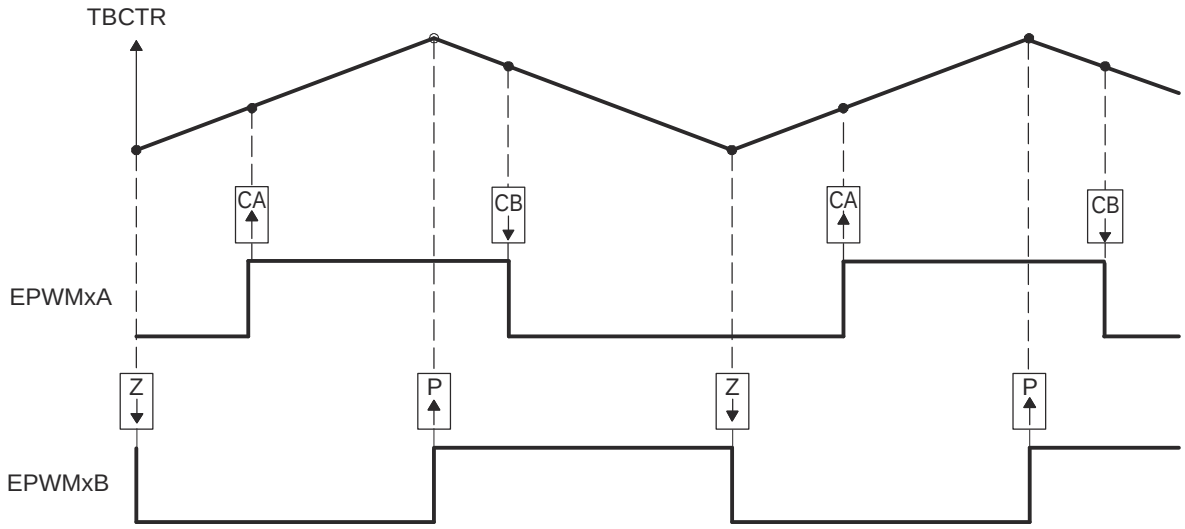
A. PWM period = 2 × TBPRD × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low, that is, low time duty proportional to CMPA
C. Duty modulation for EPWMxB is set by CMPB and is active high, that is, high time duty proportional to CMPB
D. Outputs EPWMx can drive upper/lower (complementary) 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.

Figure 3-26. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Complementary

Example 3-5. Code Sample for Figure 3-26

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                        // Period = 2´600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 350;               // Compare A = 350 TBCLK counts
EPwm1Regs.CMPB = 400;                         // Compare B = 400 TBCLK counts
EPwm1Regs.TBPHS = 0;                          // Set Phase register to zero
EPwm1Regs.TBCTR = 0;                          // clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetric
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Phase loading disabled
EPwm1Regs.TBCTL.bit.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
```

Example 3-6 contains a code sample showing initialization and run time for the waveforms in Figure 3-27. Use the constant definitions in the device-specific header file.



A. PWM period = 2 × 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 (that is, Set ! Clear and Clear Set).
F. Duty modulation for EPWMxB is fixed at 50% (utilizes spare action resources for EPWMxB)

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

Example 3-6. Code Sample for Figure 3-27

```
// Initialization Time
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
EPwm1Regs.TBPRD = 600;                          //  Period = 2 ´ 600 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 250;                 //  Compare A = 250 TBCLK counts
EPwm1Regs.CMPB = 450;                           //  Compare B = 450 TBCLK counts
EPwm1Regs.TBPHS = 0;                            //  Set Phase register to zero
EPwm1Regs.TBCTR = 0;                            //  clear TB counter
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; //  Symmetric
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;         //  Phase loading disabled
EPwm1Regs.TBCTL.bit.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.

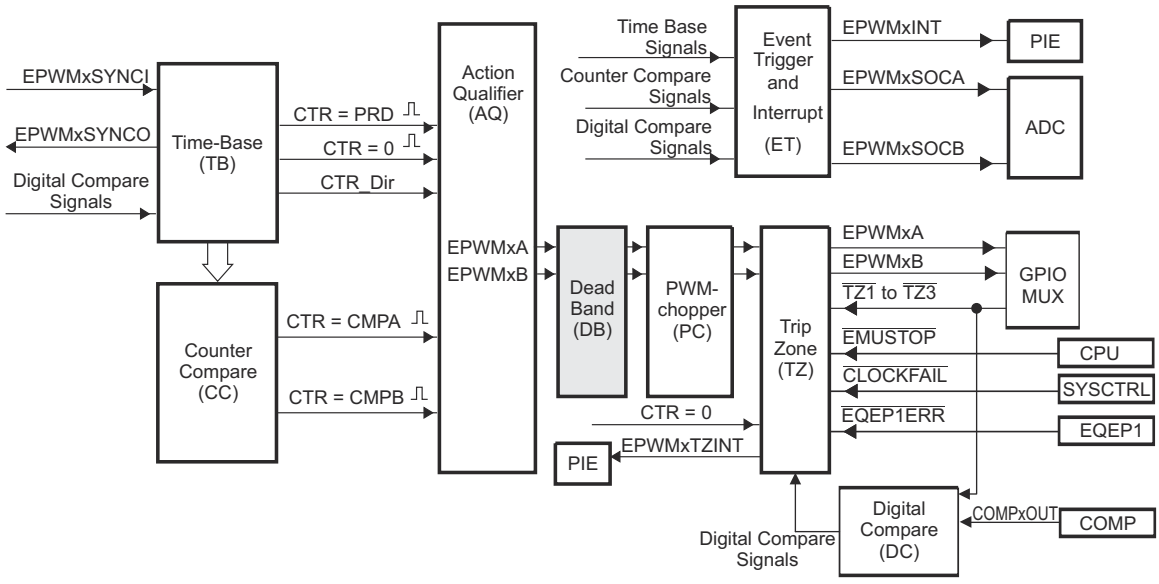


Figure 3-28. Dead-Band Submodule

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	Address Offset	Shadowed	Description	Bit Description
DBCTL	0x000F	No	Dead-Band Control Register	Section 3.4.4.1
DBRED	0x0010	No	Dead-Band Rising Edge Delay Count Register	Section 3.4.4.2
DBFED	0x0011	No	Dead-Band Falling Edge Delay Count Register	Section 3.4.4.3

3.2.5.3 Operational Highlights for the Dead-Band Submodule

The following sections provide the operational highlights. The dead-band submodule has two groups of independent selection options as shown in Figure 3-29.

- **Input Source Selection:** The input signals to the dead-band module are the EPWMxA and EPWMxB output signals from the action-qualifier. In this section they will be referred to as EPWMxA In and EPWMxB In. Using the DBCTL[IN_MODE) control bits, the signal source for each delay, falling-edge or rising-edge, can be selected:
  - EPWMxA In is the source for both falling-edge and rising-edge delay. This is the default mode.
  - EPWMxA In is the source for falling-edge delay, EPWMxB In is the source for rising-edge delay.
  - EPWMxA In is the source for rising edge delay, EPWMxB In is the source for falling-edge delay.
  - EPWMxB In is the source for both falling-edge and rising-edge delay.
- **Half Cycle Clocking:** The dead-band submodule can be clocked using half cycle clocking to double the resolution (that is, counter clocked at 2× TBCLK).
- **Output Mode Control:** The output mode is configured by way of the DBCTL[OUT_MODE] bits. These bits determine if the falling-edge delay, rising-edge delay, neither, or both are applied to the input signals.
- **Polarity Control:** The polarity control (DBCTL[POLSEL]) allows you to specify whether the rising-edge delayed signal and/or the falling-edge delayed signal is to be inverted before being sent out of the dead-band submodule.

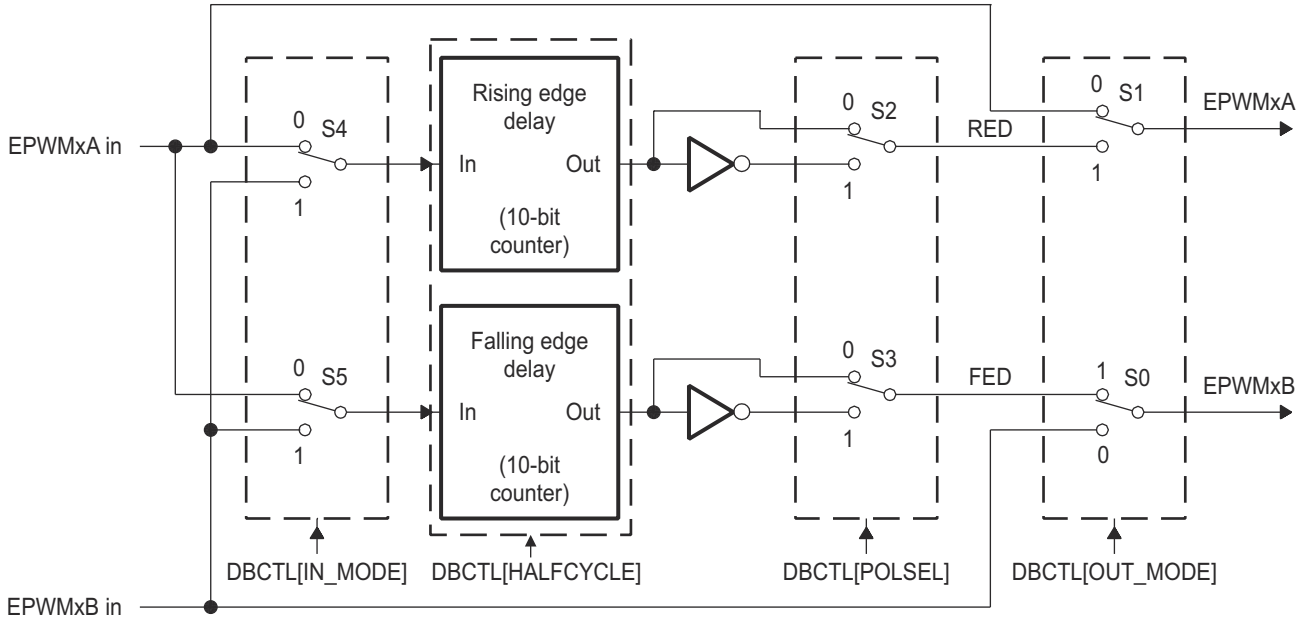


Figure 3-29. Configuration Options for the Dead-Band Submodule

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

- **Mode 1: Bypass both falling-edge delay (FED) and rising-edge delay (RED).** Allows you to fully disable the dead-band submodule from the PWM signal path.
- **Mode 2-5: Classical Dead-Band Polarity Settings.** These represent typical polarity configurations that should address all the active high/low modes required by available industry power switch gate drivers. The waveforms for these typical cases are shown in Figure 3-30. Note that to generate equivalent waveforms to Figure 3-30, configure the action-qualifier submodule to generate the signal as shown for EPWMxA.

- **Mode 6: Bypass rising-edge-delay and Mode 7: Bypass falling-edge-delay.** Finally the last two entries in Table 3-14 show combinations where either the falling-edge-delay (FED) or rising-edge-delay (RED) blocks are bypassed.

Table 3-14. Classical Dead-Band Operating Modes

Mode	Mode Description	DBCTL[POLSEL]		DBCTL[OUT_MODE]	
		S3	S2	S1	S0
1	EPWMxA and EPWMxB Passed Through (No Delay)	X	X	0	0
2	Active High Complementary (AHC)	1	0	1	1
3	Active Low Complementary (ALC)	0	1	1	1
4	Active High (AH)	0	0	1	1
5	Active Low (AL)	1	1	1	1
6	EPWMxA Out = EPWMxA In (No Delay) 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

The dead-band submodule supports independent values for rising-edge (RED) and falling-edge (FED) delays. The amount of delay is programmed using the DBRED and DBFED registers. These are 10-bit registers and their value represents the number of time-base clock, TBCLK, periods a signal edge is delayed by. For example, the formulas to calculate falling-edge-delay and rising-edge-delay are:

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

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

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

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

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

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

Figure 3-30 shows waveforms for typical cases where 0% < duty < 100%.

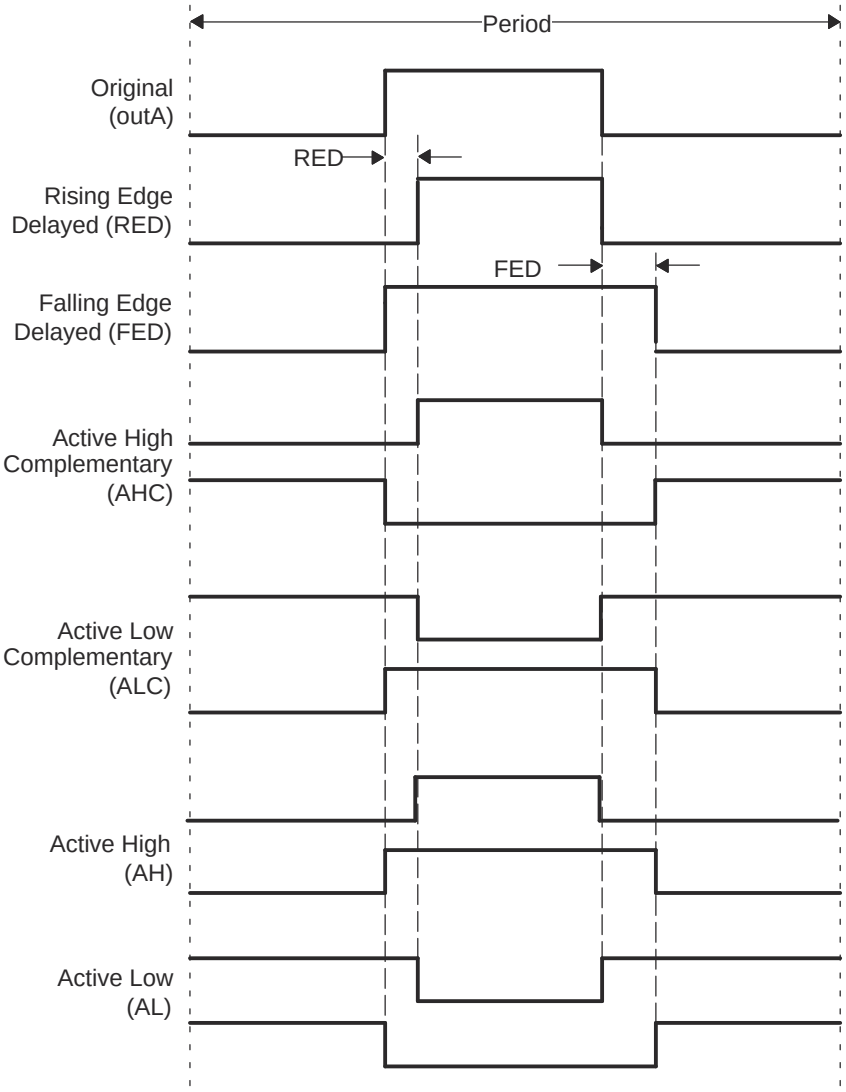


Figure 3-30. Dead-Band Waveforms for Typical Cases (0% < Duty < 100%)

3.2.6 PWM-Chopper (PC) Submodule

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

The PWM-chopper submodule allows a high-frequency carrier signal to modulate the PWM waveform generated by the action-qualifier and dead-band submodules. This capability is important if you need pulse transformer-based gate drivers to control the power switching elements.

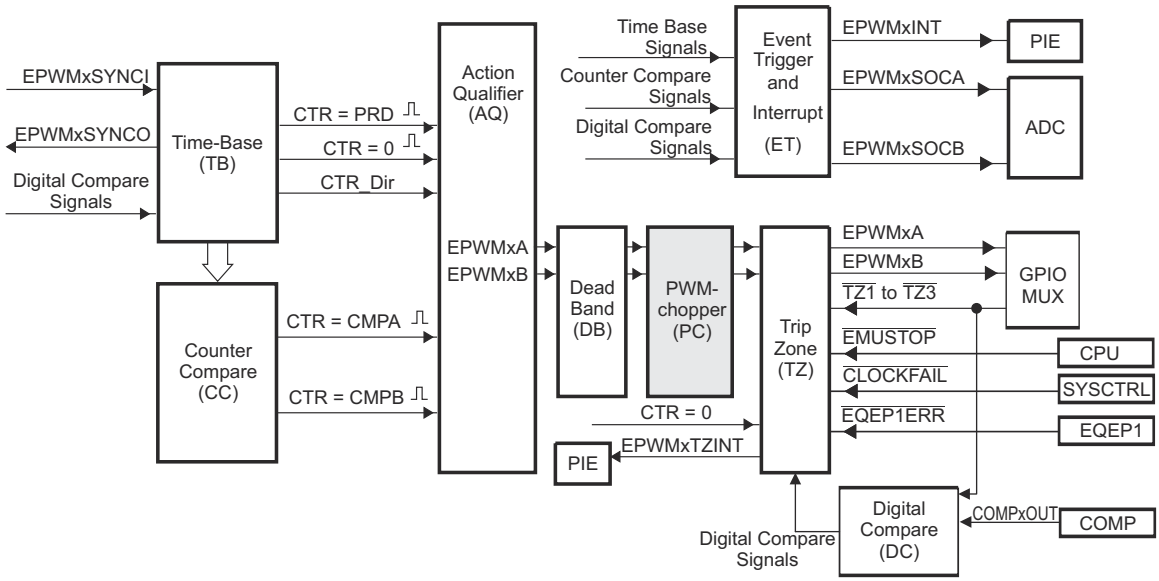


Figure 3-31. PWM-Chopper Submodule

3.2.6.1 Purpose of the PWM-Chopper Submodule

The key functions of the PWM-chopper submodule are:

- Programmable chopping (carrier) frequency
- Programmable pulse width of first pulse
- Programmable duty cycle of second and subsequent pulses
- Can be fully bypassed if not required

3.2.6.2 Controlling the PWM-Chopper Submodule

The PWM-chopper submodule operation is controlled via the registers in Table 3-15.

Table 3-15. PWM-Chopper Submodule Registers

Register	Address Offset	Shadowed	Description	Bit Description
PCCTL	0x001E	No	PWM-chopper Control Register	Section 3.4.7.1

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.

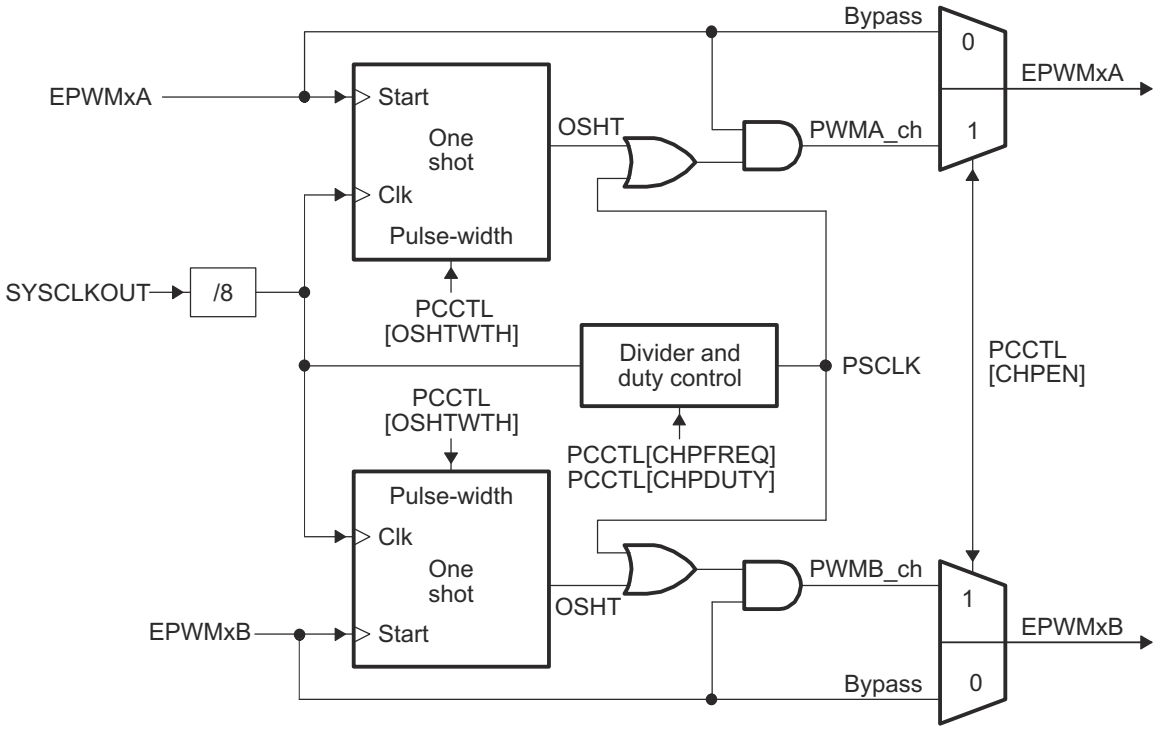


Figure 3-32. PWM-Chopper Submodule Operational Details

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.

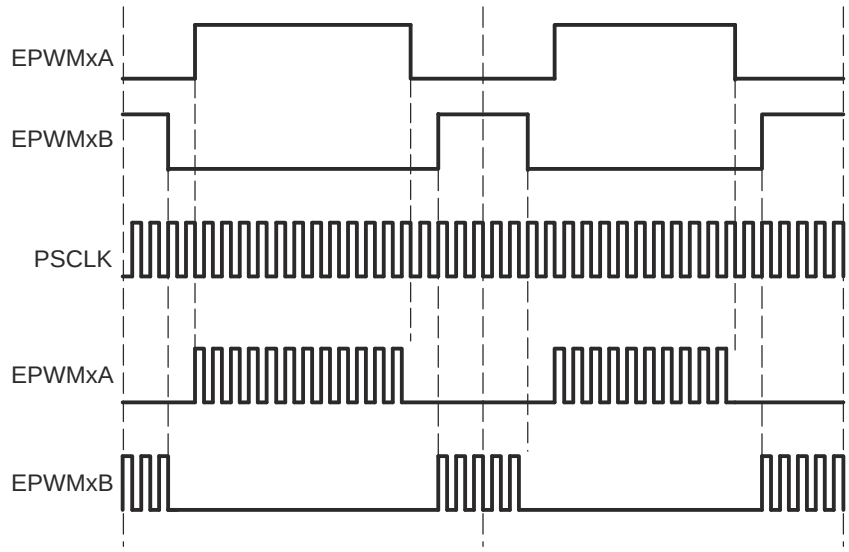


Figure 3-33. Simple PWM-Chopper Submodule Waveforms Showing Chopping Action Only

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 3-16 gives the possible pulse width values for a SYSCLKOUT = 60 MHz.

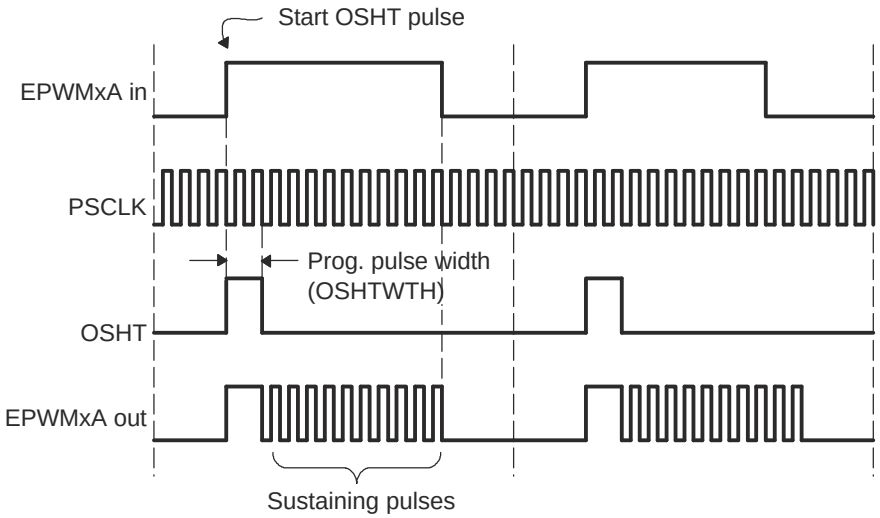


Figure 3-34. PWM-Chopper Submodule Waveforms Showing the First Pulse and Subsequent Sustaining Pulses

Table 3-16. Possible Pulse Width Values for SYSCLKOUT = 60 MHz

OSHTWTHz (hex)	Pulse Width (nS)
0	133
1	267
2	400
3	533
4	667
5	800
6	933
7	1067
8	1200
9	1333
A	1467
B	1600
C	1733
D	1867
E	2000
F	2133

3.2.6.4.2 Duty Cycle Control

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

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

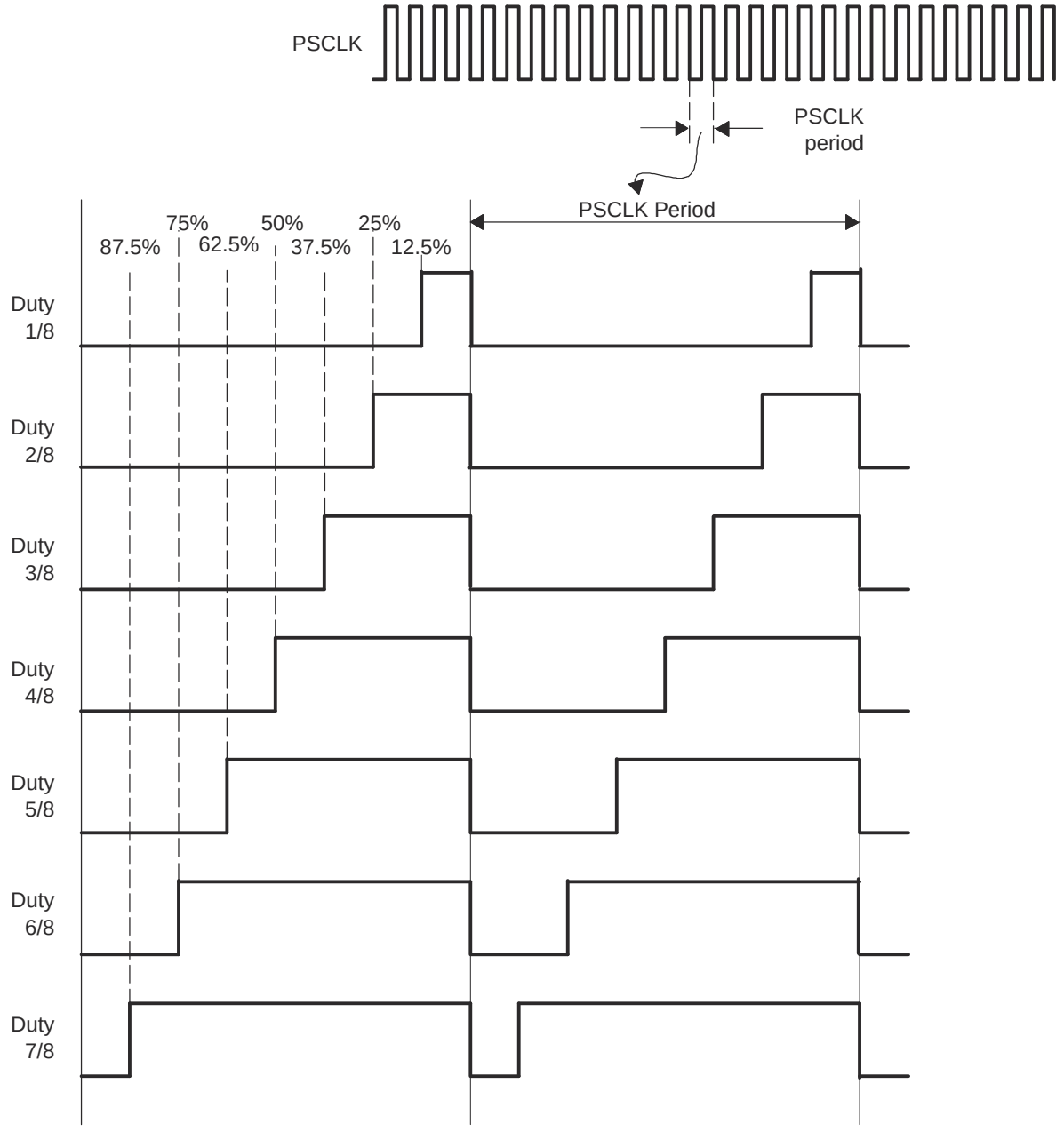


Figure 3-35. PWM-Chopper Submodule Waveforms Showing the Pulse Width (Duty Cycle) Control of Sustaining Pulses

3.2.7 Trip-Zone (TZ) Submodule

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

Each ePWM module is connected to six $\overline{TZn}$ signals ($\overline{TZ1}$ to $\overline{TZ6}$). $\overline{TZ1}$ to $\overline{TZ3}$ are sourced from the GPIO mux. $\overline{TZ4}$ is sourced from an inverted EQEP1ERR signal on those devices with an EQEP1 module. $\overline{TZ5}$ is connected to the system clock fail logic, and $\overline{TZ6}$ is sourced from the EMUSTOP output from the CPU. These signals indicate external fault or trip conditions, and the ePWM outputs can be programmed to respond accordingly when faults occur.

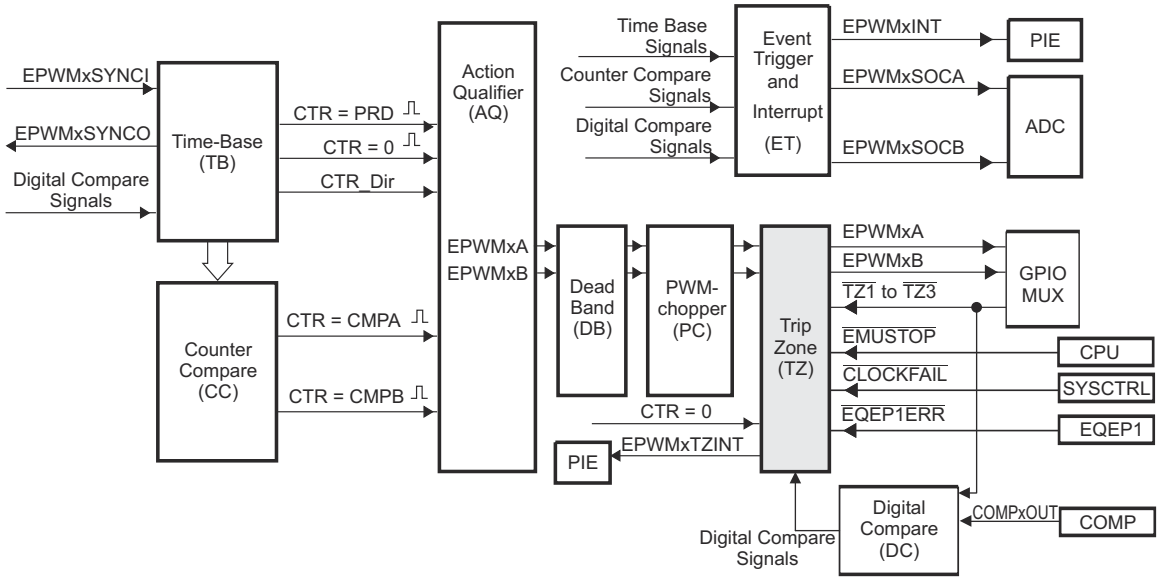


Figure 3-36. Trip-Zone Submodule

3.2.7.1 Purpose of the Trip-Zone Submodule

The key functions of the Trip-Zone submodule are:

- Trip inputs $\overline{TZ1}$ to $\overline{TZ6}$ can be flexibly mapped to any ePWM module.
- Upon a fault condition, outputs EPWMxA and EPWMxB can be forced to one of the following:
  - High
  - Low
  - High-impedance
  - No action taken
- Support for one-shot trip (OSHT) for major short circuits or over-current conditions.
- Support for cycle-by-cycle tripping (CBC) for current limiting operation.
- Support for digital compare tripping (DC) based on state of on-chip analog comparator module outputs and/or $\overline{TZ1}$ to $\overline{TZ3}$ signals.
- Each trip-zone input and digital compare (DC) submodule DCAEVT1/2 or DCBEVT1/2 force event can be allocated to either one-shot or cycle-by-cycle operation.
- Interrupt generation is possible on any trip-zone input.
- Software-forced tripping is also supported.
- The trip-zone submodule can be fully bypassed if it is not required.

3.2.7.2 Controlling and Monitoring the Trip-Zone Submodule

The trip-zone submodule operation is controlled and monitored through the following registers:

Table 3-17. Trip-Zone Submodule Registers

Register	Address Offset	Shadowed	Description (1)	Bit Description
TZSEL	0x0012	No	Trip-Zone Select Register	Section 3.4.5.1
TZDCSEL	0x0013	No	Trip-zone Digital Compare Select Register (2)	Section 3.4.5.2
TZCTL	0x0014	No	Trip-Zone Control Register	Section 3.4.5.3
TZEINT	0x0015	No	Trip-Zone Enable Interrupt Register	Section 3.4.5.4
TZFLG	0x0016	No	Trip-Zone Flag Register	Section 3.4.5.5
TZCLR	0x0017	No	Trip-Zone Clear Register	Section 3.4.5.6
TZFRC	0x0018	No	Trip-Zone Force Register	Section 3.4.5.7

(1) All trip-zone registers are EALLOW protected and can be modified only after executing the EALLOW instruction. For more information, see the *System Control and Interrupts* chapter.
(2) This register is discussed in more detail in Section 3.2.9.

3.2.7.3 Operational Highlights for the Trip-Zone Submodule

The following sections describe the operational highlights and configuration options for the trip-zone submodule.

The trip-zone signals $\overline{TZ1}$ to $\overline{TZ6}$ (also collectively referred to as $\overline{TZn}$) are active low input signals. When one of these signals goes low, or when a DCAEVT1/2 or DCBEVT1/2 force happens based on the TZDCSEL register event selection, it indicates that a trip event has occurred. Each ePWM module can be individually configured to ignore or use each of the trip-zone signals or DC events. Which trip-zone signals or DC events are used by a particular ePWM module is determined by the TZSEL register for that specific ePWM module. The trip-zone signals may or may not be synchronized to the system clock (SYSCLKOUT) and digitally filtered within the GPIO MUX block. A minimum of 3*TBCLK low pulse width on $\overline{TZn}$ inputs is sufficient to trigger a fault condition on the ePWM module. If the pulse width is less than this, the trip condition may not be latched by CBC or OST latches. The asynchronous trip makes sure that if clocks are missing for any reason, the outputs can still be tripped by a valid event present on $\overline{TZn}$ inputs . The GPIOs or peripherals must be appropriately configured. For more information, see the *System Control and Interrupts* chapter.

Each $\overline{TZn}$ input can be individually configured to provide either a cycle-by-cycle or one-shot trip event for an ePWM module. DCAEVT1 and DCBEVT1 events can be configured to directly trip an ePWM module or provide a one-shot trip event to the module. Likewise, DCAVET2 and DCBEVT2 events can also be configured to directly trip an ePWM module or provide a cycle-by-cycle trip event to the module. This configuration is determined by the TZSEL[DCAEVT1/2], TZSEL[DCBEVT1/2], TZSEL[CBCn], and TZSEL[OSHTn] control bits (where n corresponds to the trip input) respectively.

- **Cycle-by-Cycle (CBC):** When a cycle-by-cycle trip event occurs, the action specified in the TZCTL[TZA] and TZCTL[TZB] bits is carried out immediately on the EPWMxA and/or EPWMxB output. Table 3-18 lists the possible actions. In addition, the cycle-by-cycle trip event flag (TZFLG[CBC]) is set and a EPWMx_TZINT interrupt is generated if it is enabled in the TZEINT register and PIE peripheral.

  If the CBC interrupt is enabled via the TZEINT register, and DCAEVT2 or DCBEVT2 are selected as CBC trip sources via the TZSEL register, it is not necessary to also enable the DCAEVT2 or DCBEVT2 interrupts in the TZEINT register, as the DC events trigger interrupts through the CBC mechanism.

  The specified condition on the inputs is automatically cleared when the ePWM time-base counter reaches zero (TBCTR = 0x0000) if the trip event is no longer present. Therefore, in this mode, the trip event is cleared or reset every PWM cycle. The TZFLG[CBC] flag bit will remain set until it is manually cleared by writing to the TZCLR[CBC] bit. If the cycle-by-cycle trip event is still present when the TZFLG[CBC] bit is cleared, then it will again be immediately set.

- **One-Shot (OSHT):** When a one-shot trip event occurs, the action specified in the TZCTL[TZA] and TZCTL[TZB] bits is carried out immediately on the EPWMxA and/or EPWMxB output. Table 3-18 lists the possible actions. In addition, the one-shot trip event flag (TZFLG[OST]) is set and a EPWMx_TZINT interrupt is generated if it is enabled in the TZEINT register and PIE peripheral. The one-shot trip condition must be cleared manually by writing to the TZCLR[OST] bit.

  If the one-shot interrupt is enabled via the TZEINT register, and DCAEVT1 or DCBEVT1 are selected as OSHT trip sources via the TZSEL register, it is not necessary to also enable the DCAEVT1 or DCBEVT1 interrupts in the TZEINT register, as the DC events trigger interrupts through the OSHT mechanism.
- **Digital Compare Events (DCAEVT1/2 and DCBEVT1/2):** A digital compare DCAEVT1/2 or DCBEVT1/2 event is generated based on a combination of the DCAH/DCAL and DCBH/DCBL signals as selected by the TZDCSEL register. The signals which source the DCAH/DCAL and DCBH/DCBL signals are selected via the DCTRIPSEL register and can be either trip zone input pins or analog comparator COMPxOUT signals. For more information on the digital compare submodule signals, see Section 3.2.9.

  When a digital compare event occurs, the action specified in the TZCTL[DCAEVT1/2] and TZCTL[DCBEVT1/2] bits is carried out immediately on the EPWMxA and/or EPWMxB output. Table 3-18 lists the possible actions. In addition, the relevant DC trip event flag (TZFLG[DCAEVT1/2] / TZFLG[DCBEVT1/2]) is set and a EPWMx_TZINT interrupt is generated if it is enabled in the TZEINT register and PIE peripheral.

  The specified condition on the pins is automatically cleared when the DC trip event is no longer present. The TZFLG[DCAEVT1/2] or TZFLG[DCBEVT1/2] flag bit will remain set until it is manually cleared by writing to the TZCLR[DCAEVT1/2] or TZCLR[DCBEVT1/2] bit. If the DC trip event is still present when the TZFLG[DCAEVT1/2] or TZFLG[DCBEVT1/2] flag is cleared, then it will again be immediately set.

The action taken when a trip event occurs can be configured individually for each of the ePWM output pins by way of the TZCTL register bit fields. One of four possible actions, shown in Table 3-18, can be taken on a trip event.

Table 3-18. Possible Actions On a Trip Event

TZCTL Register Bit Settings	EPWMxA and/or EPWMxB	Comment
0,0	High-Impedance	Tripped
0,1	Force to High State	Tripped
1,0	Force to Low State	Tripped
1,1	No Change	Do Nothing. No change is made to the output.

Example 3-7. Trip-Zone Configurations

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

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

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

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

3.2.7.4 Generating Trip Event Interrupts

Figure 3-37 and Figure 3-38 illustrate the trip-zone submodule control and interrupt logic, respectively. DCAEVT1/2 and DCBEVT1/2 signals are described in further detail in Section 3.2.9.

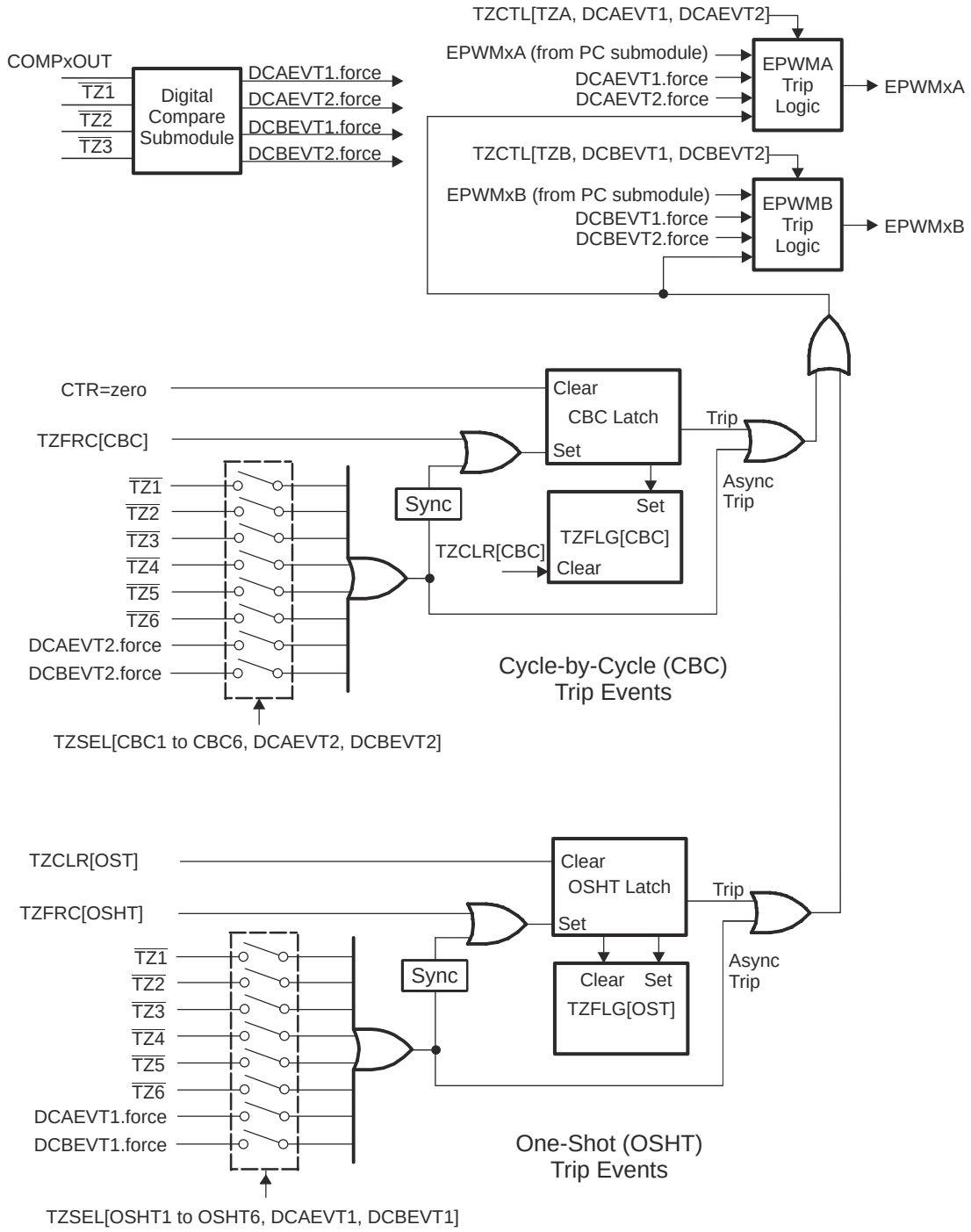


Figure 3-37. Trip-Zone Submodule Mode Control Logic

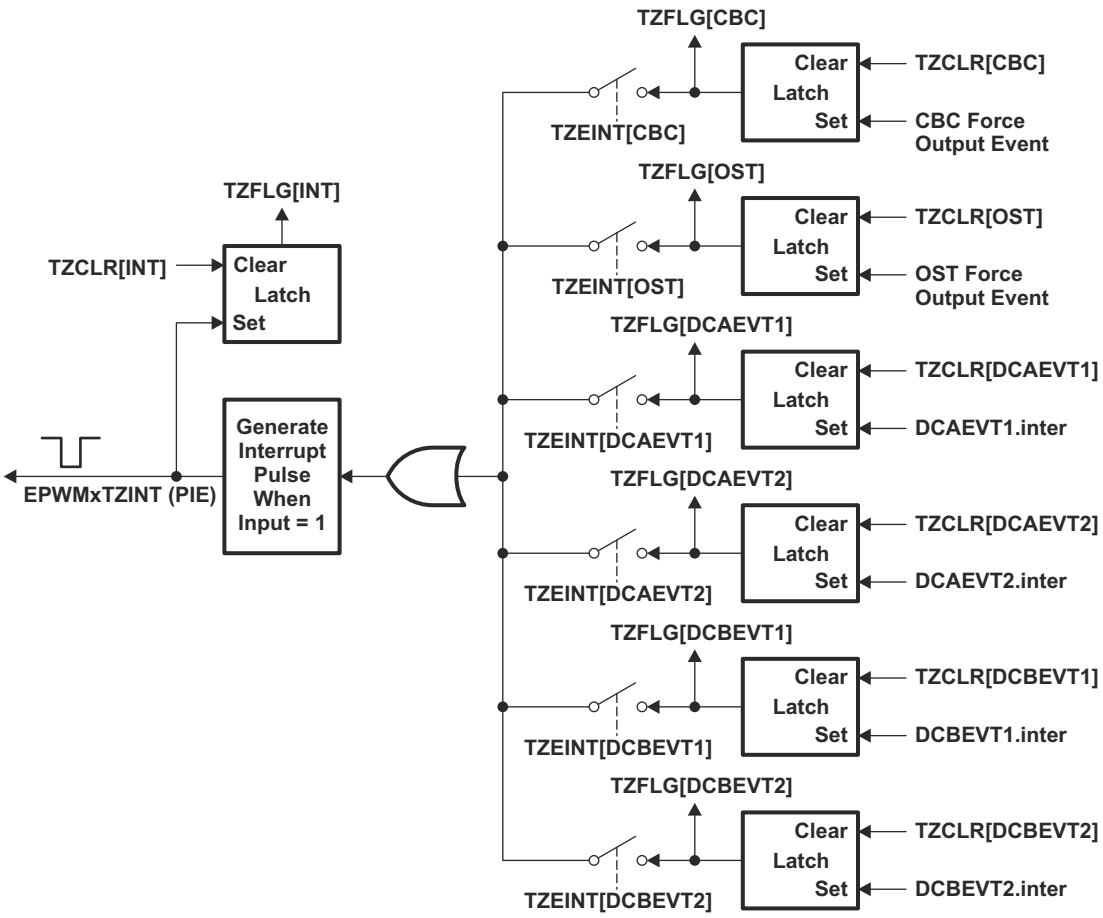


Figure 3-38. Trip-Zone Submodule Interrupt Logic

3.2.8 Event-Trigger (ET) Submodule

The key functions of the event-trigger submodule are:

- Receives event inputs generated by the time-base, counter-compare, and digital-compare submodules
- Uses the time-base direction information for up/down event qualification
- Uses prescaling logic to issue interrupt requests and ADC start of conversion at:
  - Every event
  - Every second event
  - Every third event
- Provides full visibility of event generation via event counters and flags
- Allows software forcing of Interrupts and ADC start of conversion

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

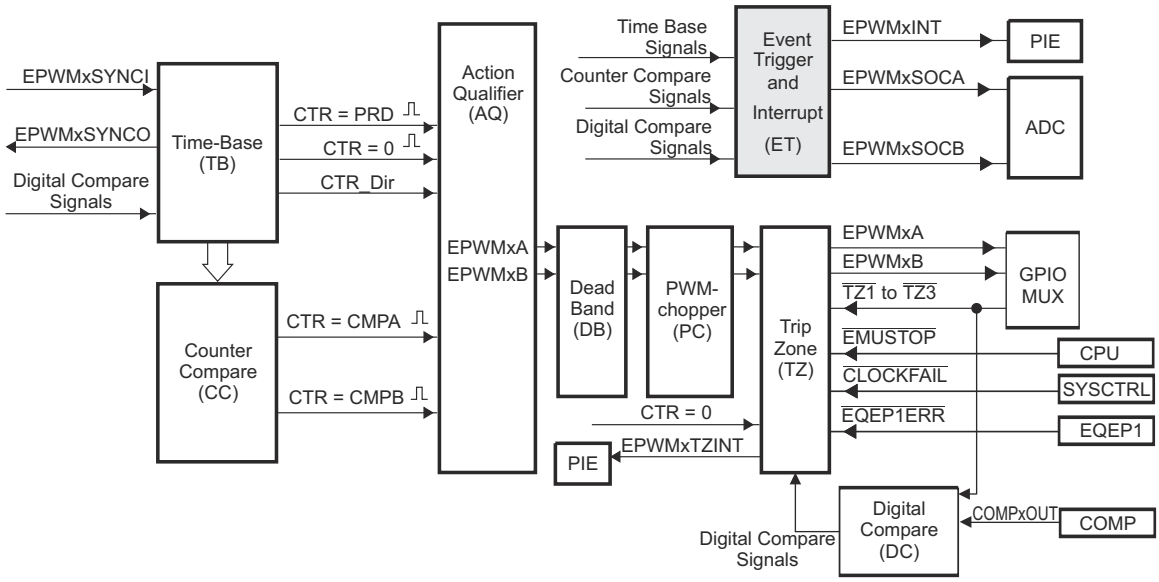


Figure 3-39. Event-Trigger Submodule

3.2.8.1 Purpose of the Event-Trigger Submodule

The following sections describe the event-trigger submodule.

Each ePWM module has one interrupt request line connected to the PIE and two start of conversion signals connected to the ADC module. As shown in Figure 3-40, ADC start of conversion for all ePWM modules are connected to individual ADC trigger inputs to the ADC, and hence multiple modules can initiate an ADC start of conversion via the ADC trigger inputs.

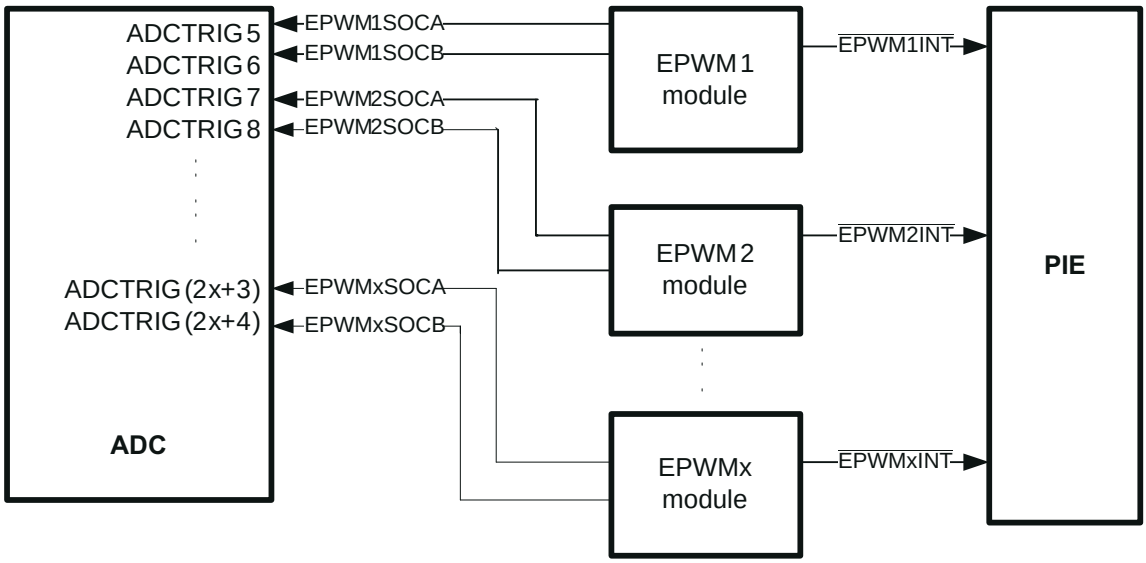


Figure 3-40. Event-Trigger Submodule Inter-Connectivity of ADC Start of Conversion

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

- Every event
- Every second event
- Every third event

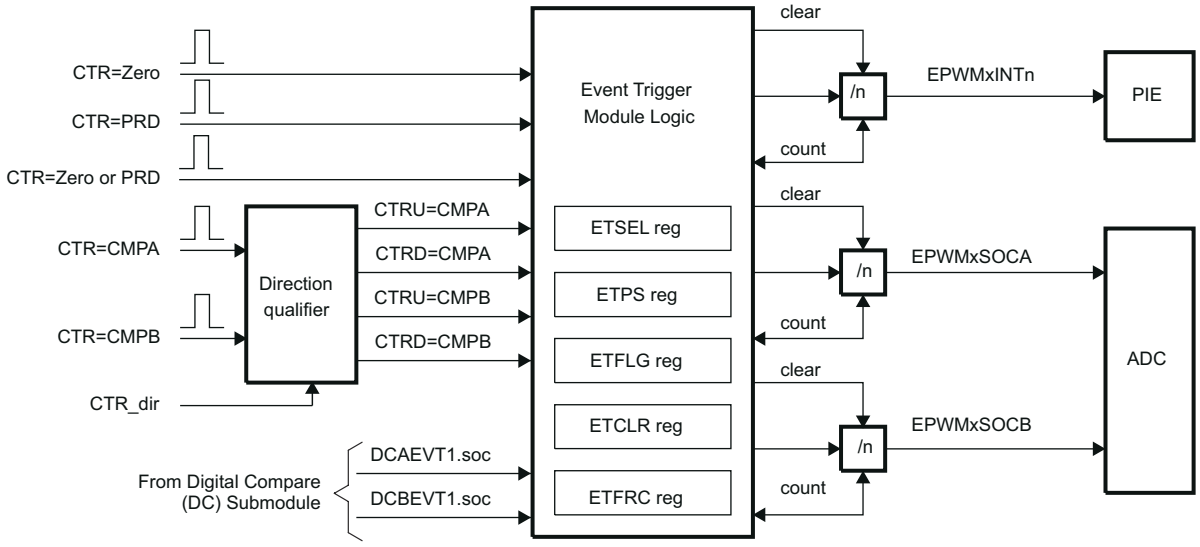


Figure 3-41. Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs

3.2.8.2 Controlling and Monitoring the Event-Trigger Submodule

The key registers used to configure the event-trigger submodule are shown in Table 3-19.

Table 3-19. Event-Trigger Submodule Registers

Register	Address Offset	Shadowed	Description	Bit Description
ETSEL	0x0019	No	Event-trigger Selection Register	Section 3.4.6.1
ETPS	0x001A	No	Event-trigger Prescale Register	Section 3.4.6.2
ETFLG	0x001B	No	Event-trigger Flag Register	Section 3.4.6.3
ETCLR	0x001C	No	Event-trigger Clear Register	Section 3.4.6.4
ETFRC	0x001D	No	Event-trigger Force Register	Section 3.4.6.5

- ETSEL—This selects which of the possible events will trigger an interrupt or start an ADC conversion
- ETPS—This programs the event prescaling options mentioned above.
- ETFLG—These are flag bits indicating status of the selected and prescaled events.
- ETCLR—These bits allow you to clear the flag bits in the ETFLG register via software.
- ETFRC—These bits allow software forcing of an event. Useful for debugging or s/w intervention.

3.2.8.3 Operational Highlights for the Event-Trigger Submodule

A more detailed look at how the various register bits interact with the Interrupt and ADC start of conversion logic are shown in Figure 3-42, Figure 3-43, and Figure 3-44.

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

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

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

- Time-base counter equal to zero (TBCTR = 0x0000).
- Time-base counter equal to period (TBCTR = TBPRD).
- Time-base counter equal to zero or period (TBCTR = 0x0000 || TBCTR = TBPRD)
- Time-base counter equal to the compare A register (CMPA) when the timer is incrementing.
- Time-base counter equal to the compare A register (CMPA) when the timer is decrementing.
- Time-base counter equal to the compare B register (CMPB) when the timer is incrementing.
- Time-base counter equal to the compare B register (CMPB) when the timer is decrementing.

The number of events that have occurred can be read from the interrupt event counter (ETPS[INTCNT]) register bits. That is, when the specified event occurs the ETPS[INTCNT] bits are incremented until they reach the value specified by ETPS[INTPRD]. When ETPS[INTCNT] = ETPS[INTPRD] the counter stops counting and its output is set. The counter is only cleared when an interrupt is sent to the PIE.

When ETPS[INTCNT] reaches ETPS[INTPRD] the following behaviors will occur:

- If interrupts are enabled, ETSEL[INTEN] = 1 and the interrupt flag is clear, ETFLG[INT] = 0, then an interrupt pulse is generated and the interrupt flag is set, ETFLG[INT] = 1, and the event counter is cleared ETPS[INTCNT] = 0. The counter will begin counting events again.
- If interrupts are disabled, ETSEL[INTEN] = 0, or the interrupt flag is set, ETFLG[INT] = 1, the counter stops counting events when it reaches the period value ETPS[INTCNT] = ETPS[INTPRD].
- If interrupts are enabled, but the interrupt flag is already set, then the counter will hold its output high until the ETFLG[INT] flag is cleared. This allows for one interrupt to be pending while one is serviced.

When writing INTPRD values the following occur:

- Writing to the INTPRD bits will automatically clear the counter INTCNT = 0 and the counter output will be reset (so no interrupts are generated).
- Writing an INTPRD value that is GREATER or equal to the current counter value will reset the INTCNT = 0.
- Writing an INTPRD value that is equal to the current counter value will trigger an interrupt if it is enabled and the status flag is cleared (and INTCNT will also be cleared to 0)
- Writing an INTPRD value that is LESS than the current counter value will result in undefined behavior (that is, INTCNT stops counting because INTPRD is below INTCNT, and interrupt will never fire).
- Writing a 1 to the ETFRC[INT] bit will increment the event counter INTCNT. The counter will behave as described above when INTCNT = INTPRD.
- When INTPRD = 0, the counter is disabled and hence no events will be detected and the ETFRC[INT] bit is also ignored.

The above definition means that you can generate an interrupt on every event, on every second event, or on every third event. An interrupt cannot be generated on every fourth or more events.

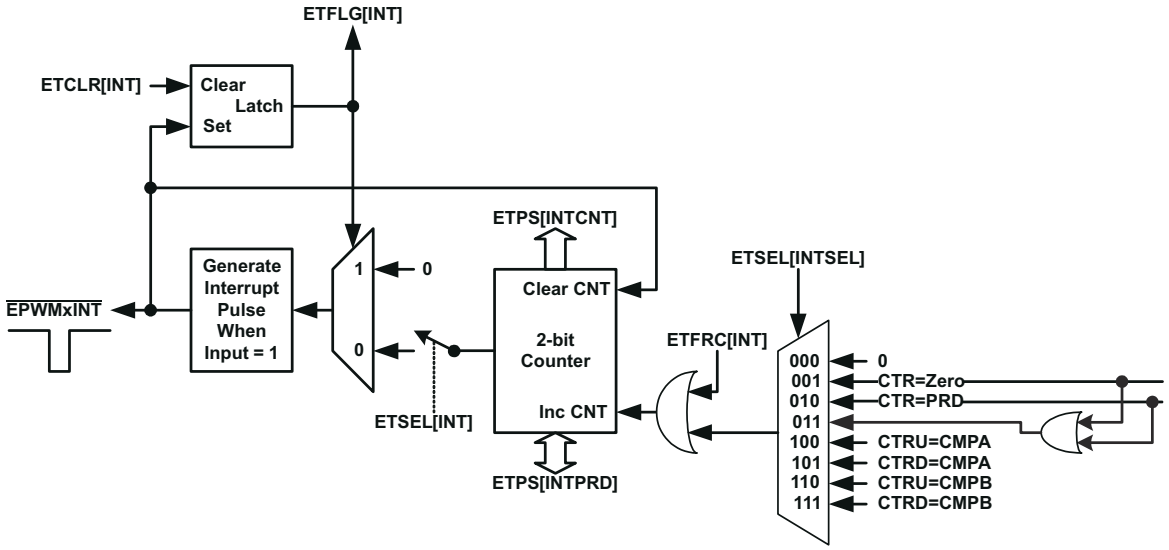
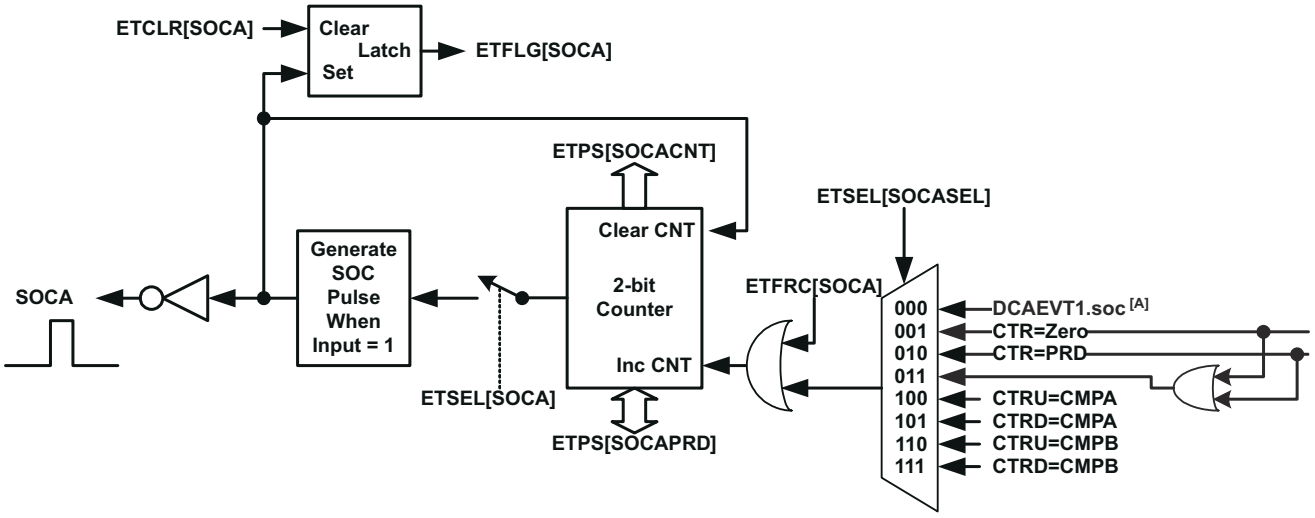


Figure 3-42. Event-Trigger Interrupt Generator

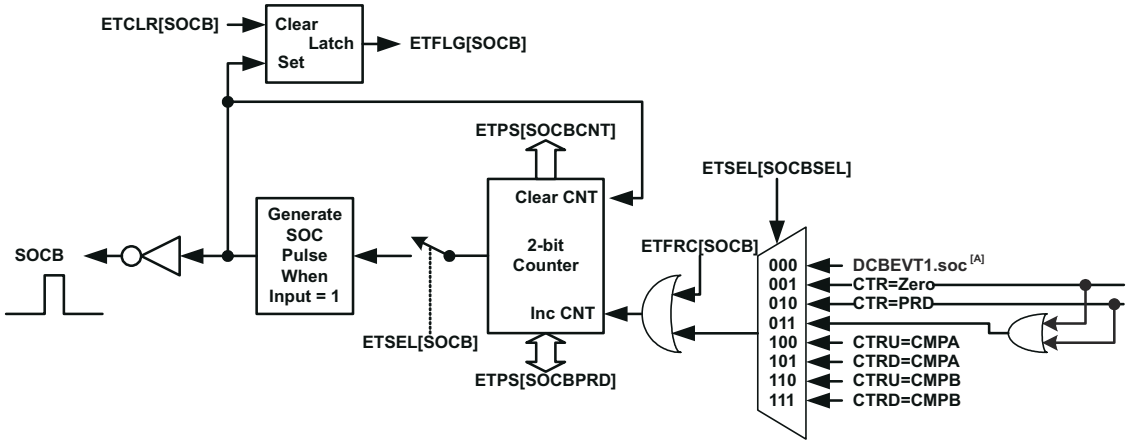
Figure 3-43 shows the operation of the event-trigger's start-of-conversion-A (SOCA) pulse generator. The ETPS[SOCACNT] counter and ETPS[SOCAPRD] period values behave similarly to the interrupt generator except that the pulses are continuously generated. That is, the pulse flag ETFLG[SOCA] is latched when a pulse is generated, but it does not stop further pulse generation. The enable/disable bit ETSEL[SOCAEN] stops pulse generation, but input events can still be counted until the period value is reached as with the interrupt generation logic. The event that will trigger an SOCA and SOCB pulse can be configured separately in the ETSEL[SOCASEL] and ETSEL[SOCBSEL] bits. The possible events are the same events that can be specified for the interrupt generation logic with the addition of the DCAEVT1.soc and DCBEVT1.soc event signals from the digital compare (DC) submodule.



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

Figure 3-43. Event-Trigger SOCA Pulse Generator

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



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

Figure 3-44. Event-Trigger SOCB Pulse Generator

3.2.9 Digital Compare (DC) Submodule

Figure 3-45 illustrates where the digital compare (DC) submodule signals interface to other submodules in the ePWM system.

The digital compare (DC) submodule compares signals external to the ePWM module (for instance, COMPxOUT signals from the analog comparators) to directly generate PWM events/actions which then feed to the event-trigger, trip-zone, and time-base submodules. Additionally, blanking window functionality is supported to filter noise or unwanted pulses from the DC event signals.

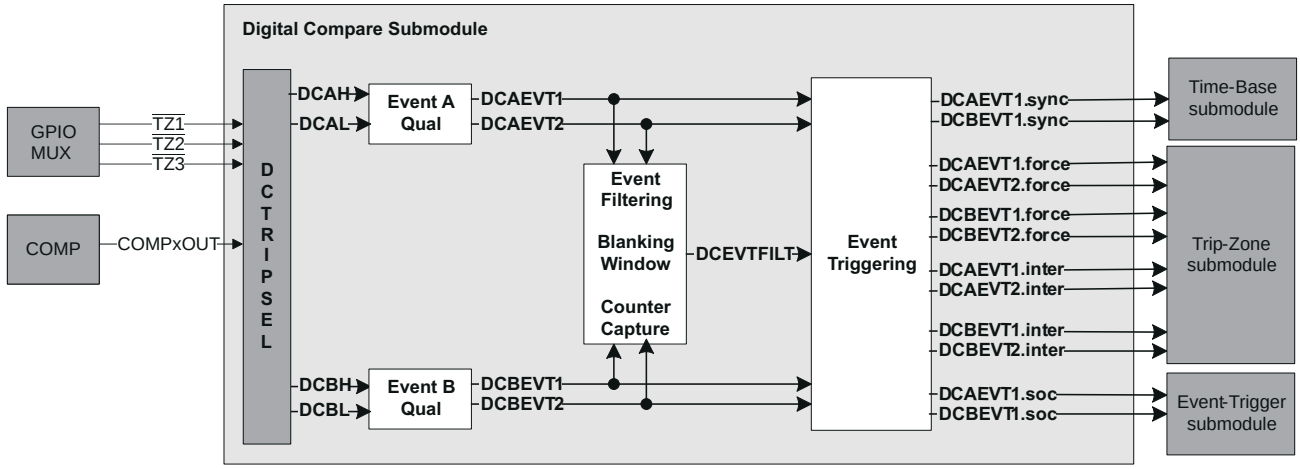


Figure 3-45. Digital-Compare Submodule High-Level Block Diagram

3.2.9.1 Purpose of the Digital Compare Submodule

The key functions of the digital compare submodule are:

- Analog Comparator (COMP) module outputs and $\overline{TZ1}$, $\overline{TZ2}$, and $\overline{TZ3}$ inputs generate Digital Compare A High/Low (DCAH, DCAL) and Digital Compare B High/Low (DCBH, DCBL) signals.
- DCAH/L and DCBH/L signals trigger events which can then either be filtered or fed directly to the trip-zone, event-trigger, and time-base submodules to:
  - generate a trip zone interrupt
  - generate an ADC start of conversion
  - force an event
  - generate a synchronization event for synchronizing the ePWM module TBCTR.
- Event filtering (blanking window logic) can optionally blank the input signal to remove noise.

3.2.9.2 Controlling and Monitoring the Digital Compare Submodule

The digital compare submodule operation is controlled and monitored through the following registers.

Table 3-20. Digital Compare Submodule Registers

Register	Address Offset	Shadowed	Description	Bit Description
TZDCSEL(1) (2)	0x13	No	Trip Zone Digital Compare Select Register	Section 3.4.5.2
DCTRIPSEL(2)	0x30	No	Digital Compare Trip Select Register	Section 3.4.8.1
DCACTL(2)	0x31	No	Digital Compare A Control Register	Section 3.4.8.2
DCBCTL(2)	0x32	No	Digital Compare B Control Register	Section 3.4.8.3
DCFCTL(2)	0x33	No	Digital Compare Filter Control Register	Section 3.4.8.4
DCCAPCTL(2)	0x34	No	Digital Compare Capture Control Register	Section 3.4.8.5
DCFOFFSET	0x35	Writes	Digital Compare Filter Offset Register	Section 3.4.8.6
DCFOFFSETCNT	0x36	No	Digital Compare Filter Offset Counter Register	Section 3.4.8.7
DCFWINDOW	0x37	No	Digital Compare Filter Window Register	Section 3.4.8.8
DCFWINDOWCNT	0x38	No	Digital Compare Filter Window Counter Register	Section 3.4.8.9
DCCAP	0x39	Yes	Digital Compare Counter Capture Register	Section 3.4.8.10

(1) The TZDCSEL register is part of the trip-zone submodule but is shown here because of its functional significance to the digital compare submodule.
(2) These registers are EALLOW protected and can be modified only after executing the EALLOW instruction. For more information, see the *System Control and Interrupts* chapter.

3.2.9.3 Operation Highlights of the Digital Compare Submodule

The following sections describe the operational highlights and configuration options for the digital compare submodule.

3.2.9.3.1 Digital Compare Events

As shown in Figure 3-45 earlier in this section, trip zone inputs ($\overline{TZ1}$, $\overline{TZ2}$, and $\overline{TZ3}$) and COMPxOUT signals from the analog comparator (COMP) module can be selected via the DCTRIPSEL bits to generate the Digital Compare A High and Low (DCAH/L) and Digital Compare B High and Low (DCBH/L) signals. Then, the configuration of the TZDCSEL register qualifies the actions on the selected DCAH/L and DCBH/L signals, which generate the DCAEVT1/2 and DCBEVT1/2 events (Event Qualification A and B).

Note

The $\overline{TZn}$ signals, when used as a DCEVT tripping functions, are treated as a normal input signal and can be defined to be active high or active low inputs. EPWM outputs are asynchronously tripped when either the $\overline{TZn}$, DCAEVTx.force, or DCBEVTx.force signals are active. For the condition to remain latched, a minimum of 3*TBCLK sync pulse width is required. If pulse width is < 3*TBCLK sync pulse width, the trip condition may or may not get latched by CBC or OST latches.

The DCAEVT1/2 and DCBEVT1/2 events can then be filtered to provide a filtered version of the event signals (DCEVTFILT) or the filtering can be bypassed. Filtering is discussed further in Section 3.2.9.3.2. Either the DCAEVT1/2 and DCBEVT1/2 event signals or the filtered DCEVTFILT event signals can generate a force to the trip zone module, a TZ interrupt, an ADC SOC, or a PWM sync signal.

- **force signal:** DCAEVT1/2.force signals force trip zone conditions which either directly influence the output on the EPWMxA pin (via TZCTL[DCAEVT1 or DCAEVT2] configurations) or, if the DCAEVT1/2 signals are selected as one-shot or cycle-by-cycle trip sources (via the TZSEL register), the DCAEVT1/2.force signals can effect the trip action via the TZCTL[TZA] configuration. The DCBEVT1/2.force signals behaves similarly, but affect the EPWMxB output pin instead of the EPWMxA output pin.

  The priority of conflicting actions on the TZCTL register is as follows (highest priority overrides lower priority):
  – Output EPWMxA: TZA (highest) -> DCAEVT1 -> DCAEVT2 (lowest)
  – Output EPWMxB: TZB (highest) -> DCBEVT1 -> DCBEVT2 (lowest)

- **interrupt signal:** DCAEVT1/2.interrupt signals generate trip zone interrupts to the PIE. To enable the interrupt, the user must set the DCAEVT1, DCAEVT2, DCBEVT1, or DCBEVT2 bits in the TZEINT register. Once one of these events occurs, an EPWMxTZINT interrupt is triggered, and the corresponding bit in the TZCLR register must be set in order to clear the interrupt.
- **soc signal:** The DCAEVT1.soc signal interfaces with the event-trigger submodule and can be selected as an event which generates an ADC start-of-conversion-A (SOCA) pulse via the ETSEL[SOCASEL] bit. Likewise, the DCBEVT1.soc signal can be selected as an event which generates an ADC start-of-conversion-B (SOCB) pulse via the ETSEL[SOCBSEL] bit.
- **sync signal:** The DCAEVT1.sync and DCBEVT1.sync events are ORed with the EPWMxSYNCI input signal and the TBCTL[SWFSYNC] signal to generate a synchronization pulse to the time-base counter.

Figure 3-46 and Figure 3-47 show how the DCAEVT1, DCAEVT2, and DCEVTFILT signals are processed to generate the digital compare A event force, interrupt, soc, and sync signals.

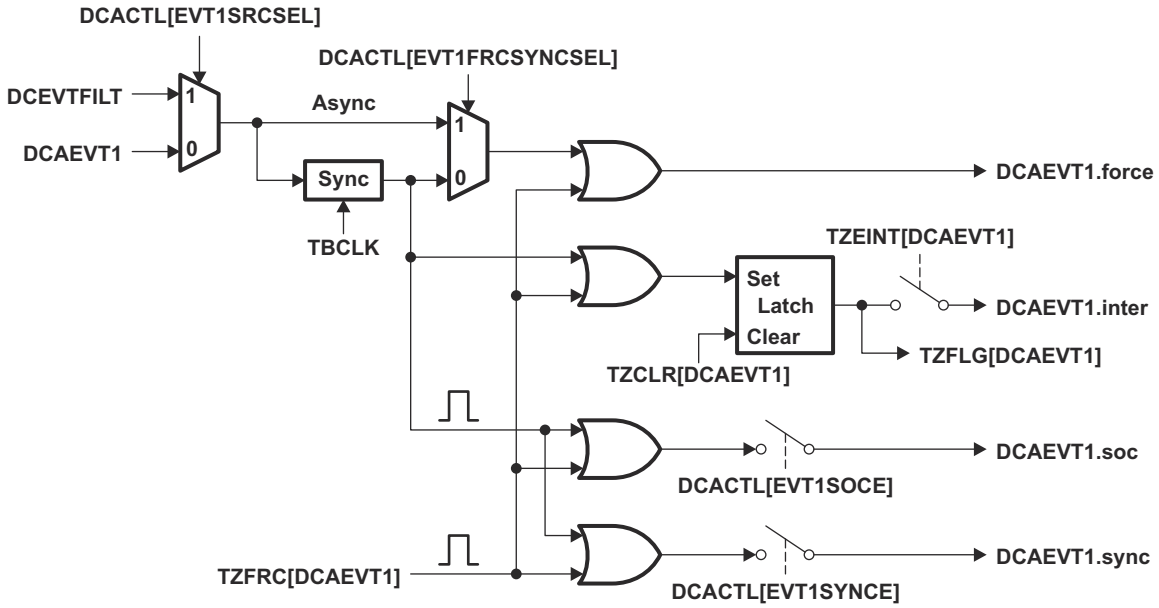


Figure 3-46. DCAEVT1 Event Triggering

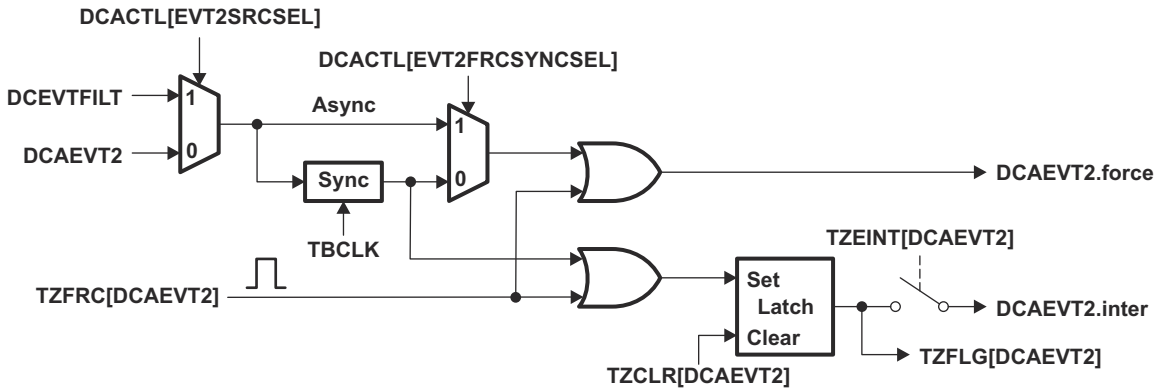


Figure 3-47. DCAEVT2 Event Triggering

Figure 3-48 and Figure 3-49 show how the DCBEVT1, DCBEVT2, and DCEVTFILT signals are processed to generate the digital compare B event force, interrupt, soc and sync signals.

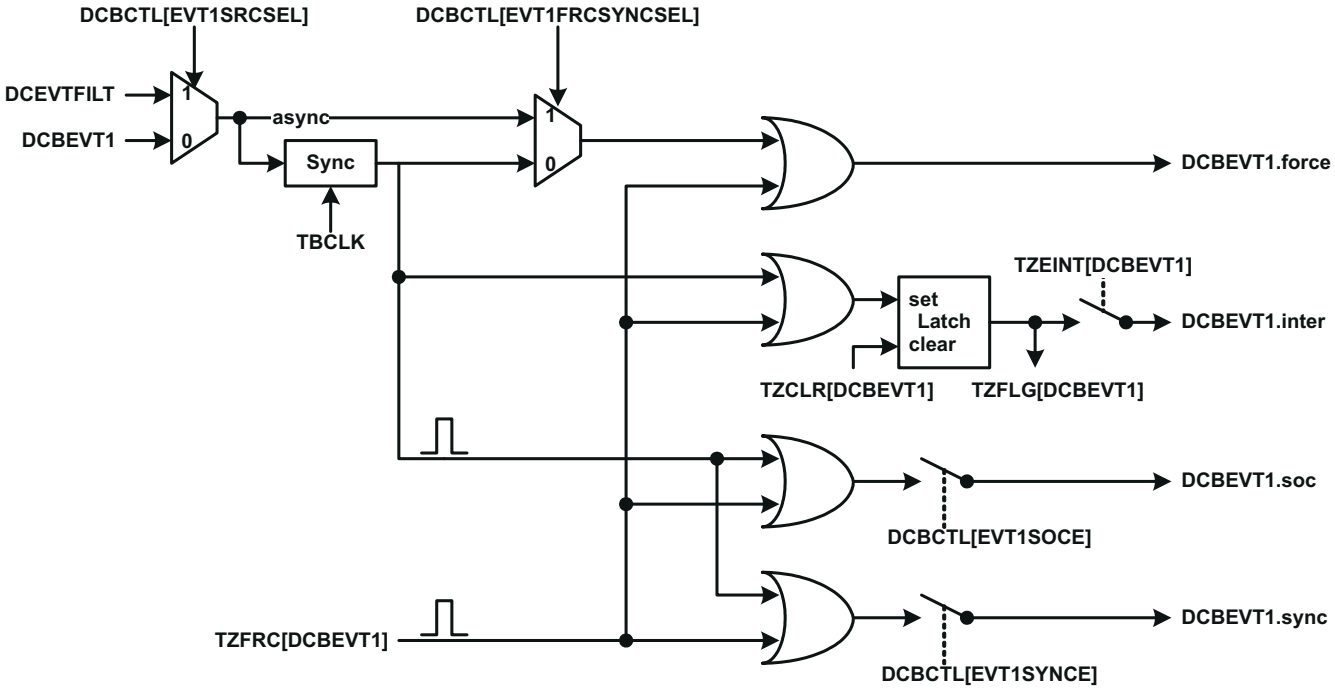


Figure 3-48. DCBEVT1 Event Triggering

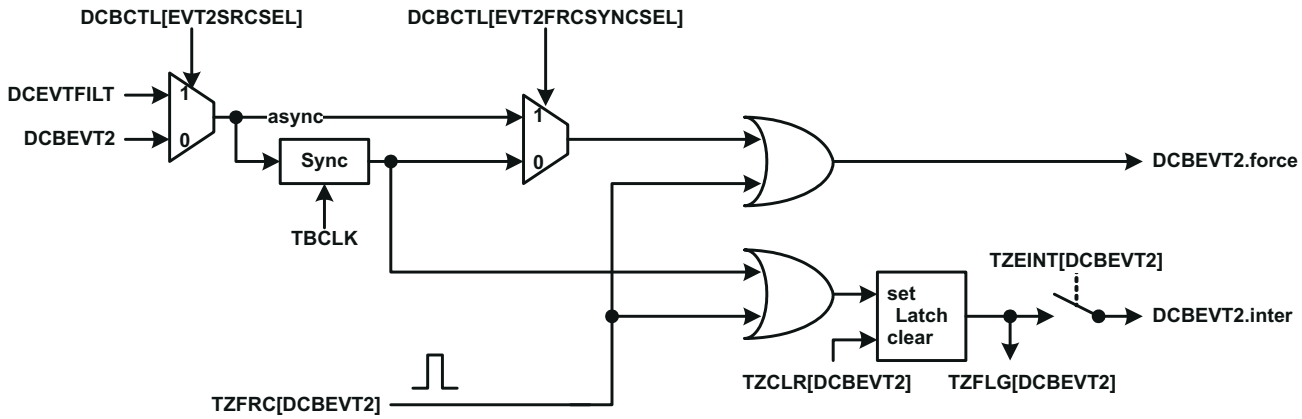


Figure 3-49. DCBEVT2 Event Triggering

3.2.9.3.2 Event Filtering

The DCAEVT1/2 and DCBEVT1/2 events can be filtered via event filtering logic to remove noise by optionally blanking events for a certain period of time. This is useful for cases where the analog comparator outputs may be selected to trigger DCAEVT1/2 and DCBEVT1/2 events, and the blanking logic is used to filter out potential noise on the signal prior to tripping the PWM outputs or generating an interrupt or ADC start-of-conversion. The event filtering can also capture the TBCTR value of the trip event. Figure 3-50 shows the details of the event filtering logic.

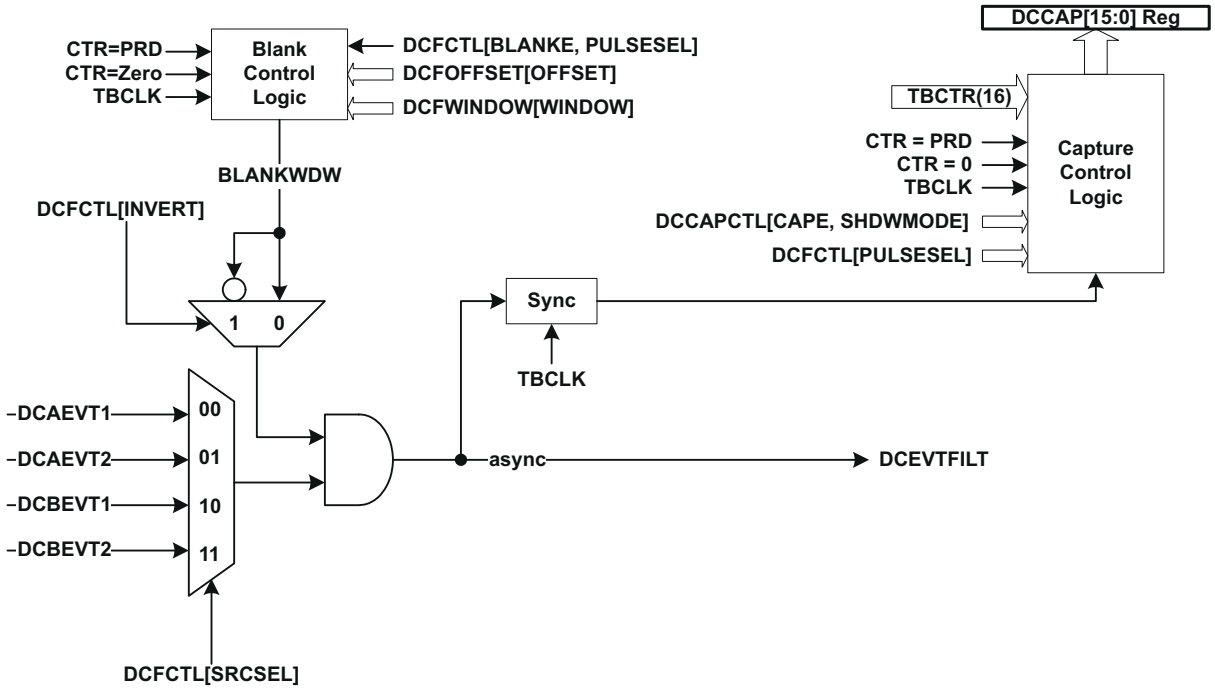


Figure 3-50. Event Filtering

If the blanking logic is enabled, one of the digital compare events – DCAEVT1, DCAEVT2, DCBEVT1, DCBEVT2 – is selected for filtering. The blanking window, which filters out all event occurrences on the signal while it is active, will be aligned to either a CTR = PRD pulse or a CTR = 0 pulse (configured by the DCFCTL[PULSESEL] bits). An offset value in TBCLK counts is programmed into the DCFOFFSET register, which determines at what point after the CTR = PRD or CTR = 0 pulse the blanking window starts. The duration of the blanking window, in number of TBCLK counts after the offset counter expires, is written to the DCFWINDOW register by the application. During the blanking window, all events are ignored. Before and after the blanking window ends, events can generate soc, sync, interrupt, and force signals as before.

Figure 3-51 illustrates several timing conditions for the offset and blanking window within an ePWM period. Notice that if the blanking window crosses the CTR = 0 or CTR = PRD boundary, the next window still starts at the same offset value after the CTR = 0 or CTR = PRD pulse.

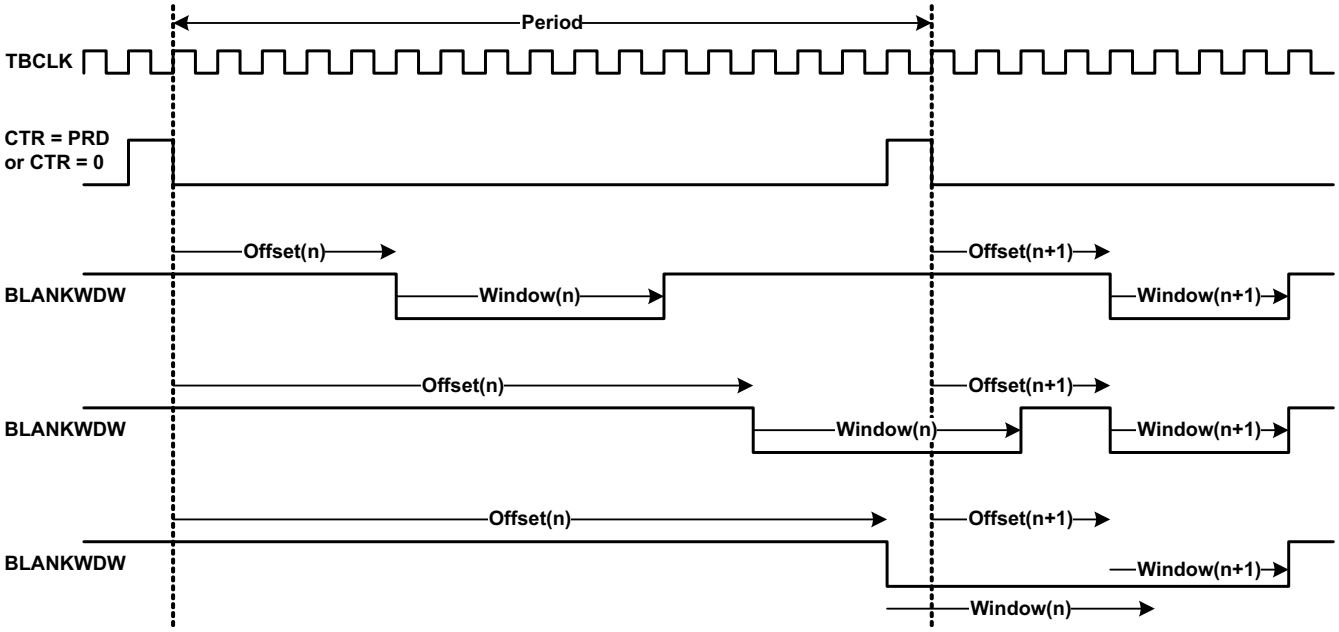


Figure 3-51. Blanking Window Timing Diagram

3.3 Applications to Power Topologies

An ePWM module has all the local resources necessary to operate completely as a standalone module or to operate in synchronization with other identical ePWM modules.

3.3.1 Overview of Multiple Modules

Previously all discussions have described the operation of a single module. To facilitate the understanding of multiple modules working together in a system, the ePWM module described in reference is represented by the more simplified block diagram shown in Figure 3-52. This simplified ePWM block shows only the key resources needed to explain how a multiswitch power topology is controlled with multiple ePWM modules working together.

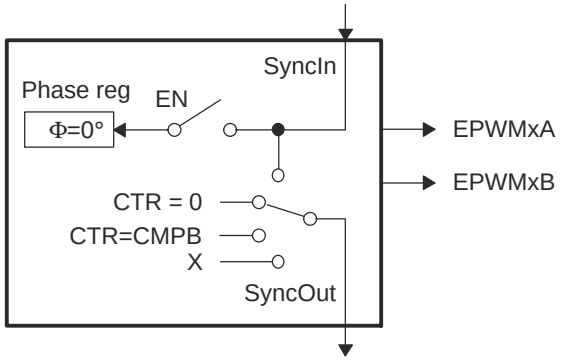


Figure 3-52. Simplified ePWM Module

3.3.2 Key Configuration Capabilities

The key configuration choices available to each module are as follows:

- Options for SyncIn
- Load own counter with phase register on an incoming sync strobe—enable (EN) switch closed
- Do nothing or ignore incoming sync strobe—enable switch open
- Sync flow-through - SyncOut connected to SyncIn
- Master mode, provides a sync at PWM boundaries—SyncOut connected to CTR = PRD
- Master mode, provides a sync at any programmable point in time—SyncOut connected to CTR = CMPB
- Module is in standalone mode and provides No sync to other modules—SyncOut connected to X (disabled)
- Options for SyncOut
  - Sync flow-through - SyncOut connected to SyncIn
  - Master mode, provides a sync at PWM boundaries—SyncOut connected to CTR = PRD
  - Master mode, provides a sync at any programmable point in time—SyncOut connected to CTR = CMPB
  - Module is in standalone mode and provides No sync to other modules—SyncOut connected to X (disabled)

For each choice of SyncOut, a module may also choose to load its own counter with a new phase value on a SyncIn strobe input or choose to ignore it, that is, with he enable switch. Although various combinations are possible, the two most common—master module and slave module modes—are shown in Figure 3-53.

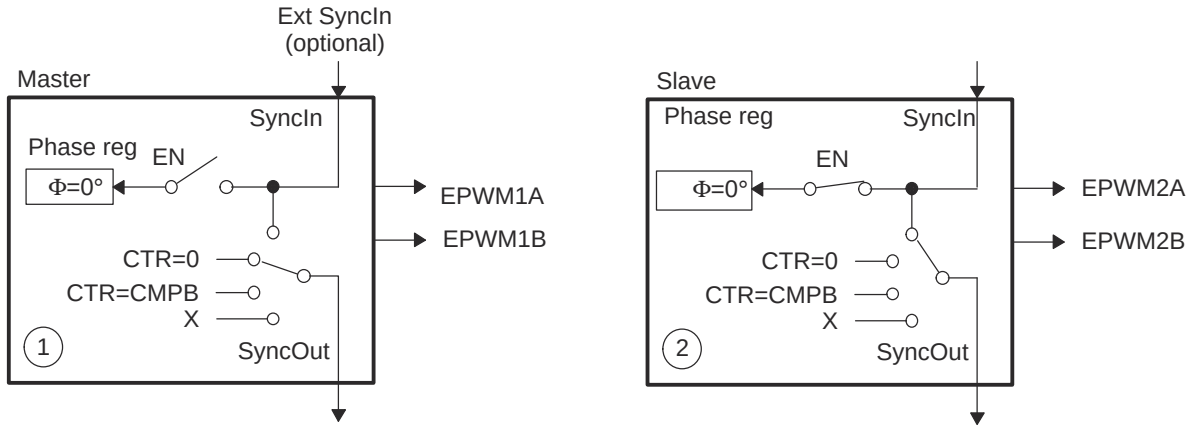
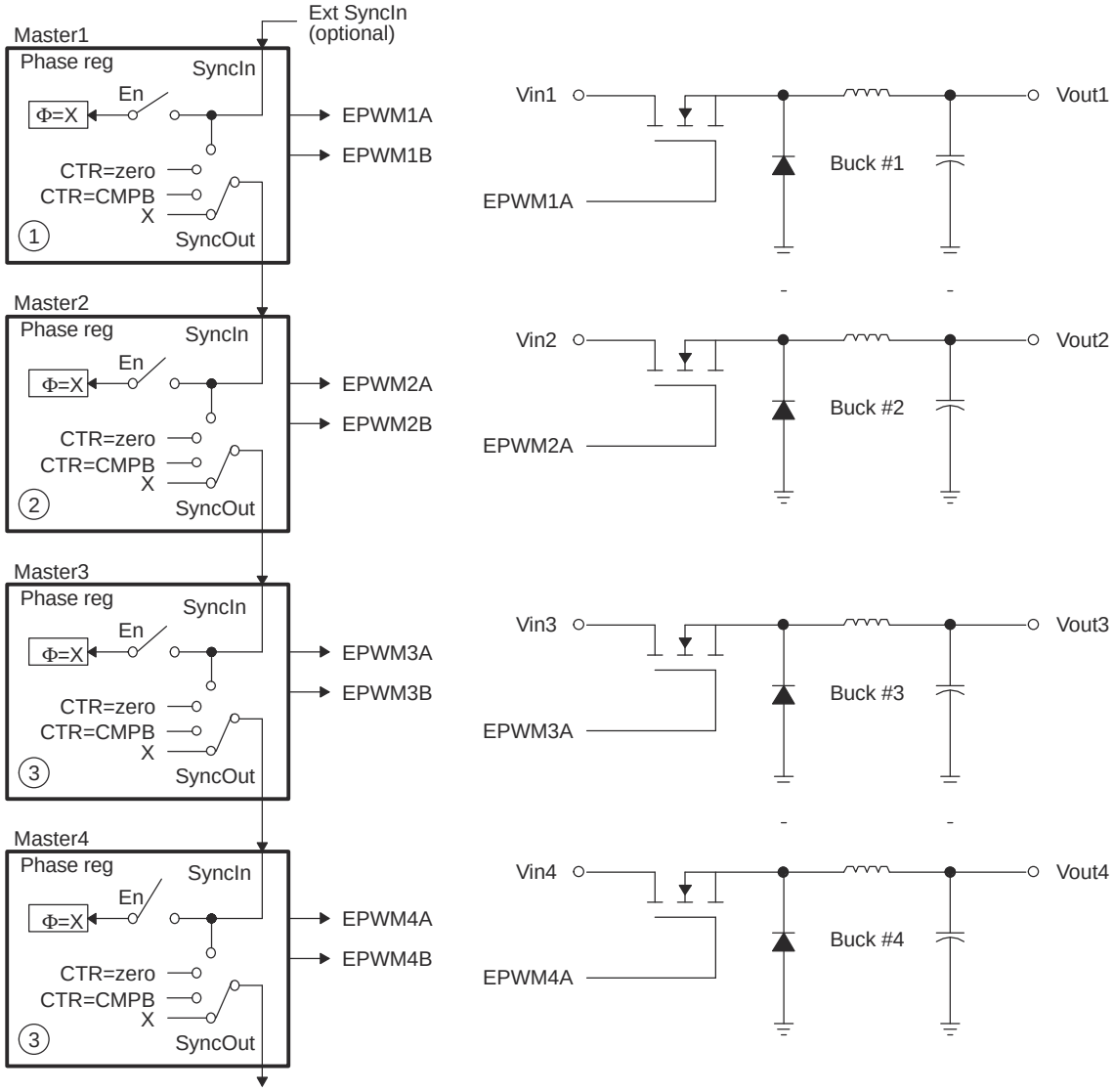


Figure 3-53. EPWM1 Configured as a Typical Master, EPWM2 Configured as a Slave

3.3.3 Controlling Multiple Buck Converters With Independent Frequencies

One of the simplest power converter topologies is the buck. A single ePWM module configured as a master can control two buck stages with the same PWM frequency. If independent frequency control is required for each buck converter, then one ePWM module must be allocated for each converter stage. Figure 3-54 shows four buck stages, each running at independent frequencies. In this case, all four ePWM modules are configured as masters and no synchronization is used. Figure 3-55 shows the waveforms generated by the setup shown in Figure 3-54; note that only three waveforms are shown, although there are four stages.



A. Θ = X indicates value in phase register is a "don't care"

Figure 3-54. Control of Four Buck Stages. Here $F_{PWM1} \neq F_{PWM2} \neq F_{PWM3} \neq F_{PWM4}$

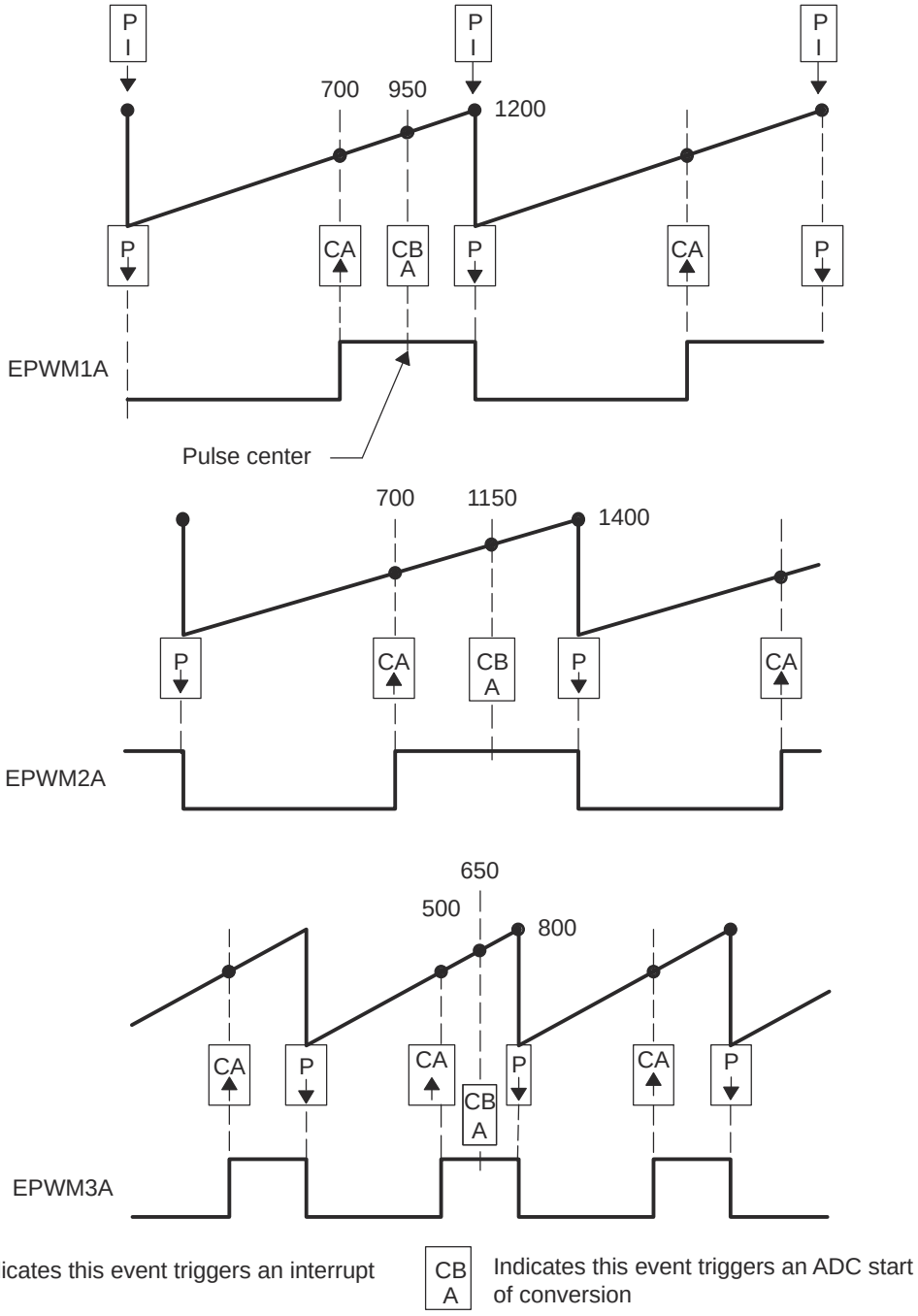


Figure 3-55. Buck Waveforms for Figure 3-54 (Note: Only three bucks shown here)

Example 3-8. Code Snippet for Configuration in Figure 3-54

```
//=====================================================================
// (Note: code for only 3 modules shown)
// Initialization Time
//=======================
// EPWM Module 1 config
   EPwm1Regs.TBPRD = 1200;                          // Period = 1201 TBCLK counts
   EPwm1Regs.TBPHS.half.TBPHS = 0;                  // Set Phase register to zero
   EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;       // Asymmetrical mode
   EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;          // Phase loading disabled
   EPwm1Regs.TBCTL.bit.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 3-56 shows such a configuration; Figure 3-57 shows the waveforms generated by the configuration.

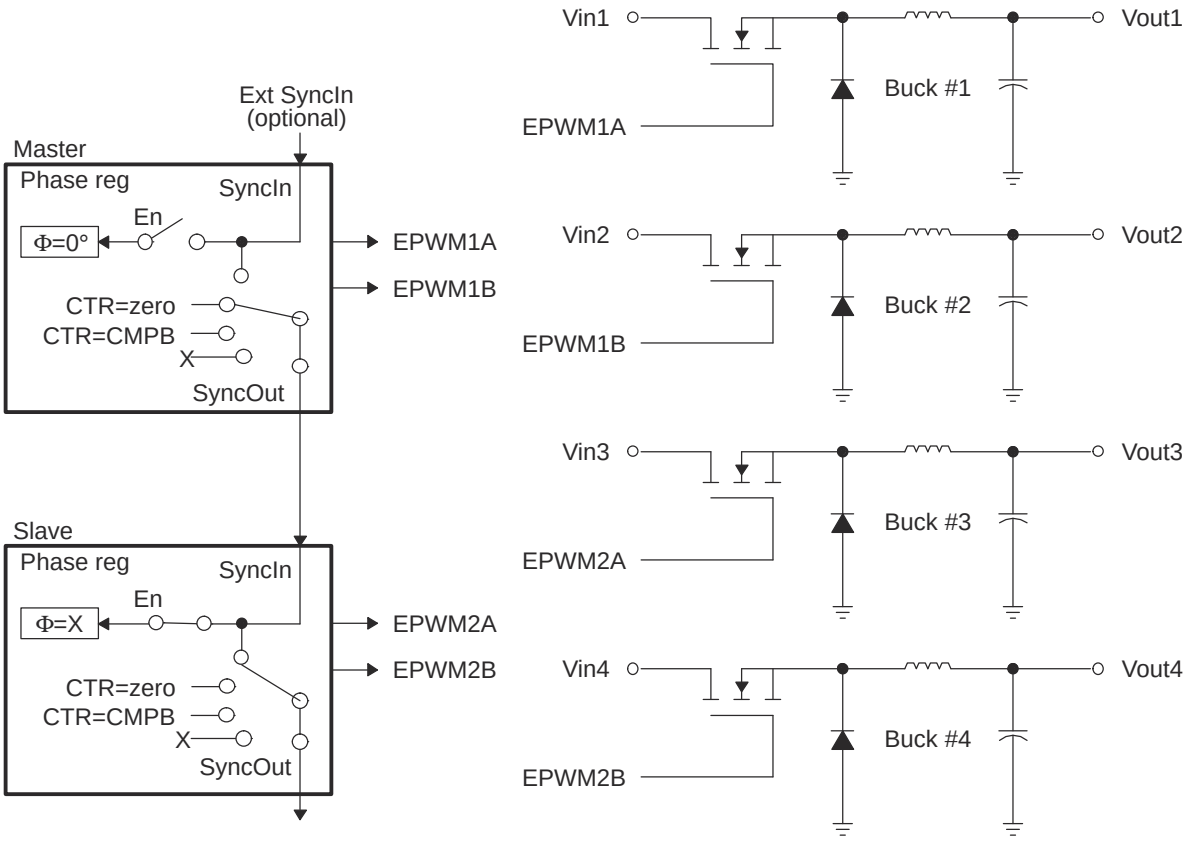


Figure 3-56. Control of Four Buck Stages. (Note: F_{PWM2} = N x F_{PWM1})

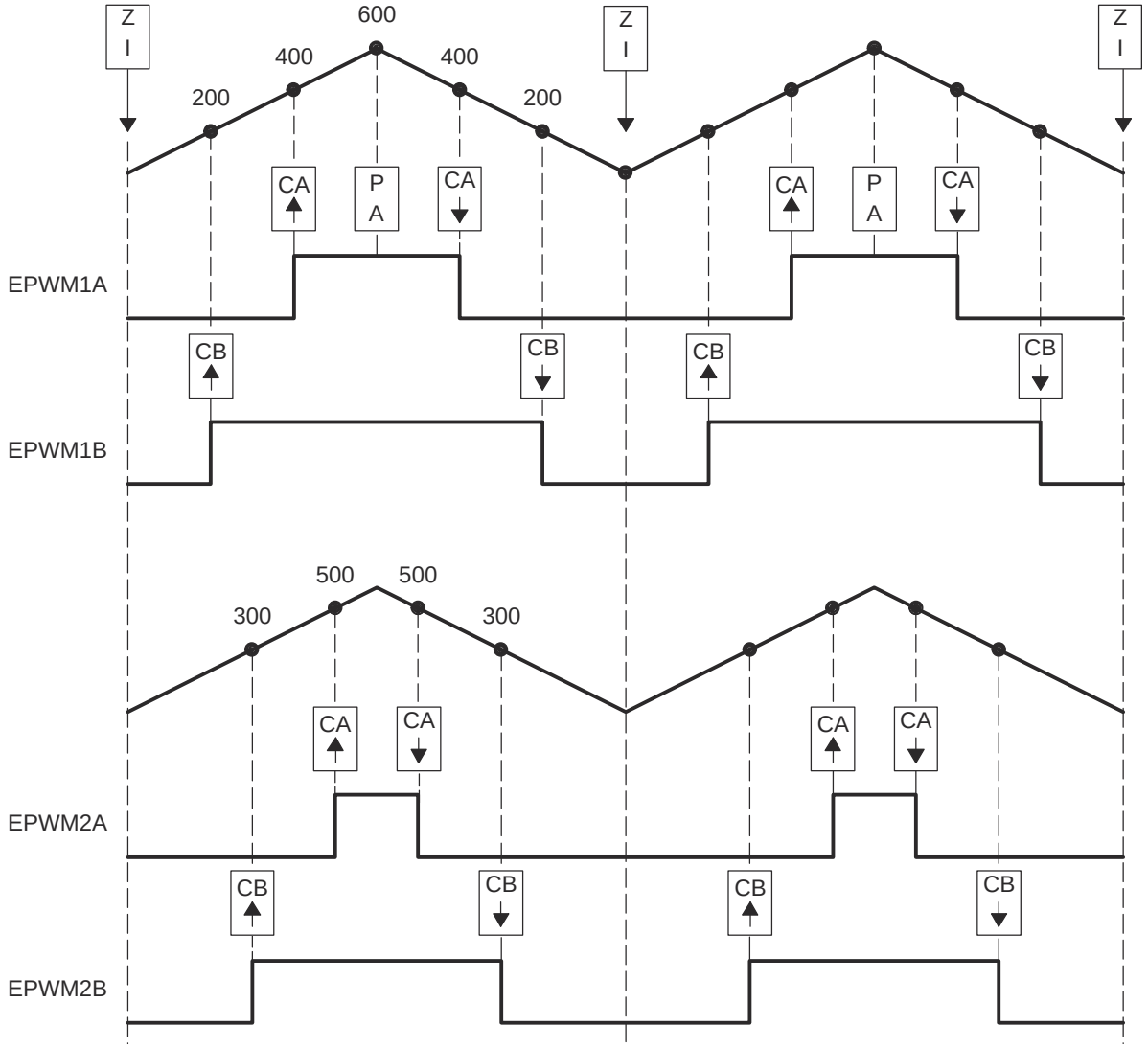


Figure 3-57. Buck Waveforms for Figure 3-56 (Note: $F_{PWM2} = F_{PWM1)}$)

Example 3-9. Code Snippet for Configuration in Figure 3-56

```
//=======================
// EPWM Module 1 config
     EPwm1Regs.TBPRD = 600;                          // Period = 1200 TBCLK counts
     EPwm1Regs.TBPHS.half.TBPHS = 0;                 // Set Phase register to zero
     EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
     EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;         // Master module
     EPwm1Regs.TBCTL.bit.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 3-58 shows control of two synchronized Half-H bridge stages where stage 2 can operate at integer multiple (N) frequencies of stage 1. Figure 3-59 shows the waveforms generated by the configuration shown in Figure 3-58.

Module 2 (slave) is configured for Sync flow-through; if required, this configuration allows for a third Half-H bridge to be controlled by PWM module 3 and also, most importantly, to remain in synchronization with master module 1.

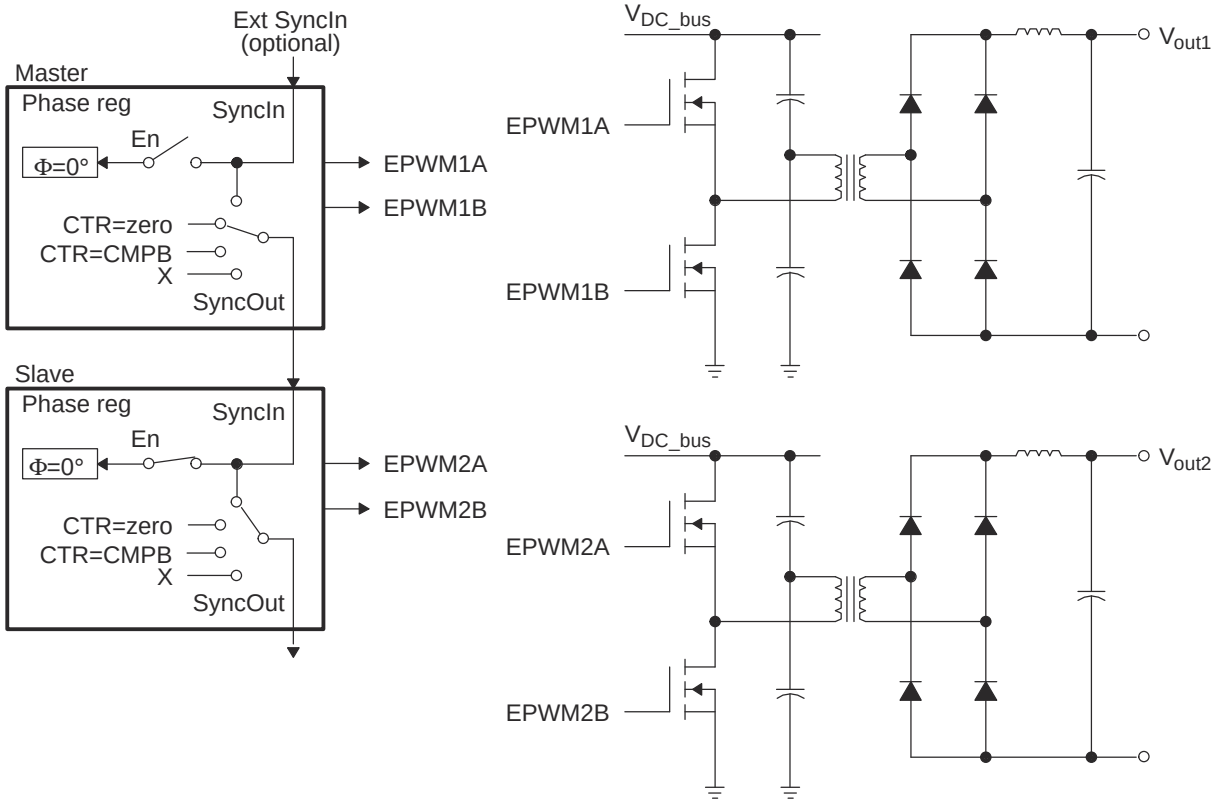


Figure 3-58. Control of Two Half-H Bridge Stages (F_{PWM2} = N x F_{PWM1})

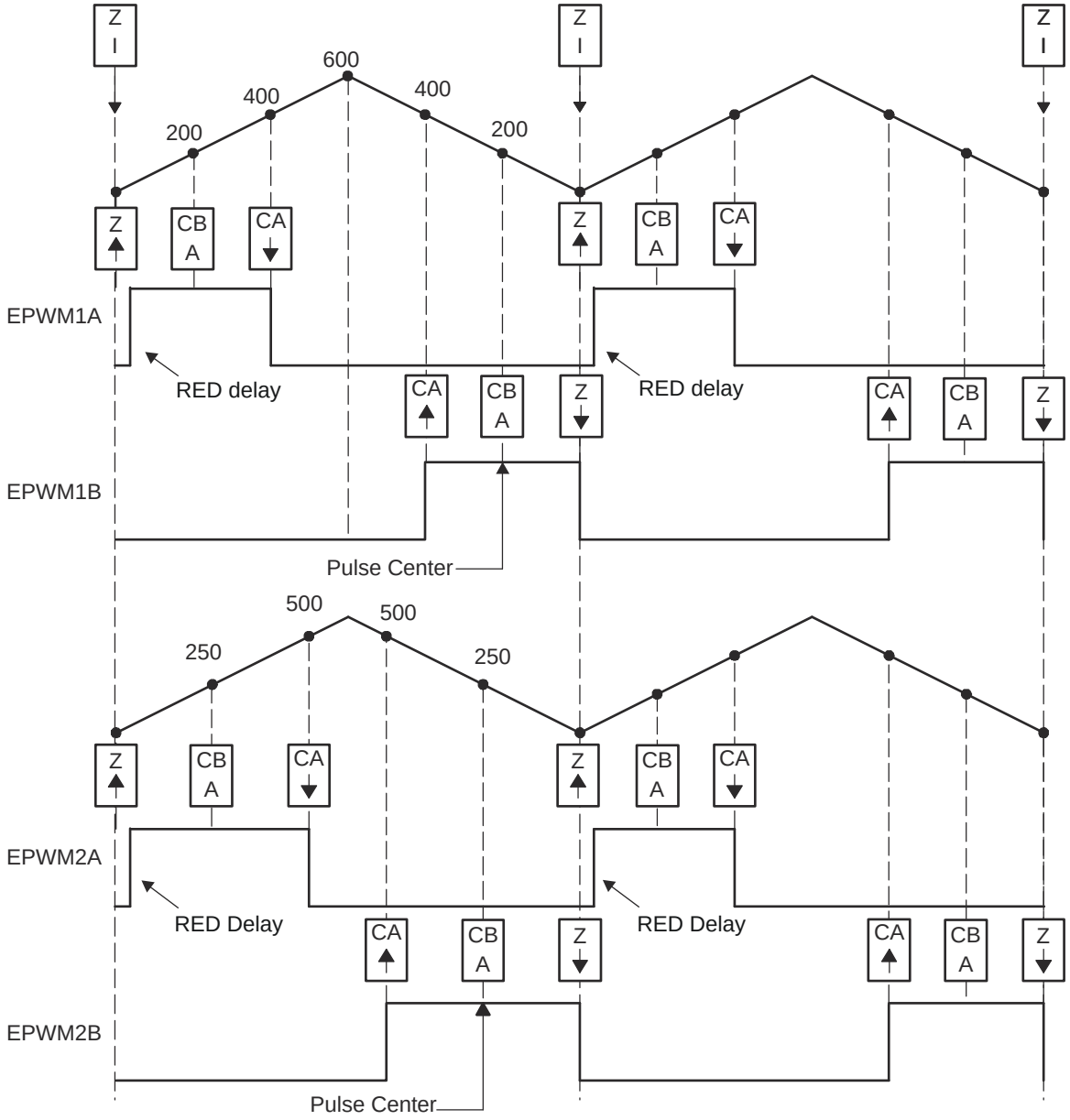


Figure 3-59. Half-H Bridge Waveforms for Figure 3-58 (Note: Here $F_{PWM2} = F_{PWM1}$)

Example 3-10. Code Snippet for Configuration in Figure 3-58

```
//=====================================================================
// Config
//=====================================================================
// Initialization Time
//=======================
// EPWM Module 1 config
     EPwm1Regs.TBPRD = 600;                          // Period = 1200 TBCLK counts
     EPwm1Regs.TBPHS.half.TBPHS = 0;                 // Set Phase register to zero
     EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
     EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;         // Master module
     EPwm1Regs.TBCTL.bit.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;
     EPwm1Regs.DBCTL.bit.OUT_MODE = 2;               // delay on EPWM1A
     EPwm1Regs.DBRED = 10;                           // delay value
// EPWM Module 2 config
     EPwm2Regs.TBPRD = 600;                          // Period = 1200 TBCLK counts
     EPwm2Regs.TBPHS.half.TBPHS = 0;                 // Set Phase register to zero
     EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
     EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;          // Slave module
     EPwm2Regs.TBCTL.bit.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;
     EPwm2Regs.DBCTL.bit.OUT_MODE = 2;               // delay on EPWM2A
     EPwm2Regs.DBRED = 10;                           // delay value
//=========================================================
     EPwm1Regs.CMPA.half.CMPA = 400;                 // adjust duty for output EPWM1A & EPWM1B
     EPwm1Regs.CMPB = 200;                           // adjust point-in-time for ADCSOC trigger
     EPwm2Regs.CMPA.half.CMPA = 500;                 // adjust duty for output EPWM2A & EPWM2B
     EPwm2Regs.CMPB = 250;                           // adjust point-in-time for ADCSOC trigger
```

3.3.6 Controlling Dual 3-Phase Inverters for Motors (ACI and PMSM)

The idea of multiple modules controlling a single power stage can be extended to the 3-phase Inverter case. In such a case, six switching elements can be controlled using three PWM modules, one for each leg of the inverter. Each leg must switch at the same frequency and all legs must be synchronized. A master + two slaves configuration can easily address this requirement. Figure 3-60 shows how six PWM modules can control two independent 3-phase Inverters; each running a motor.

As in the cases shown in the previous sections, we have a choice of running each inverter at a different frequency (module 1 and module 4 are masters as in Figure 3-60), or both inverters can be synchronized by using one master (module 1) and five slaves. In this case, the frequency of modules 4, 5, and 6 (all equal) can be integer multiples of the frequency for modules 1, 2, 3 (also all equal).

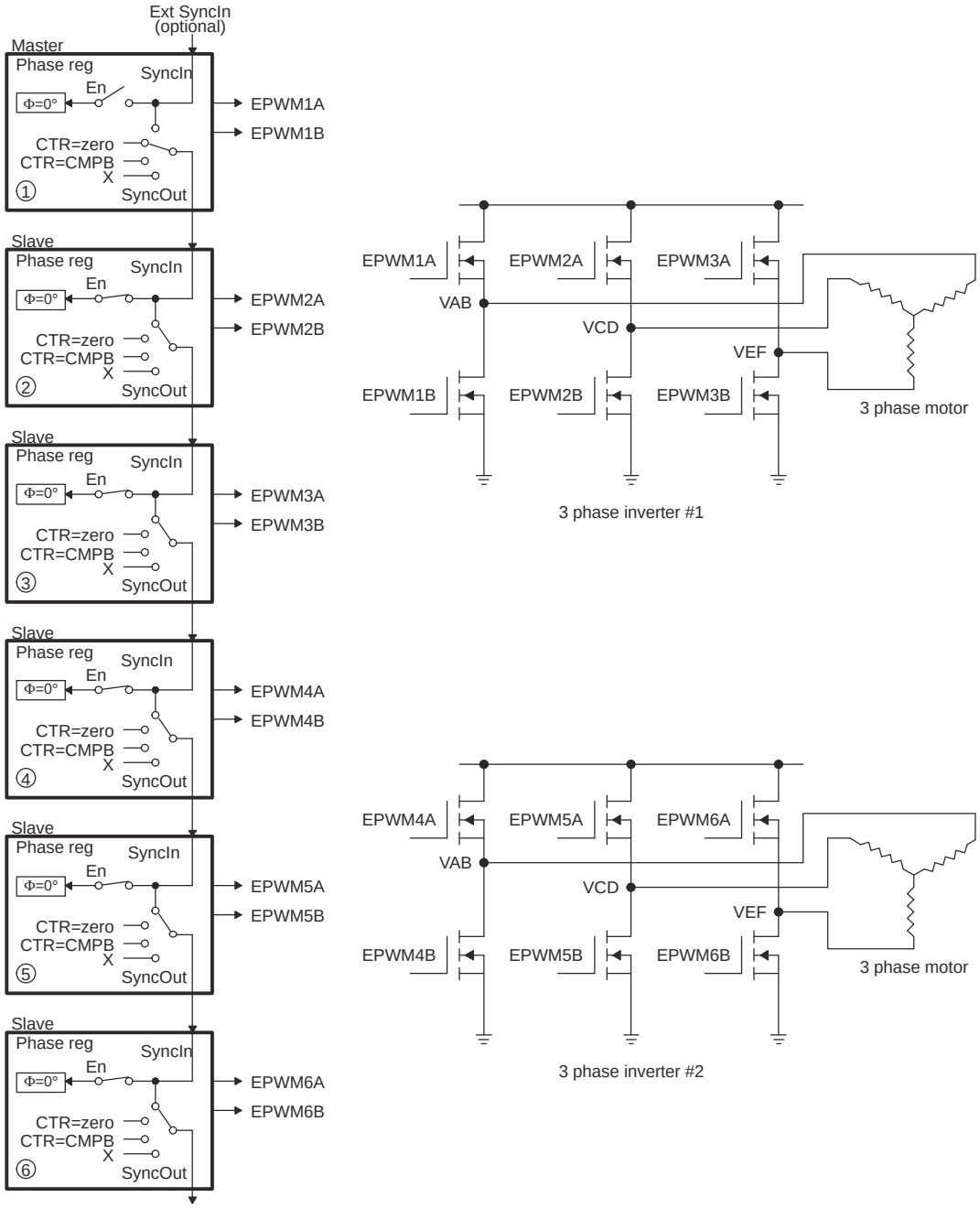


Figure 3-60. Control of Dual 3-Phase Inverter Stages as Is Commonly Used in Motor Control

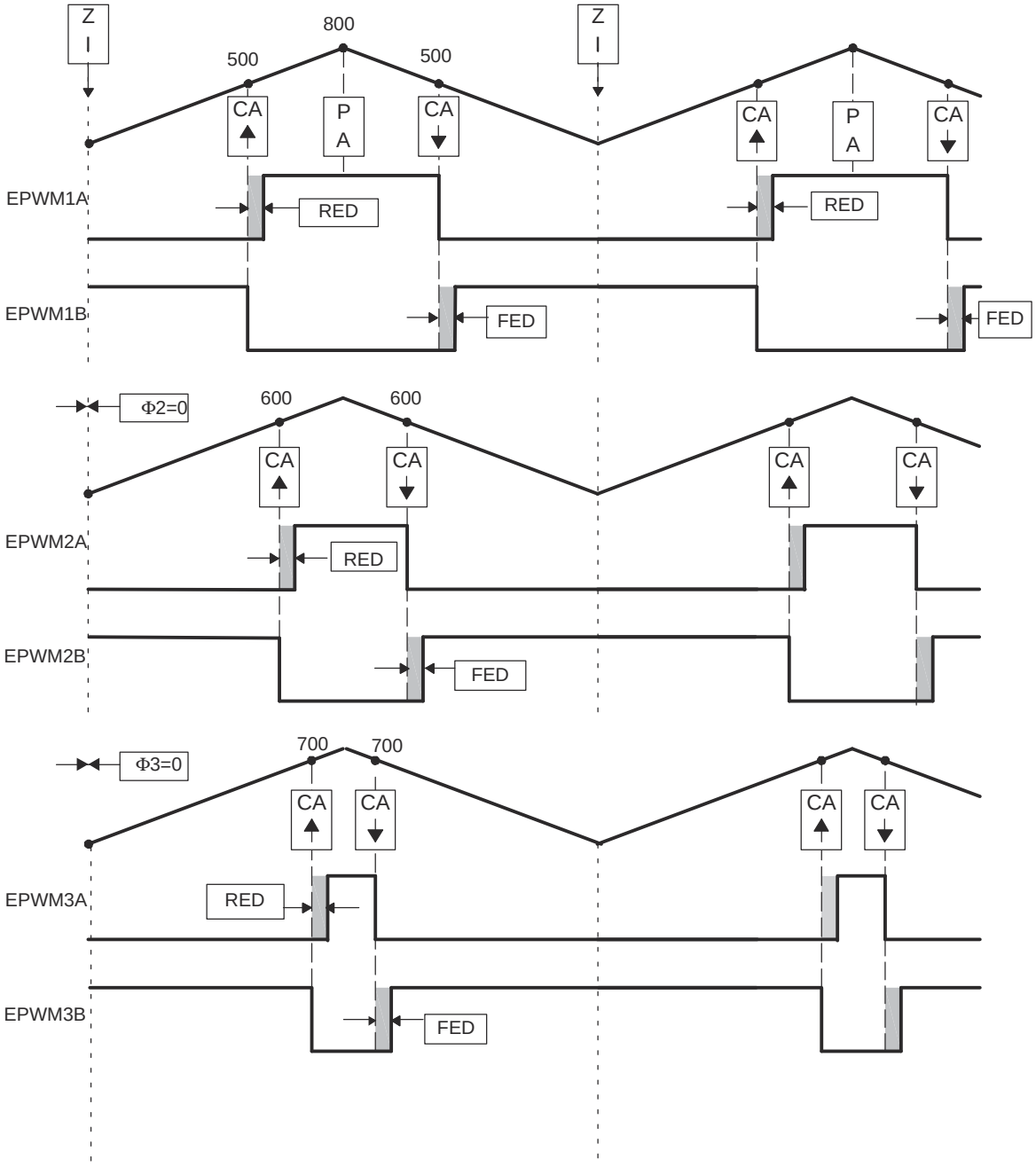


Figure 3-61. 3-Phase Inverter Waveforms for Figure 3-60 (Only One Inverter Shown)

Example 3-11. Code Snippet for Configuration in Figure 3-60

```
//=============================================================================
// Configuration
//=============================================================================
// Initialization Time
//=========================
// EPWM Module 1 config
     EPwm1Regs.TBPRD = 800;                              // Period = 1600 TBCLK counts
     EPwm1Regs.TBPHS.half.TBPHS = 0;                     // Set Phase register to zero
     EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;      // Symmetrical mode
     EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;             // Master module
     EPwm1Regs.TBCTL.bit.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 3-62 shows a master and slave module with a phase relationship of 120°, that is, the slave leads the master.

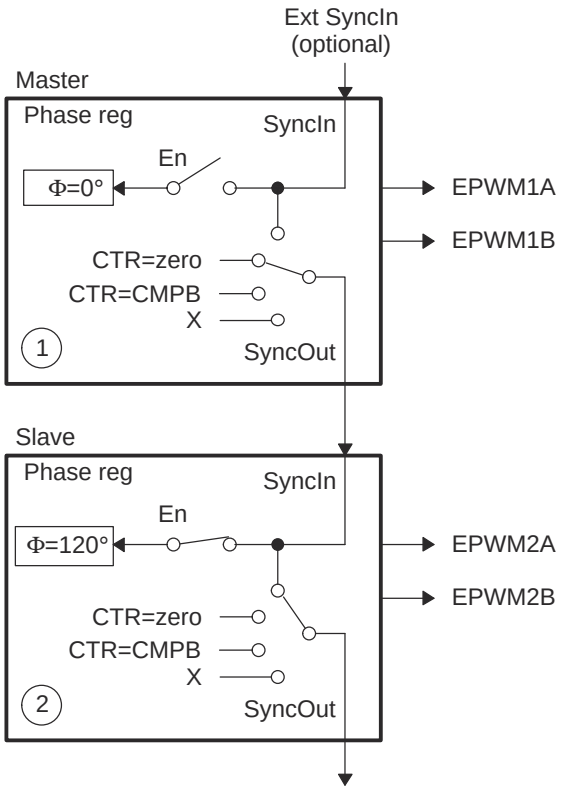


Figure 3-62. Configuring Two PWM Modules for Phase Control

Figure 3-63 shows the associated timing waveforms for this configuration. Here, TBPRD = 600 for both master and slave. For the slave, TBPHS = 200 (200/600 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°.

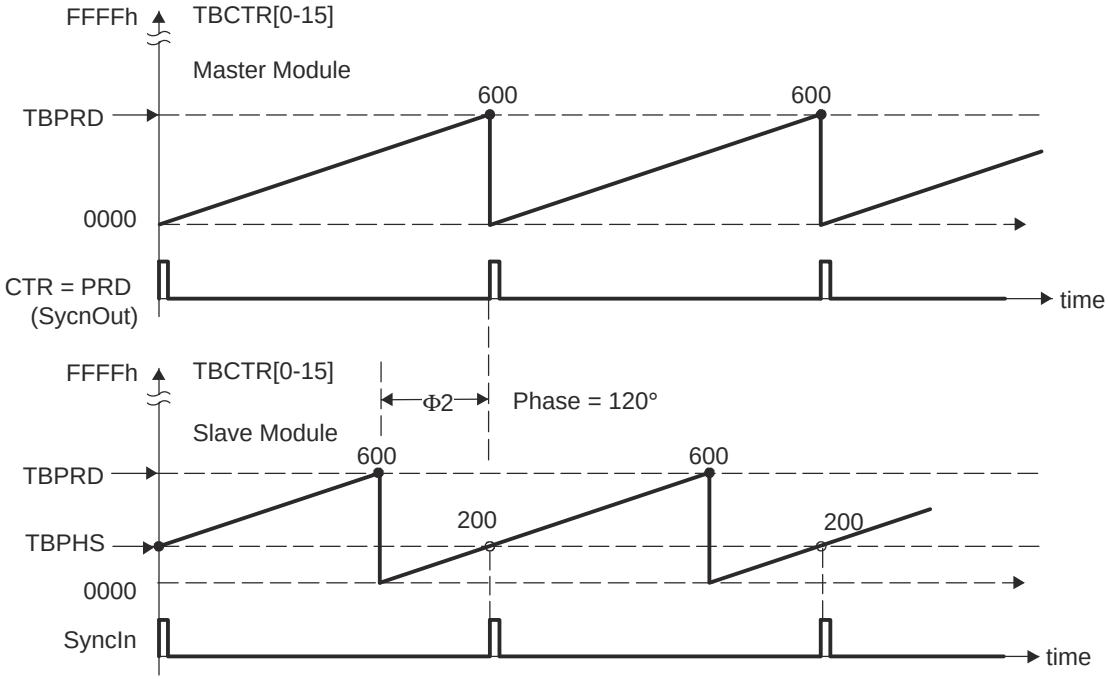


Figure 3-63. Timing Waveforms Associated With Phase Control Between 2 Modules

3.3.8 Controlling a 3-Phase Interleaved DC/DC Converter

A popular power topology that makes use of phase-offset between modules is shown in Figure 3-64. This system uses three PWM modules, with module 1 configured as the master. To work, the phase relationship between adjacent modules must be F = 120°. This is achieved by setting the slave TBPHS registers 2 and 3 with values of 1/3 and 2/3 of the period value, respectively. For example, if the period register is loaded with a value of 600 counts, then TBPHS (slave 2) = 200 and TBPHS (slave 3) = 400. Both slave modules are synchronized to the master 1 module.

This concept can be extended to four or more phases, by setting the TBPHS values appropriately. The following formula gives the TBPHS values for N phases:

TBPHS(N,M) = (TBPRD/N) x (—1)

Where:

N = number of phases

M = PWM module number

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

TBPHS(3,2) = (600/3) x (2-1) = 200 (that is, Phase value for Slave module 2)

TBPHS(3,3) = 400 (Phase value for Slave module 3)

Figure 3-65 shows the waveforms for the configuration in Figure 3-64.

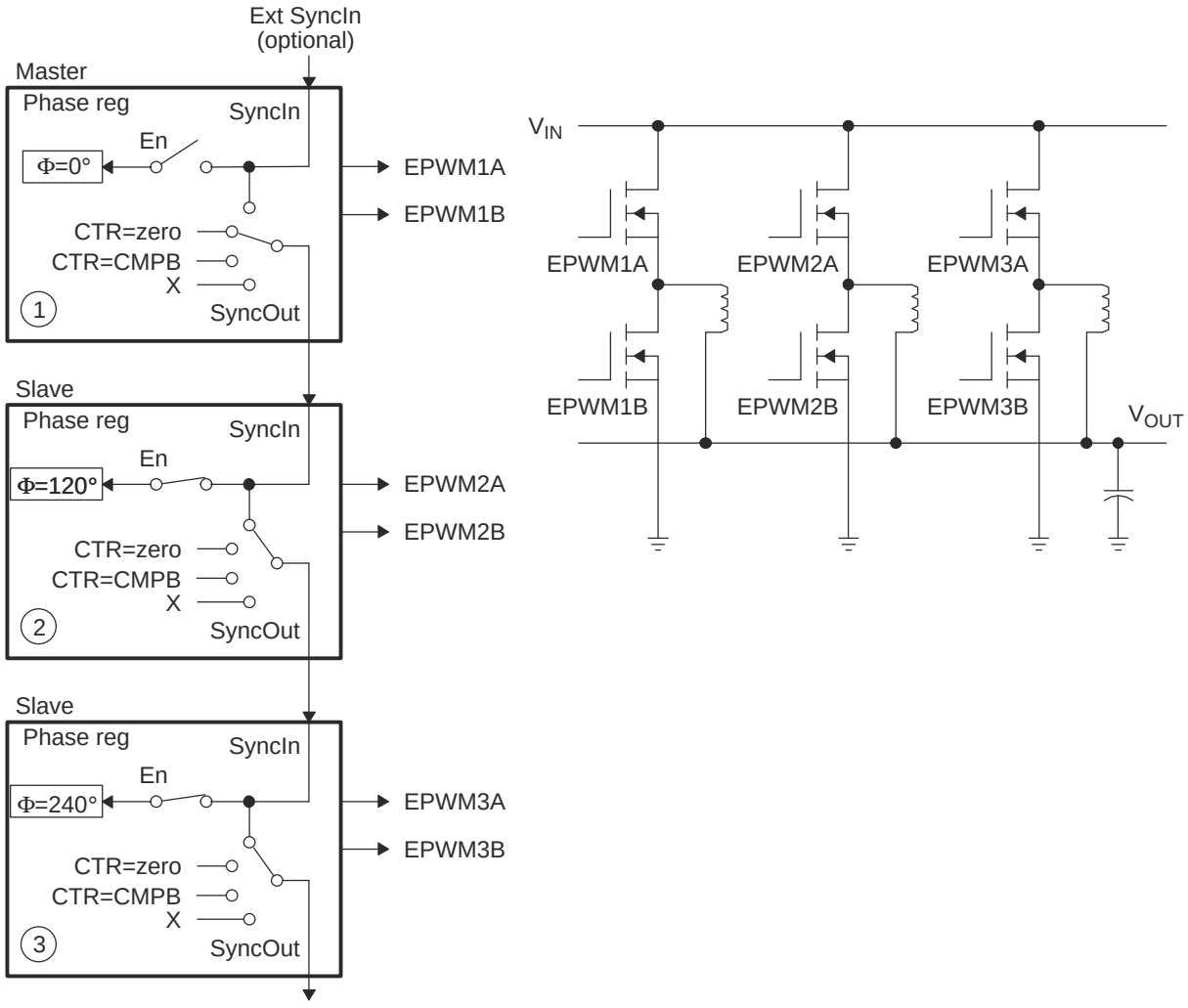


Figure 3-64. Control of a 3-Phase Interleaved DC/DC Converter

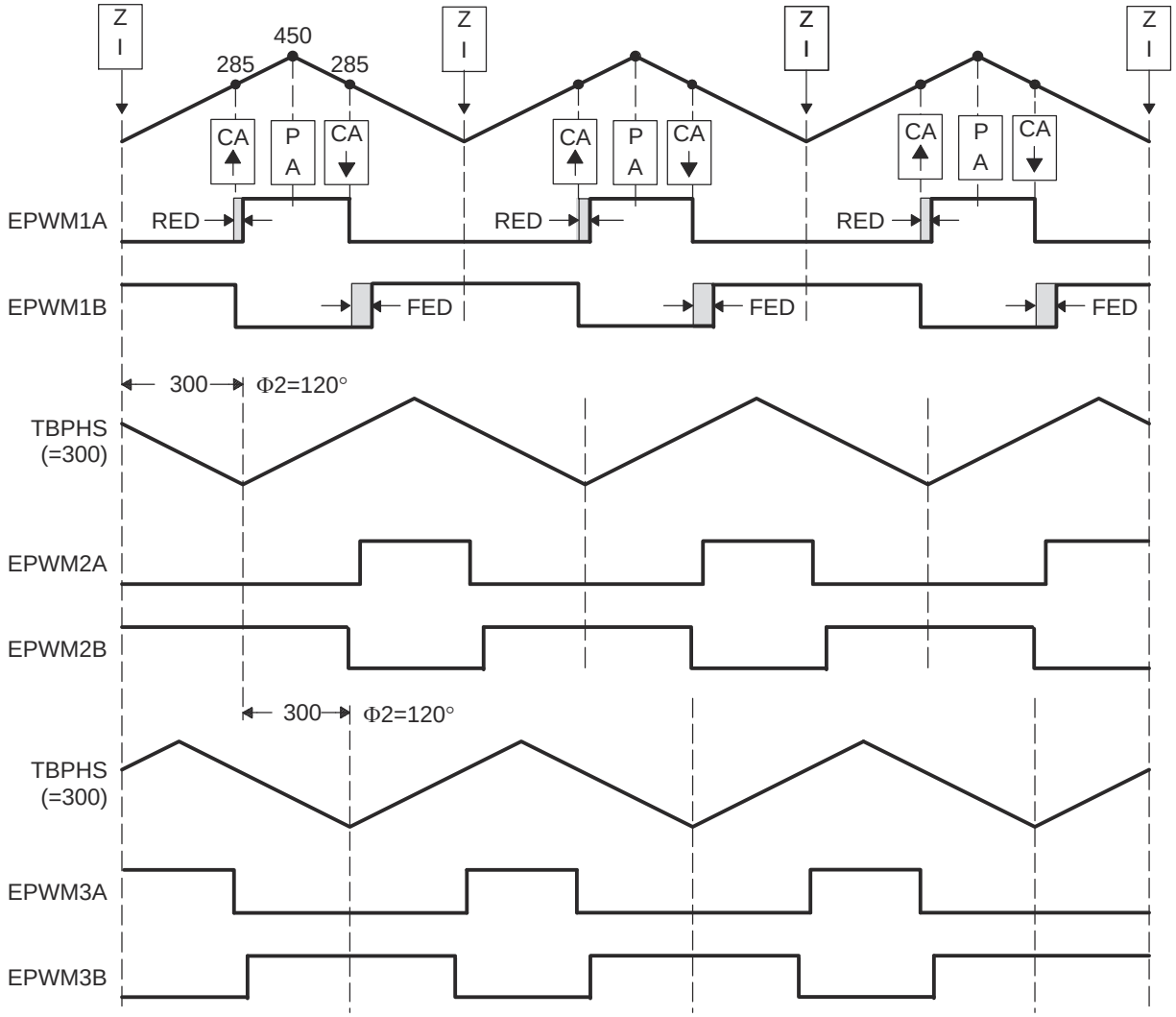


Figure 3-65. 3-Phase Interleaved DC/DC Converter Waveforms for Figure 3-64

Example 3-12. Code Snippet for Configuration in Figure 3-64

```
//=====================================================================
// Config
// Initialization Time
//=====================================================================
// EPWM Module 1 config
EPwm1Regs.TBPRD = 450;                              // Period = 900 TBCLK counts
EPwm1Regs.TBPHS.half.TBPHS = 0;                     // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;      // Symmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;             // Master module
EPwm1Regs.TBCTL.bit.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 3-66 assumes a static or constant phase relationship between legs (modules). In such a case, control is achieved by modulating the duty cycle. It is also possible to dynamically change the phase value on a cycle-by-cycle basis. This feature lends itself to controlling a class of power topologies known as *phase-shifted full bridge*, or *zero voltage switched full bridge.* Here the controlled parameter is not duty cycle (this is kept constant at approximately 50 percent); instead it is the phase relationship between legs. Such a system can be implemented by allocating the resources of two PWM modules to control a single power stage, which in turn requires control of four switching elements. Figure 3-67 shows a master/slave module combination synchronized together to control a full H-bridge. In this case, both master and slave modules are required to switch at the same PWM frequency. The phase is controlled by using the slave's phase register (TBPHS). The master's phase register is not used and therefore can be initialized to zero.

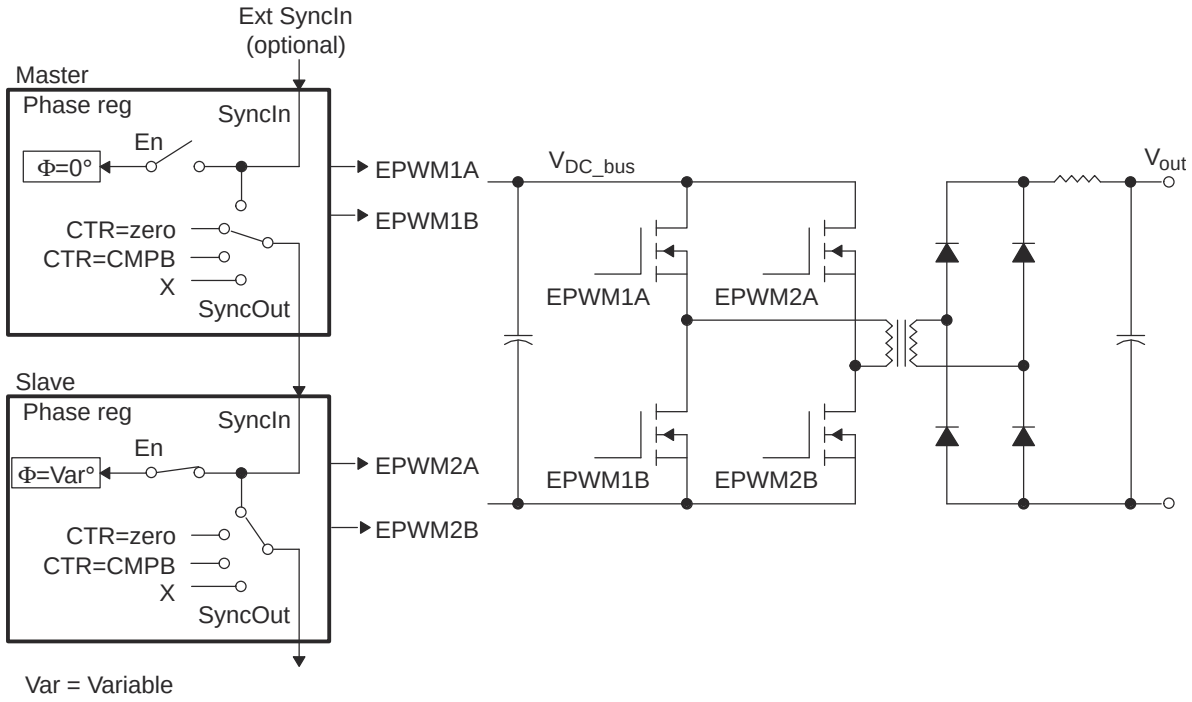


Figure 3-66. Controlling a Full-H Bridge Stage (F_{PWM2} = F_{PWM1})

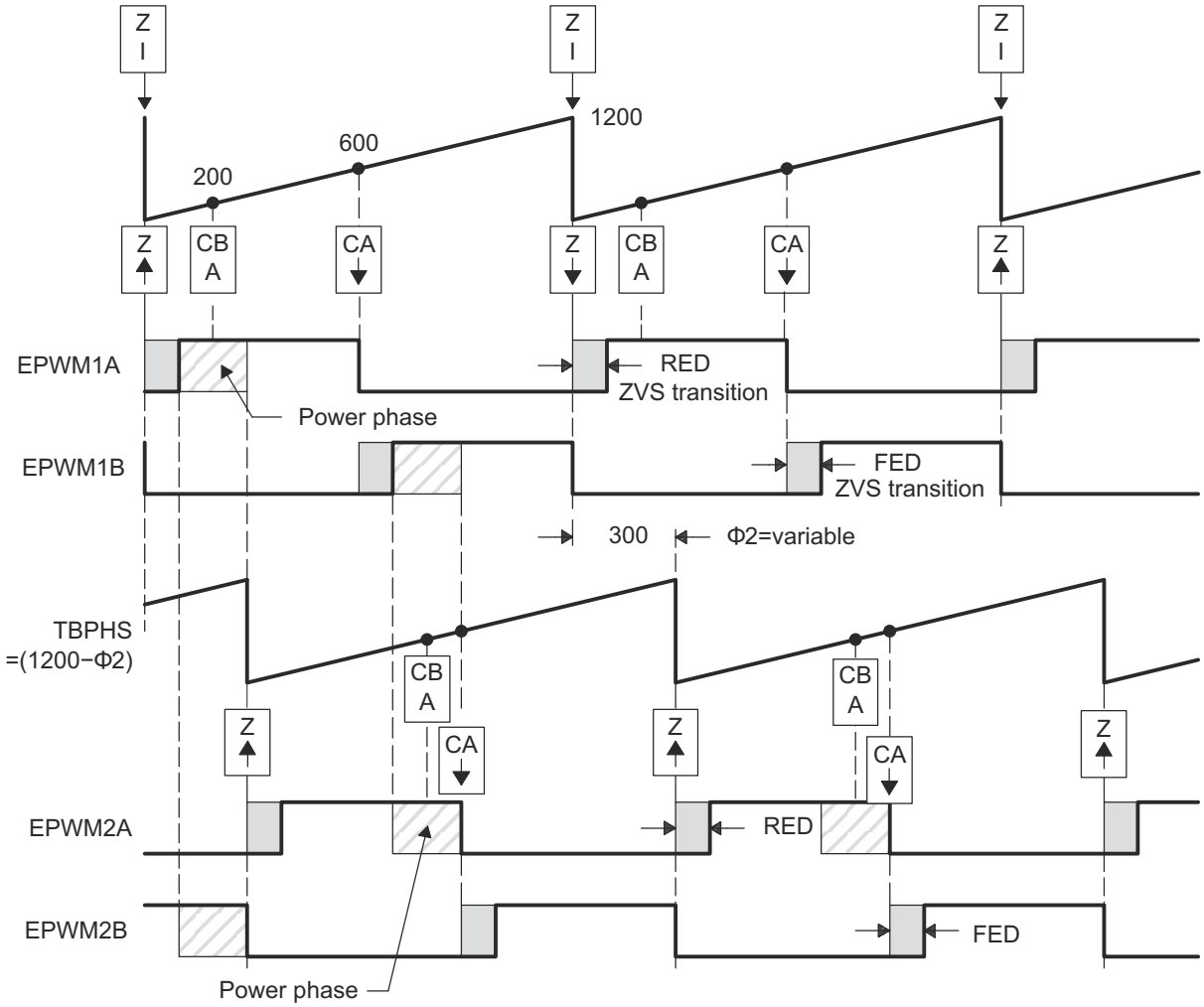


Figure 3-67. ZVS Full-H Bridge Waveforms

Example 3-13. Code Snippet for Configuration in Figure 3-66

```
//=========================================================================
// Config
//=========================================================================
// Initialization Time
//=========================
// EPWM Module 1 config
EPwm1Regs.TBPRD = 1200;                                 // Period = 1201 TBCLK counts
EPwm1Regs.CMPA.half.CMPA = 600;                         // Set 50% fixed duty for EPWM1A
EPwm1Regs.TBPHS.half.TBPHS = 0;                         // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;              // Asymmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;                 // Master module
EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;             // Sync down-stream module
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;           // load on CTR=Zero
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;           // load on CTR=Zero
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                      // set actions for EPWM1A
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;          // enable Dead-band module
EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;               // Active Hi complementary
EPwm1Regs.DBFED = 50;                                   // FED = 50 TBCLKs initially
EPwm1Regs.DBRED = 70;                                   // RED = 70 TBCLKs initially
                                                        // EPWM Module 2 config
EPwm2Regs.TBPRD = 1200;                                 // Period = 1201 TBCLK counts
EPwm2Regs.CMPA.half.CMPA = 600;                         // Set 50% fixed duty EPWM2A
EPwm2Regs.TBPHS.half.TBPHS = 0;                         // Set Phase register to zero initially
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;              // Asymmetrical mode
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;                  // Slave module
EPwm2Regs.TBCTL.bit.PRDLD = TB_SHADOW;
EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;              // sync flow-through
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;           // load on CTR=Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;           // load on CTR=Zero
EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;                      // set actions for EPWM2A
EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;          // enable Dead-band module
EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;               // Active Hi complementary
EPwm2Regs.DBFED = 30;                                   // FED = 30 TBCLKs initially
EPwm2Regs.DBRED = 40;                                   // RED = 40 TBCLKs initially
// Run Time (Note: Example execution of one run-time instant)
//=============================================================
EPwm2Regs.TBPHS = 1200-300;                             // Set Phase reg to
                                                        // 300/1200 * 360 = 90 deg
EPwm1Regs.DBFED = FED1_NewValue;                        // Update ZVS transition interval
EPwm1Regs.DBRED = RED1_NewValue;                        // Update ZVS transition interval
EPwm2Regs.DBFED = FED2_NewValue;                        // Update ZVS transition interval
EPwm2Regs.DBRED = RED2_NewValue;                        // Update ZVS transition interval
EPwm1Regs.CMPB = 200;                                   // Adjust point-in-time for ADCSOC
trigger
```

3.3.10 Controlling a Peak Current Mode Controlled Buck Module

Peak current control techniques offer a number of benefits like automatic over current limiting, fast correction for input voltage variations, and reducing magnetic saturation. Figure 3-68 shows the use of ePWM1A along with the on-chip analog comparator for buck converter topology. The output current is sensed through a current sense resistor and fed to the positive terminal of the on-chip comparator. The internal programmable 10-bit DAC can be used to provide a reference peak current at the negative terminal of the comparator. Alternatively, an external reference could be connected at this input. The comparator output is an input to the digital compare submodule. The ePWM module is configured in such a way so as to trip the ePWM1A output as soon as the sensed current reaches the peak reference value. A cycle-by-cycle trip mechanism is used. Figure 3-69 shows the waveforms generated by the configuration.

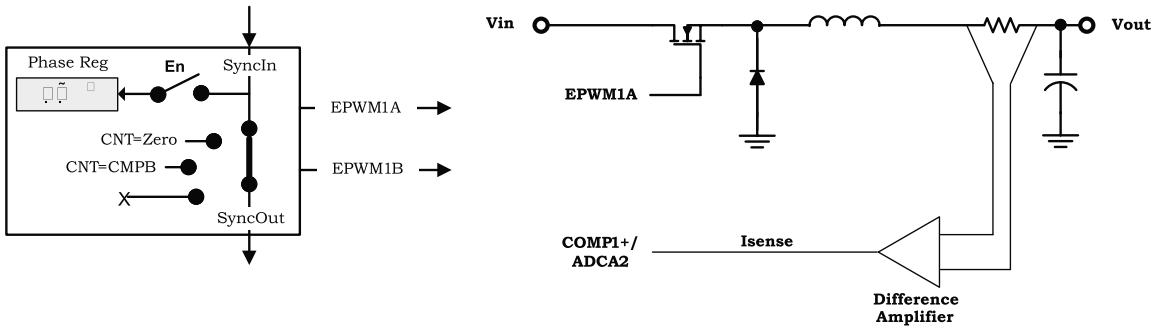


Figure 3-68. Peak Current Mode Control of a Buck Converter

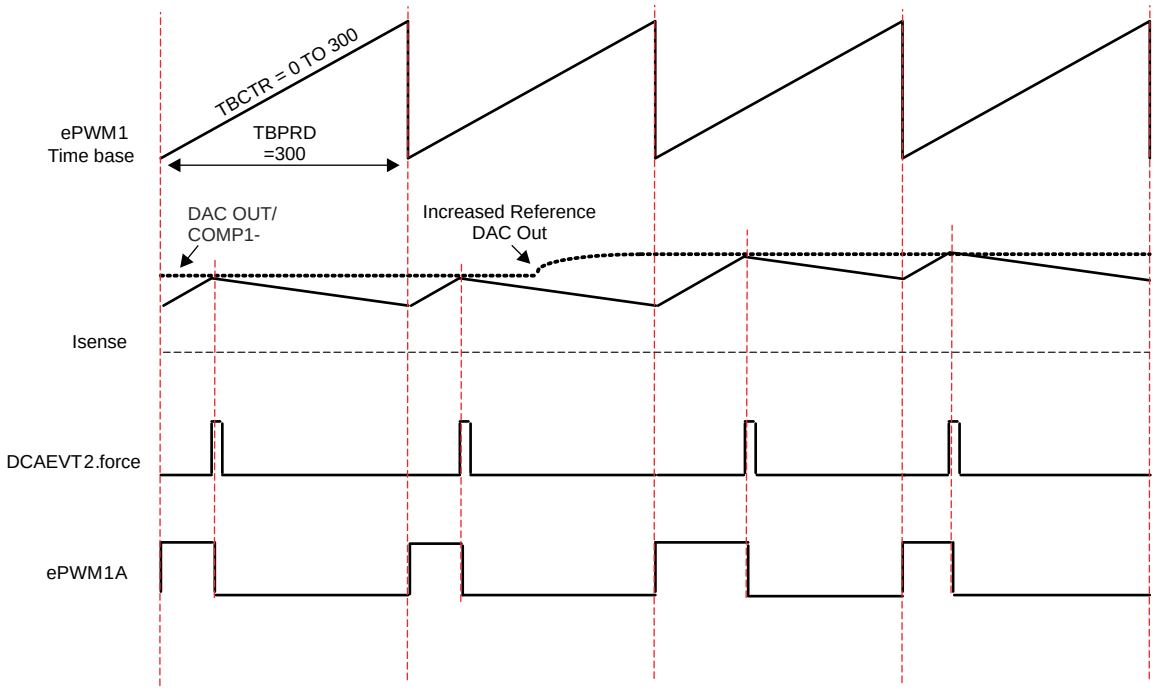


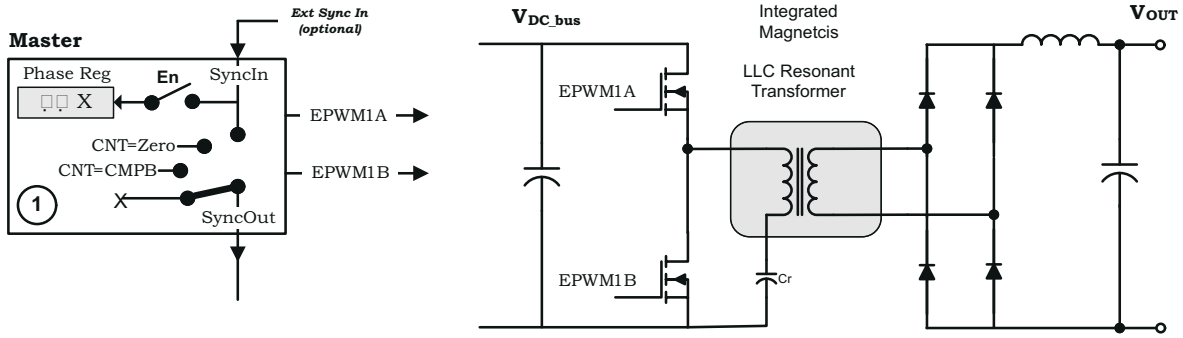
Figure 3-69. Peak Current Mode Control Waveforms for Figure 3-68

Example 3-14. Code Snippet for Configuration in Figure 3-68

```
//=============================================================================
// Config
// Initialization Time
//=============================================================================
EPwm1Regs.TBPRD = 300;
// Period = 300 TBCLK counts                        // (200 KHz @ 60MHz clock)
EPwm1Regs.TBPHS.half.TBPHS = 0;                     // Set Phase register to zero
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;          // Asymmetrical mode
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;             // Phase loading disabled
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;            // Clock ratio to SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
Pwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                   // Set PWM1A on Zero
                                                    // Define an event (DCAEVT2) based on
                                                    // Comparator 1 Output
EPwm1Regs.DCTRIPSEL.bit.DCAHCOMPSEL = DC_COMP1OUT;  // DCAH = Comparator 1 output

EPwm1Regs.TZDCSEL.bit.DCAEVT2 = TZ_DCAH_HI;         // DCAEVT2 = DCAH high(will become active
                                                    // as Comparator output goes high)
EPwm1Regs.DCACTL.bit.EVT2SRCSEL = DC_EVT2;          // DCAEVT2 = DCAEVT2 (not filtered)
EPwm1Regs.DCACTL.bit.EVT2FRCSYNCSEL = DC_EVT_ASYNC; // Take async path // Enable DCAEVT2 as a
                                                    // one-shot trip source
                                                    // Note: DCxEVT1 events can be defined as
                                                    // one-shot.
                                                    // DCxEVT2 events can be defined as
                                                    // cycle-by-cycle.
EPwm1Regs.TZSEL.bit.DCAEVT2 = 1;                    // What do we want the DCAEVT1 and DCBEVT1
                                                    // events to do?
                                                    // DCAEVTx events can force EPWMxA
                                                    // DCBEVTx events can force EPWMxB
EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO;              // EPWM1A will go low
//=============================================================================
// Run Time
//=============================================================================
// Adjust reference peak current to Comparator 1 negative input
```

3.3.11 Controlling H-Bridge LLC Resonant Converter

For many years, various topologies of resonant converters have been well-known in the field of power electronics. In addition to these, H-bridge LLC resonant converter topology has recently gained popularity in many consumer electronics applications where high efficiency and power density are required. In this example, the single channel configuration of ePWM1 is detailed, yet the configuration can easily be extended to multichannel. Here, the controlled parameter is not duty cycle (this is kept constant at approximately 50 percent); instead it is frequency. Although the deadband is not controlled and kept constant as 300ns (that is, 18@60MHz TBCLK), it is up to the user to update it in real time to enhance the efficiency by adjusting enough time delay for soft switching.



NOTE: Θ = X indicates value in phase register is a "don't care"

Figure 3-70. Control of Two Resonant Converter Stages

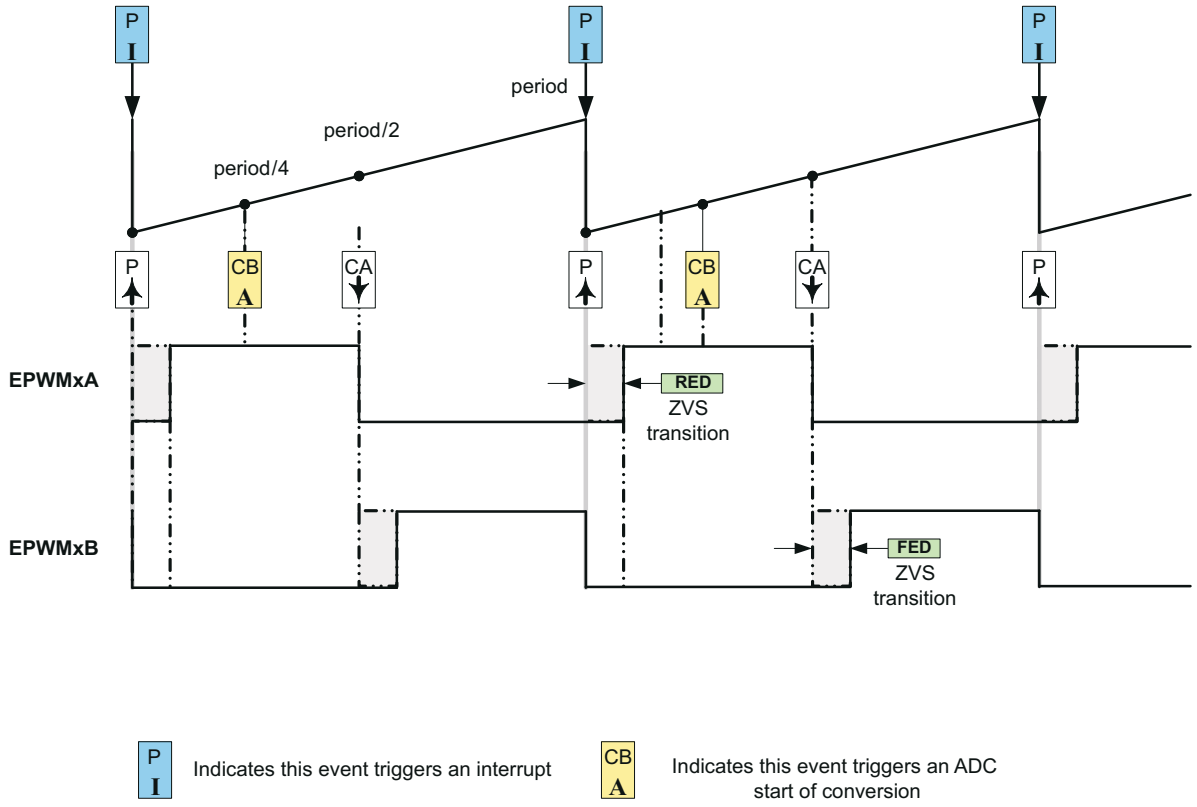


Figure 3-71. H-Bridge LLC Resonant Converter PWM Waveforms

Example 3-15. Code Snippet for Configuration in Figure 3-70

```
//=========================================================================
// Config
//========================================================================= //
Initialization Time
//======================== //
EPWMxA & EPWMxB config
EPwm1Regs.TBCTL.bit.PRDLD = TB_IMMEDIATE;             // Set immediate load
EPwm1Regs.TBPRD = period;                             // PWM frequency = 1 / period
EPwm1Regs.CMPA.half.CMPA = period/2;                  // Set duty as 50%
EPwm1Regs.CMPB = period/4;                            // Set duty as 25%
EPwm1Regs.TBPHS.half.TBPHS = 0;                       // Set as master, phase =0
EPwm1Regs.TBCTR = 0;                                  // Time base counter =0
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;            // Count-up mode: used for asymmetric PWM
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;               // Disable phase loading
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;           // Used to sync EPWM(n+1)"down-stream"
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;              // Set the clock rate
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;                 // Set the clock rate
EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_PRD;          // Load on CTR=PRD
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_PRD;          // Load on CTR=PRD
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;           // Shadow mode. Operates as a double buffer.
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;           // Shadow mode. Operates as a double buffer.
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                    // Set PWM1A on Zero
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;                  // Clear PWM1A on event A, up count
EPwm1Regs.AQCTLB.bit.CAU = AQ_SET;                    // Set PWM1B on event A, up count
EPwm1Regs.AQCTLB.bit.PRD = AQ_CLEAR;                  // Clear PWM1B on PRD
EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;                // EPWMxA is the source for both delays
EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;        // Enable Dead-band module
EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;             // Active High Complementary (AHC)
EPwm1Regs.DBRED = 30;                                 // RED = 30 TBCLKs initially
EPwm1Regs.DBFED = 30;                                 // FED = 30 TBCLKs initially
                                                      // Configure TZ1 for short cct
                                                      // protection EALLOW;
EPwm1Regs.TZSEL.bit.OSHT1 = 1;                        // one-shot source EPwm1Regs.TZCTL.bit.TZA =
TZ_FORCE_LO;
                                                      // set EPWM1A to low at fault
EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_LO;                // set EPWM1B to low at fault instant
EPwm1Regs.TZEINT.bit.OST = 1;                         // Enable TZ interrupt EDIS;
                                                      // Enable HiRes option EALLOW;
EPwm1Regs.HRCNFG.all = 0x0;
EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP;
EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP;
EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_PRD;
EDIS;                                                 // Run Time (Note: Example execution of
                                                      // one run-time instant)
//===========================================================
EPwm1Regs.TBPRD = period_new value;                   // Update new period
                                                      // EPwm1Regs.CMPA.half.CMPA= period_new
                                                      // value/2;
                                                      // Update new CMPA EPwm1Regs.CMPB= period_new
                                                      // value/4;
                                                      // Update new CMPB
                                                      // Update new CMPB
                                                      // Update new CMPB
```

3.4 Registers

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

3.4.1 Time-Base Submodule Registers

This section describes the time-base submodule registers.

3.4.1.1 Time-Base Control Register (TBCTL)

Figure 3-72. Time-Base Control Register (TBCTL)

15	14	13	12	10	9	8
FREE, SOFT		PHSDIR	CLKDIV		HSPCLKDIV	
R/W-0		R/W-0	R/W-0		R/W-0,0,1	

7	6	5	4	3	2	1	0
HSPCLKDIV	SWFSYNC	SYNCOSEL		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 3-21. 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 3-21. 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, or when a digital compare sync event occurs.
1-0	CTRMODE		Counter Mode
			The time-base counter mode is normally configured once and not changed during normal operation. If you change the mode of the counter, the change will take effect at the next TBCLK edge and the current counter value shall increment or decrement from the value before the mode change.
			These bits set the time-base counter mode of operation as follows:
		00	Up-count mode
		01	Down-count mode
		10	Up-down-count mode
		11	Stop-freeze counter operation (default on reset)

3.4.1.2 Time-Base Status Register (TBSTS)

Figure 3-73. 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; *-n* = value after reset

Table 3-22. 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.

3.4.1.3 Time-Base Phase High Resolution Register (TBPHSHR)

Figure 3-74. Time-Base Phase High Resolution Register (TBPHSHR)

15	8
TBPHSHR	
R/W-0	

7	0
Reserved	
R-0	

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

Table 3-23. Time-Base Phase High Resolution Register (TBPHSHR) Field Descriptions

Bit	Field	Value	Description
15-8	TBPHSHR	00-FFh	Time base phase high-resolution bits
7-0	Reserved		Reserved

3.4.1.4 Time-Base Phase Register (TBPHS)

Figure 3-75. Time-Base Phase Register (TBPHS)

15	0
TBPHS	
R/W-0	

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

Table 3-24. Time-Base Phase Register (TBPHS) Field Descriptions

Bits	Name	Value	Description
15-0	TBPHS	0000-FFFF	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.

3.4.1.5 Time-Base Counter Register (TBCTR)

Figure 3-76. Time-Base Counter Register (TBCTR)

15	0
TBCTR	
R/W-0	

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

Table 3-25. Time-Base Counter Register (TBCTR) Field Descriptions

Bits	Name	Value	Description
15-0	TBCTR	0000-FFFF	Reading these bits gives the current time-base counter value. Writing to these bits sets the current time-base counter value. The update happens as soon as the write occurs; the write is NOT synchronized to the time-base clock (TBCLK) and the register is not shadowed.

3.4.1.6 Time-Base Period Register (TBPRD)

Figure 3-77. Time-Base Period Register (TBPRD)

15	0
TBPRD	
R/W-0	

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

Table 3-26. Time-Base Period Register (TBPRD) Field Descriptions

Bit	Field	Value	Description
15-0	TBPRD	0000-FFFFh	These bits determine the period of the time-base counter. This sets the PWM frequency. Shadowing of this register is enabled and disabled by the TBCTL[PRDLD] bit. By default this register is shadowed. • 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.

3.4.1.7 Time Base Period High Resolution Register (TBPRDHR)

Figure 3-78. Time Base Period High Resolution Register (TBPRDHR)

15							8
TBPRDHR							
R/W-0							
7							0
Reserved							
R-0							

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

Table 3-27. Time Base Period High Resolution Register (TBPRDHR) Field Descriptions

Bit	Field	Value	Description
15-8	TBPRDHR	00-FFh	Period High Resolution Bits These 8-bits contain the high-resolution portion of the period value. The TBPRDHR register is not affected by the TBCTL[PRDLD] bit. Reads from this register always reflect the shadow register. Likewise writes are also to the shadow register. The TBPRDHR register is only used when the high resolution period feature is enabled. This register is only available with ePWM modules which support high-resolution period control.
7-0	Reserved	0	Reserved

3.4.1.8 Time-Base Period High Resolution Mirror Register (TBPRDHRM)

Figure 3-79. Time-Base Period High Resolution Mirror Register (TBPRDHRM)

15							8
TBPRDHR							
R/W-0							
7							0
Reserved							
R-0							

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

Table 3-28. Time-Base Period High Resolution Mirror Register (TBPRDHRM) Field Descriptions

Bit	Field	Value	Description
15-8	TBPRDHR	00-FFh	Period High Resolution Bits These 8-bits contain the high-resolution portion of the period value TBPRD provides backwards compatibility with earlier ePWM modules. The mirror registers (TBPRDM and TBPRDHRM) allow for 32-bit writes to TBPRDHR in one access. Due to the odd-numbered memory address location of the TBPRD legacy register, a 32-bit write is not possible with TBPRD and TBPRDHR. The TBPRDHRM register is not affected by the TBCTL[PRDLD] bit Writes to both the TBPRDHR and TBPRDM locations access the high-resolution (least significant 8-bit) portion of the Time Base Period value. The only difference is that unlike TBPRDHR, reads from the mirror register TBPRDHRM, are indeterminate (reserved for TI Test). The TBPRDHRM register is available with ePWM modules which support high-resolution period control and is used only when the high resolution period feature is enabled.
7-0	Reserved	00-FFh	Reserved for TI Test

3.4.1.9 Time-Base Period Mirror Register (TBPRDM)

Figure 3-80. Time-Base Period Mirror Register (TBPRDM)

15	0
TBPRD	
R/W-0	

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

Table 3-29. Time-Base Period Mirror Register (TBPRDM) Field Descriptions

Bit	Field	Value	Description
15-0	TBPRD	0000-FFFFh	TBPRDM and TBPRD can both be used to access the time-base period. TBPRD provides backwards compatibility with earlier ePWM modules. The mirror registers (TBPRDM and TBPRDHRM) allow for 32-bit writes to TBPRDHR in one access. Due to the odd address memory location of the TBPRD legacy register, a 32-bit write is not possible. By default writes to this register are shadowed. Unlike the TBPRD register, reads of TBPRDM always return the active register value. Shadowing is enabled and disabled by the TBCTL[PRDLD] bit. • If TBCTL[PRDLD] = 0, then the shadow is enabled and any write will automatically go to the shadow register. In this case the active register will be loaded from the shadow register when the time-base counter equals zero. Reads return the active value. • If TBCTL[PRDLD] = 1, then the shadow is disabled and any write to this register will go directly to the active register controlling the hardware. Likewise reads return the active value.

3.4.2 Counter-Compare Submodule Registers

This section describes the counter-compare submodule control and status registers.

3.4.2.1 Counter-Compare Control (CMPCTL) Register

Figure 3-81. Counter-Compare Control (CMPCTL) Register

15					10	9	8
Reserved						SHDWBFULL	SHDWAFULL
R-0						R-0	R-0
7	6	5	4	3	2	1	0
Reserved	SHDWBMODE	Reserved	SHDWAMODE	LOADBMODE		LOADAMODE	
R-0	R/W-0	R-0	R/W-0	R/W-0		R/W-0	

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

Table 3-30. Counter-Compare Control (CMPCTL) Register Field Descriptions

Bits	Name	Value	Description
15-10	Reserved		Reserved
9	SHDWBFULL		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)

Table 3-30. Counter-Compare Control (CMPCTL) Register Field Descriptions (continued)

Bits	Name	Value	Description
1-0	LOADAMODE		Active Counter-Compare A (CMPA) Load From Shadow Select Mode. This bit has no effect in immediate mode (CMPCTL[SHDWAMODE] = 1).
		00	Load on CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000)
		01	Load on CTR = PRD: Time-base counter equal to period (TBCTR = TBPRD)
		10	Load on either CTR = Zero or CTR = PRD
		11	Freeze (no loads possible)

3.4.2.2 Compare A High Resolution (CMPAHR) Register

Figure 3-82. Compare A High Resolution (CMPAHR) Register

15	8
CMPAHR	
R/W-0	

7	0
Reserved	
R-0	

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

Table 3-31. Compare A High Resolution (CMPAHR) Register Field Descriptions

Bit	Field	Value	Description
15-8	CMPAHR	00-FFh	These 8-bits contain the high-resolution portion (least significant 8-bits) of the counter-compare A value. CMPA:CMPAHR can be accessed in a single 32-bit read/write. Shadowing is enabled and disabled by the CMPCTL[SHDWAMODE] bit as described for the CMPA register.
7-0	Reserved		Reserved for TI Test

3.4.2.3 Counter-Compare A (CMPA) Register

Figure 3-83. Counter-Compare A (CMPA) Register

15	0
CMPA	
R/W-0	

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

Table 3-32. Counter-Compare A (CMPA) Register Field Descriptions

Bits	Name	Description
15-0	CMPA	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.

3.4.2.4 Counter-Compare B (CMPB) Register

Figure 3-84. Counter-Compare B (CMPB) Register

15	0
CMPB	
R/W-0	

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

Table 3-33. Counter-Compare B (CMPB) Register Field Descriptions

Bits	Name	Description
15-0	CMPB	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.

3.4.2.5 Compare A High-Resolution Mirror (CMPAHRM) Register

Figure 3-85. Compare A High-Resolution Mirror (CMPAHRM) Register

15							8
CMPAHR							
R/W-0							
7							0
Reserved							
R-0							

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

Table 3-34. Compare A High-Resolution Mirror (CMPAHRM) Register Field Descriptions

Bit	Field	Value	Description
15-8	CMPAHR	00-FFh	Compare A High Resolution Bits Writes to both the CMPAHR and CMPAHRM locations access the high-resolution (least significant 8-bit) portion of the Counter Compare A value. The only difference is that unlike CMPAHR, reads from the mirror register, CMPAHRM, are indeterminate (reserved for TI Test). By default writes to this register are shadowed. Shadowing is enabled and disabled by the CMPCTL[SHDWAMODE] bit as described for the CMPAM register.
7-0	Reserved		Reserved for TI Test

3.4.2.6 Counter-Compare A Mirror (CMPAM) Register

Figure 3-86. Counter-Compare A Mirror (CMPAM) Register

15	0
CMPA	
R/W-0	

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

Table 3-35. Counter-Compare A Mirror (CMPAM) Register Field Descriptions

Bit	Field	Value	Description
15-0	CMPA	0000-FFFFh	CMPA and CMPAM can both be used to access the counter-compare A value. The only difference is that the mirror register always reads back the active value. By default writes to this register are shadowed. Unlike the CMPA register, reads of CMPAM always return the active register value. Shadowing is enabled and disabled by the CMPCTL[SHDWAMODE] bit. • If CMPCTL[SHDWAMODE] = 0, then the shadow is enabled and any write will automatically go to the shadow register. All reads will reflect the active register value. In this case, the CMPCTL[LOADAMODE] bit field determines which event will load the active register from the shadow register. • Before a write, the CMPCTL[SHDWAFULL] bit can be read to determine if the shadow register is currently full. • If CMPCTL[SHDWAMODE] = 1, then the shadow register is disabled and any write will go directly to the active register, that is the register actively controlling the hardware.

3.4.3 Action-Qualifier Submodule Registers

This section describes the action-qualifier submodule registers.

3.4.3.1 Action-Qualifier Output A Control Register (AQCTLA)

Figure 3-87. Action-Qualifier Output A Control Register (AQCTLA)

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

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

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

Table 3-36. 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.

Table 3-36. Action-Qualifier Output A Control Register (AQCTLA) Field Descriptions (continued)

Bits	Name	Value	Description
1-0	ZRO		Action when counter equals zero.
			Note: By definition, in count up-down mode when the counter equals 0 the direction is defined as 1 or counting up.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxA output low.
		10	Set: force EPWMxA output high.
		11	Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.

3.4.3.2 Action-Qualifier Output B Control Register (AQCTLB)

Figure 3-88. Action-Qualifier Output B Control Register (AQCTLB)

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

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

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

Table 3-37. Action-Qualifier Output B Control Register (AQCTLB) Field Descriptions

Bits	Name	Value	Description
15-12	Reserved		
11-10	CBD		Action when the counter equals the active CMPB register and the counter is decrementing.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxB output low.
		10	Set: force EPWMxB output high.
		11	Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.
9-8	CBU		Action when the counter equals the active CMPB register and the counter is incrementing.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxB output low.
		10	Set: force EPWMxB output high.
		11	Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.
7-6	CAD		Action when the counter equals the active CMPA register and the counter is decrementing.
		00	Do nothing (action disabled)
		01	Clear: force EPWMxB output low.
		10	Set: force EPWMxB output high.
		11	Toggle EPWMxB output: low output signal will be forced high, and a high signal will be forced low.

Table 3-37. Action-Qualifier Output B Control Register (AQCTLB) Field Descriptions (continued)

Bits	Name	Value	Description
5-4	CAU		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.

3.4.3.3 Action-Qualifier Software Force Register (AQSFRC)

Figure 3-89. Action-Qualifier Software Force Register (AQSFRC)

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
RLDCSF		OTSFB	ACTSFB		OTSFA	ACTSFA	
R/W-0		R/W-0	R/W-0		R/W-0	R/W-0	

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

Table 3-38. 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 (that is, a forced event is initiated).
			This is a one-shot forced event. It can be overridden by another subsequent event on output B.
		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 (that is, 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)

3.4.3.4 Action-Qualifier Continuous Software Force Register (AQCSFRC)

Figure 3-90. 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 3-39. 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

This section describes the dead-band submodule registers.

3.4.4.1 Dead-Band Generator Control (DBCTL) Register

Figure 3-91. Dead-Band Generator Control (DBCTL) Register

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

7	6	5	4	3	2	1	0
Reserved		IN_MODE		POLSEL		OUT_MODE	
R-0		R/W-0		R/W-0		R/W-0	

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

Table 3-40. Dead-Band Generator Control (DBCTL) Register Field Descriptions

Bits	Name	Value	Description
15	HALFCYCLE		Half Cycle Clocking Enable Bit:
		0	Full cycle clocking enabled. The dead-band counters are clocked at the TBCLK rate.
		1	Half cycle clocking enabled. The dead-band counters are clocked at TBCLK*2.
14-6	Reserved		Reserved
5-4	IN_MODE		Dead Band Input Mode Control Bit 5 controls the S5 switch and bit 4 controls the S4 switch shown in Figure 3-29. 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 3-40. Dead-Band Generator Control (DBCTL) Register 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].

3.4.4.2 Dead-Band Generator Rising Edge Delay (DBRED) Register

Figure 3-92. Dead-Band Generator Rising Edge Delay (DBRED) Register

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 3-41. Dead-Band Generator Rising Edge Delay (DBRED) Register Field Descriptions

Bits	Name	Value	Description
15-10	Reserved		Reserved
9-0	DEL		Rising Edge Delay Count. 10-bit counter.

3.4.4.3 Dead-Band Generator Falling Edge Delay (DBFED) Register

Figure 3-93. Dead-Band Generator Falling Edge Delay (DBFED) Register

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 3-42. Dead-Band Generator Falling Edge Delay (DBFED) Register Field Descriptions

Bits	Name	Description
15-10	Reserved	Reserved
9-0	DEL	Falling Edge Delay Count. 10-bit counter

3.4.5 Trip-Zone Submodule Control and Status Registers

This section describes the trip-zone submodule control and status registers.

3.4.5.1 Trip-Zone Select Register (TZSEL)

Figure 3-94. Trip-Zone Select Register (TZSEL)

15	14	13	12	11	10	9	8
DCBEVT1	DCAEVT1	OSHT6	OSHT5	OSHT4	OSHT3	OSHT2	OSHT1
R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
DCBEVT2	DCAEVT2	CBC6	CBC5	CBC4	CBC3	CBC2	CBC1
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 3-43. Trip-Zone Submodule Select Register (TZSEL) Field Descriptions

Bits	Name	Value	Description
One-Shot (OSHT) Trip-zone enable/disable. When any of the enabled pins go low, a one-shot trip event occurs for this ePWM module. When the event occurs, the action defined in the TZCTL register (Section 3.4.5.3) is taken on the EPWMxA and EPWMxB outputs. The one-shot trip condition remains latched until the user clears the condition via the TZCLR register (Section 3.4.5.6).			
15	DCBEVT1		Digital Compare Output B Event 1 Select
		0	Disable DCBEVT1 as one-shot-trip source for this ePWM module.
		1	Enable DCBEVT1 as one-shot-trip source for this ePWM module.
14	DCAEVT1		Digital Compare Output A Event 1 Select
		0	Disable DCAEVT1 as one-shot-trip source for this ePWM module.
		1	Enable DCAEVT1 as one-shot-trip source for this ePWM module.
13	OSHT6		Trip-zone 6 ($\overline{TZ6}$) 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

Table 3-43. Trip-Zone Submodule Select Register (TZSEL) Field Descriptions (continued)

Bits	Name	Value	Description
Cycle-by-Cycle (CBC) Trip-zone enable/disable. When any of the enabled pins go low, a cycle-by-cycle trip event occurs for this ePWM module. When the event occurs, the action defined in the TZCTL register (Section 3.4.5.3) is taken on the EPWMxA and EPWMxB outputs. A cycle-by-cycle trip condition is automatically cleared when the time-base counter reaches zero.			
7	DCBEVT2		Digital Compare Output B Event 2 Select
		0	Disable DCBEVT2 as a CBC trip source for this ePWM module
		1	Enable DCBEVT2 as a CBC trip source for this ePWM module
6	DCAEVT2		Digital Compare Output A Event 2 Select
		0	Disable DCAEVT2 as a CBC trip source for this ePWM module
		1	Enable DCAEVT2 as a CBC trip source for this ePWM module
5	CBC6		Trip-zone 6 ($\overline{TZ6}$) Select
		0	Disable $\overline{TZ6}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ6}$ as a CBC trip source for this ePWM module
4	CBC5		Trip-zone 5 ($\overline{TZ5}$) Select
		0	Disable $\overline{TZ5}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ5}$ as a CBC trip source for this ePWM module
3	CBC4		Trip-zone 4 ($\overline{TZ4}$) Select
		0	Disable $\overline{TZ4}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ4}$ as a CBC trip source for this ePWM module
2	CBC3		Trip-zone 3 ($\overline{TZ3}$) Select
		0	Disable $\overline{TZ3}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ3}$ as a CBC trip source for this ePWM module
1	CBC2		Trip-zone 2 ($\overline{TZ2}$) Select
		0	Disable $\overline{TZ2}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ2}$ as a CBC trip source for this ePWM module
0	CBC1		Trip-zone 1 ($\overline{TZ1}$) Select
		0	Disable $\overline{TZ1}$ as a CBC trip source for this ePWM module
		1	Enable $\overline{TZ1}$ as a CBC trip source for this ePWM module

3.4.5.2 Trip Zone Digital Compare Event Select Register (TZDCSEL)

Figure 3-95. Trip Zone Digital Compare Event Select Register (TZDCSEL)

15 12	11 9	8 6	5 3	2 0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1
R-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 3-44. Trip Zone Digital Compare Event Select Register (TZDCSEL) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Reserved

Table 3-44. Trip Zone Digital Compare Event Select Register (TZDCSEL) Field Descriptions (continued)

Bit	Field	Value	Description
11-9	DCBEVT2		Digital Compare Output B Event 2 Selection
		000	Event disabled
		001	DCBH = low, DCBL = don't care
		010	DCBH = high, DCBL = don't care
		011	DCBL = low, DCBH = don't care
		100	DCBL = high, DCBH = don't care
		101	DCBL = high, DCBH = low
		110	reserved
		111	reserved
8-6	DCBEVT1		Digital Compare Output B Event 1 Selection
		000	Event disabled
		001	DCBH = low, DCBL = don't care
		010	DCBH = high, DCBL = don't care
		011	DCBL = low, DCBH = don't care
		100	DCBL = high, DCBH = don't care
		101	DCBL = high, DCBH = low
		110	reserved
		111	reserved
5-3	DCAEVT2		Digital Compare Output A Event 2 Selection
		000	Event disabled
		001	DCAH = low, DCAL = don't care
		010	DCAH = high, DCAL = don't care
		011	DCAL = low, DCAH = don't care
		100	DCAL = high, DCAH = don't care
		101	DCAL = high, DCAH = low
		110	reserved
		111	reserved
2-0	DCAEVT1		Digital Compare Output A Event 1 Selection
		000	Event disabled
		001	DCAH = low, DCAL = don't care
		010	DCAH = high, DCAL = don't care
		011	DCAL = low, DCAH = don't care
		100	DCAL = high, DCAH = don't care
		101	DCAL = high, DCAH = low
		110	reserved
		111	reserved

3.4.5.3 Trip-Zone Control Register (TZCTL)

Figure 3-96. Trip-Zone Control Register (TZCTL)

15	12	11	10	9	8
Reserved		DCBEVT2		DCBEVT1	
R-0		R/W-0		R/W-0	

7	6	5	4	3	2	1	0
DCAEVT2		DCAEVT1		TZB		TZA	
R/W-0		R/W-0		R/W-0		R/W-0	

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

Table 3-45. Trip-Zone Control Register Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Reserved
11-10	DCBEVT2		Digital Compare Output B Event 2 Action On EPWMxB:
		00	High-impedance (EPWMxB = High-impedance state)
		01	Force EPWMxB to a high state.
		10	Force EPWMxB to a low state.
		11	Do Nothing, trip action is disabled
9-8	DCBEVT1		Digital Compare Output B Event 1 Action On EPWMxB:
		00	High-impedance (EPWMxB = High-impedance state)
		01	Force EPWMxB to a high state.
		10	Force EPWMxB to a low state.
		11	Do Nothing, trip action is disabled
7-6	DCAEVT2		Digital Compare Output A Event 2 Action On EPWMxA:
		00	High-impedance (EPWMxA = High-impedance state)
		01	Force EPWMxA to a high state.
		10	Force EPWMxA to a low state.
		11	Do Nothing, trip action is disabled
5-4	DCAEVT1		Digital Compare Output A Event 1 Action On EPWMxA:
		00	High-impedance (EPWMxA = High-impedance state)
		01	Force EPWMxA to a high state.
		10	Force EPWMxA to a low state.
		11	Do Nothing, trip action is disabled
3-2	TZB		When a trip event occurs the following action is taken on output EPWMxB. Which trip-zone pins can cause an event is defined in the TZSEL register.
		00	High-impedance (EPWMxB = High-impedance state)
		01	Force EPWMxB to a high state
		10	Force EPWMxB to a low state
		11	Do nothing, no action is taken on EPWMxB.

Table 3-45. Trip-Zone Control Register Field Descriptions (continued)

Bit	Field	Value	Description
1-0	TZA		When a trip event occurs the following action is taken on output EPWMxA. Which trip-zone pins can cause an event is defined in the TZSEL register.
		00	High-impedance (EPWMxA = High-impedance state)
		01	Force EPWMxA to a high state
		10	Force EPWMxA to a low state
		11	Do nothing, no action is taken on EPWMxA.

3.4.5.4 Trip-Zone Enable Interrupt Register (TZEINT)

Figure 3-97. Trip-Zone Enable Interrupt Register (TZEINT)

15							8
Reserved							
R -0							
7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	Reserved
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

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

Table 3-46. Trip-Zone Enable Interrupt Register (TZEINT) Field Descriptions

Bits	Name	Value	Description
15-3	Reserved		Reserved
6	DCBEVT2		Digital Comparator Output B Event 2 Interrupt Enable
		0	Disabled
		1	Enabled
5	DCBEVT1		Digital Comparator Output B Event 1 Interrupt Enable
		0	Disabled
		1	Enabled
4	DCAEVT2		Digital Comparator Output A Event 2 Interrupt Enable
		0	Disabled
		1	Enabled
3	DCAEVT1		Digital Comparator Output A Event 1 Interrupt Enable
		0	Disabled
		1	Enabled
2	OST		Trip-zone One-Shot Interrupt Enable
		0	Disable one-shot interrupt generation
		1	Enable Interrupt generation; a one-shot trip event will cause a EPWMx_TZINT PIE interrupt.
1	CBC		Trip-zone Cycle-by-Cycle Interrupt Enable
		0	Disable cycle-by-cycle interrupt generation.
		1	Enable interrupt generation; a cycle-by-cycle trip event will cause an EPWMx_TZINT PIE interrupt.
0	Reserved		Reserved

3.4.5.5 Trip-Zone Flag Register (TZFLG)

Figure 3-98. Trip-Zone Flag Register (TZFLG)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	INT
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 3-47. Trip-Zone Flag Register Field Descriptions

Bit	Field	Value	Description
15-7	Reserved		Reserved
6	DCBEVT2		Latched Status Flag for Digital Compare Output B Event 2
		0	Indicates no trip event has occurred on DCBEVT2
		1	Indicates a trip event has occurred for the event defined for DCBEVT2
5	DCBEVT1		Latched Status Flag for Digital Compare Output B Event 1
		0	Indicates no trip event has occurred on DCBEVT1
		1	Indicates a trip event has occurred for the event defined for DCBEVT1
4	DCAEVT2		Latched Status Flag for Digital Compare Output A Event 2
		0	Indicates no trip event has occurred on DCAEVT2
		1	Indicates a trip event has occurred for the event defined for DCAEVT2
3	DCAEVT1		Latched Status Flag for Digital Compare Output A Event 1
		0	Indicates no trip event has occurred on DCAEVT1
		1	Indicates a trip event has occurred for the event defined for DCAEVT1
2	OST		Latched Status Flag for A One-Shot Trip Event
		0	No one-shot trip event has occurred.
		1	Indicates a trip event has occurred on a pin selected as a one-shot trip source. This bit is cleared by writing the appropriate value to the TZCLR register .
1	CBC		Latched Status Flag for Cycle-By-Cycle Trip Event
		0	No cycle-by-cycle trip event has occurred.
		1	Indicates a trip event has occurred on a signal selected as a cycle-by-cycle trip source. The TZFLG[CBC] bit will remain set until it is manually cleared by the user. If the cycle-by-cycle trip event is still present when the CBC bit is cleared, then CBC will be immediately set again. The specified condition on the signal is automatically cleared when the ePWM time-base counter reaches zero (TBCTR = 0x0000) if the trip condition is no longer present. The condition on the signal is only cleared when the TBCTR = 0x0000 no matter where in the cycle the CBC flag is cleared. This bit is cleared by writing the appropriate value to the TZCLR register .

Table 3-47. Trip-Zone Flag Register Field Descriptions (continued)

Bit	Field	Value	Description
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 .

3.4.5.6 Trip-Zone Clear Register (TZCLR)

Figure 3-99. Trip-Zone Clear Register (TZCLR)

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	INT
R-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W-0	R/W-0	R/W-0

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

Table 3-48. Trip-Zone Clear Register (TZCLR) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved		Reserved
6	DCBEVT2		Clear Flag for Digital Compare Output B Event 2
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCBEVT2 event trip condition.
5	DCBEVT1		Clear Flag for Digital Compare Output B Event 1
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCBEVT1 event trip condition.
4	DCAEVT2		Clear Flag for Digital Compare Output A Event 2
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCAEVT2 event trip condition.
3	DCAEVT1		Clear Flag for Digital Compare Output A Event 1
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 clears the DCAEVT1 event trip condition.
2	OST		Clear Flag for One-Shot Trip (OST) Latch
		0	Has no effect. Always reads back a 0.
		1	Clears this Trip (set) condition.
1	CBC		Clear Flag for Cycle-By-Cycle (CBC) Trip Latch
		0	Has no effect. Always reads back a 0.
		1	Clears this Trip (set) condition.
0	INT		Global Interrupt Clear Flag
		0	Has no effect. Always reads back a 0.
		1	Clears the trip-interrupt flag for this ePWM module (TZFLG[INT]).
			NOTE: No further EPWMx_TZINT PIE interrupts will be generated until the flag is cleared. If the TZFLG[INT] bit is cleared and any of the other flag bits are set, then another interrupt pulse will be generated. Clearing all flag bits will prevent further interrupts.

3.4.5.7 Trip-Zone Force Register (TZFRC)

Figure 3-100. Trip-Zone Force Register (TZFRC)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
Reserved	DCBEVT2	DCBEVT1	DCAEVT2	DCAEVT1	OST	CBC	Reserved
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R- 0

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

Table 3-49. Trip-Zone Force Register (TZFRC) Field Descriptions

Bits	Name	Value	Description
15-7	Reserved		Reserved
6	DCBEVT2		Force Flag for Digital Compare Output B Event 2
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 forces the DCBEVT2 event trip condition and sets the TZFLG[DCBEVT2] bit.
5	DCBEVT1		Force Flag for Digital Compare Output B Event 1
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 forces the DCBEVT1 event trip condition and sets the TZFLG[DCBEVT1] bit.
4	DCAEVT2		Force Flag for Digital Compare Output A Event 2
		0	Writing 0 has no effect. This bit always reads back 0.
		1	Writing 1 forces the DCAEVT2 event trip condition and sets the TZFLG[DCAEVT2] bit.
3	DCAEVT1		Force Flag for Digital Compare Output A Event 1
		0	Writing 0 has no effect. This bit always reads back 0
		1	Writing 1 forces the DCAEVT1 event trip condition and sets the TZFLG[DCAEVT1] bit.
2	OST		Force a One-Shot Trip Event via Software
		0	Writing of 0 is ignored. Always reads back a 0.
		1	Forces a one-shot trip event and sets the TZFLG[OST] bit.
1	CBC		Force a Cycle-by-Cycle Trip Event via Software
		0	Writing of 0 is ignored. Always reads back a 0.
		1	Forces a cycle-by-cycle trip event and sets the TZFLG[CBC] bit.
0	Reserved		Reserved

3.4.6 Event-Trigger Submodule Registers

This section describes the event-trigger submodule registers.

3.4.6.1 Event-Trigger Selection Register (ETSEL)

Figure 3-101. Event-Trigger Selection Register (ETSEL)

15	14	12	11	10	8
SOCBEN	SOCBSEL		SOCAEN	SOCASEL	
R/W-0	R/W-0		R/W-0	R/W-0	

7	4	3	2	0
Reserved		INTEN	INTSEL	
R-0		R/W-0	R/W-0	

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

Table 3-50. Event-Trigger Selection Register (ETSEL) Field Descriptions

Bits	Name	Value	Description
15	SOCBEN		Enable the ADC Start of Conversion B (EPWMxSOCB) Pulse
		0	Disable EPWMxSOCB.
		1	Enable EPWMxSOCB pulse.
14-12	SOCBSEL		EPWMxSOCB Selection Options
			These bits determine when a EPWMxSOCB pulse will be generated.
		000	Enable DCBEVT1.soc event
		001	Enable event time-base counter equal to zero. (TBCTR = 0x0000)
		010	Enable event time-base counter equal to period (TBCTR = TBPRD)
		011	Enable event time-base counter equal to zero or period (TBCTR = 0x0000 or TBCTR = TBPRD). This mode is useful in up-down count mode.
		100	Enable event time-base counter equal to CMPA when the timer is incrementing.
		101	Enable event time-base counter equal to CMPA when the timer is decrementing.
		110	Enable event: time-base counter equal to CMPB when the timer is incrementing.
		111	Enable event: time-base counter equal to CMPB when the timer is decrementing.
11	SOCAEN		Enable the ADC Start of Conversion A (EPWMxSOCA) Pulse
		0	Disable EPWMxSOCA.
		1	Enable EPWMxSOCA pulse.
10-8	SOCASEL		EPWMxSOCA Selection Options
			These bits determine when a EPWMxSOCA pulse will be generated.
		000	Enable DCAEVT1.soc event
		001	Enable event time-base counter equal to zero. (TBCTR = 0x0000)
		010	Enable event time-base counter equal to period (TBCTR = TBPRD)
		011	Enable event time-base counter equal to zero or period (TBCTR = 0x0000 or TBCTR = TBPRD). This mode is useful in up-down count mode.
		100	Enable event time-base counter equal to CMPA when the timer is incrementing.
		101	Enable event time-base counter equal to CMPA when the timer is decrementing.
		110	Enable event: time-base counter equal to CMPB when the timer is incrementing.
		111	Enable event: time-base counter equal to CMPB when the timer is decrementing.
7-4	Reserved		Reserved

Table 3-50. Event-Trigger Selection Register (ETSEL) Field Descriptions (continued)

Bits	Name	Value	Description
3	INTEN		Enable ePWM Interrupt (EPWMx_INT) Generation
		0	Disable EPWMx_INT generation
		1	Enable EPWMx_INT generation
2-0	INTSEL		ePWM Interrupt (EPWMx_INT) Selection Options
		000	Reserved
		001	Enable event time-base counter equal to zero. (TBCTR = 0x0000)
		010	Enable event time-base counter equal to period (TBCTR = TBPRD)
		011	Enable event time-base counter equal to zero or period (TBCTR = 0x0000 or TBCTR = TBPRD). This mode is useful in up-down count mode.
		100	Enable event time-base counter equal to CMPA when the timer is incrementing.
		101	Enable event time-base counter equal to CMPA when the timer is decrementing.
		110	Enable event: time-base counter equal to CMPB when the timer is incrementing.
		111	Enable event: time-base counter equal to CMPB when the timer is decrementing.

3.4.6.2 Event-Trigger Prescale Register (ETPS)

Figure 3-102. Event-Trigger Prescale Register (ETPS)

15	14	13	12	11	10	9	8
SOCBCNT		SOCBPRD		SOCACNT		SOCAPRD	
R-0		R/W-0		R-0		R/W-0	
7			4	3	2	1	0
Reserved				INTCNT		INTPRD	
R-0				R-0		R/W-0	

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

Table 3-51. Event-Trigger Prescale Register (ETPS) Field Descriptions

Bits	Name		Description
15-14	SOCBCNT		ePWM ADC Start-of-Conversion B Event (EPWMxSOCB) Counter Register
			These bits indicate how many selected ETSEL[SOCBSEL] events have occurred:
		00	No events have occurred.
		01	1 event has occurred.
		10	2 events have occurred.
		11	3 events have occurred.
13-12	SOCBPRD		ePWM ADC Start-of-Conversion B Event (EPWMxSOCB) Period Select
			These bits determine how many selected ETSEL[SOCBSEL] events need to occur before an EPWMxSOCB pulse is generated. To be generated, the pulse must be enabled (ETSEL[SOCBEN] = 1). The SOCB pulse will be generated even if the status flag is set from a previous start of conversion (ETFLG[SOCB] = 1). Once the SOCB pulse is generated, the ETPS[SOCBCNT] bits will automatically be cleared.
		00	Disable the SOCB event counter. No EPWMxSOCB pulse will be generated
		01	Generate the EPWMxSOCB pulse on the first event: ETPS[SOCBCNT] = 0,1
		10	Generate the EPWMxSOCB pulse on the second event: ETPS[SOCBCNT] = 1,0
		11	Generate the EPWMxSOCB pulse on the third event: ETPS[SOCBCNT] = 1,1

Table 3-51. Event-Trigger Prescale Register (ETPS) Field Descriptions (continued)

Bits	Name		Description
11-10	SOCACNT		ePWM ADC Start-of-Conversion A Event (EPWMxSOCA) Counter Register
			These bits indicate how many selected ETSEL[SOCASEL] events have occurred:
		00	No events have occurred.
		01	1 event has occurred.
		10	2 events have occurred.
		11	3 events have occurred.
9-8	SOCAPRD		ePWM ADC Start-of-Conversion A Event (EPWMxSOCA) Period Select
			These bits determine how many selected ETSEL[SOCASEL] events need to occur before an EPWMxSOCA pulse is generated. To be generated, the pulse must be enabled (ETSEL[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.
1-0	INTPRD		ePWM Interrupt (EPWMx_INT) Period Select
			These bits determine how many selected ETSEL[INTSEL] events need to occur before an interrupt is generated. To be generated, the interrupt must be enabled (ETSEL[INT] = 1). If the interrupt status flag is set from a previous interrupt (ETFLG[INT] = 1) then no interrupt will be generated until the flag is cleared via the ETCLR[INT] bit. This allows for one interrupt to be pending while another is still being serviced. Once the interrupt is generated, the ETPS[INTCNT] bits will automatically be cleared.
			Writing a INTPRD value that is the same as the current counter value will trigger an interrupt if it is enabled and the status flag is clear.
			Writing a INTPRD value that is less than the current counter value will result in an undefined state.
			If a counter event occurs at the same instant as a new zero or non-zero INTPRD value is written, the counter is incremented.
		00	Disable the interrupt event counter. No interrupt will be generated and ETFRC[INT] is ignored.
		01	Generate an interrupt on the first event INTCNT = 01 (first event)
		10	Generate interrupt on ETPS[INTCNT] = 1,0 (second event)
		11	Generate interrupt on ETPS[INTCNT] = 1,1 (third event)

3.4.6.3 Event-Trigger Flag Register (ETFLG)

Figure 3-103. Event-Trigger Flag Register (ETFLG)

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 3-52. 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. See Figure 3-42.

3.4.6.4 Event-Trigger Clear Register (ETCLR)

Figure 3-104. 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 3-53. 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

3.4.6.5 Event-Trigger Force Register (ETFRC)

Figure 3-105. 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 3-54. 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.

3.4.7 PWM-Chopper Submodule Control Register

This section describes the PWM-chopper submodule control register.

3.4.7.1 PWM-Chopper Control (PCCTL) Register

Figure 3-106. PWM-Chopper Control (PCCTL) Register

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 3-55. PWM-Chopper Control (PCCTL) Register Field Descriptions

Bits	Name	Value	Description
15-11	Reserved		Reserved
10-8	CHPDUTY		Chopping Clock Duty Cycle
		000	Duty = 1/8 (12.5%)
		001	Duty = 2/8 (25.0%)
		010	Duty = 3/8 (37.5%)
		011	Duty = 4/8 (50.0%)
		100	Duty = 5/8 (62.5%)
		101	Duty = 6/8 (75.0%)
		110	Duty = 7/8 (87.5%)
		111	Reserved
7-5	CHPFREQ		Chopping Clock Frequency
		000	Divide by 1 (no prescale, = 11.25 MHz at 90 MHz SYSCLKOUT)
		001	Divide by 2 (5.63 MHz at 90 MHz SYSCLKOUT)
		010	Divide by 3 (3.75 MHz at 90 MHz SYSCLKOUT)
		011	Divide by 4 (2.81 MHz at 90 MHz SYSCLKOUT)
		100	Divide by 5 (2.25 MHz at 90 MHz SYSCLKOUT)
		101	Divide by 6 (1.88 MHz at 90 MHz SYSCLKOUT)
		110	Divide by 7 (1.61 MHz at 90 MHz SYSCLKOUT)
		111	Divide by 8 (1.41 MHz at 90 MHz SYSCLKOUT)

Table 3-55. PWM-Chopper Control (PCCTL) Register Field Descriptions (continued)

Bits	Name	Value	Description
4-1	OSHTWTH		One-Shot Pulse Width
		0000	1 x SYSCLKOUT / 8 wide (= 133.3 nS at 60 MHz SYSCLKOUT)
		0001	2 x SYSCLKOUT / 8 wide (= 266.7 nS at 60 MHz SYSCLKOUT)
		0010	3 x SYSCLKOUT / 8 wide (= 400 nS at 60 MHz SYSCLKOUT)
		0011	4 x SYSCLKOUT / 8 wide (= 533.3 nS at 60 MHz SYSCLKOUT)
		0100	5 x SYSCLKOUT / 8 wide (= 666.7 nS at 60 MHz SYSCLKOUT)
		0101	6 x SYSCLKOUT / 8 wide (= 800 nS at 60 MHz SYSCLKOUT)
		0110	7 x SYSCLKOUT / 8 wide (= 933.3 nS at 60 MHz SYSCLKOUT)
		0111	8 x SYSCLKOUT / 8 wide (= 1067 nS at 60 MHz SYSCLKOUT)
		1000	9 x SYSCLKOUT / 8 wide (= 1200 nS at 60 MHz SYSCLKOUT)
		1001	10 x SYSCLKOUT / 8 wide (= 1333 nS at 60 MHz SYSCLKOUT)
		1010	11 x SYSCLKOUT / 8 wide (= 1467 nS at 60 MHz SYSCLKOUT)
		1011	12 x SYSCLKOUT / 8 wide (= 1600 nS at 60 MHz SYSCLKOUT)
		1100	13 x SYSCLKOUT / 8 wide (= 1733 nS at 60 MHz SYSCLKOUT)
		1101	14 x SYSCLKOUT / 8 wide (= 1867 nS at 60 MHz SYSCLKOUT)
		1110	15 x SYSCLKOUT / 8 wide (= 2000 nS at 60 MHz SYSCLKOUT)
		1111	16 x SYSCLKOUT / 8 wide (= 2133 nS at 60 MHz SYSCLKOUT)
0	CHPEN		PWM-chopping Enable
		0	Disable (bypass) PWM chopping function
		1	Enable chopping function

3.4.8 Digital Compare Submodule Registers

This section describes the digital compare submodule registers.

3.4.8.1 Digital Compare Trip Select (DCTRIPSEL) Register

Figure 3-107. Digital Compare Trip Select (DCTRIPSEL) Register

15	12	11	8
DCBLCOMPSEL		DCBHCOMPSEL	
R/W-0		R/W-0	

7	4	3	0
DCALCOMPSEL		DCAHCOMPSEL	
R/W-0		R/W-0	

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

Table 3-56. Digital Compare Trip Select (DCTRIPSEL) Register Field Descriptions

Bit	Field	Value	Description
15-12	DCBLCOMPSEL		Digital Compare B Low Input Select Defines the source for the DCBL input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.
11-8	DCBHCOMPSEL		Digital Compare B High Input Select Defines the source for the DCBH input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.

Table 3-56. Digital Compare Trip Select (DCTRIPSEL) Register Field Descriptions (continued)

Bit	Field	Value	Description
7-4	DCALCOMPSEL		Digital Compare A Low Input Select Defines the source for the DCAL input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.
3-0	DCAHCOMPSEL		Digital Compare A High Input Select Defines the source for the DCAH input. The TZ signals, when used as trip signals, are treated as normal inputs and can be defined as active high or active low.
		0000	$\overline{TZ1}$ input
		0001	$\overline{TZ2}$ input
		0010	$\overline{TZ3}$ input
		1000	COMP1OUT input
		1001	COMP2OUT input
		1010	COMP3OUT input
			Values not shown are reserved. If a device does not have a particular comparator, then that option is reserved.

3.4.8.2 Digital Compare A Control (DCACTL) Register

Figure 3-108. Digital Compare A Control (DCACTL) Register

15	10	9	8
Reserved		EVT2FRC SYNCSEL	EVT2SRCSEL
R-0		R/W-0	R/W-0

7	4	3	2	1	0
Reserved		EVT1SYNCE	EVT1SOCE	EVT1FRC SYNCSEL	EVT1SRCSEL
R-0		R/W-0	R/W-0	R/W-0	R/W-0

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

Table 3-57. Digital Compare A Control (DCACTL) Register Field Descriptions

Bit	Field	Value	Description
15-10	Reserved		Reserved
9	EVT2FRC SYNCSEL		DCAEVT2 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
8	EVT2SRCSEL		DCAEVT2 Source Signal Select
		0	Source Is DCAEVT2 Signal
		1	Source Is DCEVTFILT Signal
7-4	Reserved		Reserved
3	EVT1SYNCE		DCAEVT1 SYNC, Enable/Disable
		0	SYNC Generation Disabled
		1	SYNC Generation Enabled
2	EVT1SOCE		DCAEVT1 SOC, Enable/Disable
		0	SOC Generation Disabled
		1	SOC Generation Enabled
1	EVT1FRC SYNCSEL		DCAEVT1 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
0	EVT1SRCSEL		DCAEVT1 Source Signal Select
		0	Source Is DCAEVT1 Signal
		1	Source Is DCEVTFILT Signal

3.4.8.3 Digital Compare B Control (DCBCTL) Register

Figure 3-109. Digital Compare B Control (DCBCTL) Register

15	10	9	8
Reserved		EVT2FRC SYNCSEL	EVT2SRCSEL
R-0		R/W-0	R/W-0

7	4	3	2	1	0
Reserved		EVT1SYNCE	EVT1SOCE	EVT1FRC SYNCSEL	EVT1SRCSEL
R-0		R/W-0	R/W-0	R/W-0	R/W-0

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

Table 3-58. Digital Compare B Control (DCBCTL) Register Field Descriptions

Bit	Field	Value	Description
15-10	Reserved		Reserved
9	EVT2FRC SYNCSEL		DCBEVT2 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
8	EVT2SRCSEL		DCBEVT2 Source Signal Select
		0	Source Is DCBEVT2 Signal
		1	Source Is DCEVTFILT Signal
7-4	Reserved		Reserved
3	EVT1SYNCE		DCBEVT1 SYNC, Enable/Disable
		0	SYNC Generation Disabled
		1	SYNC Generation Enabled
2	EVT1SOCE		DCBEVT1 SOC, Enable/Disable
		0	SOC Generation Disabled
		1	SOC Generation Enabled
1	EVT1FRC SYNCSEL		DCBEVT1 Force Synchronization Signal Select
		0	Source Is Synchronous Signal
		1	Source Is Asynchronous Signal
0	EVT1SRCSEL		DCBEVT1 Source Signal Select
		0	Source Is DCBEVT1 Signal
		1	Source Is DCEVTFILT Signal

3.4.8.4 Digital Compare Filter Control (DCFCTL) Register

Figure 3-110. Digital Compare Filter Control (DCFCTL) Register

15	13	12	8
Reserved		Reserved	
R-0		R-0	

7	6	5	4	3	2	1	0
Reserved	Reserved	PULSESEL		BLANKINV	BLANKE	SRCSEL	
R-0	R-0	R/W-0		R/W-0	R/W-0	R/W-0	

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

Table 3-59. Digital Compare Filter Control (DCFCTL) Register Field Descriptions

Bit	Field	Value	Description
15-13	Reserved		Reserved
12-8	Reserved		Reserved for TI Test
7	Reserved		Reserved
6	Reserved		Reserved for TI Test
5-4	PULSESEL		Pulse Select For Blanking & Capture Alignment
		00	Time-base counter equal to period (TBCTR = TBPRD)
		01	Time-base counter equal to zero (TBCTR = 0x0000)
		10	Reserved
		11	Reserved
3	BLANKINV		Blanking Window Inversion
		0	Blanking window not inverted
		1	Blanking window inverted
2	BLANKE		Blanking Window Enable/Disable
		0	Blanking window is disabled
		1	Blanking window is enabled
1-0	SRCSEL		Filter Block Signal Source Select
		00	Source Is DCAEVT1 Signal
		01	Source Is DCAEVT2 Signal
		10	Source Is DCBEVT1 Signal
		11	Source Is DCBEVT2 Signal

3.4.8.5 Digital Compare Capture Control (DCCAPCTL) Register

Figure 3-111. Digital Compare Capture Control (DCCAPCTL) Register

15	8
Reserved	
R-0	

7	2	1	0
Reserved		SHDWMODE	CAPE
R-0		R/W-0	R/W-0

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

Table 3-60. Digital Compare Capture Control (DCCAPCTL) Register Field Descriptions

Bit	Field	Value	Description
15-2	Reserved		Reserved
1	SHDWMODE		TBCTR Counter Capture Shadow Select Mode
		0	Enable shadow mode. The DCCAP active register is copied to shadow register on a TBCTR = TBPRD or TBCTR = zero event as defined by the DCFCTL[PULSESEL] bit. CPU reads of the DCCAP register will return the shadow register contents.
		1	Active Mode. In this mode the shadow register is disabled. CPU reads from the DCCAP register will always return the active register contents.
0	CAPE		TBCTR Counter Capture Enable/Disable
		0	Disable the time-base counter capture.
		1	Enable the time-base counter capture.

3.4.8.6 Digital Compare Filter Offset (DCFOFFSET) Register

Figure 3-112. Digital Compare Filter Offset (DCFOFFSET) Register

15	0
DCOFFSET	
R-0	

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

Table 3-61. Digital Compare Filter Offset (DCFOFFSET) Register Field Descriptions

Bit	Field	Value	Description
15-0	OFFSET	0000- FFFFh	Blanking Window Offset These 16-bits specify the number of TBCLK cycles from the blanking window reference to the point when the blanking window is applied. The blanking window reference is either period or zero as defined by the DCFCTL[PULSESEL] bit. This offset register is shadowed and the active register is loaded at the reference point defined by DCFCTL[PULSESEL]. The offset counter is also initialized and begins to count down when the active register is loaded. When the counter expires, the blanking window is applied. If the blanking window is currently active, then the blanking window counter is restarted.

3.4.8.7 Digital Compare Filter Offset Counter (DCFOFFSETCNT) Register

Figure 3-113. Digital Compare Filter Offset Counter (DCFOFFSETCNT) Register

15	0
OFFSETCNT	
R-0	

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

Table 3-62. Digital Compare Filter Offset Counter (DCFOFFSETCNT) Register Field Descriptions

Bit	Field	Value	Description
15-0	OFFSETCNT	0000- FFFFh	Blanking Offset Counter These 16-bits are read only and indicate the current value of the offset counter. The counter counts down to zero and then stops until it is re-loaded on the next period or zero event as defined by the DCFCTL[PULSESEL] bit. The offset counter is not affected by the free/soft emulation bits. That is, it will always continue to count down if the device is halted by a emulation stop.

3.4.8.8 Digital Compare Filter Window (DCFWINDOW) Register

Figure 3-114. Digital Compare Filter Window (DCFWINDOW) Register

15	8
Reserved	
R-0	

7	0
WINDOW	
R/W-0	

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

Table 3-63. Digital Compare Filter Window (DCFWINDOW) Register Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		Reserved
7-0	WINDOW		Blanking Window Width
		00h	No blanking window is generated.
		01-FFh	Specifies the width of the blanking window in TBCLK cycles. The blanking window begins when the offset counter expires. When this occurs, the window counter is loaded and begins to count down. If the blanking window is currently active and the offset counter expires, the blanking window counter is restarted.
			The blanking window can cross a PWM period boundary.

3.4.8.9 Digital Compare Filter Window Counter (DCFWINDOWCNT) Register

Figure 3-115. Digital Compare Filter Window Counter (DCFWINDOWCNT) Register

15	8
Reserved	
R-0	

7	0
WINDOWCNT	
R-0	

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

Table 3-64. Digital Compare Filter Window Counter (DCFWINDOWCNT) Register Field Descriptions

Bit	Field	Value	Description
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7-0	WINDOWCNT	00-FF	Blanking Window Counter
			These 8 bits are read only and indicate the current value of the window counter. The counter counts down to zero and then stops until it is re-loaded when the offset counter reaches zero again.

3.4.8.10 Digital Compare Counter Capture (DCCAP) Register

Figure 3-116. Digital Compare Counter Capture (DCCAP) Register

15	0
DCCAP	
R-0	

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

Table 3-65. Digital Compare Counter Capture (DCCAP) Register Field Descriptions

Bit	Field	Value	Description
15-0	DCCAP	0000-FFFFh	Digital Compare Time-Base Counter Capture To enable time-base counter capture, set the DCCAPCLT[CAPE] bit to 1. If enabled, reflects the value of the time-base counter (TBCTR) on the low to high edge transition of a filtered (DCEVTFLT) event. Further capture events are ignored until the next period or zero as selected by the DCFCTL[PULSESEL] bit. Shadowing of DCCAP is enabled and disabled by the DCCAPCTL[SHDWMODE] bit. By default this register is shadowed. • If DCCAPCTL[SHDWMODE] = 0, then the shadow is enabled. In this mode, the active register is copied to the shadow register on the TBCTR = TBPRD or TBCTR = zero as defined by the DCFCTL[PULSESEL] bit. CPU reads of this register will return the shadow register value. • If DCCAPCTL[SHDWMODE] = 1, then the shadow register is disabled. In this mode, CPU reads will return the active register value. The active and shadow registers share the same memory map address.

3.4.9 Proper Interrupt Initialization Procedure

When the ePWM peripheral clock is enabled it may be possible that interrupt flags may be set due to spurious events due to the ePWM registers not being properly initialized. The proper procedure for initializing the ePWM peripheral is as follows:

1. Disable global interrupts (CPU INTM flag)
2. Disable ePWM interrupts
3. Set TBCLKSYNC=0
4. Initialize peripheral registers
5. Set TBCLKSYNC=1
6. Clear any spurious ePWM flags (including PIEIFR)
7. Enable ePWM interrupts
8. Enable global interrupts

This page intentionally left blank.

Chapter 4

High-Resolution Pulse Width Modulator (HRPWM)

This chapter is used in conjunction with Chapter 3. The HRPWM module is a type 1 HRPWM. See the *C2000 Real-Time Control Peripheral Reference Guide* for a list of all devices with an HRPWM module of the same type, to determine the differences between types, and for a list of device-specific differences within a type.

The HRPWM module extends the time resolution capabilities of the conventionally derived digital pulse width modulator (PWM). HRPWM is typically used when PWM resolution falls below ~9-10 bits. The key features of HRPWM are:

- Extended time resolution capability
- Used in both duty cycle and phase-shift control methods
- Finer time granularity control or edge positioning using extensions to the Compare A and Phase registers
- Implemented using the A signal path of PWM, that is, on the EPWMxA output
- Self-check diagnostics software mode to check if the micro edge positioner (MEP) logic is running optimally
- Enables high-resolution output on B signal path of PWM via PWM A and B channel path swapping
- Enables high-resolution output on B signal output via inversion of A signal output
- Enables high-resolution period control on the ePWMxA output on devices with a type 1 ePWM module. See the device-specific data manual to determine if your device has a type 1 ePWM module for high-resolution period support. The ePWMxB output will have a ±1-2 cycle jitter in this mode.

4.1 Introduction

The ePWM peripheral is used to perform a function that is mathematically equivalent to a digital-to-analog converter (DAC). As shown in Figure 4-1, the effective resolution for conventionally generated PWM is a function of PWM frequency (or period) and system clock frequency.

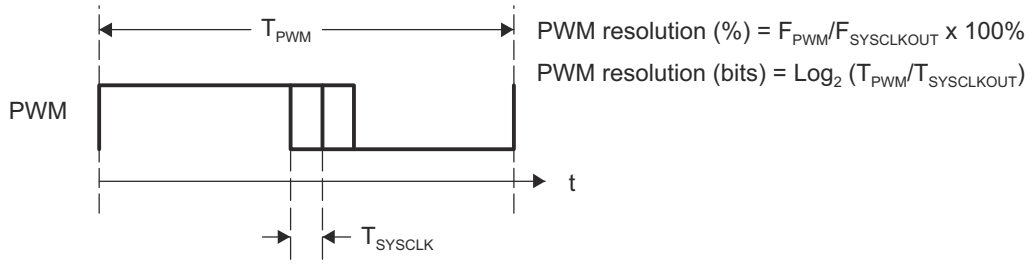


Figure 4-1. Resolution Calculations for Conventionally Generated PWM

If the required PWM operating frequency does not offer sufficient resolution in PWM mode, you may want to consider HRPWM. As an example of improved performance offered by HRPWM, Table 4-1 shows resolution in bits for various PWM frequencies. These values assume a MEP step size of 180 ps. See your device-specific data sheet for typical and maximum performance specifications for the MEP.

Table 4-1. Resolution for PWM and HRPWM

PWM Frequency	Regular Resolution (PWM)				High Resolution (HRPWM)	
	60 MHz SYSCLKOUT		50 MHz SYSCLKOUT			
(kHz)	Bits	%	Bits	%	Bits	%
20	11.6	0.0	11.3	0	18.1	0.000
50	10.2	0.1	10	0.1	16.8	0.001
100	9.2	0.2	9	0.2	15.8	0.002
150	8.6	0.3	8.4	0.3	15.2	0.003
200	8.2	0.3	8	0.4	14.8	0.004
250	7.9	0.4	7.6	0.5	14.4	0.005
500	6.9	0.8	6.6	1	13.4	0.009
1000	5.9	1.7	5.6	2	12.4	0.018
1500	5.3	2.5	5.1	3	11.9	0.027
2000	4.9	3.3	4.6	4	11.4	0.036

Although each application may differ, typical low-frequency PWM operation (below 250 kHz) may not require HRPWM. HRPWM capability is most useful for high-frequency PWM requirements of power conversion topologies such as:

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

4.2 Operational Description of HRPWM

The HRPWM is based on micro edge positioner (MEP) technology. MEP logic is capable of positioning an edge very finely by sub-dividing one coarse system clock of a conventional PWM generator. The time step accuracy is on the order of 150 ps. See the device-specific data sheet for the typical MEP step size on a particular device. The HRPWM also has a self-check software diagnostics mode to check if the MEP logic is running optimally, under all operating conditions. Details on software diagnostics and functions are in Section 4.2.4.

Figure 4-2 shows the relationship between one coarse system clock and edge position in terms of MEP steps, which are controlled with an 8-bit field in the Compare A extension register (CMPAHR).

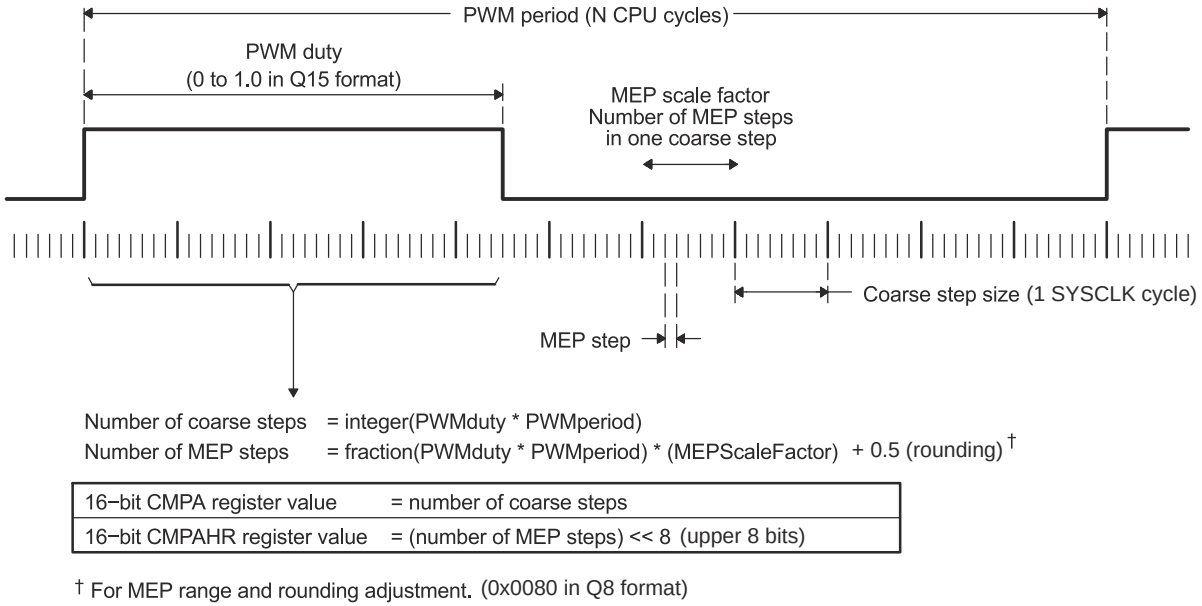


Figure 4-2. Operating Logic Using MEP

To generate an HRPWM waveform, configure the TBM, CCM, and AQM registers as you would to generate a conventional PWM of a given frequency and polarity. The HRPWM works together with the TBM, CCM, and AQM registers to extend edge resolution, and should be configured accordingly. Although many programming combinations are possible, only a few are needed and practical. These methods are described in Section 4.2.5.

Registers discussed but not found in this chapter are in Chapter 3.

The HRPWM operation is controlled and monitored using the registers listed in Table 4-2.

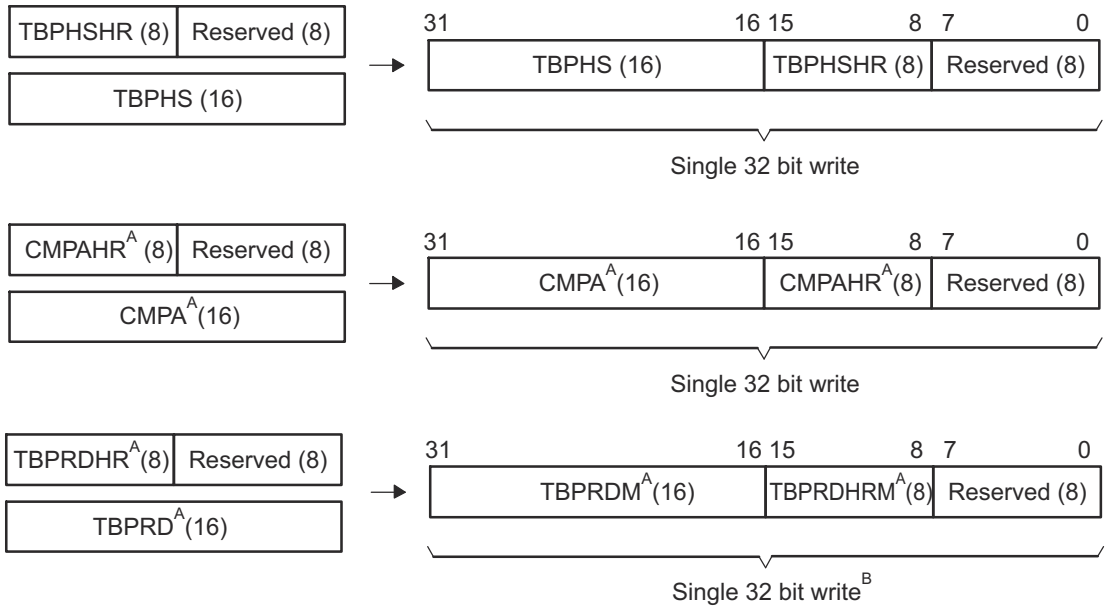
Table 4-2. HRPWM Registers

Register	Address Offset	Shadowed	Description
TBPHSHR	0x0002	No	Extension Register for HRPWM Phase (8 bits)
TBPRDHR	0x0006	Yes	Extension Register for HRPWM Period (8 bits)
CMPAHR	0x0008	Yes	Extension Register for HRPWM Duty (8 bits)
HRCNFG	0x0020	No	HRPWM Configuration Register
HRPWR	0x0021	No	HRPWM Power Register
HRMSTEP	0x0026	No	HRPWM MEP Step Register
HRPCTL	0x0028	No	High Resolution Period Control Register
TBPRDHRM	0x002A	Yes	Extension Mirror Register for HRPWM Period (8 bits)
CMPAHRM	0x002C	Yes	Extension Mirror Register for HRPWM Duty (8 bits)

4.2.1 Controlling the HRPWM Capabilities

The MEP of the HRPWM is controlled by three extension registers, each 8-bits wide. These HRPWM registers are concatenated with the 16-bit TBPHS, TBPRD, and CMPA registers used to control PWM operation, see Figure 4-3.

- TBPHSHR - Time-Base Phase High Resolution Register
- CMPAHR - Counter-Compare A High Resolution Register
- TBPRDHR - Time-Base Period High Resolution Register (available on some devices)

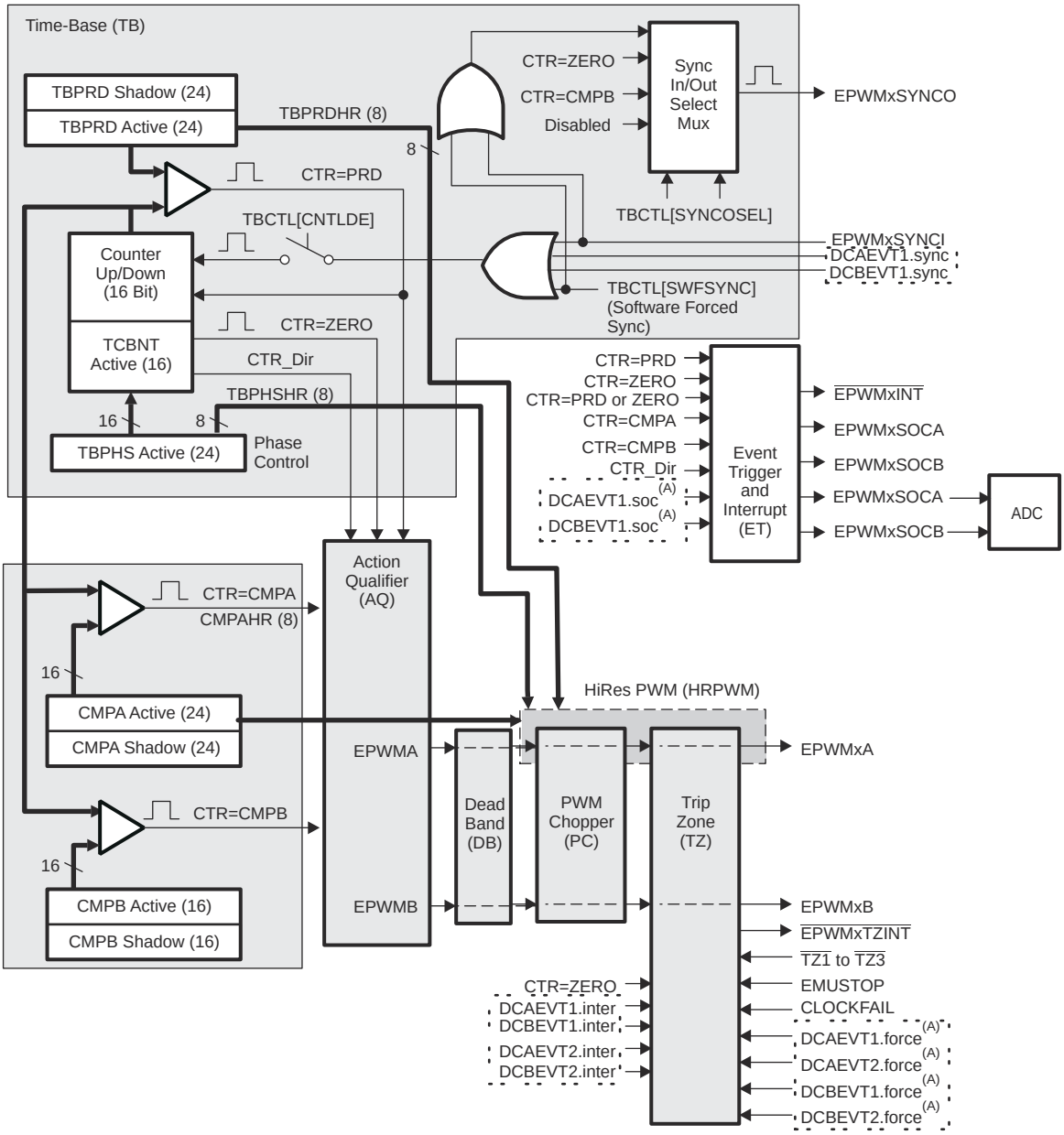


A. These registers are mirrored and can be written to at two different memory locations (mirrored registers have an "M" suffix (that is, CMPA mirror = CMPAM). Reads of the high-resolution mirror registers will result in indeterminate values.

B. TBPRDHR and TBPRD may be written to as a 32-bit value only at the mirrored address
Not all devices may have TBPRD and TBPRDHR registers. See device-specific data sheet for more information

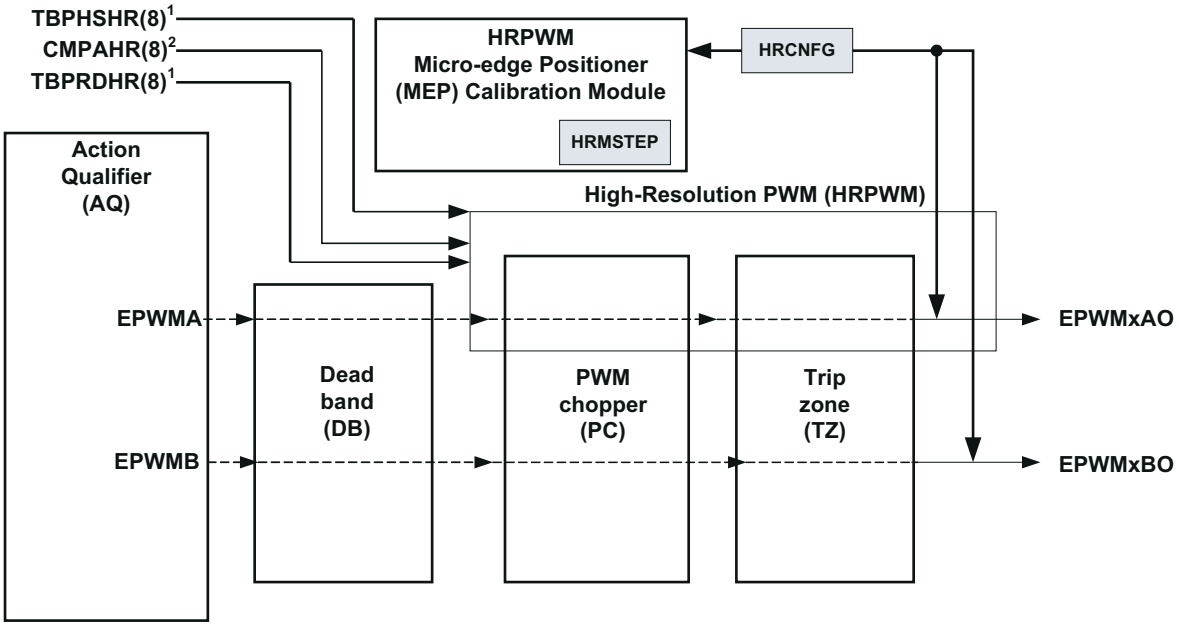
Figure 4-3. HRPWM Extension Registers and Memory Configuration

HRPWM capabilities are controlled using the Channel A PWM signal path. HRPWM support on the channel B signal path is available by properly configuring the HRCNFG register. Figure 4-4 shows how the HRPWM interfaces with the 8-bit extension registers.



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

Figure 4-4. HRPWM System Interface



1. From ePWM time-base (TB) submodule.
2. From ePWM counter-compare (CC) submodule.

Figure 4-5. HRPWM Block Diagram

4.2.2 Configuring the HRPWM

Once the ePWM has been configured to provide conventional PWM of a given frequency and polarity, the HRPWM is configured by programming the HRCNFG register located at offset address 20h. This register provides the following configuration options:

Edge Mode	The MEP can be programmed to provide precise position control on the rising edge (RE), falling edge (FE), or both edges (BE) at the same time. FE and RE are used for power topologies requiring duty cycle control (CMPA high-resolution control), while BE is used for topologies requiring phase shifting, for example, phase shifted full bridge (TBPHS or TBPRD high-resolution control).
Control Mode	The MEP is programmed to be controlled either from the CMPAHR register (duty cycle control) or the TBPHSHR register (phase control). RE or FE control mode should be used with CMPAHR register. BE control mode should be used with TBPHSHR register. When the MEP is controlled from the TBPRDHR register (period control) the duty cycle and phase can also be controlled via their respective high-resolution registers.
Shadow Mode	This mode provides the same shadowing (double buffering) option as in regular PWM mode. This option is valid only when operating from the CMPAHR and TBPRDHR registers and should be chosen to be the same as the regular load option for the CMPA register. If TBPHSHR is used, then this option has no effect.
High-Resolution B Signal Control	The B signal path of an ePWM channel can generate a high-resolution output by either swapping the A and B outputs (the high- resolution signal will appear on ePWMxB instead of ePWMxA) or by outputting an inverted version of the high-resolution ePWMxA signal on the ePWMxB pin.
Auto-conversion Mode	This mode is used in conjunction with the scale factor optimization software only. For a type 1 HRPWM module, if auto-conversion is enabled, CMPAHR = fraction(PWMduty*PWMperiod)<<8. The scale factor optimization software will calculate the MEP scale factor in background code and automatically update the HRMSTEP register with the calculated number of MEP steps per coarse step. The MEP Calibration Module will then use the values in the HRMSTEP and CMPAHR register to automatically calculate the appropriate number of MEP steps represented by the fractional duty cycle and move the high-resolution ePWM signal edge accordingly. If auto-conversion is disabled, the CMPAHR register behaves like a type 0 HRPWM module and CMPAHR = (fraction(PWMduty * PWMperiod) * MEP Scale Factor + 0.5)<<8). All of these calculations will need to be performed by user code in this mode, and the HRMSTEP register is ignored. Auto-conversion for high-resolution period has the same behavior as auto-conversion for high-resolution duty cycle. Auto-conversion must always be enabled for high-resolution period mode.

4.2.3 Principle of Operation

The MEP logic is capable of placing an edge in one of 255 (8 bits) discrete time steps (see device-specific data sheet for typical MEP step size). The MEP works with the TBM and CCM registers to be certain that time steps are optimally applied and that edge placement accuracy is maintained over a wide range of PWM frequencies, system clock frequencies, and other operating conditions. Table 4-3 shows the typical range of operating frequencies supported by the HRPWM.

Table 4-3. Relationship Between MEP Steps, PWM Frequency, and Resolution

System (MHz)	MEP Steps Per SYSCLKOUT (1) (2) (3)	PWM Minimum (Hz) (4)	PWM Maximum (MHz)	Resolution @ Maximum (Bits) (5)
50.0	111	763	2.50	11.1
60.0	93	916	3.00	10.9

(1) System frequency = SYSCLKOUT, that is, CPU clock. TBCLK = SYSCLKOUT.
(2) Table data based on a MEP time resolution of 180 ps (this is an example value. See the device-specific data sheet for MEP limits)
(3) MEP steps applied = $T_{SYSCLKOUT}$/180 ps in this example.
(4) PWM minimum frequency is based on a maximum period value, that is, TBPRD = 65535. PWM mode is asymmetrical up-count.
(5) Resolution in bits is given for the maximum PWM frequency stated.

4.2.3.1 Edge Positioning

In a typical power control loop (for example, switch modes, digital motor control [DMC], uninterruptible power supply [UPS]), a digital controller (PID, 2pole/2zero, lag/lead, and so on) issues a duty command, usually expressed in a per unit or percentage terms. Assume that for a particular operating point, the demanded duty cycle is 0.405 or 40.5% on time and the required converter PWM frequency is 1.25 MHz. In conventional PWM generation with a system clock of 60 MHz, the duty cycle choices are in the vicinity of 40.5%. In Figure 4-6, a compare value of 19 counts (that is, duty = 39.6%) is the closest to 40.5% that you can attain. This is equivalent to an edge position of 316.7 ns instead of the desired 324 ns. This data is shown in Table 4-4.

By utilizing the MEP, you can achieve an edge position much closer to the desired point of 324 ns. Table 4-4 shows that in addition to the CMPA 44 steps of the MEP (CMPAHR register) will position the edge at 323.92 ns, resulting in almost zero error. In this example, it is assumed that the MEP has a step resolution of 180 ps.

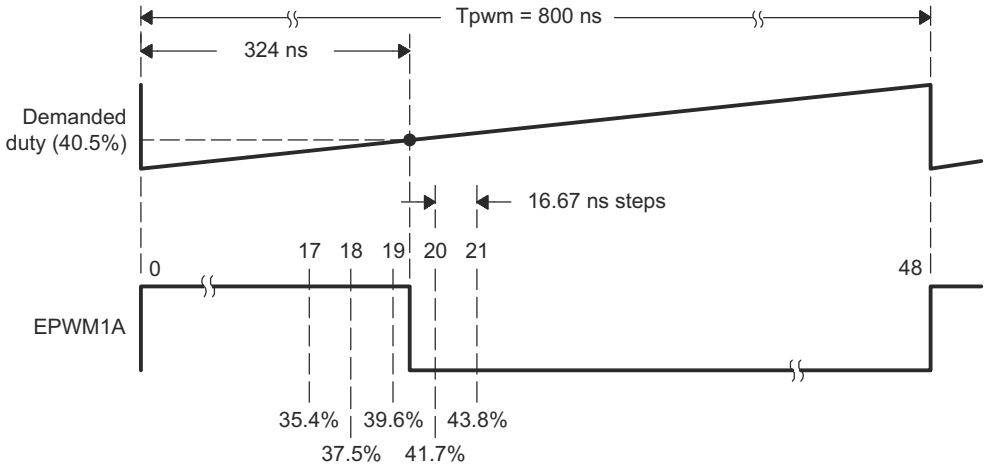


Figure 4-6. Required PWM Waveform for a Requested Duty = 30.0%

Table 4-4. CMPA versus Duty (left), and [CMPA:CMPAHR] versus Duty (right)

CMPA (count)(1) (2) (3)	Duty	High Time (ns)	CMPA (count)	CMPAHR (count)	Duty	High Time (ns)
15	31.25%	250	19	40	40.40%	323.2
16	33.33%	267	19	41	40.42%	323.38
17	35.42%	283	19	42	40.45%	323.56
18	37.50%	300	19	43	40.47%	323.74
19	39.58%	316	19	44	40.49%	323.92
20	41.67%	333	19	45	40.51%	324.1
21	43.75%	350	19	46	40.54%	324.28
			19	47	40.56%	324.46
Required			19	48	40.58%	324.64
19.4	40.50%	324	19	49	40.60%	324.82

(1) System clock, SYSCLKOUT and TBCLK = 60 MHz, 16.67 ns
(2) For a PWM Period register value of 48 counts, PWM Period = 48 x 16.67 ns = 800 ns, PWM frequency = 1/800 ns = 1.25 MHz
(3) Assumed MEP step size for the above example = 180 ps
See the device-specific data manual for typical and maximum MEP values.

4.2.3.2 Scaling Considerations

The mechanics of how to position an edge precisely in time has been demonstrated using the resources of the standard CMPA and MEP (CMPAHR) registers. In a practical application, however, it is necessary to seamlessly provide the CPU a mapping function from a per-unit (fractional) duty cycle to a final integer (non-fractional) representation that is written to the [CMPA:CMPAHR] register combination. This section describes the mapping from a per-unit duty cycle only. The method for mapping from a per-unit period is described in Section 4.2.3.4.

To do this, first examine the scaling or mapping steps involved. It is common in control software to express duty cycle in a per-unit or percentage basis. This has the advantage of performing all needed math calculations without concern for the final absolute duty cycle, expressed in clock counts or high time in ns. Furthermore, it makes the code more transportable across multiple converter types running different PWM frequencies.

To implement the mapping scheme, a two-step scaling procedure is required.

Assumptions for this example:

System clock , SYSCLKOUT	=	16.67 ns (60MHz)
PWM frequency	=	1.25 MHz (1/800 ns)
Required PWM duty cycle, **PWMDuty**	=	0.300 (30.0%)
PWM period in terms of coarse steps, **PWMperiod** (800 ns/ 11.1 ns)	=	48
Number of MEP steps per coarse step at 180 ps (11.1 ns/ 180 ps), **MEP_ScaleFactor**	=	93
Value to keep CMPAHR within the range of 1-255 and fractional rounding constant (default value). In the event that frac(**PWMDuty * PWMperiod**) * MEP_ScaleFactor results in a value with a decimal portion ≥ 0.5, this rounding constant will round the CMPAHR value up 1 MEP step.	=	0.5 (0 080h in Q8 format)

Step 1: Percentage Integer Duty value conversion for CMPA register

CMPA register value	=	int(**PWMDuty*****PWMperiod**); int means integer part
	=	int(0.405* 48)
CMPA register value	=	19 (13h)

Step 2: Fractional value conversion for CMPAHR register

CMPAHR register value	=	(frac(**PWMDuty*****PWMperiod**)***MEP_ScaleFactor**+0.5) << 8; frac means fractional part
	=	(frac(19.4)* 93+ 0.5) <<8; Shift is to move the value as CMPAHR high byte
	=	((0.4 * 93 + 0.5) << 8)
	=	((37.2 + 0.5) <<8)
	=	37.7 * 256 ; Shifting left by 8 is the same as multiplying by 256.
	=	9,651
CMPAHR value	=	25B3h; lower 8 bits will be ignored by hardware.

Note

If the AUTOCONV bit (HRCNFG.6) is set and the MEP_ScaleFactor is in the HRMSTEP register, then CMPAHR register value = frac (PWMDuty*PWMperiod<<8). The rest of the conversion calculations are performed automatically in hardware, and the correct MEP-scaled signal edge appears on the ePWM channel output. If AUTOCONV is not set, the above calculations must be performed by software.

Note

The MEP scale factor (MEP_ScaleFactor) varies with the system clock and DSP operating conditions. TI provides an MEP scale factor optimizing (SFO) software C function, which uses the built in diagnostics in each HRPWM and returns the best scale factor for a given operating point.

The scale factor varies slowly over a limited range so the optimizing C function can be run very slowly in a background loop.

The CMPA and CMPAHR registers are configured in memory so that the 32-bit data capability of the 28x CPU can write this as a single concatenated value, [CMPA:CMPAHR]. The TBPRDM and TBPRDHRM (mirror) registers are similarly configured in memory.

The mapping scheme has been implemented in both C and assembly, as shown in Section 4.2.5. The actual implementation takes advantage of the 32-bit CPU architecture of the 28xx, and is somewhat different from the steps shown in Section 4.2.3.2.

For time critical control loops where every cycle counts, the assembly version is recommended. This is a cycle optimized function (11 SYSCLKOUT cycles) that takes a Q15 duty value as input and writes a single [CMPA:CMPAHR] value.

4.2.3.3 Duty Cycle Range Limitation

In high resolution mode, the MEP is not active for 100% of the PWM period. It becomes operational:

- 3 SYSCLK cycles after the period starts when high-resolution period (TBPRDHR) control is not enabled.
- When high resolution period (TBPRDHR) control is enabled via the HRPCTL register:
  - In up-count mode: 3 SYSCLK cycles after the period starts until 3 SYSCLK cycles before the period ends.
  - In up-down count mode: when counting up, 3 cycles after CTR = 0 until 3 cycles before CTR = PRD, and when counting down, 3 cycles after CTR = PRD until 3 cycles before CTR = 0.

To better understand the useable duty cycle range, see Table 4-5. Duty cycle range limitations are illustrated in Figure 4-7 to Figure 4-10. This limitation imposes a duty cycle limit on the MEP. For example, precision edge control is not available all the way down to 0% duty cycle. When high-resolution period control is disabled, although for the first three cycles, the HRPWM capabilities are not available, regular PWM duty control is still fully operational down to 0% duty. In most applications, this should not be an issue as the controller regulation point is usually not designed to be close to 0% duty cycle. When high-resolution period control is enabled (HRPCTL[HRPE]=1), the duty cycle must not fall within the restricted range. Otherwise, there may be undefined behavior on the ePWMxA output.

Table 4-5. Duty Cycle Range Limitation for 3 SYSCLK/TBCLK Cycles

PWM Frequency (1) (kHz)	3 Cycles Minimum Duty	3 Cycles Maximum Duty(2)
200	0.67%	99.00%
400	1.33%	98.00%
600	2.00%	97.00%
800	2.67%	96.00%
1000	3.33%	95.00%
1200	4.00%	94.00%
1400	4.67%	93.00%
1600	5.33%	92.00%
1800	6.00%	91.00%
2000	6.67%	90.00%

(1) System clock - $T_{SYSCLKOUT}$ = 16.67 ns System clock = TBCLK = 60 MHz
(2) This limitation applies only if high-resolution period (TBPRDHR) control is enabled.

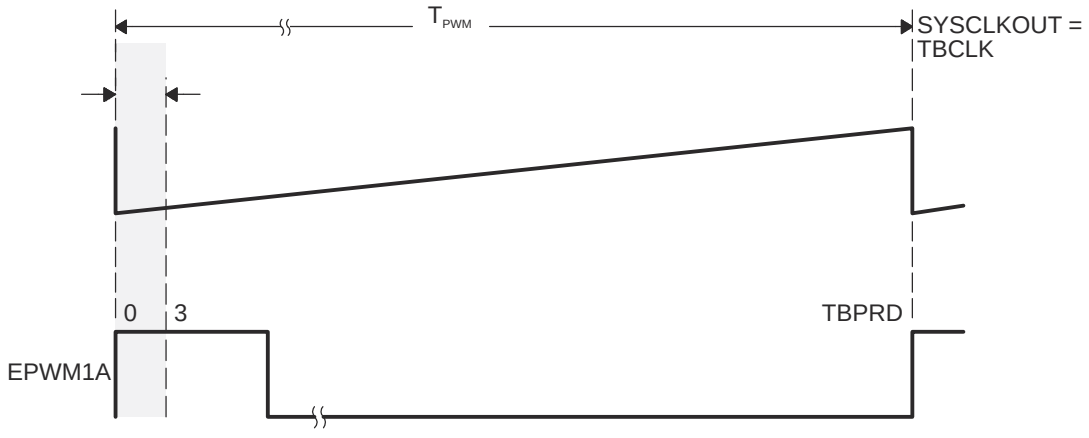


Figure 4-7. Low % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0)

If the application demands HRPWM operation in the low percent duty cycle region, then the HRPWM can be configured to operate in count-down mode with the rising edge position (REP) controlled by the MEP when high-resolution period is disabled (HRPCTL[HRPE] = 0). This is illustrated in Figure 4-8. In this case, low percent duty limitation is no longer an issue. However, there will be a maximum duty limitation with same percent numbers as given in Table 4-5.

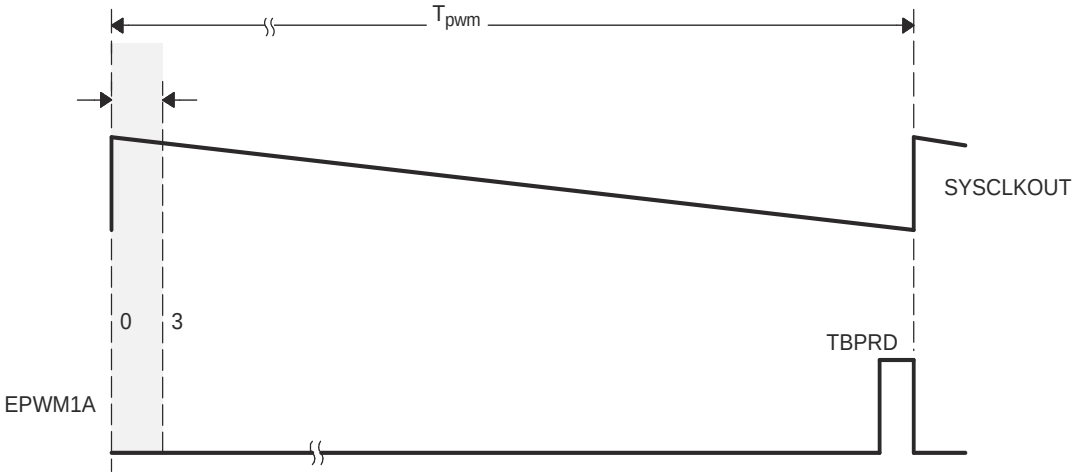


Figure 4-8. High % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0)

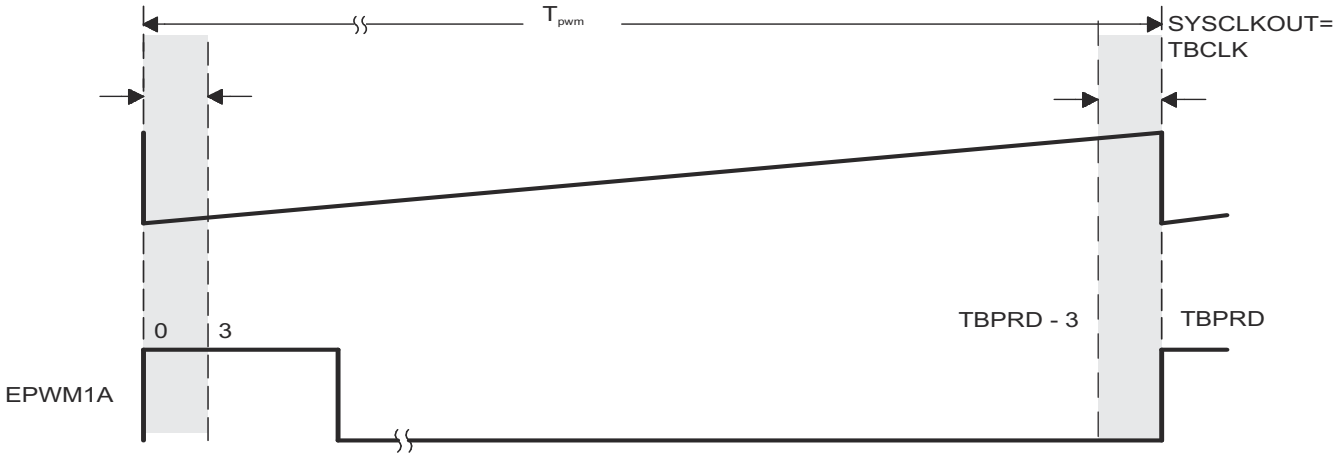


Figure 4-9. Up-Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)

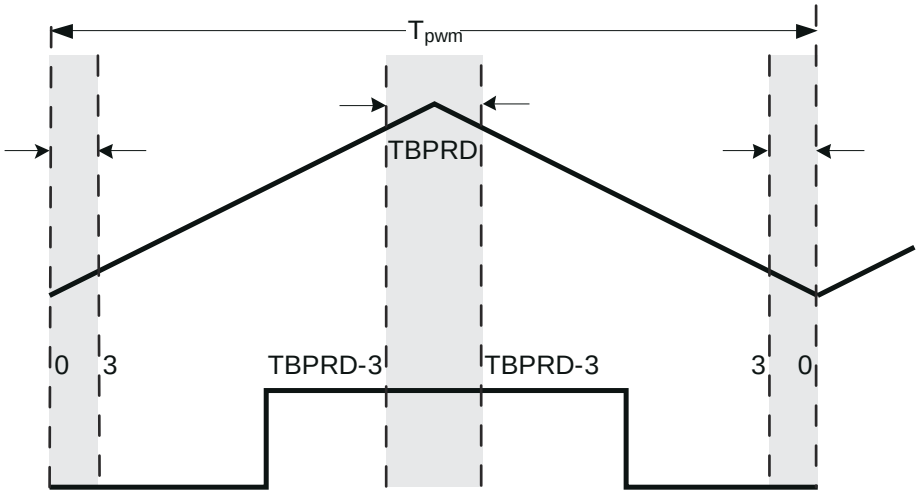


Figure 4-10. Up-Down Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)

Note

If the application has enabled high-resolution period control (HRPCTL[HRPE]=1), the duty cycle must not fall within the restricted range. Otherwise, there will be undefined behavior on the ePWM output.

4.2.3.4 High Resolution Period

High resolution period control using the MEP logic is supported on devices with a Type 1 ePWM module via the TBPRDHR(M) register.

Note

When high-resolution period control is enabled, the ePWMxB output will have +/- 1 TBCLK cycle jitter in up-count mode and +/- 2 TBCLK cycle jitter in up-down count mode.

The scaling procedure described for duty cycle in Section 4.2.3.2 applies for high-resolution period as well:

Assumptions for this example:

System clock , SYSCLKOUT	= 16.67 ns (60 MHz)
Required PWM frequency	= 175 kHz (TBPRD value of 342.857)
Number of MEP steps per coarse step at 180 ps (MEP_ScaleFactor)	= 93 (16.67 ns/180 ps)
Value to keep TBPRDHR within range of 1-255 and fractional rounding constant (default value)	= 0.5 (0080h in Q8 format)

Problem:

In up-count mode:

If TBPRD = 342, then PWM frequency = 174.93 kHz (period = (342+1)* T_{TBCLK}).

TBPRD = 341, then PWM frequency = 175.44 kHz (period = (341+1)* T_{TBCLK}).

In up-down count mode:

If TBPRD = 172, then PWM frequency = 174.42 kHz (period = (172*2)* T_{TBCLK}).

If TBPRD = 171, then PWM frequency = 175.44 kHz (period = (171*2)* T_{TBCLK}).

Solution:

With 93 MEP steps per coarse step at 180 ps each:

Step 1: Percentage Integer Period value conversion for TBPRD register

Integer period value	= 342 * T_{TBCLK}
	= int (342.857) * T_{TBCLK}
	= int (PWMperiod) * T_{TBCLK}
In up-count mode:	
TBPRD register value	= 341 (TBPRD = period value - 1)
	= 0155h
In up-down count mode:	= 171 (TBPRD = period value / 2)
TBPRD register value	= 00ABh

Step 2: Fractional value conversion for TBPRDHR register

TBPRDHR register value	= (frac(PWMperiod) * MEP_ScaleFactor + 0.5) (shift is to move the value as TBPRDHR high byte)
If auto-conversion enabled and HRMSTEP =	
MEP_ScaleFactor value (93):	=frac (PWMperiod)<<8
TBPRDHR register value	=frac (342.857)<<8
	=0.857 × 256
	=DB00h
The autoconversion will then automatically perform the calculation such that TBPRDHR MEP delay is scaled by hardware to:	=((TBPRDHR(15:0) >> 8) × HRMSTEP + 80h) >> 8
	= (00DBh × 93 + 80h) >> 8
	=(500Fh) >> 8
Period MEP delay	=0050h MEP Steps

4.2.3.4.1 High-Resolution Period Configuration

To use High Resolution Period, the ePWMx module must be initialized, following the steps in this exact order:

1. Enable ePWMx clock
2. Disable TBCLKSYNC
3. Configure ePWMx registers - AQ, TBPRD, CC, etc.
   - ePWMx may only be configured for up-count or up-down count modes. High-resolution period is not compatible with down-count mode.
   - TBCLK must equal SYSCLKOUT
   - TBPRD and CC registers must be configured for shadow loads.
   - CMPCTL[LOADAMODE]
     - In up-count mode:CMPCTL[LOADAMODE] = 1 (load on CTR = PRD)
     - In up-down count mode: CMPCTL[LOADAMODE] = 2 (load on CTR=0 or CTR=PRD)
4. Configure HRPWM register such that:
   - HRCNFG[HRLOAD] = 2 (load on either CTR = 0 or CTR = PRD)
   - HRCNFG[AUTOCONV] = 1 (Enable auto-conversion)
   - HRCNFG[EDGMODE] = 3 (MEP control on both edges)
5. For TBPHS: TBPHSHR synchronization with high-resolution period, set both HRPCTL[TBPSHRLOADE] = 1 and TBCTL[PHSEN] = 1. In up-down count mode these bits must be set to 1 regardless of the contents of TBPHSHR.
6. Enable high-resolution period control (HRPCTL[HRPE] = 1)
7. Enable TBCLKSYNC
8. TBCTL[SWFSYNC] = 1
9. HRMSTEP must contain an accurate MEP scale factor (# of MEP steps per SYSCLKOUT coarse step) because auto-conversion is enabled. The MEP scale factor can be acquired via the SFO() function described in Section 4.3.
10. To control high-resolution period, write to the TBPRDHR(M) registers.

Note

When high-resolution period mode is enabled, an EPWMxSYNC pulse will introduce +/- 1 - 2 cycle jitter to the PWM (+/- 1 cycle in up-count mode and +/- 2 cycle in up-down count mode). For this reason, TBCTL[SYNCOSEL] should not be set to 1 (CTR = 0 is EPWMxSYNCO source) or 2 (CTR = CMPB is EPWMxSYNCO source). Otherwise, the jitter will occur on every PWM cycle with the synchronization pulse.

When TBCTL[SYNCOSEL] = 0 (EPWMxSYNCI is EPWMxSYNCO source), a software synchronization pulse should be issued only once during high-resolution period initialization. If a software sync pulse is applied while the PWM is running, the jitter will appear on the PWM output at the time of the sync pulse.

4.2.4 Scale Factor Optimizing Software (SFO)

The micro edge positioner (MEP) logic is capable of placing an edge in one of 255 discrete time steps. As previously mentioned, the size of these steps is on the order of 150 ps (see device-specific data sheet for typical MEP step size on your device). The MEP step size varies based on worst-case process parameters, operating temperature, and voltage. MEP step size increases with decreasing voltage and increasing temperature and decreases with increasing voltage and decreasing temperature. Applications that use the HRPWM feature should use the TI-supplied MEP scale factor optimizer (SFO) software function. The SFO function helps to dynamically determine the number of MEP steps per SYSCLKOUT period while the HRPWM is in operation.

To utilize the MEP capabilities effectively during the Q15 duty (or period) to [CMPA:CMPAHR] or [TBPRD(M):TBPRDHR(M)] mapping function (see Section 4.2.3.2), the correct value for the MEP scaling factor (MEP_ScaleFactor) needs to be known by the software. To accomplish this, the HRPWM module has built in self-check and diagnostics capabilities that can be used to determine the optimum MEP_ScaleFactor value for any operating condition. TI provides a C-callable library containing one SFO function that utilizes this hardware and determines the optimum MEP_ScaleFactor. As such, MEP Control and Diagnostics registers are reserved for TI use.

A detailed description of the SFO library - SFO_TI_Build_V6.lib software can be found in Section 4.3.

4.2.5 HRPWM Examples Using Optimized Assembly Code

The best way to understand how to use the HRPWM capabilities is through two real examples:

1. Simple buck converter using asymmetrical PWM (that is, count-up) with active high polarity.
2. DAC function using simple R+C reconstruction filter.

The following examples all have Initialization/configuration code written in C. To make these easier to understand, the #defines shown below are used. Note that the #defines introduced in Chapter 3 are also used.

Example 4-1 assumes an MEP step size of 150 ps and does not use the SFO library.

Example 4-1. #Defines for HRPWM Header Files

```
// HRPWM (High Resolution PWM) //
================================
// HRCNFG
#define HR_Disable 0x0
#define HR_REP 0x1          // Rising Edge position
#define HR_FEP 0x2          // Falling Edge position
#define HR_BEP 0x3          // Both Edge position #define HR_CMP 0x0 // CMPAHR controlled
#define HR_PHS 0x1          // TBPHSHR controlled #define HR_CTR_ZERO 0x0 // CTR = Zero event
#define HR_CTR_PRD 0x1      // CTR = Period event
#define HR_CTR_ZERO_PRD 0x2 // CTR = ZERO or Period event
#define HR_NORM_B  0x0      // Normal ePWMxB output
#define HR_INVERT_B 0x1     // ePWMxB is inverted ePWMxA output
```

4.2.5.1 Implementing a Simple Buck Converter

In this example, the PWM requirements for SYSCLKOUT = 60 MHz are:

- PWM frequency = 600 kHz (TBPRD = 100)
- PWM mode = asymmetrical, up-count
- Resolution = 12.7 bits (with a MEP step size of 150 ps)

Figure 4-11 and Figure 4-12 show the required PWM waveform. As explained previously, configuration for the ePWM1 module is almost identical to the normal case except that the appropriate MEP options need to be enabled/selected.

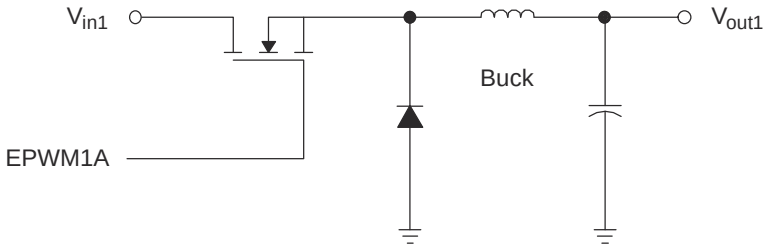


Figure 4-11. Simple Buck Controlled Converter Using a Single PWM

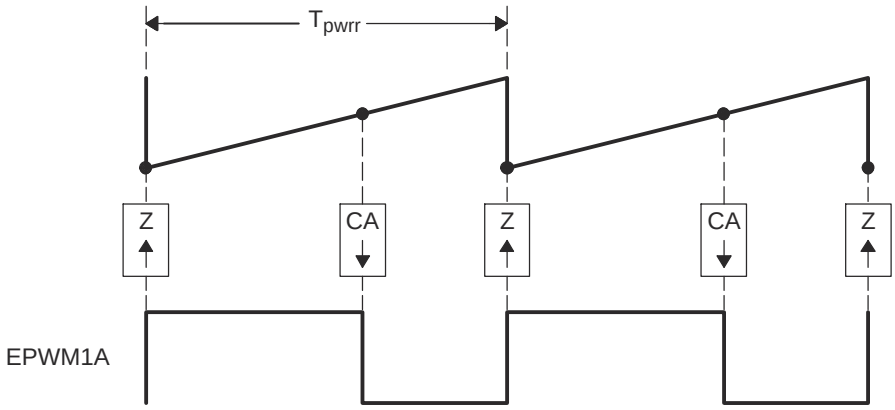


Figure 4-12. PWM Waveform Generated for Simple Buck Controlled Converter

The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

Example 4-2 shows the Initialization code. The first part is configured for conventional PWM. The second part sets up the HRPWM resources.

This example assumes an MEP step size of 150 ps and does not use the SFO library.

Example 4-2. HRPWM Buck Converter Initialization Code

```
void HrBuckDrvCnf(void)
 {
// Config for conventional PWM first
EPwm1Regs.TBCTL.bit.PRDLD = TB_IMMEDIATE;         // set Immediate load
EPwm1Regs.TBPRD = 100;                            // Period set for 600 kHz PWM
hrbuck_period = 200;                              // Used for Q15 to Q0 scaling
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;           // EPWM1 is the Master
EPwm1Regs.TBCTL.bit.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 = 111*256;                        // Start with typical Scale Factor
                                                  // value for 60 MHz
                                                  // Note: Use SFO functions to update
                                                  // MEP_ScaleFactor dynamically
}
```

Example 4-3 shows an assembly example of run-time code for the HRPWM buck converter.

Example 4-3. HRPWM Buck Converter Run-Time Code

```
EPWM1_BASE .set 0x6800
CMPAHR1 .set EPWM1_BASE+0x8
;================================================
HRBUCK_DRV; (can execute within an ISR or loop)
;================================================
     MOVW DP, #_HRBUCK_In
     MOVL XAR2,@_HRBUCK_In      ; Pointer to Input Q15 Duty (XAR2)
     MOVL XAR3,#CMPAHR1         ; Pointer to HRPWM CMPA reg (XAR3)
; Output for EPWM1A (HRPWM)
     MOV T,*XAR2 ; T <= Duty
     MPYU ACC,T,@_hrbuck_period ; Q15 to Q0 scaling based on Period
     MOV T,@_MEP_ScaleFactor    ; MEP scale factor (from optimizer s/w)
     MPYU P,T,@AL               ; P <= T * AL, Optimizer scaling
     MOVH @AL,P                 ; AL <= P, move result back to ACC
     ADD ACC, #0x080            ; MEP range and rounding adjustment
     MOVL *XAR3,ACC             ; CMPA:CMPAHR(31:8) <= ACC
; Output for EPWM1B (Regular Res) Optional - for comparison purpose only
     MOV *+XAR3[2],AH           ; Store ACCH to regular CMPB
```

4.2.5.2 Implementing a DAC Function Using an R+C Reconstruction Filter

In this example, the PWM requirements are:

- PWM frequency = 400 kHz (that is, TBPRD = 150)
- PWM mode = Asymmetrical, Up-count
- Resolution = 14 bits (MEP step size = 150 ps)

Figure 4-13 and Figure 4-14 show the DAC function and the required PWM waveform. As explained previously, configuration for the ePWM1 module is almost identical to the normal case except that the appropriate MEP options need to be enabled/selected.

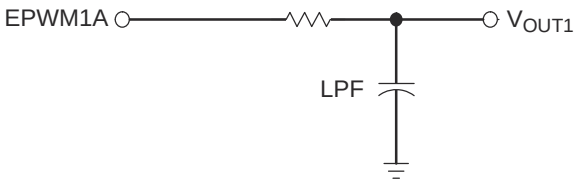


Figure 4-13. Simple Reconstruction Filter for a PWM Based DAC

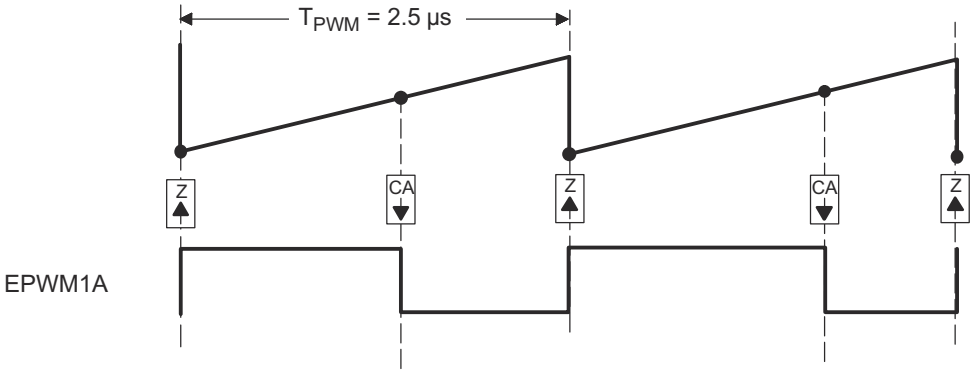


Figure 4-14. PWM Waveform Generated for the PWM DAC Function

The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

This example assumes a typical MEP_ScaleFactor and does not use the SFO library.

Example 4-4 shows the Initialization code. The first part is configured for conventional PWM. The second part sets up the HRPWM resources.

Example 4-4. PWM DAC Function Initialization Code

```
void HrPwmDacDrvCnf(void)
{
// Config for conventional PWM first
EPwm1Regs.TBCTL.bit.PRDLD = TB_IMMEDIATE;      // Set Immediate load
EPwm1Regs.TBPRD = 150;                         // Period set for 400 kHz PWM
hrDAC_period = 150;                            // Used for Q15 to Q0 scaling
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // EPWM1 is the Master
EPwm1Regs.TBCTL.bit.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 = 111*256;                     // Start with typical Scale Factor
                                               // value for 60 MHz.
                                               // Use SFO functions to update MEP_ScaleFactor
                                               // dynamically.
}
```

Example 4-5 shows an assembly example of run-time code that can execute in a high-speed ISR loop.

Example 4-5. PWM DAC Function Run-Time Code

```
EPWM1_BASE .set 0x6800
CMPAHR1 .set EPWM1_BASE+0x8
;================================================
HRPWM_DAC_DRV; (can execute within an ISR or loop)
;================================================
      MOVW DP, #_HRDAC_In
      MOVL XAR2,@_HRDAC_In            ; Pointer to input Q15 duty (XAR2)
      MOVL XAR3,#CMPAHR1              ; Pointer to HRPWM CMPA reg (XAR3)
; Output for EPWM1A (HRPWM
      MOV T,*XAR2                      ; T <= duty
      MPY ACC,T,@_hrDAC_period         ; Q15 to Q0 scaling based on period
      ADD ACC,@_HrDAC_period<<15       ; Offset for bipolar operation
      MOV T,@_MEP_ScaleFactor          ; MEP scale factor (from optimizer s/w)
      MPYU P,T,@AL                     ; P <= T * AL, optimizer scaling
      MOVH @AL,P                       ; AL <= P, move result back to ACC
      ADD ACC, #0x080                  ; MEP range and rounding adjustment
      MOVL *XAR3,ACC                   ; CMPA:CMPAHR(31:8) <= ACC
; Output for EPWM1B (Regular Res) Optional - for comparison purpose only
      MOV *+XAR3[2],AH                 ; Store ACCH to regular CMPB
```

4.3 SFO Library Software - SFO_TI_Build_V6.lib

Table 4-6 lists several features of the SFO_TI_Build_V6.lib library.

Table 4-6. SFO Library Features

	SYSCLK Freq	SFO_TI_Build_V6.lib	Unit
Maximum HRPWM channels supported	-	8	channels
Total static variable memory size	-	11	words
Completion-checking?	-	Yes	-
Typical time required for SFO() to update MEP_ScaleFactor, if called repetitively without interrupts	80 MHz	1.3	milliseconds
	60 MHz	2.23	milliseconds

Following is a functional description of the SFO library routine, SFO().

4.3.1 Scale Factor Optimizer Function - int SFO()

This routine drives the micro-edge positioner (MEP) calibration module to run SFO diagnostics and determine the appropriate MEP scale factor (number of MEP steps per coarse SYSCLKOUT step) for a device at any given time.

If SYSCLKOUT = TBCLK = 60 MHz and assuming the MEP step size is 150 ps, the typical scale factor value at 60 MHz = 111 MEP steps per TBCLK unit (16.67 ns)

The function returns a MEP scale factor value:

MEP_ScaleFactor = Number of MEP steps/SYSCLKOUT.

Constraints when using this function:

- SFO() can be used with a minimum SYSCLKOUT = TBCLK = 50 MHz. MEP diagnostics logic uses SYSCLKOUT and not TBCLK, so the SYSCLKOUT restriction is an important constraint. Below 50 MHz, with device process variation, the MEP step size may decrease under cold temperature and high core voltage conditions to such a point, that 255 MEP steps will not span an entire SYSCLKOUT cycle.
- At any time, SFO() can be called to run SFO diagnostics on the MEP calibration module

Usage:

- SFO() can be called at any time in the background while the ePWM channels are running in HRPWM mode. The scale factor result obtained in MEP_ScaleFactor can be applied to all ePWM channels running in HRPWM mode because the function makes use of the diagnostics logic in the MEP calibration module (which runs independently of ePWM channels).
- This routine returns a 1 when calibration is finished, and a new scale factor has been calculated or a 0 if calibration is still running. The routine returns a 2 if there is an error, and the MEP_ScaleFactor is greater than the maximum 255 fine steps per coarse SYSCLKOUT cycle. In this case, the HRMSTEP register will maintain the last MEP scale factor value less than 256 for auto conversion.
- All ePWM modules operating in HRPWM incur only a 3-SYSCLKOUT cycle minimum duty cycle limitation when high-resolution period control is not used. If high-resolution period control is enabled, there is an additional duty cycle limitation 3-SYSCLKOUT cycles before the end of the PWM period (see Section 4.2.3.3).
- In SFO_TI_Build_V6b.lib, the SFO() function also updates the HRMSTEP register with the scale factor result. If the HRCNFG[AUTOCONV] bit is set, the application software is responsible only for setting CMPAHR = fraction(PWMduty*PWMperiod)<<8 or TBPRDHR = fraction (PWMperiod) while running SFO() in the background. The MEP Calibration Module will then use the values in the HRMSTEP and CMPAHR/TBPRDHR register to automatically calculate the appropriate number of MEP steps represented by the fractional duty cycle or period and move the high-resolution ePWM signal edge accordingly. In SFO_TI_Build_V6.lib, the SFO() function does not automatically update the HRMSTEP register. Therefore, after the SFO function completes, the application software must write MEP_ScaleFactor to the HRMSTEP register (EALLOW-protected).

- If the HRCNFG[AUTOCONV] bit is clear, the HRMSTEP register is ignored. The application software will need to perform the necessary calculations manually so that:
  - CMPAHR = (fraction(PWMduty * PWMperiod) * MEP Scale Factor)<<8 + 0x080.
  - Similar behavior applies for TBPHSHR. Auto-conversion must be enabled when using TBPRDHR.

The routine can be run as a background tasks in a slow loop requiring negligible CPU cycles. The repetition rate at which an SFO function needs to be executed depends on the application's operating environment. As with all digital CMOS devices temperature and supply voltage variations have an effect on MEP operation. However, in most applications these parameters vary slowly and therefore it is often sufficient to execute the SFO function once every 5 to 10 seconds or so. If more rapid variations are expected, then execution may have to be performed more frequently to match the application. Note, there is no high limit restriction on the SFO function repetition rate, hence it can execute as quickly as the background loop is capable.

While using the HRPWM feature, HRPWM logic will not be active for the first 3 SYSCLKOUT cycles of the PWM period (and the last 3 SYSCLKOUT cycles of the PWM period if TBPRDHR is used). While running the application in this configuration, if high-resolution period control is disabled (HRPCTL[HRPE=0]) and the CMPA register value is less than 3 cycles, then its CMPAHR register must be cleared to zero. If high-resolution period control is enabled (HRPCTL[HRPE=1]), the CMPA register value must not fall below 3 or above TBPRD-3.This would avoid any unexpected transitions on the PWM signal.

4.3.2 Software Usage

The software library function SFO(), calculates the MEP scale factor for the HRPWM-supported ePWM modules. The scale factor is an integer value in the range 1-255, and represents the number of micro step edge positions available for a system clock period. The scale factor value is returned in an integer variable called MEP_ScaleFactor. For example, see Table 4-7.

Table 4-7. Factor Values

Software Function call	Functional Description	Updated Variables
SFO()	Returns MEP scale factor in MEP_ScaleFactor Returns MEP scale factor in the HRMSTEP register in SFO_TI_Build_V6b.lib	MEP_ScaleFactor and HRMSTEP register.

To use the HRPWM feature of the ePWMs it is recommended that the SFO function be used as described here.

Step 1. Add "Include" Files

The SFO_V6.h file needs to be included as follows. This include file is mandatory while using the SFO library function. For the TMS320F2802x devices, the F2802x C/C++ Header Files and Peripheral Examples in controlSUITE DSP2802x_Device.h and DSP2802x_Epwm_defines.h are necessary. For other device families, the device-specific equivalent files in the header files and peripheral examples software packages for those devices should be used. These include files are optional if customized header files are used in the end applications.

Example 4-6. A Sample of How to Add "Include" Files

```
#include "DSP2802x_Device.h"       // DSP2802x Headerfile
#include "DSP2802x_EPwm_defines.h" // init defines
#include "SFO_V6.h"              // SFO lib functions (needed for HRPWM)
```

Step 2. Element Declaration

Declare an integer variable for the scale factor value as shown in Example 4-7.

Example 4-7. Declaring an Element

```
int MEP_ScaleFactor = 0;   //scale factor value
volatile struct EPWM_REGS *ePWM[] = {0, &EPwm1Regs, &EPwm2Regs, &EPwm3Regs,
&EPwm4Regs};
```

Step 3. MEP_ScaleFactor Initialization

The SFO() function does not require a starting scale factor value in MEP_ScaleFactor. Prior to using the MEP_ScaleFactor variable in application code, SFO() should be called to drive the MEP calibration module to calculate an MEP_ScaleFactor value.

As part of the one-time initialization code prior to using MEP_ScaleFactor, include the following shown in Example 4-8.

Example 4-8. Initializing With a Scale Factor Value

```
MEP_ScaleFactor initialized using function SFO ()
while (SFO() == 0) {} // MEP_ScaleFactor calculated by MEP Cal Module
```

Step 4. Application Code

While the application is running, fluctuations in both device temperature and supply voltage may be expected. To be sure that optimal Scale Factors are used for each ePWM module, the SFO function should be re-run periodically as part of a slower back-ground loop. Some examples of this are shown in Example 4-9.

Note

See the HRPWM_SFO example in the device-specific C/C++ header files and peripheral examples available from the TI website.

Example 4-9. SFO Function Calls

```
main ()
 {
 int status;
    // User code
    // ePWM1, 2, 3, 4 are running in HRPWM mode
    // The status variable returns 1 once a new MEP_ScaleFactor has been
    // calculated by the MEP Calibration Module running SFO
    // diagnostics.
status = SFO();
if(status==2) {ESTOP0;}   // The function returns a 2 if MEP_ScaleFactor is greater
                          // than the maximum 255 allowed (error condition)
 }
```

4.3.3 SFO Library Version Software Differences

There are two different versions of the SFO library - SFO_TI_Build_V6.lib, and SFO_TI_Build_V6b.lib. SFO_TI_Build_V6.lib does not update the HRMSTEP register with the value in MEP_ScaleFactor, while SFO_TI_Build_V6b.lib updates the register. Therefore, if using SFO_TI_Build_V6.lib and auto-conversion is enabled, the application should write MEP_Scalefactor in the HRMSTEP register as shown in Example 4-10.

Example 4-10. Manually Updating the HRMSTEP Register if Using SFO_TI_Build_V6b.lib

```
main ()
 {
     int status;

     status = SFO_INCOMPLETE;
     while (status==SFO_INCOMPLETE) {
            status = SFO();
 }
     if(status!=SFO_ERROR) { // IF SFO() is complete with no errors
        EALLOW;
        EPwm1Regs.HRMSTEP=MEP_ScaleFactor;
        EDIS;
 }
```

4.4 HRPWM Registers

A summary of the registers required for the HRPWM is listed in Table 4-8.

Table 4-8. HRPWM Registers

Name	Offset	Size(x16)	Shadow	Description	Bit Description
Time-Base Registers					
TBCTL	0x0000	1		Time-Base Control Register	Section 3.4.1.1
TBSTS	0x0001	1		Time-Base Status Register	Section 3.4.1.2
TBPHSHR	0x0002	1		Time-Base Phase High Resolution Register	Section 3.4.1.3
TBPHS	0x0003	1		Time-Base Phase Register	Section 3.4.1.4
TBCTR	0x0004	1		Time-Base Counter Register	Section 3.4.1.5
TBPRD	0x0005	1	Yes	Time-Base Period Register Set	Section 3.4.1.6
TBPRDHR	0x0006	1	Yes	Time-Base Period High Resolution Register Set	Section 3.4.1.7
Conter-Compare Registers					
CMPCTL	0x0007	1		Counter-Compare Control Register	Section 3.4.2.1
CMPAHR	0x0008	1	Yes	Counter-Compare A High Resolution Register	Section 3.4.2.2
CMPA	0x0009	1	Yes	Counter-Compare A Register	Section 3.4.2.3
CMPB	0x000A	1	Yes	Counter-Compare B Register	Section 3.4.2.4
HRPWM Registers					
HRCNFG	0x0020	1		HRPWM Configuration Register	Section 4.4.1
HRPWR	0x0021	1		HRPWM Power Register	Section 4.4.2
HRMSTEP	0x0026	1		HRPWM MEP Step Register	Section 4.4.3
High Resolution Period and Mirror Registers					
HRPCTL	0x0028	1		High Resolution Period Control Register	Section 4.4.4
TBPRDHRM	0x002A	1	Writes	Time-Base Period High Resolution Mirror Register	Section 3.4.1.8
TBPRDM	0x002B	1	Writes	Time-Base Period Mirror Register	Section 3.4.1.9
CMPAHRM	0x002C	1	Writes	Counter-Compare A High Resolution Mirror Register	Section 3.4.2.5
CMPAM	0x002D	1	Writes	Counter-Compare A Mirror Register	Section 3.4.2.6

4.4.1 HRPWM Configuration (HRCNFG) Register

Figure 4-15. HRPWM Configuration (HRCNFG) Register

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
SWAPAB	AUTOCONV	SELOUTB	HRLOAD		CTLMODE	EDGMODE	
R/W-0	R/W-0	R/W-0	R/W-0		R/W-0	R/W-0	

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

Table 4-9. HRPWM Configuration (HRCNFG) Register Field Descriptions

Bit	Field	Value	Description (1)
15-8	Reserved		Reserved
7	SWAPAB		Swap ePWM A and B Output Signals
			This bit enables the swapping of the A and B signal outputs. The selection is as follows:
		0	ePWMxA and ePWMxB outputs are unchanged.
		1	ePWMxA signal appears on ePWMxB output and ePWMxB signal appears on ePWMxA output.
6	AUTOCONV		Auto Convert Delay Line Value
			Selects whether the fractional duty cycle/period/phase in the CMPAHR/TBPRDHR/TBPHSHR register is automatically scaled by the MEP scale factor in the HRMSTEP register or manually scaled by calculations in application software. The SFO library function automatically updates the HRMSTEP register with the appropriate MEP scale factor.
		0	Automatic HRMSTEP scaling is disabled.
		1	Automatic HRMSTEP scaling is enabled.
			If application software is manually scaling the fractional duty cycle, or phase (that is, software sets CMPAHR = (fraction(PWMduty * PWMperiod) * MEP Scale Factor)<<8 + 0x080 for duty cycle), then this mode must be disabled.
5	SELOUTB		EPWMxB Output Select Bit
			This bit selects which signal is output on the ePWMxB channel output.
		0	ePWMxB output is normal.
		1	ePWMxB output is inverted version of ePWMxA signal.
4-3	HRLOAD		Shadow Mode Bit
			Selects the time event that loads the CMPAHR shadow value into the active register.
		00	Load on CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000)
		01	Load on CTR = PRD: Time-base counter equal to period (TBCTR = TBPRD)
		10	Load on either CTR = Zero or CTR = PRD
		11	Reserved
2	CTLMODE		Control Mode Bits
			Selects the register (CMP/TBPRD or TBPHS) that controls the MEP:
		0	CMPAHR(8) or TBPRDHR(8) Register controls the edge position (that is, this is duty or period control mode). (Default on Reset)
		1	TBPHSHR(8) Register controls the edge position (that is, this is phase control mode).
1-0	EDGMODE		Edge Mode Bits
			Selects the edge of the PWM that is controlled by the micro-edge position (MEP) logic:
		00	HRPWM capability is disabled (default on reset)
		01	MEP control of rising edge (CMPAHR)
		10	MEP control of falling edge (CMPAHR)
		11	MEP control of both edges (TBPHSHR or TBPRDHR)

(1) This register is EALLOW protected.

4.4.2 High Resolution Power (HRPWR) Register

Figure 4-16. High Resolution Power (HRPWR) Register

15	10	9	6	5	0
Reserved		MEPOFF		Reserved	
R/W-0		R/W-0		R/W-0	

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

Table 4-10. High Resolution Power (HRPWR) Register Field Descriptions

Bit	Field	Value	Description (1)
15-10	Reserved		Reserved for TI Test
9-6	MEPOFF	0-Fh	MEP Calibration OFF bits When not using MEP calibration, setting these bits to all 1's disables MEP calibration logic in the HRPWM and reduces power consumption.
5-0	Reserved		Reserved for TI Test

(1) This register is EALLOW protected.

4.4.3 High Resolution Micro Step (HRMSTEP) Register

Figure 4-17. High Resolution Micro Step (HRMSTEP) Register

15	8	7	0
Reserved		HRMSTEP	
R-0		R/W-0	

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

Table 4-11. High Resolution Micro Step (HRMSTEP) Register Field Descriptions

Bit	Field	Value	Description (1)
15-8	Reserved		Reserved
7-0	HRMSTEP	00-FFh	High Resolution MEP Step When auto-conversion is enabled (HRCNFG[AUTOCONV] = 1), This 8-bit field contains the MEP_ScaleFactor (number of MEP steps per coarse steps) used by the hardware to automatically convert the value in the CMPAHR, TBPHSHR, or TBPRDHR register to a scaled micro-edge delay on the high-resolution ePWM output. The value in this register is written by the SFO calibration software at the end of each calibration run.

(1) This register is EALLOW protected.

4.4.4 High Resolution Period Control (HRPCTL) Register

Figure 4-18. High Resolution Period Control (HRPCTL) Register

15				8
Reserved				
R-0				
7		**3**	**2**	**1**	**0**
Reserved			TBPHSHR LOADE	PWMSYNCSEL	HRPE
R-0			R/W-0	R/W-0	R/W-0

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

Table 4-12. High Resolution Period Control (HRPCTL) Register Field Descriptions

Bit	Field	Value	Description (1) (2)
15-3	Reserved		Reserved
2	TBPHSHRLOADE		TBPHSHR Load Enable This bit allows you to synchronize ePWM modules with a high-resolution phase on a SYNCIN, TBCTL[SWFSYNC] or digital compare event. This allows for multiple ePWM modules operating at the same frequency to be phase aligned with high-resolution.
		0	Disables synchronization of high-resolution phase on a SYNCIN, TBCTL[SWFSYNC] or digital compare event:
		1	Synchronize the high-resolution phase on a SYNCIN, TBCTL[SWFSYNC] or digital comparator synchronization event. The phase is synchronized using the contents of the high-resolution phase TBPHSHR register. The TBCTL[PHSEN] bit which enables the loading of the TBCTR register with TBPHS register value on a SYNCIN or TBCTL[SWFSYNC] event works independently. However, users need to enable this bit also if they want to control phase in conjunction with the high-resolution period feature. This bit and the TBCTL[PHSEN] bit must be set to 1 when high-resolution period is enabled for up-down count mode even if TBPHSHR = 0x0000. This bit does not need to be set when only high-resolution duty is enabled.
1	PWMSYNCSEL		PWMSYNC Source Select Bit This bit selects the source for the PWMSYNC signal. The PWMSYNC signal is used by external modules (such as COMP+DAC) for synchronizing timing to the selected ePWM module. **Note:** This bit is not used for high-resolution period control.
		0	PWMSYNC is generated by TBCTR = PRD pulse.
		1	PWMSYNC is generated by TBCTR = 0 pulse.
0	HRPE		High Resolution Period Enable Bit
		0	High resolution period feature disabled. In this mode the ePWM behaves as a Type 0 ePWM.
		1	High resolution period enabled. In this mode the HRPWM module can control high-resolution of both the duty and frequency. When high-resolution period is enabled, TBCTL[CTRMODE] = 0,1 (down-count mode) is not supported.

(1) This register is EALLOW protected.
(2) This register is used with Type 1 ePWM modules (support high-resolution period) only.

Chapter 5

High-Resolution Capture (HRCAP) Module

This chapter describes the operation of the high-resolution capture (HRCAP) module. The HRCAP module described here is a Type 0 HRCAP. HRCAP measures the width of external pulses with a typical resolution within hundreds of picoseconds. See the *TMS320x28xx, 28xxx DSP Peripheral Reference Guide* for a list of all devices with an HRCAP module of the same type, to determine the differences between types, and for a list of device-specific differences within a type.

5.1 Introduction

Uses for the HRCAP include:

- Capacitive touch applications
- High-resolution period and duty-cycle measurements of pulse-train cycles
- Instantaneous speed measurements
- Instantaneous frequency measurements
- Voltage measurements across an isolation boundary
- Distance and sonar measurement and scanning

5.1.1 Features

The HRCAP module includes the following features:

- Pulse-width capture in either non-high-resolution or high-resolution modes
- Difference (Delta) mode pulse-width capture
- Typical high-resolution capture on the order of 300 ps resolution on each edge
- Interrupt on either falling or rising edge
- Continuous mode capture of pulse widths in 2-deep buffer
- Calibration logic for precision high-resolution capture
- All of the above resources are dedicated to a single input pin.

5.1.2 HRCAP Related Collateral

Getting Started Materials

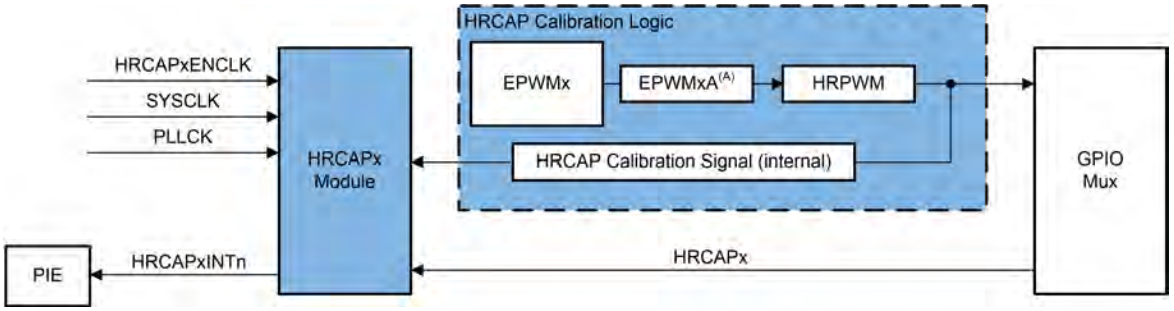
- *Leveraging High Resolution Capture (HRCAP) for Single Wire Data Transfer Application Report*

5.2 Description

The HRCAP module includes one capture channel in addition to a high-resolution calibration block, which connects internally to an HRPWM channel during calibration. See the device-specific data manual to determine which HRPWM channel output the HRCAP module is internally tied to during calibration.

Each HRCAP channel has the following independent key resources:

- Dedicated input capture pin
- 16-bit HRCAP clock (HCCAPCLK) that is either equal to the PLL output frequency (asynchronous to SYSCLKOUT) or equal to the SYSCLKOUT frequency (synchronous to SYSCLKOUT).
- High-resolution pulse width capture in a two-deep buffer
- High-resolution calibration logic utilizing an internal connection to an HRPWM output

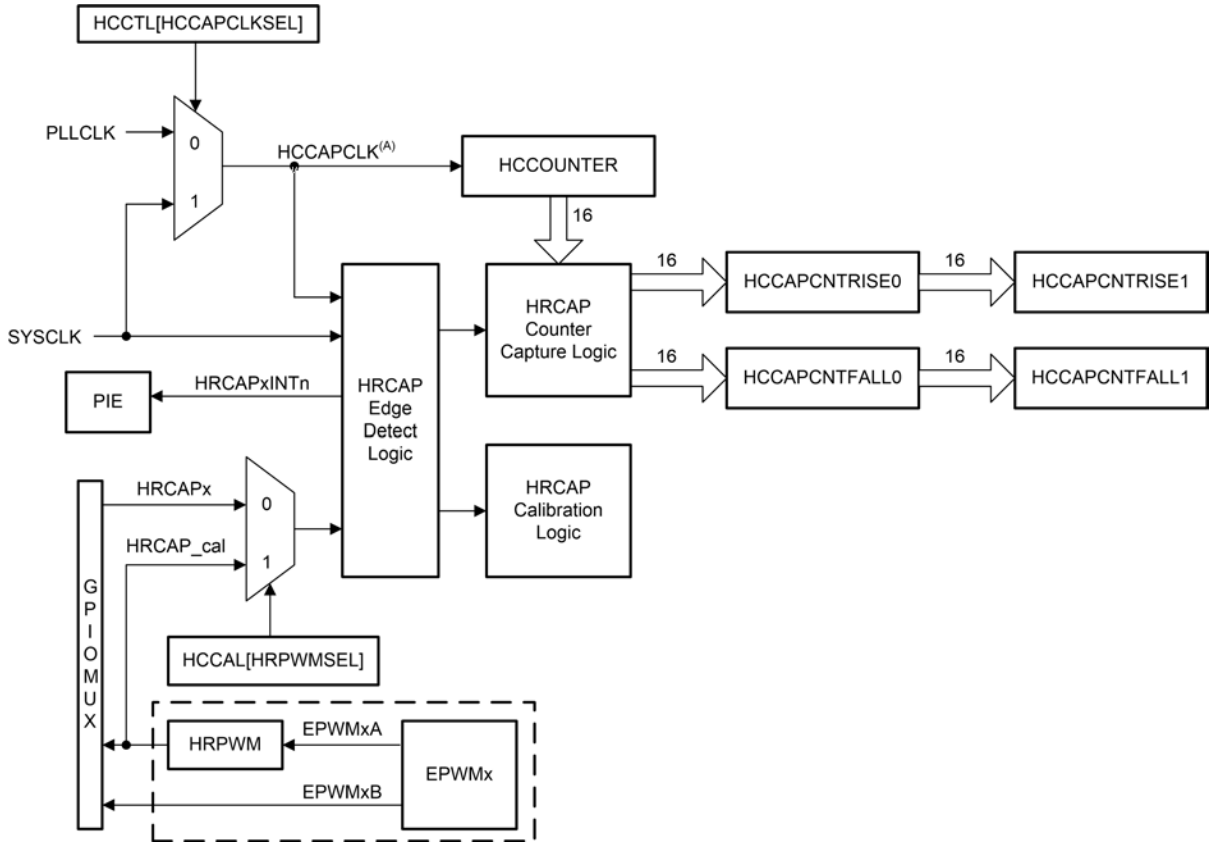


A. In general, the largest numerical instance of an HRPWM module channel A output is the internal HRCAP calibration signal input. For instance, on devices where there are seven HRPWM instances, ePWM7A HRPWM output is the internal HRCAP calibration signal input.

Figure 5-1. HRCAP Module System Block Diagram

5.3 Operational Details

Figure 5-2 shows the various components that implement the high-resolution, pulse-width capture functionality of the module.



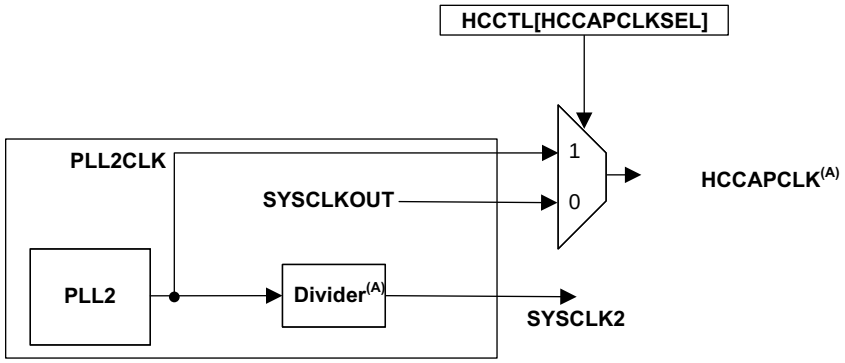
A. If PLLCLK is selected as the source for HCCAPCLK, HCCAPCLK is asynchronous to SYSCLK.

Figure 5-2. HRCAP Block Diagram

5.3.1 HRCAP Clocking

Although the HRCAP module is clocked by the system clock, the 16-bit counter (HCCOUNTER) and edge detection logic used for capturing high-resolution pulses is clocked by HCCAPCLK. HCCAPCLK must fall within the frequency range specified in the *Electricals* section of the device-specific data manual. HCCAPCLK can either be clocked by the system clock (SYSCLK), or the output of the PLL (PLLCLK) before the divider is applied. If HCCAPCLK is fed from the PLLCLK (HCCTL[HCCAPCLKSEL] = 1), then HCCAPCLK will be asynchronous to SYSCLK.

Figure 5-3 shows how the HCCAPCLK that clocks the HCCOUNTER and edge detection logic is generated.



A. On this device, the clock divider is a result of the PLLSTS[DIVSEL] bits.

Figure 5-3. HCCAPCLK Generation

5.3.2 HRCAP Modes of Operation

The HRCAP module has two modes of operation:

- **Normal capture mode:** The HRCAP module captures pulse widths in normal resolution within +/- 1 SYSCLK (where SYSCLK is sourced from the same PLL output clock that sources HCCAPCLK . This mode requires less software overhead than high-resolution capture mode.
- **High-resolution capture mode:** The HRCAP module captures pulse widths with the resolution of each edge captured within +/- 300 ps typical and requires the usage of the HCCal calibration library provided by Texas Instruments. In this mode, one HRCAP channel and the ePWM module connected to the HRCAP calibration input must be dedicated to HRCAP calibration and are not functionally available to the application during calibration.

5.3.2.1 HRCAP Counter

Both modes of operation utilize HCCOUNTER, which resets to 0 and starts counting HCCAPCLK cycles again under the following conditions:

- SOFTRESET
- Detection of rising edge
- Detection of falling edge
- Device reset and reenable of HRCAP clock

When a rising edge is detected, the value in HCCOUNTER is captured into the 16-bit HCCAPCNTRISE0 register before the counter resets to 0. When a falling edge is detected, the value in HCCOUNTER is captured into the 16-bit HCCAPCNTFALL0 register before the counter resets to 0. Because the HCCOUNTER starts counting at 0 after an edge is detected, the actual low and high pulse widths (non-high-resolution) are HCCAPCNTFALL0 + 1 and HCCAPCNTRISE0 + 1, respectively, where the “+1” is added to account for the “0” HCCAPCLK cycle. This behavior is illustrated for high pulse width capture in Figure 5-4.

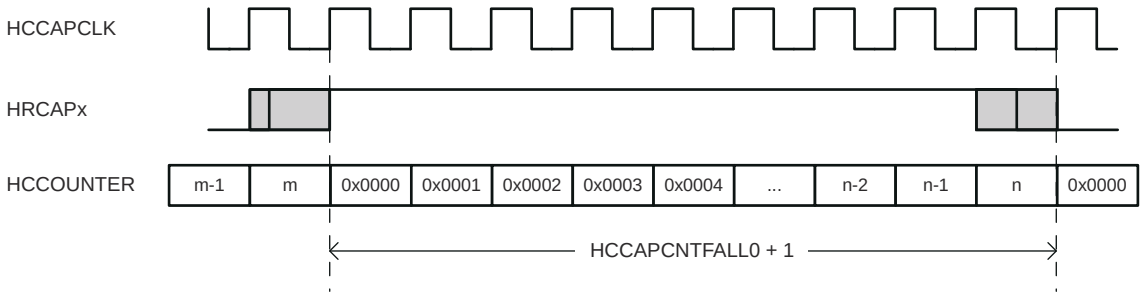


Figure 5-4. HCCOUNTER Behavior During High Pulse Width Capture

Because the HCCOUNTER starts counting immediately after SOFTRESET, the first capture result into the capture “0” registers should be discarded. The value captured will be the number of HCCAPCLK cycles since the last SOFTRESET or the HRCAPCLKEN bit was set rather than an actual pulse width measurement.

5.3.2.2 HCCAP0 - HCCAP1 Registers

The HRCAP capture registers include a 2-deep FIFO buffer to store two pulse widths’ worth of data.

When a rising edge event occurs, HCCAPCNTRISE0 is loaded with the pulse width data from the last falling edge to the current rising edge (low pulse width). At the next rising edge event, the value in the HCCAPCNTRISE0 register is loaded into HCCAPCNTRISE1.

When a falling edge event occurs, HCCAPCNTFALL0 is loaded with the pulse width data from the last rising edge to the current falling edge (high pulse width). For falling edge events, the HRCAP logic operates such that the HCCAPCNTFALL0 register value is then loaded into the HCCAPCNTFALL1 register at the next rising edge event rather than waiting until the next falling edge event.

5.3.2.3 RISE versus FALL Capture Events

HRCAP capture registers can be read either during rising edge capture events or during falling edge capture events. They should not be read during both events.

There are a number of differences with regard to using RISE events to read captured registers versus using FALL events to read captured registers as shown in Figure 5-5.

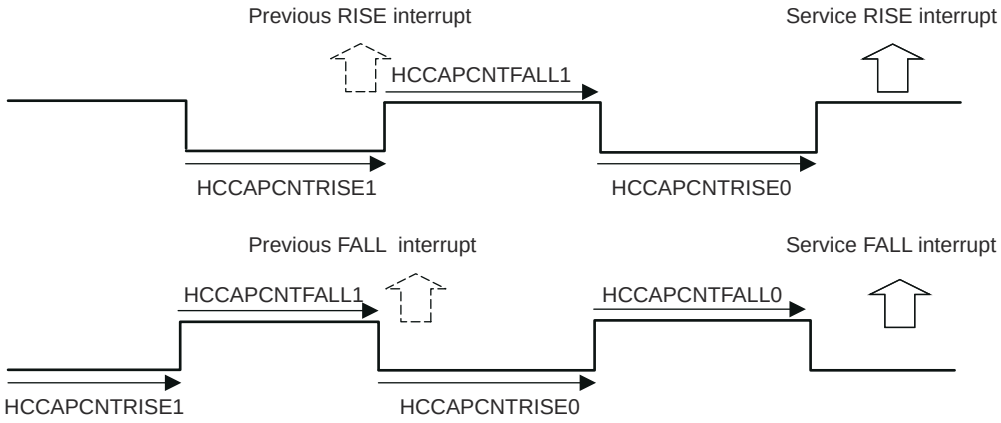


Figure 5-5. Rise versus Fall Capture Events

5.3.2.3.1 RISE Capture Events

When a RISE event occurs, the application code has access to full valid capture data for two pulse widths (1 period) in high-resolution capture mode and three pulse widths (1.5 periods) in normal capture mode. HCCAPCNTFALL0 does not have valid data available on RISE events, as this values are not captured until the falling edge event after the current rising edge event (event has not yet occurred).

The application code has until the next RISE event to read all relevant capture data and clear the RISE event. Otherwise, the data will be overwritten and invalid. Therefore, RISE events are generally used to capture data for period signals where duty cycle may vary significantly.

Note

Because HCCOUNTER starts counting immediately after SOFTRESET, the first RISE capture result into HCCAPCNTRISE0 does not include valid pulse width data and should be discarded. When the second RISE capture event occurs, this invalid data is transferred to HCCAPCNTRISE1, and therefore the data in this register should also be discarded. After the second rise interrupt, all capture data is valid and can be used normally.

5.3.2.3.2 FALL Capture Events

When a FALL event occurs, the application code has access to full valid capture data for three pulse widths (1.5 periods) in high-resolution capture mode and four pulse widths (2 full periods) in normal capture mode.

The application code has only until the next RISE event to read all relevant registers and clear the FALL event. Otherwise, the data will be overwritten and invalid. Therefore FALL events are generally used to capture short pulse widths spaced far enough apart to read the registers safely.

Note

Because HCCOUNTER starts counting immediately after SOFTRESET, the first FALL capture result into HCCAPCNTFALL0 does not include valid pulse width data and should be discarded. By the next FALL capture event, the invalid data in the “0” register has been transferred into HCCAPCNTFALL1. Therefore the data in this register should also be discarded. After the second FALL interrupt, all capture data is valid and can be used normally.

5.3.2.4 Normal Capture Mode

In normal capture mode, when a rise event (HCIFR[RISE]=1) or a fall event (HCIFR[FALL]=1) occurs, the application code reads the HCCAPCNTRISE0/1 and HCCAPCNTFALL0/1 registers and does not require the HCCal HRCAP calibration library. The resolution of the captured result will be accurate within +/- 1 SYSCLK cycles (where SYSCLK is sourced by the same PLLCLK that generates HCCAPCLK

High pulse widths are measured in number of HCCAPCLK cycles equal to 1 + HCCAPCNTFALL0 or 1 + HCCAPCNTFALL1 as shown in Figure 5-6.

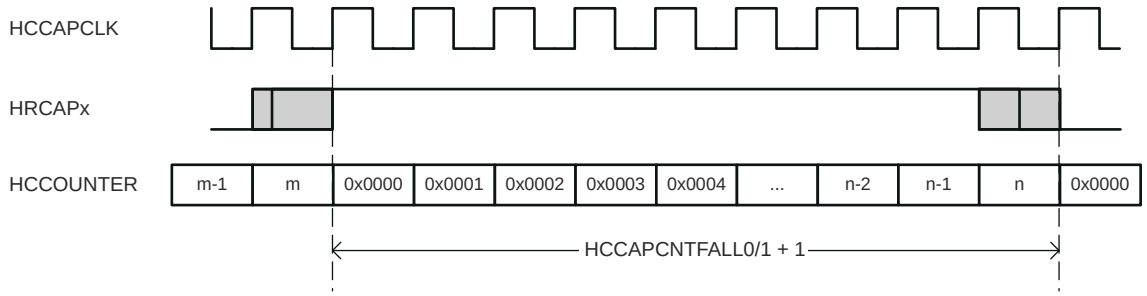


Figure 5-6. High Pulse Width Normal Mode Capture

Low pulse widths are measured in number of HCCAPCLK cycles equal to 1 + HCCAPCNTRISE0 or 1 + HCCAPCNTRISE1 as shown in Figure 5-7.

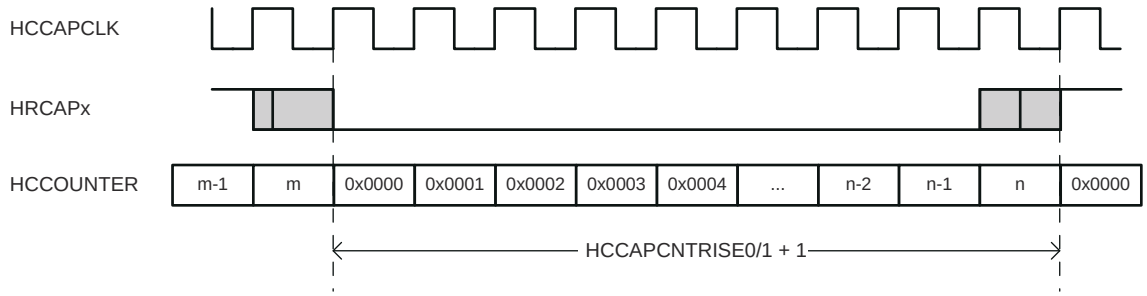


Figure 5-7. Low Pulse Width Normal Mode Capture

In both cases, 1 is added to the value in the HCCAPCNT registers to account for the HCCAPCLK cycle in which HCCOUNTER = 0.

5.3.2.5 High Resolution Capture Mode

In high-resolution capture mode, the application code utilizes the HCCal HRCAP calibration library functions to capture the high-resolution pulse width with each edge captured at a typical resolution of +/- 300 ps (with two edges, the resolution of the measured pulse width could vary by +/- 600 ps). Note that although the HRCAP logic itself can be calibrated to capture high-resolution pulse widths, if the jitter on the input signal is greater than +/- 300 ps, the captured value will also vary according to the jitter on the input signal.

In order to use high-resolution capture mode, the high-resolution capture logic must be calibrated to scale the HRCAP step size to a Q16 fraction of the HCCAPCLK. One HRCAP module and the ePWM module internally connected to the HRCAP calibration input must be dedicated only to calibration and cannot be used functionally in the application during calibration.

Texas Instruments provides a calibration function in the HCCal HRCAP calibration library to perform this calibration once prior to using the HRCAP in high-resolution capture mode and periodically in a slow loop to account for changes in the HRCAP step size due to voltage and temperature changes while the application is running.

The library also provides functions to measure the high resolution high pulse width, low pulse width, and period. The pulse and period width measurement results are returned in Q16 fixed-point format with the fractional portion of the result representing a fraction of an HCCAPCLK cycle. (For instance a pulse width may appear in the form of 500.25 HCCAPCLK cycles). Figure 5-8 shows how the high-resolution pulse width is a function of the calibration of the HRCAP step size and the values in the HCCAPCNT registers.

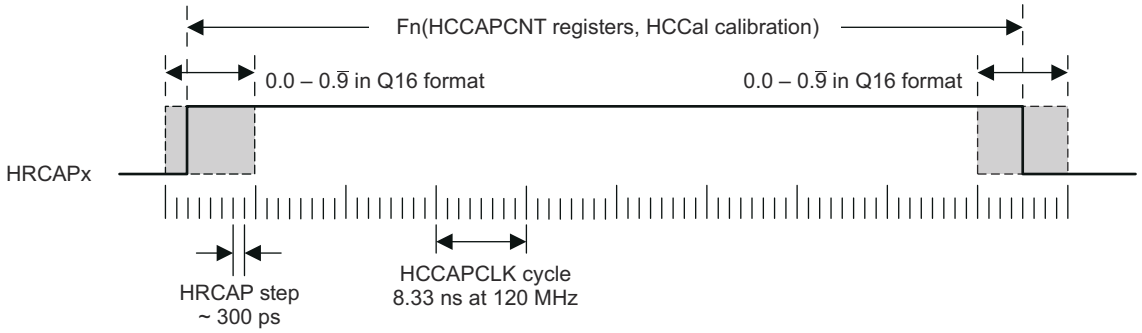


Figure 5-8. HRCAP High-Resolution Mode Operating Logic

For details on using the HCCal HRCAP calibration library in high-resolution capture mode, see Section 5.4.

5.3.3 HRCAP Interrupts

Rising edge capture (RISE), falling edge capture (FALL), and HCCOUNTER overflow (OVF) events can generate interrupts to the PIE from the HRCAP module. Additionally, if the rising edge capture flag is set (HCIFR[RISE]) when another rising edge capture event occurs, a rising edge overflow (RISEOVF) interrupt can also generate an interrupt to the PIE. The HRCAP interrupt logic is shown in Figure 5-9.

RISE/RISEOVF, FALL, and OVF events will only generate an interrupt if the corresponding interrupt enable bits in the HCCTL register are set to 1. Interrupt events can be cleared by writing a 1 to the corresponding bits in the HCICLR register. For testing purposes, interrupt events can be forced by writing a 1 to the corresponding bits in the HCIFRC register.

For proper operation, RISE and FALL interrupts should not be enabled at the same time. Capture registers should be read during rising edge interrupt events only, or during falling edge interrupt events only, and not during both interrupt events simultaneously. If RISEOVF interrupts are enabled, the RISE flag must always be acknowledged after a RISE event, otherwise a rise overflow condition will occur.

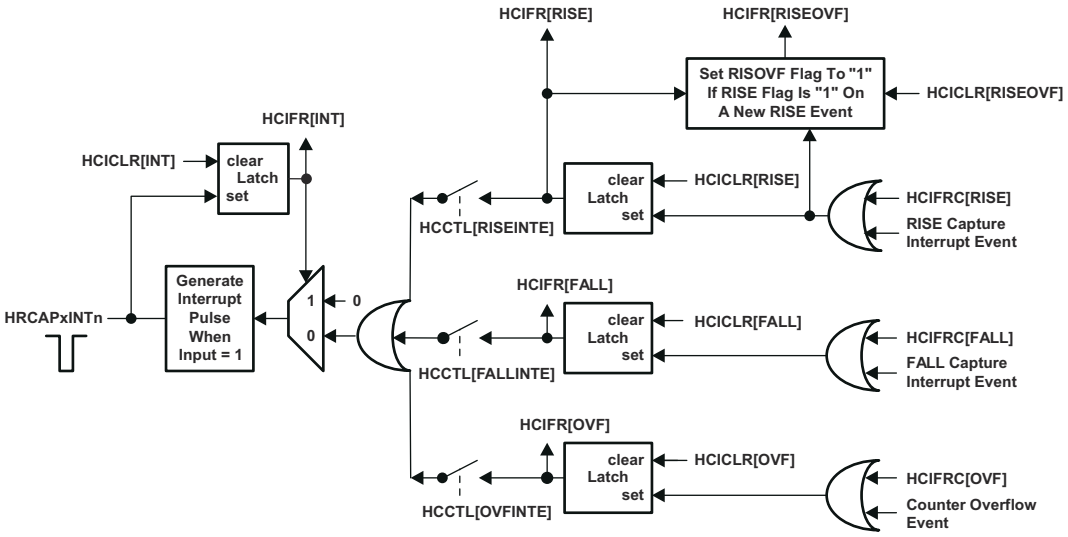


Figure 5-9. Interrupts in HRCAP Module

5.4 HRCAP Calibration Library

The HRCAP calibration (HCCal) logic is capable of capturing an edge in discrete time steps which subdivide an HCCAPCLK cycle. As previously mentioned, the size of each step is on the order of 300 ps (see device-specific data sheet for typical HRCAP step size on your device). The HRCAP_Cal() function in the HRCAP calibration library must be run periodically to be certain that time steps are optimally applied and that the edge capture accuracy is maintained over a wide range of PWM frequencies, system clock frequencies, voltages, and temperatures. The HRCAP step size varies based on worst-case process parameters, operating temperature, and voltage. HRCAP step size increases with decreasing voltage and increasing temperature and decreases with increasing voltage and decreasing temperature.

Applications that use the HRCAP in high-resolution capture mode should use the TI-supplied HRCAP calibration (HCCal) software HRCAP_Cal() function. The HRCAP_Cal function helps to dynamically scale the HRCAP step size to a fraction of the HCCAPCLK cycle while the HRCAP is in high-resolution mode. To utilize the HCCal capabilities effectively during HRCAP operation, the HRCAP calibration logic uses built-in self-check and diagnostics capabilities to scale the HRCAP step size appropriately for any operating condition.

TI provides a C-callable library containing one HRCAP calibration function that utilizes this hardware and properly calibrates the internal HRCAP step logic as a fraction of a HCCAPCLK cycle. The library supplies additional high-resolution capture functions to calculate pulse widths captured in Q16 integer + fractional HCCAPCLK cycles based on the values in the HCCAPCNT registers and the calibration results.

The contents of these functions are proprietary to Texas Instruments and will not be published.

Currently, there is 1 released version of the HCCal Type 0 library, HCCal_Type0_V1.lib, which is located in the C2000ware software package under the \libraries\calibration\hrcap\ directory.

5.4.1 HRCAP Calibration Library Functions

5.4.1.1 HRCAP_Cal

The HRCAP_Cal() function runs calibration and self-check diagnostics logic on a given HRCAP module to internally scale the HCCal step logic as a fraction of a HCCAPCLK cycle.

Prototype:

```
Uint16 HRCAP_Cal(Uint16 HRCAPModule, Uint16 PLLClk, volatile struct EPWM_REGS *ePWMModule);
```

Parameters:

HRCAPModule	The HRCAP module number as an integer value (that is, “1” for HRCAP1 and “2” for HRCAP2). This HRCAP module will be dedicated to calibration only and cannot be used functionally to capture pulse widths.
PLLClk	If 0, then HCCAPCLK is clocked by SYSCLK, where SYSCLK is the system clock that clocks the HRCAP module. If 1, then HCCAPCLK is clocked by PLLCLK, where the PLLCLK frequency is a multiple of the system clock that clocks the HRCAP module.
ePWMModule	A pointer to the address of the EPWM_REGS structure for the HRPWM module used to calibrate the HRCAP (that is, if HRPWM7A output is connected to the HRCAP’s internal calibration logic, then &EPwm7Regs is passed into this parameter. The single ePWM module used for HRCAP calibration is device-dependent.

Returns:

0	If HCCal calibration is in progress without encountering errors.
1	If HCCal has exited with errors. User should check that PLL is configured such that the HCCAPCLK frequency falls within the frequency limits designated by the HRCAP *Electricals* section of the data manual.
2	If HCCal calibration has completed without errors.

Description:

This function drives the HRCAP calibration module logic to subdivide an HCCAPCLK cycle into HRCAP time steps equivalent to a fraction of an HCCAPCLK cycle at any given time.

HRCAP_Cal() can only be used with HCCAPCLK between 98 MHz and 120 MHz (See your device-specific data manual’s HRCAP *Electricals* section for the device-specific HCCAPCLK frequency limits).

The function can be called at any time on a single HRCAP module which is dedicated to calibration only.

The calibration HRCAP module cannot be used functionally to capture pulse widths. The calibration logic driven by this function uses a single ePWMxA HRPWM channel output connected internally to the HRCAP input to run diagnostics. During calibration, that ePWM module cannot be used for normal ePWM functions in the application. For instance, on this device, while the HRCAP is in use in high-resolution capture mode, ePWM7 cannot be used functionally by the application during calibration.

5.4.1.2 LowPulseWidth0

The LowPulseWidth0() function captures the high-resolution low pulse width around HCCAPCNTRISE0 in HCCAPCLK cycles.

Prototype:

```
Uint32 LowPulseWidth0 (Uint16 * ptrHRCAPmodule);
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

Returns:

32-bit high-resolution low pulse width around HCCAPCNTRISE0 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

This function can be called for any of the HRCAP modules not used for calibration to convert the HCCAPCNTRISE0 and HRCAP calibration results into a fixed-point Q16 integer + fractional high-resolution low-pulse width in HCCAPCLK cycles. Figure 5-10 shows which low pulse widths can be captured on a RISE and FALL event.

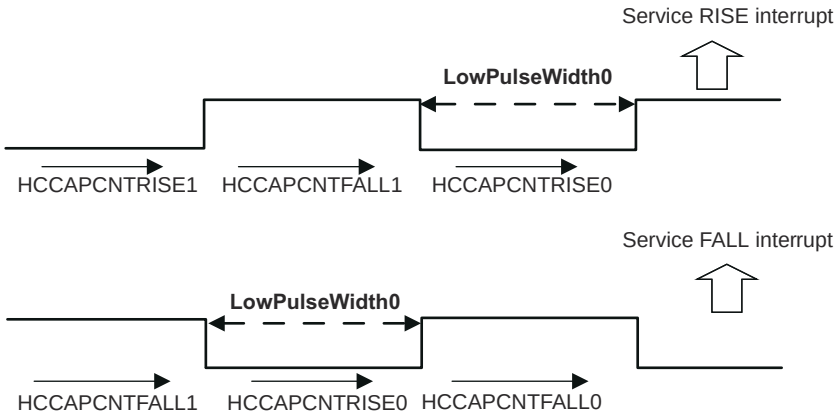


Figure 5-10. LowPulseWidth0 Capture on RISE and FALL Events

5.4.1.3 HighPulseWidth0 and HighPulseWidth1

The HighPulseWidth0()function captures the high-resolution high pulse width around HCCAPCNTFALL0 in HCCAPCLK cycles. The HighPulseWidth1()function captures the high-resolution high pulse width around HCCAPCNTFALL1 in HCCAPCLK cycles.

Prototypes:

```
Uint32 HighPulseWidth0 (Uint16 * ptrHRCAPmodule);
Uint32 HighPulseWidth1 (Uint16 * ptrHRCAPmodule);
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

Returns:

32-bit high-resolution high pulse width around HCCAPCNTFALL0/1 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

These functions can be called for any of the HRCAP modules not used for calibration. They use the calibration logic to convert the HCCAPCNTFALL0/1 register value and HCCal calibration results into a fixed-point Q16 integer + fractional high-resolution high-pulse width in HCCAPCLK cycles. Figure 5-11 shows which high pulse widths can be captured on a RISE and FALL event.

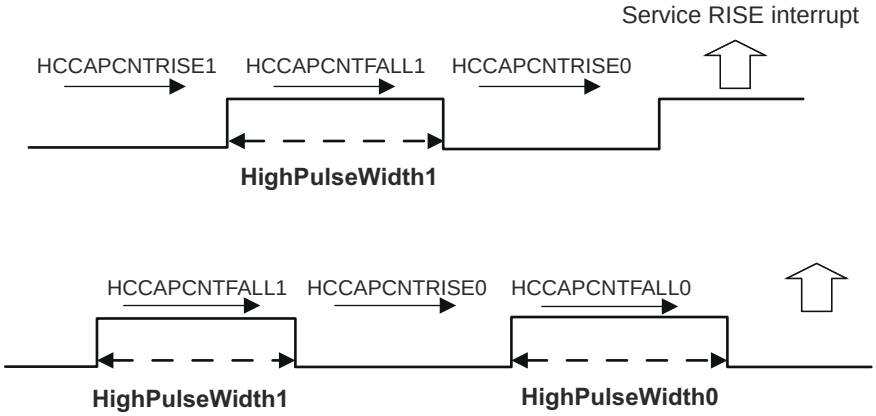


Figure 5-11. HighPulseWidth0/1 Capture on RISE and FALL Events

5.4.1.4 PeriodWidthRise0

The PeriodWidthRise0() function captures the rising edge to rising edge high-resolution period width around HCCAPCNTRISE0 and HCCAPCNTFALL1 in HCCAPCLK cycles.

Prototype:

```
Uint32 PeriodWidthRise0 (Uint16 * ptrHRCAPmodule)
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

Returns:

32-bit high-resolution rising edge to rising edge period width around HCCAPCNTRISE0+HCCAPCNTFALL1 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

This function can be called for any of the HRCAP modules not used for calibration. It uses the calibration logic to convert the HCCAPCNTRISE0 and HCCAPCNTFALL1 register values and HCCal calibration results into the function into a fixed-point Q16 integer + fractional high-resolution period width in HCCAPCLK cycles. Figure 5-12 shows which period widths can be captured on a RISE and FALL event.

5.4.1.5 PeriodWidthFall0

The PeriodWidthFall0()function captures the falling edge to falling edge high-resolution period width around HCCAPCNTFALL0 and HCCAPCNTRISE0 in HCCAPCLK cycles.

Prototype:

```
Uint32 PeriodWidthFall0 (Uint16 * ptrHRCAPmodule);
```

Parameters:

ptrHRCAPmodule	A 16-bit pointer to the first address of the HRCAP register block of the HRCAP module used to capture pulses.

Returns:

32-bit high-resolution falling edge to falling edge period width around HCCAPCNTFALL0+HCCAPCNTRISE0 as Q16 fixed-point value in number of HCCAPCLK cycles.

Description:

This function can be called for any of the HRCAP modules not used for calibration. It uses the calibration logic to convert the HCCAPCNTFALL0 and HCCAPCNTRISE0 register values and HCCal calibration results into a fixed-point Q16 integer + fractional high-resolution period width in HCCAPCLK cycles. Figure 5-12 shows which period widths can be captured on a RISE and FALL event.

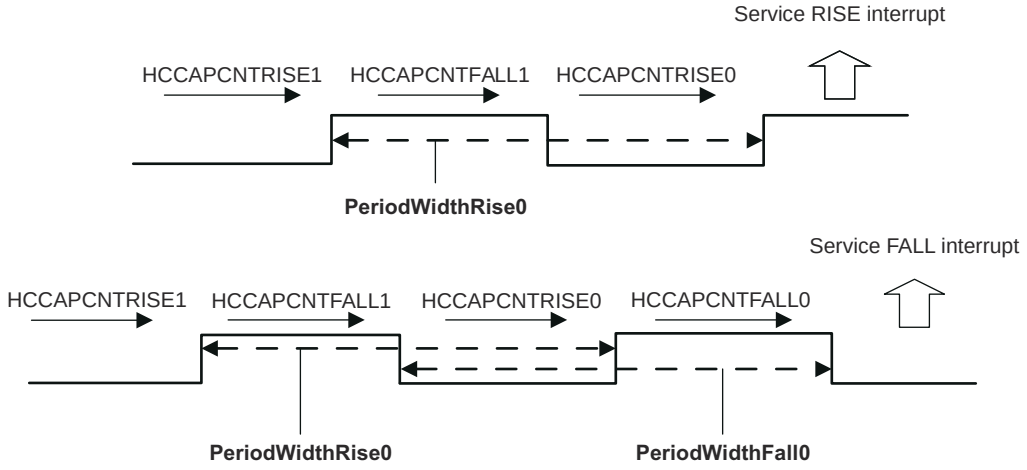


Figure 5-12. PeriodWidthRise0 and PeriodWidthFall0 Capture on RISE and FALL Events

5.4.2 HRCAP Calibration Library Software Usage

To use the HRCAP in high-resolution mode, it is recommended that the HRCAP calibration functions be used as described here.

Step 1: Add "Include" Files

The HCCal_Type0_V1.h file needs to be included as follows. This include file is mandatory while using the HRCAP calibration library functions. For these devices, the DSP2803x_Device.h and DSP2803x_Examples.h files in the F2803x C/C++ Header Files and Peripheral Examples package in C2000ware are necessary. For other device families, the device-specific equivalent files in the header files and peripheral examples software packages for those devices should be used. These include files are optional if customized header files are used in the end applications.

Example 1: A Sample of How to Add "Include" Files

```
#include "DSP2803x_Device.h"            // F2803x Headerfile
#include "DSP2803x_Examples.h"          // F2803x Examples Headerfile
#include "HCCal_Type0_V1.h"
```

Step 2: HRCAP Register Array Declaration

Declare an array of pointers to HRCAP_REGS structures which includes all available HRCAP modules on the device. Position 0 includes a 0 value which is not used by the HRCAP_Cal function.

Example 2: A Sample of How to Declare an HRCAP Register Array

```
#define NUM_HRCAP 3        // # of HRCAP modules on 2803x + 1
volatile struct HRCAP_REGS *HRCAP[NUM_HRCAP] = {0, &HRCap1Regs,
      &HRCap2Regs};
```

Step 3: HRCAP Pre-Calibration

Prior to using the HRCAP in high-resolution mode in application code, HRCAP_Cal() should be called to calibrate the HRCAP step size subdivision into HCCAPCLK.

As part of the one-time pre-calibration prior to using the HRCAP in high-resolution mode, include the following:

Example 3: A Sample of HRCAP Pre-Calibration

```
while (status!= HCCAL_COMPLETE)  // While calibration is incomplete
  {
                                 // Use HRCAP2 to calibrate with:
                                 // HCCAPCLK = PLLCLK
                                 // ePWM7A = HRCAP calibration input
       status = HRCAP_Cal(2,HCCAPCLK_PLLCLK, &EPwm7Regs);
       if (status == HCCAL_ERROR)
       {
    ESTOP0;              // Error, stop and check HCCAPCLK frequency
       }
   }
```

Step 4: Application Code Calibration

While the application is running, fluctuations in both device temperature and supply voltage may be expected. To be sure that optimal HRCAP step size is used for each HRCAP module, the HRCAP_Cal function should be re-run periodically as part of a slower back-ground loop. Some examples of this are shown here.

Note

NOTE: See the hrcap_capture_hrpwm example in the device-specific C/C++ header files and peripheral examples available in C2000ware from the TI website

Example 4: HRCAP_Cal Function Calls

```
main ()
{
    int status;
    // User code
    // HRCAP 1, is running in high-resolution mode
    // The status variable returns 2 once calibration has been
    // completed by the HCCal Calibration Module running
    // diagnostics.
    status = HRCAP_Cal(2,HCCAPCLK_PLLCLK, &EPwm7Regs);
    // The function returns a 2 if HCCAPCLK is not within the
    // appropriate frequency range.
    if(status==HCCAL_ERROR) {ESTOP0;}
}
```

Step 5: Application Code Pulse Width Measurement

While the application is running, when a RISE or FALL event occurs, pulse and period widths can be measured using the high resolution pulse width functions as shown below.

Example 5: LowPulseWidth, HighPulseWidth, and PeriodWidth Function Calls

```
interrupt void HRCAP1_Isr (void)
{
    EALLOW;
if (HRCap1Regs.HCIFR.bit.RISEOVF == 1) {
ESTOP0;         // Another rising edge detected
    }
       if (first < 1) {
            first++;            // Discard first data (because first interrupt
                                // after reset/clk enable measures time from
                                // clock start to edge - invalid pulse width)
} else {
               periodwidth =    PeriodWidthRise0((Uint16 *)&HRCap1Regs);
               pulsewidthlow =  LowPulseWidth0((Uint16 *)&HRCap1Regs);
               pulsewidthhigh = HighPulseWidth0((Uint16 *)&HRCap1Regs);
  }
    HRCap1Regs.HCICLR.bit.RISE=1;
    HRCap1Regs.HCICLR.bit.INT=1;
    PieCtrlRegs.PIEACK.bit.ACK4=1;
    EDIS;
}
```

5.5 HRCAP Register Descriptions

The HRCAP register set is shown in Table 5-1.

Table 5-1. HRCAP Register Summary

Name	Address Offset	Description	Section
HCCTL	0x00	HRCAP Control Register	Section 5.5.1
HCIFR	0x01	HRCAP Interrupt Flag Register	Section 5.5.2
HCICLR	0x02	HRCAP Interrupt Clear Register	Section 5.5.3
HCIFRC	0x03	HRCAP Interrupt Force Register	Section 5.5.4
HCCOUNTER	0x04	HRCAP 16-bit Counter Register	Section 5.5.5
HCCAPCNTRISE0	0x10	HRCAP Capture Counter On Rising Edge 0 Register	Section 5.5.6
HCCAPCNTFALL0	0x12	HRCAP Capture Counter On Falling Edge 0 Register	Section 5.5.7
HCCAPCNTRISE1	0x18	HRCAP Capture Counter On Rising Edge 1 Register	Section 5.5.8
HCCAPCNTFALL1	0x1A	HRCAP Capture Counter On Falling Edge 1 Register	Section 5.5.9

5.5.1 HRCAP Control Register (HCCTL)

The HRCAP control register (HCCTL) is shown and described in the figure and table below.

Figure 5-13. HRCAP Control Register (HCCTL)

15	9	8
Reserved		HCCAPCLKSEL
R-0		R/W-0

7	4	3	2	1	0
Reserved		OVFINTE	FALLINTE	RISEINTE	SOFTRESET
R-0		R/W-0	R/W-0	R/W-0	R/W-0

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

Table 5-2. HRCAP Control Register (HCCTL) Field Descriptions

Bit	Field	Value	Description [1]
15-9	Reserved		Reserved
8	HCCAPCLKSEL		Capture clock select bit. This bit is used to select the clock source for HCCAPCLK. This bit should be set such that HCCAPCLK falls between the frequency range limits specified in the HRCAP *Electricals* section of the device-specific data manual.
		0	HCCAPCLK = SYSCLK
		1	HCCAPCLK = PLLCLK (bypasses the clock divider)
7-4	Reserved		Reserved
3	OVFINTE		Counter overflow interrupt enable bit
		0	Disable counter overflow interrupt
		1	Enable counter overflow interrupt
2	FALLINTE		Falling edge capture interrupt enable bit
		0	Disable falling edge capture interrupt
		1	Enable rising edge capture interrupt
1	RISEINTE		Rising edge capture interrupt enable bit
		0	Disable rising edge capture interrupt
		1	Enable rising edge capture interrupt
0	SOFTRESET		Soft reset
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear HCCOUNTER, all capture registers, and the IFR register bits.

(1) This register is EALLOW protected.

5.5.2 HRCAP Interrupt Flag Register (HCIFR)

The HRCAP interrupt flag register (HCIFR) is shown and described in the figure and table below.

Figure 5-14. HRCAP Interrupt Flag Register (HCIFR)

15						8
Reserved						
R-0						

7	5	4	3	2	1	0
Reserved		RISEOVF	COUNTEROVF	FALL	RISE	INT
R-0		R-0	R-0	R-0	R-0	R-0

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

Table 5-3. HRCAP Interrupt Flag Register (HCIFR) Field Descriptions

Bit	Field	Value	Description (1)
15-5	Reserved		Reserved
4	RISEOVF		Rising edge interrupt overflow event flag
		0	No rising edge interrupt overflow event has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	This bit is set to "1" if the RISE flag is "1" when a new RISE event occurs.
3	COUNTEROVF		Counter overflow interrupt flag
		0	The HCCOUNTER has not overflowed. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	This bit is set to 1 when the 16-bit HCCOUNTER overflows (from 0xFFFF to 0x0000). This bit can also be set to 1 by writing to the corresponding bit in the HCIFRC register.
2	FALL		Falling edge capture interrupt flag:
		0	No falling edge interrupt has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	A falling edge input capture event has occurred. This bit can also be set to 1 by writing to the corresponding bit in the HCIFRC register.
1	RISE		Rising edge capture interrupt flag
		0	No rising edge interrupt has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	A rising edge input capture event has occurred. This bit can also be set to 1 by writing to the corresponding bit in the HCIFRC register.
0	INT		Global interrupt flag
		0	No HRCAP interrupt has occurred. This bit is cleared to 0 by writing to the corresponding bit in the HCICLR register. This bit is also cleared by HCCTL[SOFTRESET].
		1	An enabled RISE, FALL or COUNTEROVF interrupt has been generated. No further interrupts are generated until this bit is cleared.

(1) This register is EALLOW protected.

5.5.3 HRCAP Interrupt Clear Register (HCICLR)

The HRCAP interrupt clear register (HCICLR) is shown and described in the figure and table below.

Figure 5-15. HRCAP Interrupt Clear Register (HCICLR)

15	8
Reserved	
R-0	

7	5	4	3	2	1	0
Reserved		RISEOVF	COUNTEROVF	FALL	RISE	INT
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 5-4. HRCAP Interrupt Clear Register (HCICLR) Field Descriptions

Bit	Field	Value	Description (1)
15-5	Reserved		Reserved
4	RISEOVF		Rising edge interrupt overflow clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding RISEOVF flag bit in the HCIFR register to "0". The hardware setting of HCIFR[RISEOVF] flag bit has priority over the software clear if both happen on the same cycle.
3	COUNTEROVF		Counter overflow interrupt clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding COUNTEROVF flag bit in the HCIFR register to "0". The hardware setting of HCIFR[COUNTEROVF] flag bit has priority over the software clear if both happen on the same cycle.
2	FALL		Falling edge capture interrupt clear bit:
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding FALL flag bit in the HCIFR register to "0". The hardware setting of HCIFR[FALL] flag bit has priority over the software clear if both happen on the same cycle.
1	RISE		Rising edge capture interrupt clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding RISE flag bit in the HCIFR register to "0". The hardware setting of HCIFR[RISE] flag bit has priority over the software clear if both happen on the same cycle.
0	INT		Global interrupt clear bit
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will clear the corresponding INT flag bit in the HCIFR register to "0". The hardware setting of HCIFR[INT] flag bit has priority over the software clear if both happen on the same cycle.

(1) This register is EALLOW protected.

5.5.4 HRCAP Interrupt Force Register (HCIFRC)

The HRCAP interrupt force register (HCIFRC) is shown and described in the figure and table below.

Figure 5-16. HRCAP Interrupt Force Register (HCIFRC)

15	8
Reserved	
R-0	

7	4	3	2	1	0
Reserved		COUNTEROVF	FALL	RISE	Reserved
R-0		R/W-0	R/W-0	R/W-0	R-0

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

Table 5-5. HRCAP Interrupt Force Register (HCIFRC) Field Descriptions

Bit	Field	Value	Description [(1)]
15-4	Reserved		Reserved
3	COUNTEROVF		Counter overflow interrupt force bits
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will force the corresponding COUNTEROVF flag bit in HCIFR register to "1". The software HCCTL[SOFTRESET] clearing of the HCIFR[COUNTEROVF] bit has priority over the hardware trying to set the bit in the same cycle.
2	FALL		Falling edge interrupt force bits
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will force the corresponding FALL flag bit in HCIFR register to "1". The software HCCTL[SOFTRESET] clearing of the HCIFR[FALL] bit has priority over the hardware trying to set the bit in the same cycle.
1	RISE		Rising edge interrupt force bits
		0	Writes of "0" are ignored. This bit always reads "0".
		1	Writes of "1" to this bit will force the corresponding RISE flag bit in HCIFR register to "1". The software HCCTL[SOFTRESET] clearing of the HCIFR[RISE] bit has priority over the hardware trying to set the bit in the same cycle.
0	Reserved		Reserved

(1) This register is EALLOW protected.

5.5.5 HRCAP Counter Register (HCCOUNTER)

The HRCAP counter register (HCCOUNTER) is shown and described in the figure and table below.

Figure 5-17. HRCAP Counter Register (HCCOUNTER)

15	0
COUNTER	
R-0	

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

Table 5-6. HRCAP Counter Register (HCCOUNTER) Field Descriptions

Bit	Field	Value	Description
15-0	COUNTER	0	16-bit capture counter This free running counter is used to capture rising and falling edge events. The HCCOUNTER is incremented on every HCCAPCLK cycle. When the counter reaches 0xFFFF, it will overflow to 0x0000 on the next cycle and generate a COUNTEROVF interrupt event. The counter is reset to 0x0000 on every rising and falling edge event. The counter can also be reset to 0x0000 by a system reset or by setting the HCCTL[SOFTRESET] bit. **NOTE:** Because the counter is clocked from HCCAPCLK, which can be asynchronous to SYSCLK, CPU reads to this register should not be performed unless the clocks to the HRCAP module are disabled (HRCAPxENCLK = 0).

5.5.6 HRCAP Capture Counter On Rising Edge 0 Register (HCCAPCNTRISE0)

The HRCAP capture counter on rising edge 0 register (HCCAPCNTRISE0) is shown and described in the figure and table below.

Figure 5-18. HRCAP Capture Counter On Rising Edge 0 Register (HCCAPCNTRISE0)

15	0
HCCAPCNTRISE0	
R-0	

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

Table 5-7. HRCAP Capture Counter On Rising Edge 0 Register (HCCAPCNTRISE0) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTRISE0	0	HRCAP capture counter on rising edge 0 register This register captures the16-bit HCCOUNTER value when a rising edge event is detected.

5.5.7 HRCAP Capture Counter On Falling Edge 0 Register (HCCAPCNTFALL0)

The HRCAP capture counter on falling edge 0 register (HCCAPCNTFALL0) is shown and described in the figure and table below.

Figure 5-19. HRCAP Capture Counter On Falling Edge 0 Register (HCCAPCNTFALL0)

15	0
HCCAPCNTFALL0	
R-0	

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

Table 5-8. HRCAP Capture Counter On Falling Edge 0 Register (HCCAPCNTFALL0) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTFALL0	0	HRCAP capture counter on falling edge 0 register This register captures the16-bit HCCOUNTER value when a Falling edge event is detected.

5.5.8 HRCAP Capture Counter On Rising Edge 1 Register (HCCAPCNTRISE1)

The HRCAP capture counter on rising edge 1 register (HCCAPCNTRISE1) is shown and described in the figure and table below.

Figure 5-20. HRCAP Capture Counter On Rising Edge 1 Register (HCCAPCNTRISE1)

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

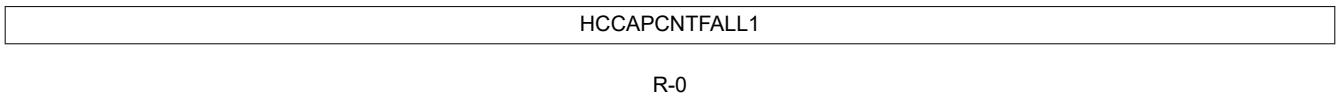
Table 5-9. HRCAP Capture Counter On Rising Edge 1 Register (HCCAPCNTRISE1) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTRISE1	0	HRCAP capture counter on rising edge 1 register On an input rising edge event, the value in the HCCAPCNTRISE0 register is copied into the HCCAPCNTRISE1 register before the HCCOUNTER value is captured into the HCCAPCNTRISE0 register.

5.5.9 HRCAP Capture Counter On Falling Edge 1 Register (HCCAPCNTFALL1)

The HRCAP capture counter on falling edge 1 register (HCCAPCNTFALL1) is shown and described in the figure and table below.

Figure 5-21. HRCAP Capture Counter On Falling Edge 1 Register (HCCAPCNTFALL1)



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

Table 5-10. HRCAP Capture Counter On Falling Edge 1 Register (HCCAPCNTFALL1) Field Descriptions

Bit	Field	Value	Description
15-0	HCCAPCNTFALL1	0	HRCAP capture counter on falling edge 1 register On an input falling edge event, the value in the HCCAPCNTFALL0 register is copied into the HCCAPCNTFALL1 register before the HCCOUNTER value is captured into the HCCAPCNTFALL0 register.

Chapter 6

Enhanced Capture (eCAP)

This chapter describes the enhanced capture (eCAP) module, which is used in systems where accurate timing of external events is important.

The enhanced capture (eCAP) module is a Type 0 eCAP. See the *C2000 Real-Time Control Peripheral Reference Guide* for a list of all devices with an eCAP module of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

6.1 Introduction

6.1.1 Features

The features of the eCAP module include:

- Speed measurements of rotating machinery (for example, toothed sprockets sensed by way of Hall sensors)
- Elapsed time measurements between position sensor pulses
- Period and duty cycle measurements of pulse train signals
- Decoding current or voltage amplitude derived from duty cycle encoded current/voltage sensors

The eCAP module features described in this chapter include:

- 4-event time-stamp registers (each 32 bits)
- Edge polarity selection for up to four sequenced time-stamp capture events
- Interrupt on either of the four events
- Single-shot capture of up to four event time-stamps
- Continuous mode capture of time stamps in a four-deep circular buffer
- Absolute time-stamp capture
- Difference (Delta) mode time-stamp capture
- All above resources are dedicated to a single input pin
- When not used in capture mode, the eCAP module can be configured as a single-channel PWM output

6.1.2 ECAP Related Collateral

Getting Started Materials

- *Leveraging High Resolution Capture (HRCAP) for Single Wire Data Transfer Application Report*

6.2 Description

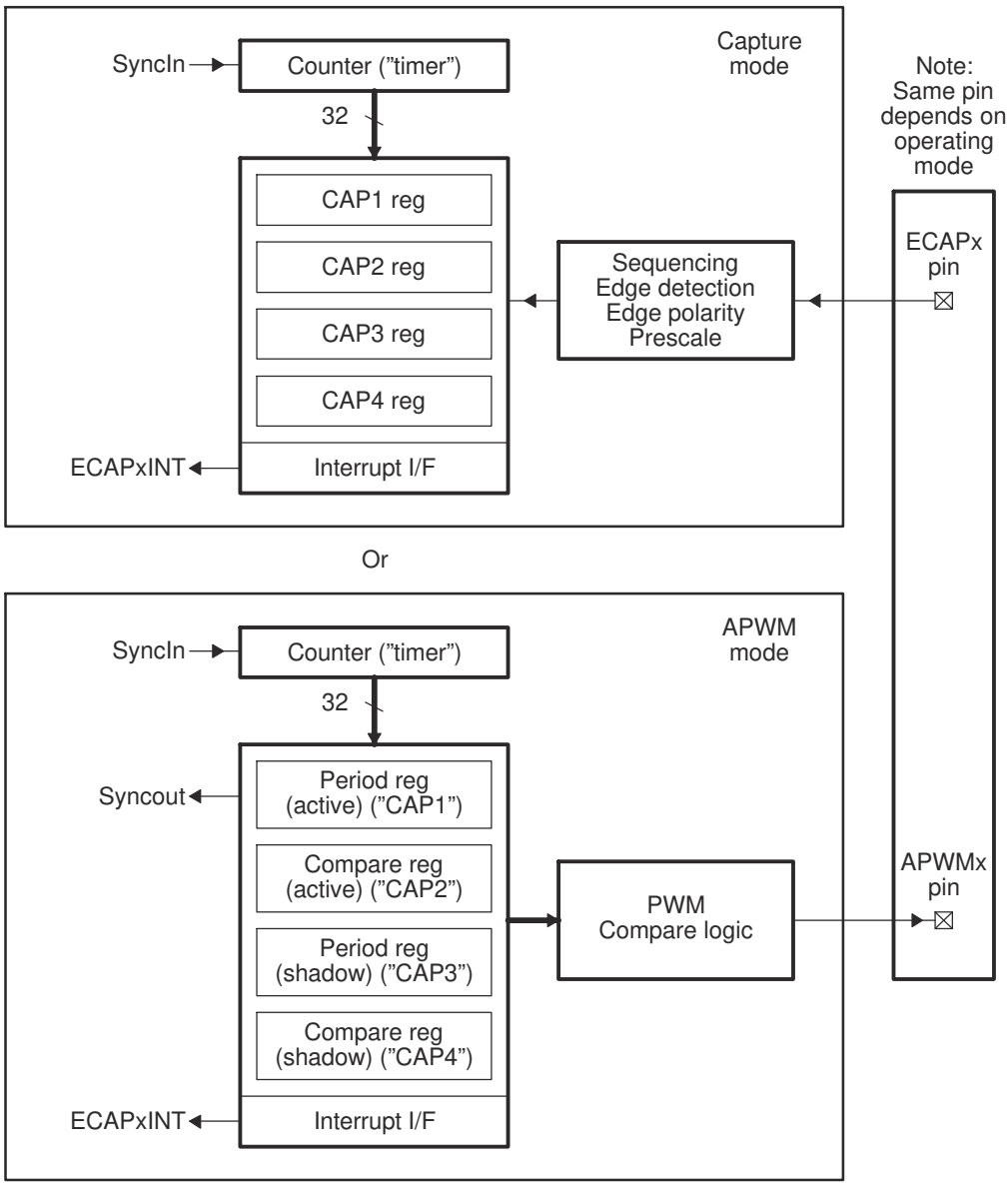
The eCAP module represents one complete capture channel that can be instantiated multiple times, depending on the target device. In the context of this guide, one eCAP channel has the following independent key resources:

- Dedicated input capture pin
- 32-bit time base (counter)
- 4 x 32-bit time-stamp capture registers (CAP1-CAP4)
- 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

6.3 Capture and APWM Operating Mode

You can use the eCAP module resources to implement a single-channel PWM generator (with 32-bit capabilities) when it is not being used for input captures. The counter operates in count-up mode, providing a time-base for asymmetrical pulse width modulation (PWM) waveforms. The CAP1 and CAP2 registers become the active period and compare registers, respectively, while CAP3 and CAP4 registers become the period and compare shadow registers, respectively. Figure 6-1 is a high-level view of both the capture and auxiliary pulse-width modulator (APWM) modes of operation.

Figure 6-2 further descries the output of the eCAP in APWM mode based on the CMP and PRD values.



A. A single pin is shared between CAP and APWM functions. In capture mode, it is an input; in APWM mode, it is an output.

B. In APWM mode, writing any value to CAP1/CAP2 active registers also writes the same value to the corresponding shadow registers CAP3/CAP4. This emulates immediate mode. Writing to the shadow registers CAP3/CAP4 invokes the shadow mode.

Figure 6-1. Capture and APWM Modes of Operation

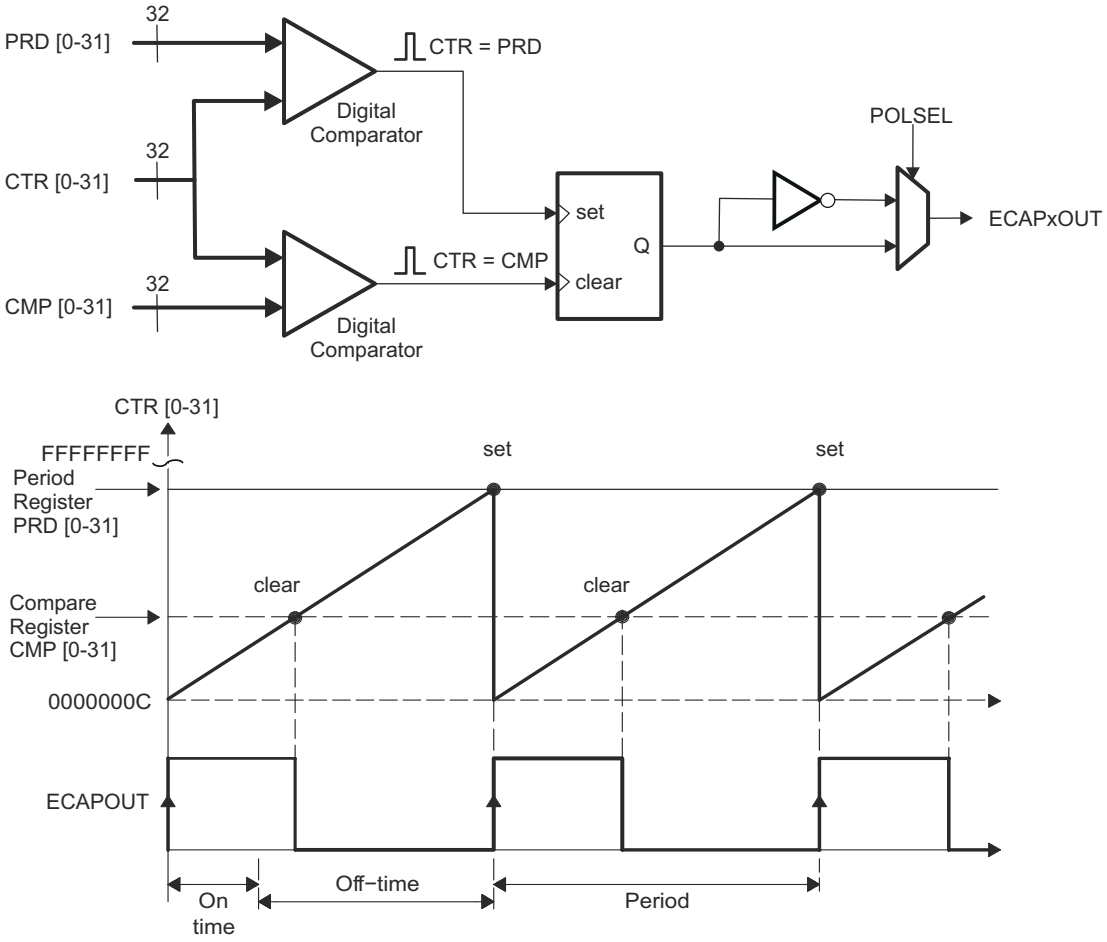


Figure 6-2. Counter Compare and PRD Effects on the eCAP Output in APWM Mode

6.4 Capture Mode Description

Figure 6-3 shows the various components that implement the capture function.

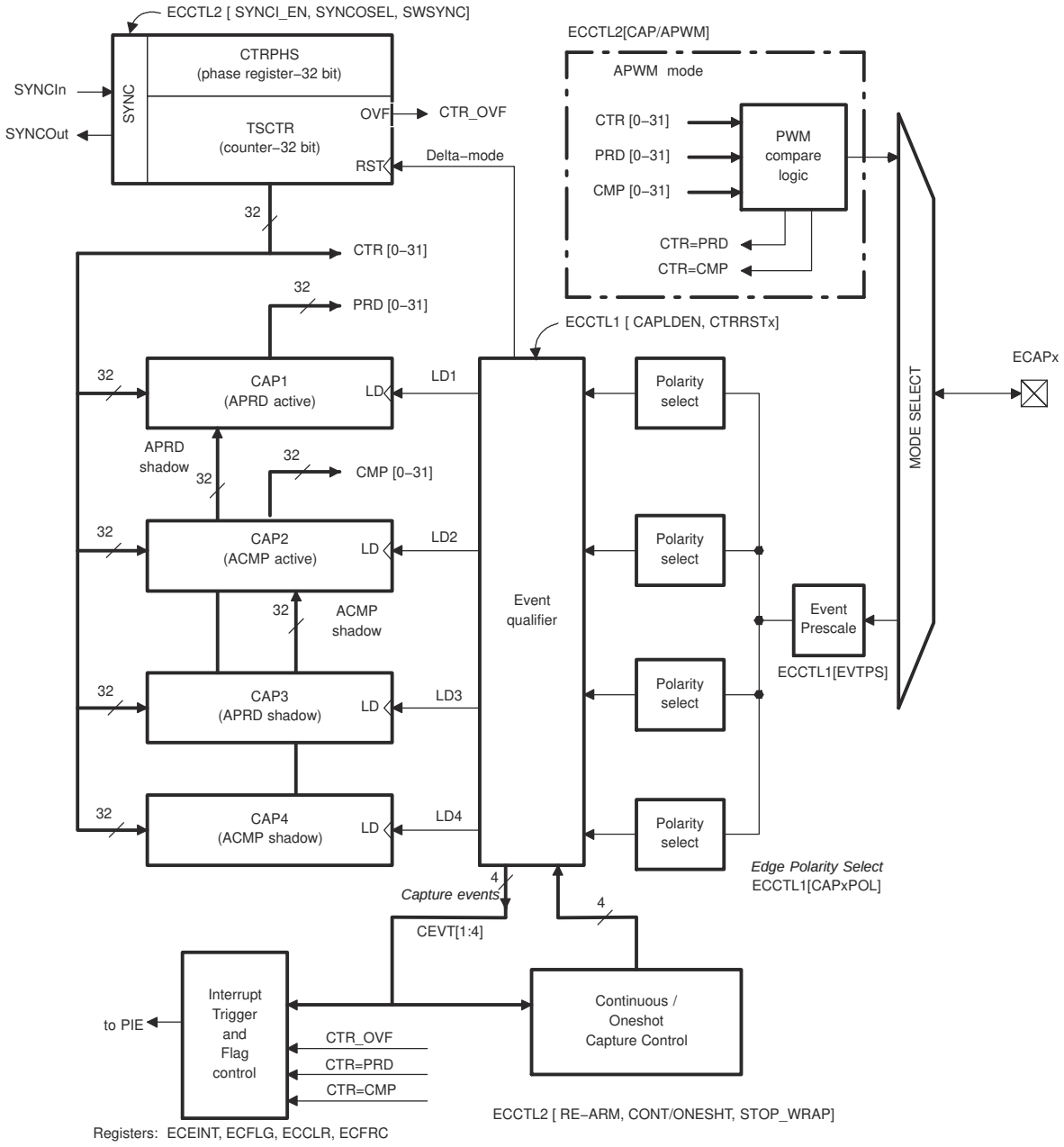
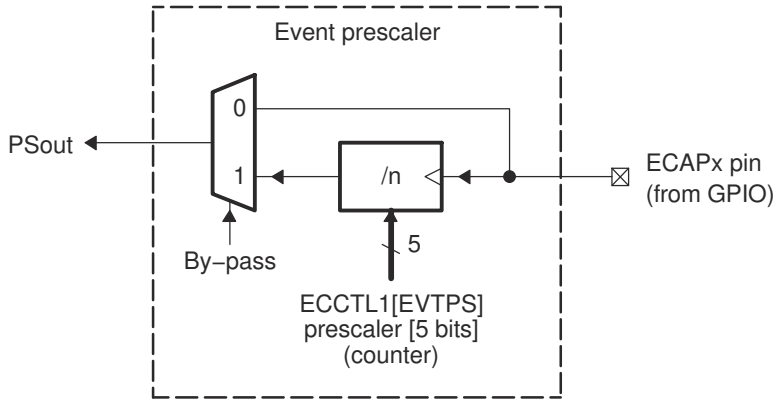


Figure 6-3. eCAP Block Diagram

6.4.1 Event Prescaler

An input capture signal (pulse train) can be prescaled by N = 2-62 (in multiples of 2) or can bypass the prescaler. This is useful when very high frequency signals are used as inputs. Figure 6-4 shows a functional diagram and Figure 6-5 shows the operation of the prescale function.



A. When a prescale value of 1 is chosen (ECCTL1[13:9] = 0,0,0,0,0), the input capture signal bypasses the prescale logic completely.
B. The first Rise edge after Prescale configuration change is not passed to Capture logic, prescaler value takes into effect on the second rising edge after the configuration.

Figure 6-4. Event Prescale Control

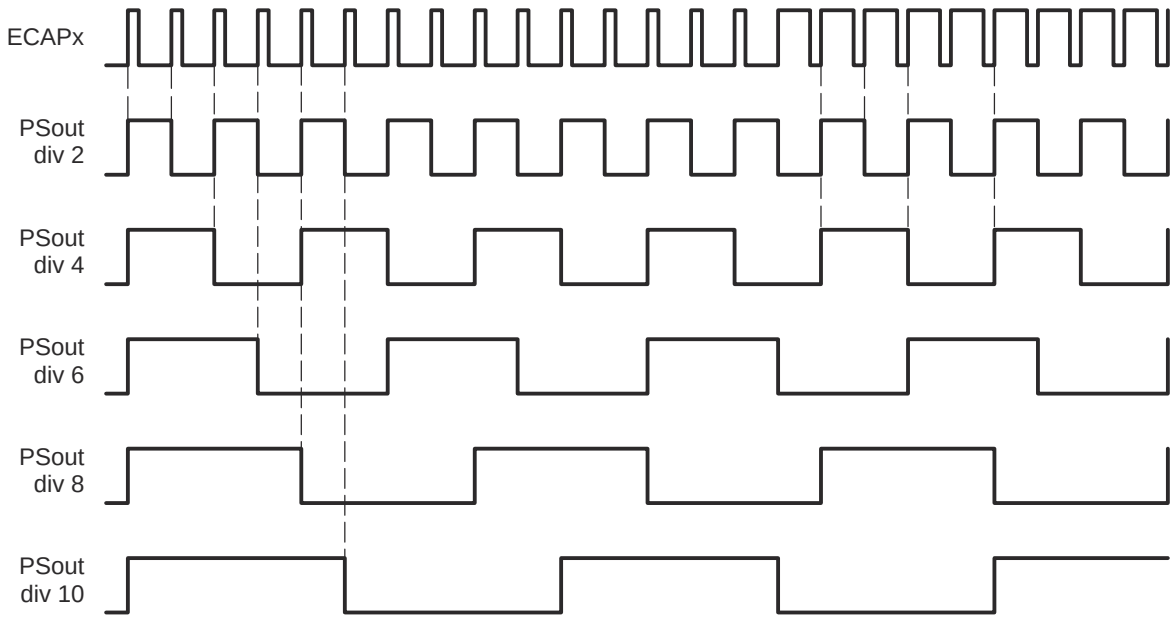


Figure 6-5. Prescale Function Waveforms

6.4.2 Edge Polarity Select and Qualifier

Functionality and features include:

- Four independent edge polarity (rising edge/falling edge) selection muxes are used, one for each capture event.
- Each edge (up to 4) is event qualified by the Modulo4 sequencer.
- The edge event is gated to its respective CAPx register by the Mod4 counter. The CAPx register is loaded on the falling edge.

6.4.3 Continuous/One-Shot Control

Operation of eCAP in Continuous/One-Shot mode:

- The Mod4 (2-bit) counter is incremented via edge qualified events (CEVT1-CEVT4).
- The Mod4 counter continues counting (0->1->2->3->0) and wraps around unless stopped.
- During one-shot operation, a 2-bit stop register (STOP_WRAP) is used to compare the Mod4 counter output, and when equal, stops the Mod4 counter and inhibits further loads of the CAP1-CAP4 registers. In this mode if TSCCTR counter is configured to reset on capture event (CEVTx) by configuring ECCTL1.CTRRSTx bit, it will still keep resetting the TSCCTR counter on capture event (CEVTx) after the STOP_WRAP value is reached and re-arm (REARM) has not occurred.

The continuous/one-shot block controls the start, stop and reset (zero) functions of the Mod4 counter, via a mono-shot type of action that can be triggered by the stop-value comparator and re-armed via software control.

Once armed, the eCAP module waits for 1-4 (defined by stop-value) capture events before freezing both the Mod4 counter and contents of CAP1-4 registers (time stamps).

Re-arming prepares the eCAP module for another capture sequence. Also, re-arming clears (to zero) the Mod4 counter and permits loading of CAP1-4 registers again, providing the CAPLDEN bit is set.

In continuous mode, the Mod4 counter continues to run (0->1->2->3->0, the one-shot action is ignored, and capture values continue to be written to CAP1-4 in a circular buffer sequence.

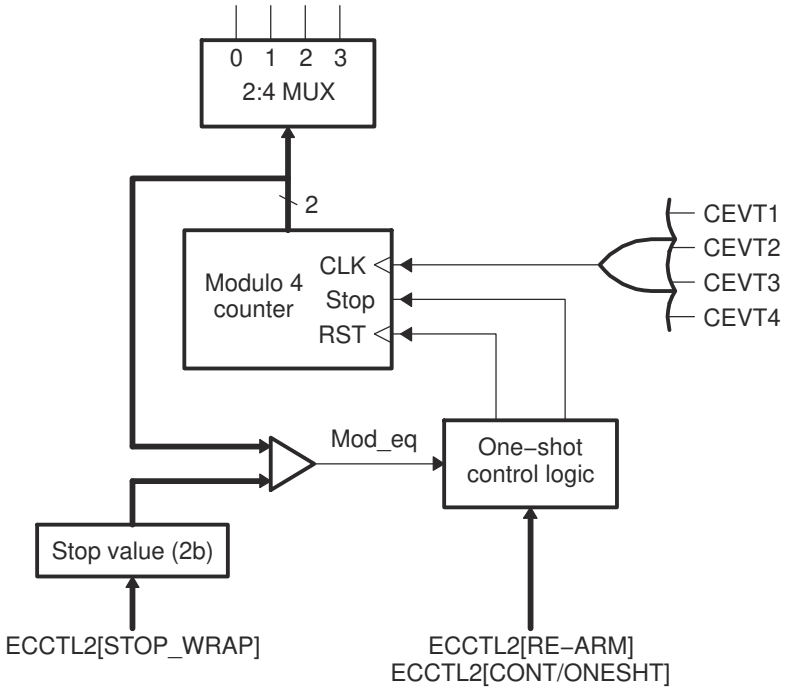


Figure 6-6. Details of the Continuous/One-shot Block

6.4.4 32-Bit Counter and Phase Control

This counter provides the time-base for event captures, and is clocked via the system clock.

A phase register is provided to achieve synchronization with other counters, via a hardware and software forced sync. This is useful in APWM mode when a phase offset between modules is needed.

On any of the four event loads, an option to reset the 32-bit counter is given. This is useful for time difference capture. The 32-bit counter value is captured first, then it is reset to 0 by any of the LD1-LD4 signals.

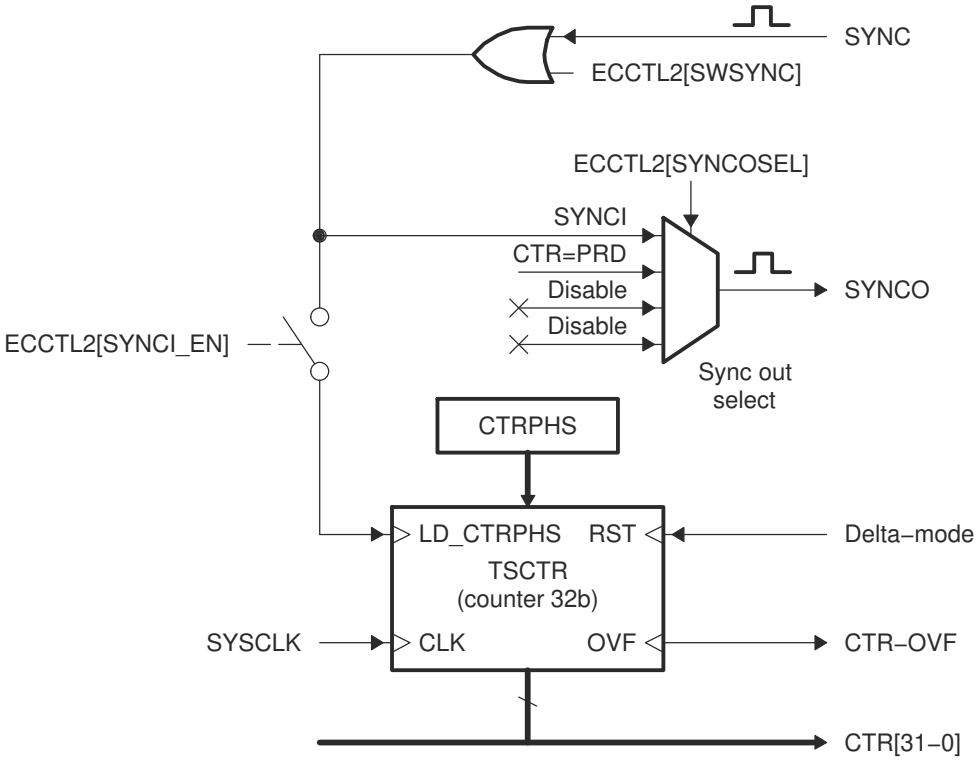


Figure 6-7. Details of the Counter and Synchronization Block

6.4.5 CAP1-CAP4 Registers

These 32-bit registers are fed by the 32-bit counter timer bus, CTR[0-31] and are loaded (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.

6.4.6 Interrupt Control

Operation and features of the eCAP interrupt control include (see Figure 6-8):

- 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.

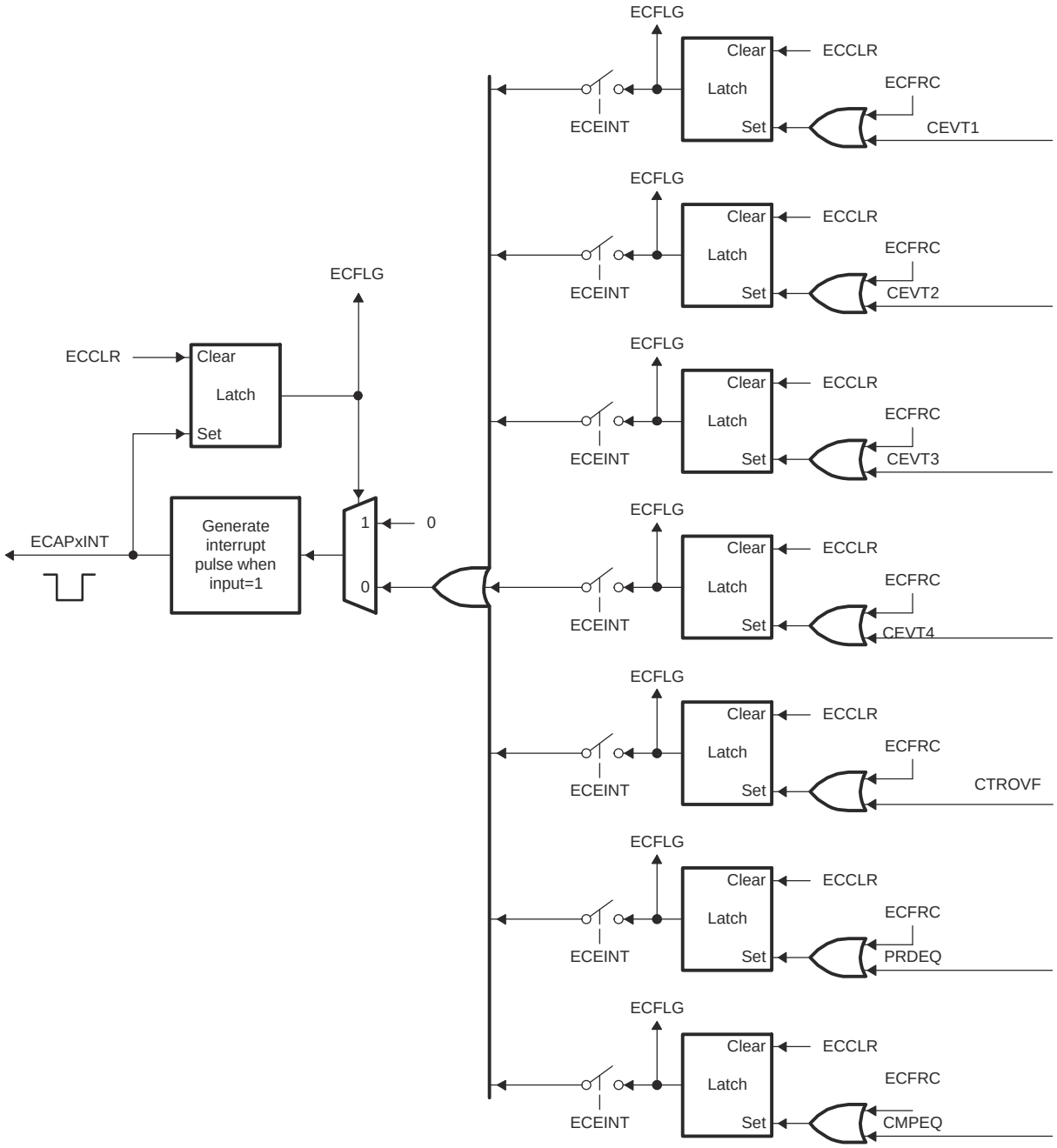


Figure 6-8. Interrupts in eCAP Module

6.4.7 Shadow Load and Lockout Control

In capture mode, this logic inhibits (locks out) any shadow loading of CAP1 or CAP2 from APRD and ACMP registers, respectively.

In APWM mode, shadow loading is active and two choices are permitted:

- Immediate - APRD or ACMP are transferred to CAP1 or CAP2 immediately upon writing a new value.
- On period equal, CTR[31:0] = PRD[31:0].

6.4.8 APWM Mode Operation

Main operating highlights of the APWM section:

- The time-stamp counter bus is made available for comparison by way of 2 digital (32-bit) comparators.
- When CAP1/2 registers are not used in capture mode, their contents can be used as Period and Compare values in APWM mode.
- Double buffering is achieved via shadow registers APRD and ACMP (CAP3/4). The shadow register contents are transferred over to CAP1/2 registers, either immediately upon a write, or on a CTR = PRD trigger.
- In APWM mode, writing to CAP1/CAP2 active registers will also write the same value to the corresponding shadow registers CAP3/CAP4. This emulates immediate mode. Writing to the shadow registers CAP3/CAP4 will invoke the shadow mode.
- During initialization, you must write to the active registers for both period and compare. This automatically copies the initial values into the shadow values. For subsequent compare updates, during run-time, you only need to use the shadow registers.

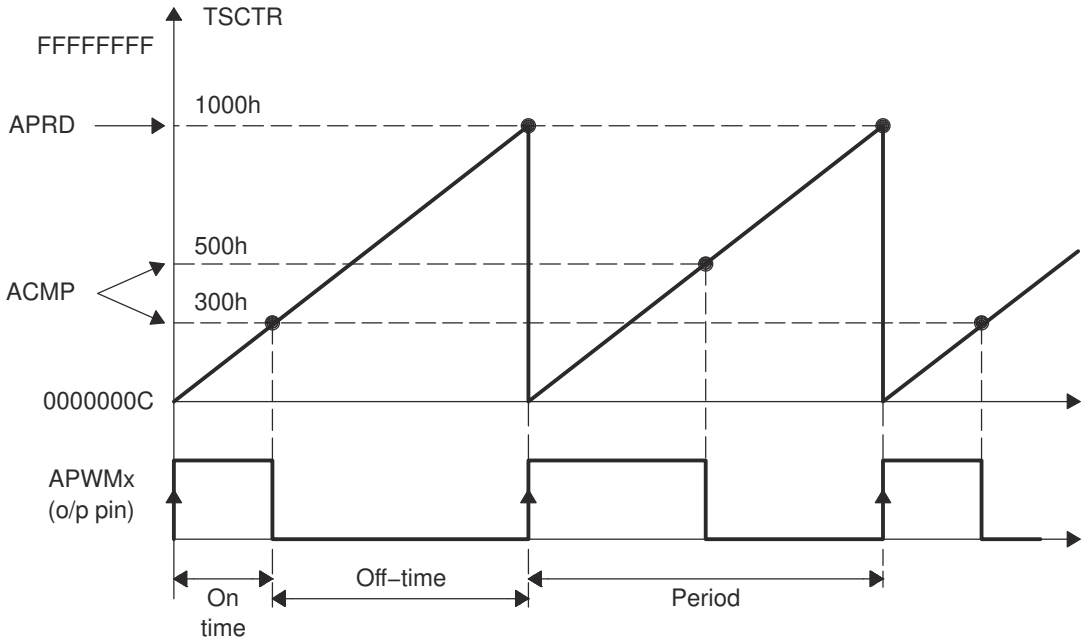


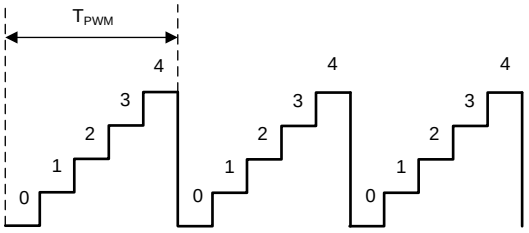
Figure 6-9. PWM Waveform Details Of APWM Mode Operation

The behavior of APWM active high mode (APWMPOL == 0) is as follows:

```
CMP = 0x00000000, output low for duration of period (0% duty)

CMP = 0x00000001, output high 1 cycle

CMP = 0x00000002, output high 2 cycles

CMP = PERIOD, output high except for 1 cycle (<100% duty)

CMP = PERIOD+1, output high for complete period (100% duty)

CMP > PERIOD+1, output high for complete period
```

The behavior of APWM active low mode (APWMPOL == 1) is as follows:

```
CMP = 0x00000000, output high for duration of period (0% duty)

CMP = 0x00000001, output low 1 cycle

CMP = 0x00000002, output low 2 cycles

CMP = PERIOD, output low except for 1 cycle (<100% duty)

CMP = PERIOD+1, output low for complete period (100% duty)

CMP > PERIOD+1, output low for complete period
```



$$T_{PWM} = (CAP1 + 1) \times T_{TSCTR}$$

$$F_{PWM} = \frac{1}{T_{PWM}}$$

Figure 6-10. Time-Base Frequency and Period Calculation

6.5 Application of the eCAP Module

The following sections will provide applications examples to show how to operate the eCAP module.

6.5.1 Example 1 - Absolute Time-Stamp Operation Rising Edge Trigger

Figure 6-11 shows an example of continuous capture operation (Mod4 counter wraps around). In this figure, TSCTR counts-up without resetting and capture events are qualified on the rising edge only, this gives period (and frequency) information.

On an event, the TSCTR contents (time-stamp) is first captured, then Mod4 counter is incremented to the next state. When the TSCTR reaches FFFFFFFF (maximum value), it wraps around to 00000000 (not shown in Figure 6-11), if this occurs, the CTROVF (counter overflow) flag is set, and an interrupt (if enabled) occurs, CTROVF (counter overflow) Flag is set, and an Interrupt (if enabled) occurs. Captured Time-stamps are valid at the point indicated by the diagram (after the 4th event), hence event CEVT4 can conveniently be used to trigger an interrupt and the CPU can read data from the CAPx registers.

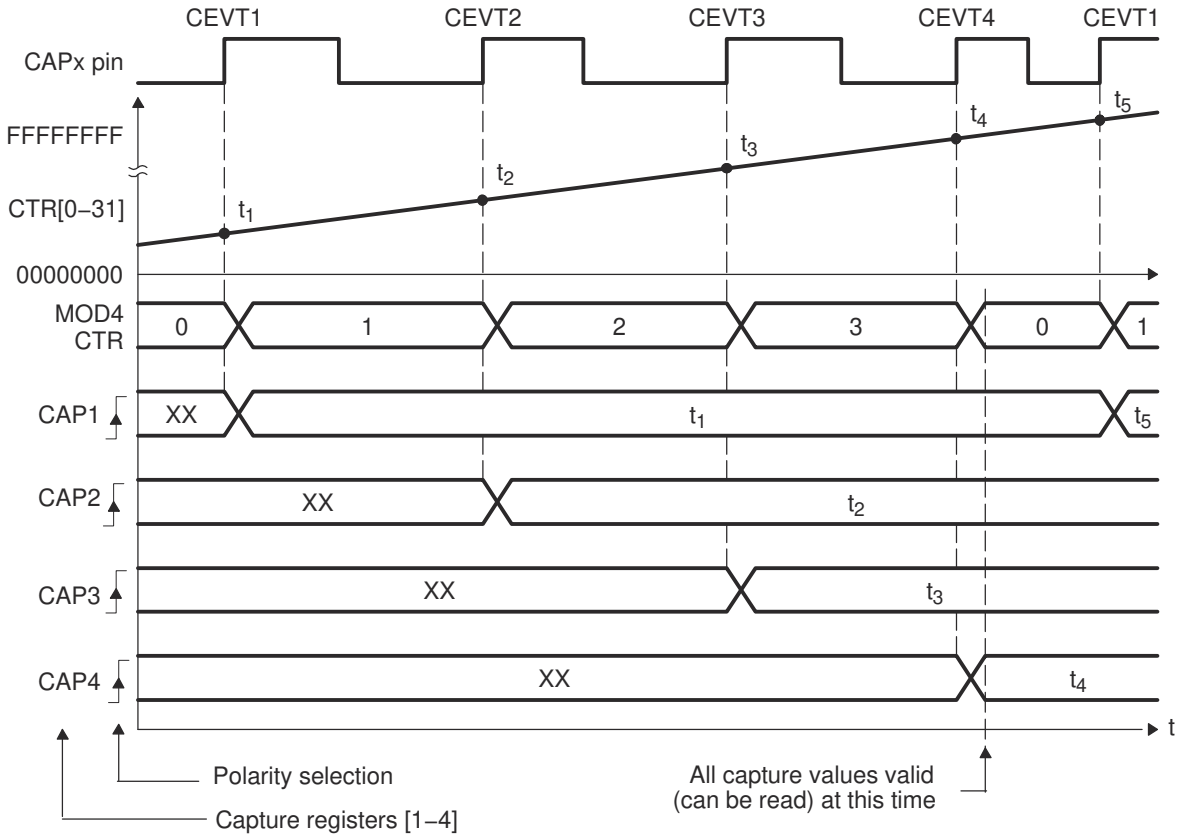


Figure 6-11. Capture Sequence for Absolute Time-stamp and Rising Edge Detect

6.5.2 Example 2 - Absolute Time-Stamp Operation Rising and Falling Edge Trigger

In Figure 6-12, the eCAP operating mode is almost the same as in the previous section except capture events are qualified as either rising or falling edge, this now gives both period and duty cycle information, that is: Period1 = $t_3 - t_1$, Period2 = $t_5 - t_3$, …and so on. Duty Cycle1 (on-time %) = $(t_2 - t_1)$ / Period1 x 100%, and so on. Duty Cycle1 (off-time %) = $(t_3 - t_2)$ / Period1 x 100%, and so on.

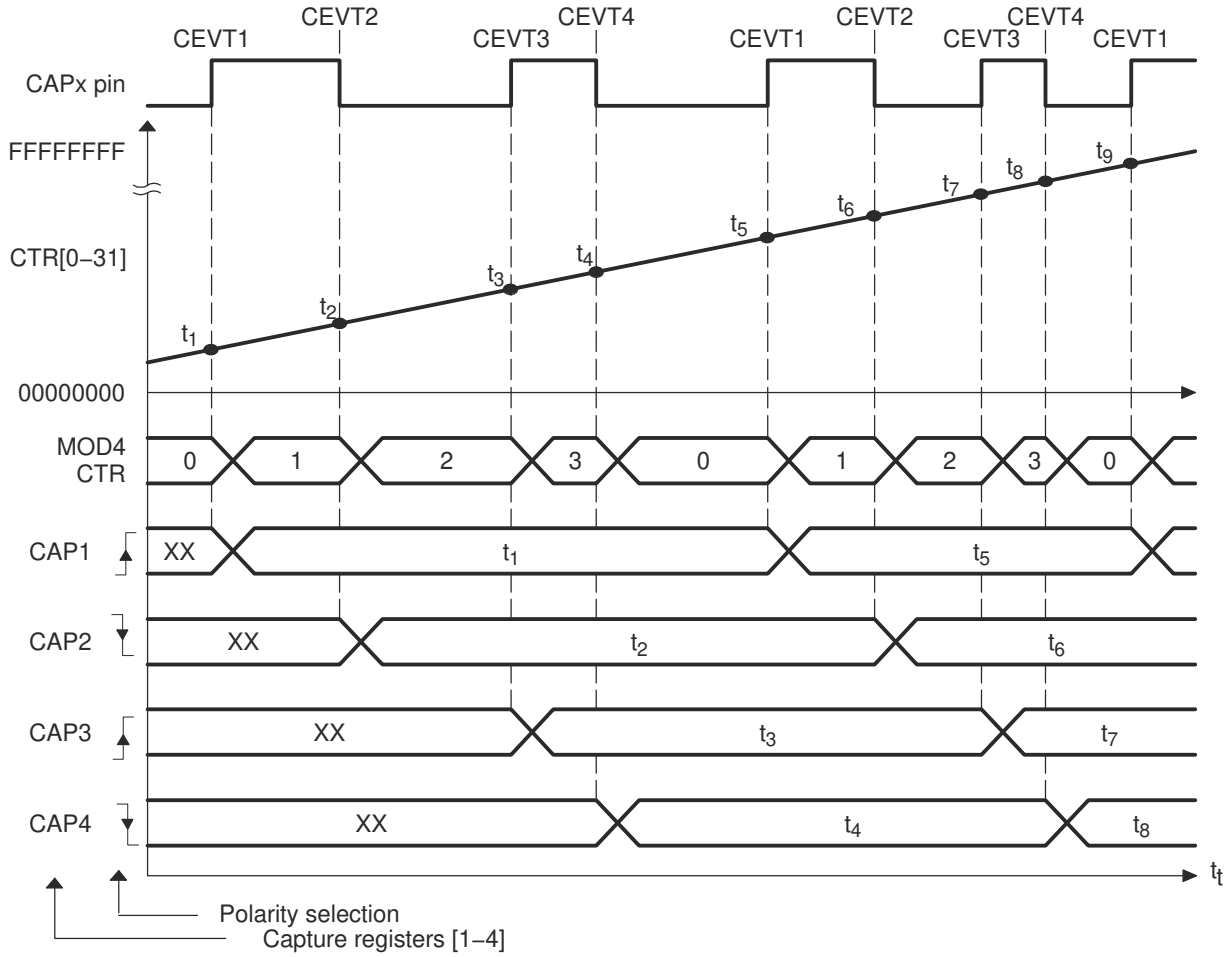


Figure 6-12. Capture Sequence for Absolute Time-stamp With Rising and Falling Edge Detect

6.5.3 Example 3 - Time Difference (Delta) Operation Rising Edge Trigger

Figure 6-13 shows how the eCAP module can be used to collect Delta timing data from pulse train waveforms. Here Continuous Capture mode (TSCTR counts-up without resetting, and Mod4 counter wraps around) is used. In Delta-time mode, TSCTR is Reset back to Zero on every valid event. Here Capture events are qualified as Rising edge only. On an event, TSCTR contents (Time-Stamp) is captured first, and then TSCTR is reset to Zero. The Mod4 counter then increments to the next state. If TSCTR reaches FFFFFFFF (Max value), before the next event, it wraps around to 00000000 and continues, a 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, and so on. As shown in Figure 6-13, the CEVT1 event is a good trigger point to read the timing data, T_1, T_2, T_3, T_4 are all valid here.

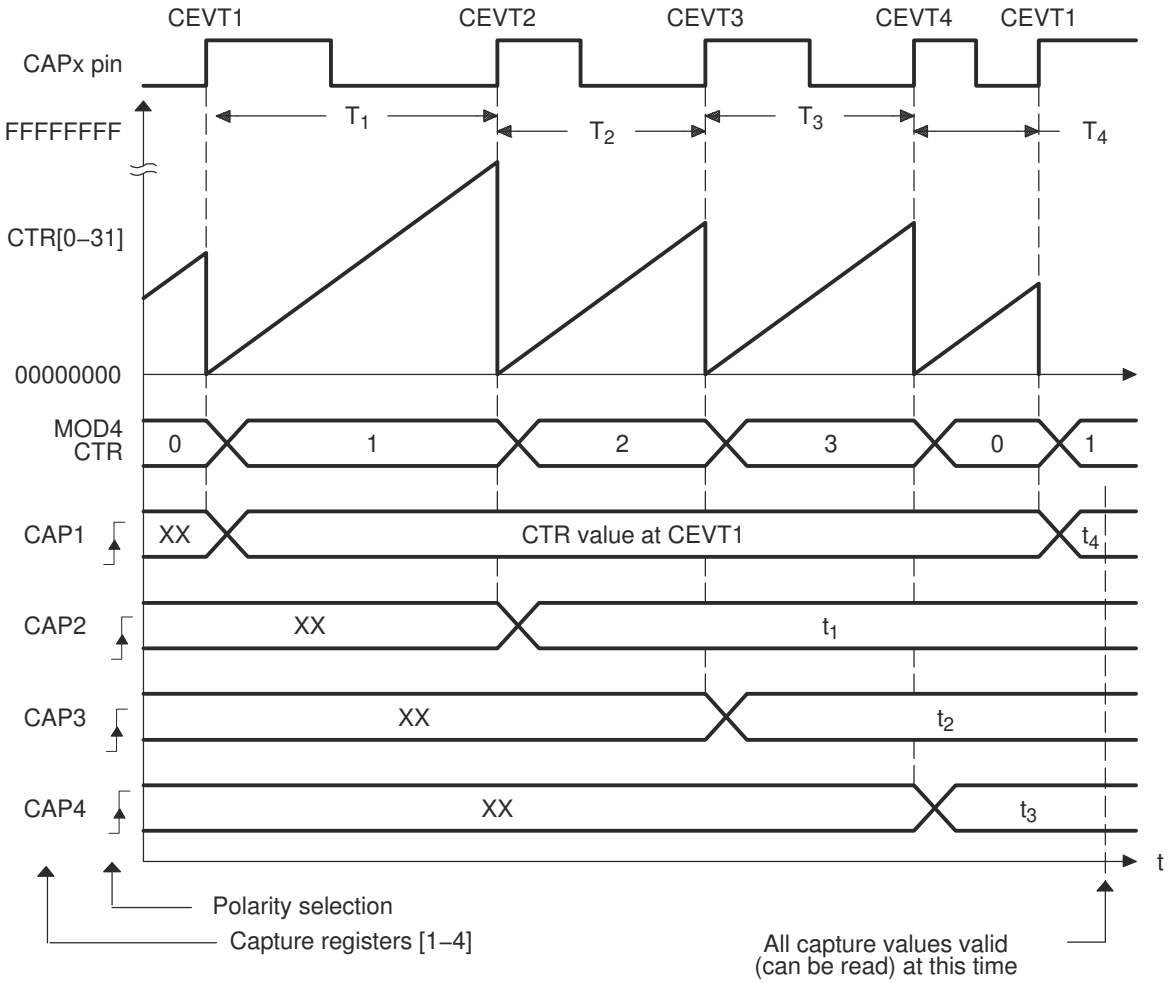


Figure 6-13. Capture Sequence for Delta Mode Time-stamp and Rising Edge Detect

6.5.4 Example 4 - Time Difference (Delta) Operation Rising and Falling Edge Trigger

In Figure 6-14, the eCAP operating mode is almost the same as in previous section except Capture events are qualified as either Rising or Falling edge, this now gives both Period and Duty cycle information, that is: 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.

During initialization, you must write to the active registers for both period and compare. This action will automatically copy the init values into the shadow values. For subsequent compare updates during run-time, the shadow registers must be used.

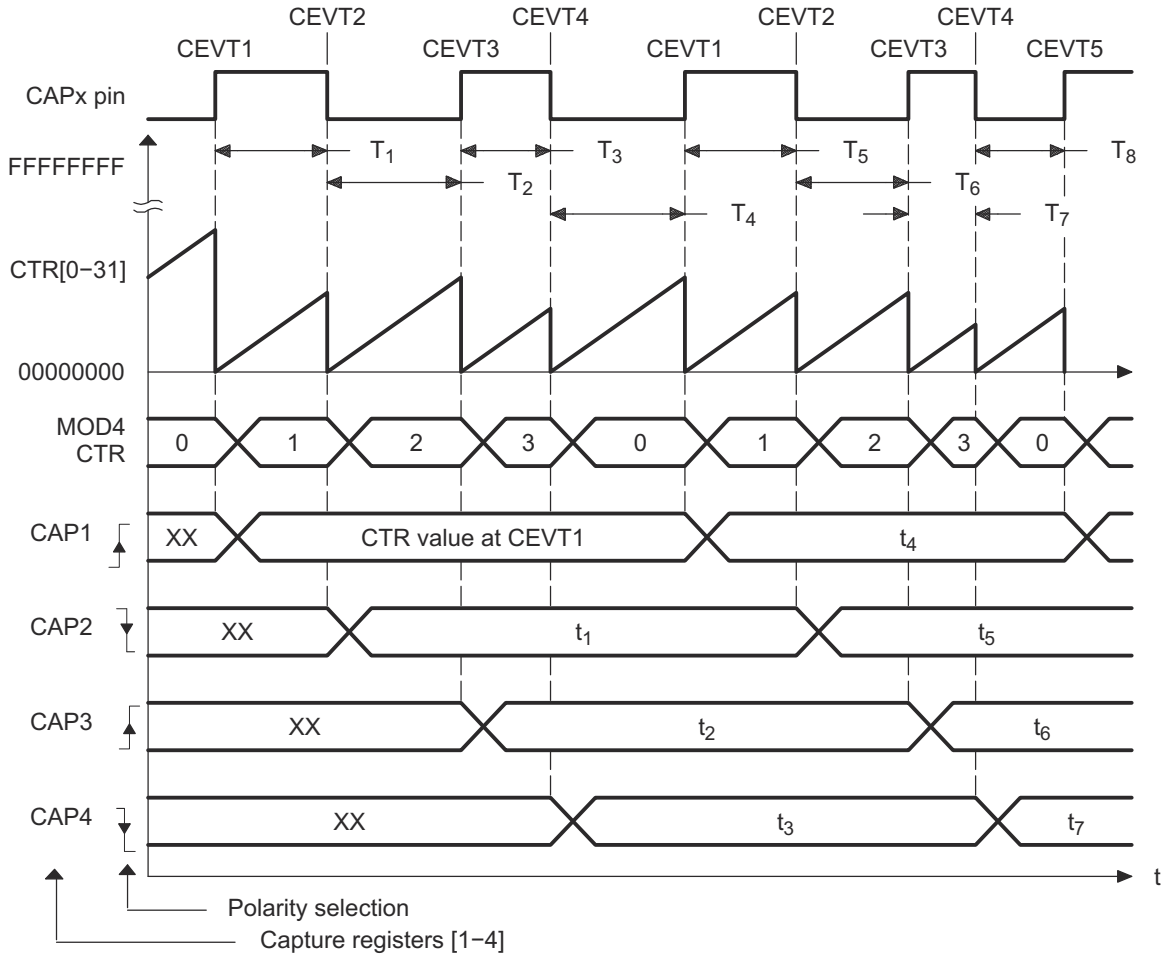


Figure 6-14. Capture Sequence for Delta Mode Time-stamp With Rising and Falling Edge Detect

6.6 Application of the APWM Mode

In this example, the eCAP module is configured to operate as a PWM generator. Here, a very simple single-channel PWM waveform is generated from the APWMx output pin. The PWM polarity is active high, which means that the compare value (CAP2 reg is now a compare register) represents the on-time (high level) of the period. Alternatively, if the APWMPOL bit is configured for active low, then the compare value represents the off-time.

6.6.1 Example 1 - Simple PWM Generation (Independent Channel/s)

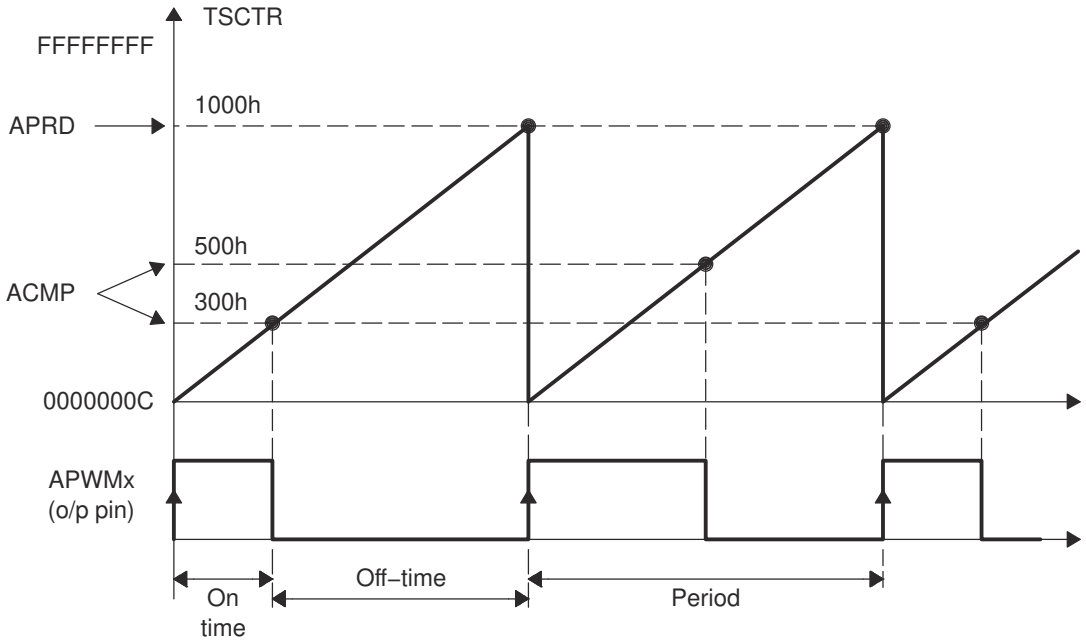


Figure 6-15. PWM Waveform Details of APWM Mode Operation

6.7 eCAP Registers

This section describes the Enhanced Capture Registers.

6.7.1 eCAP Base Addresses

Table 6-1. eCAP Base Address Table (C28)

Bit Field Name		Base Address
Instance	Structure	
ECap1Regs	ECAP_REGS	0x0000_6A00

6.7.2 ECAP_REGS Registers

Table 6-2 lists the ECAP_REGS registers. All register offset addresses not listed in Table 6-2 should be considered as reserved locations and the register contents should not be modified.

Table 6-2. ECAP_REGS Registers

Offset	Acronym	Register Name	Write Protection	Section
0h	TSCTR	Time-Stamp Counter		Section 6.7.2.1
2h	CTRPHS	Counter Phase Offset Value Register		Section 6.7.2.2
4h	CAP1	Capture 1 Register		Section 6.7.2.3
6h	CAP2	Capture 2 Register		Section 6.7.2.4
8h	CAP3	Capture 3 Register		Section 6.7.2.5
Ah	CAP4	Capture 4 Register		Section 6.7.2.6
14h	ECCTL1	Capture Control Register 1		Section 6.7.2.7
15h	ECCTL2	Capture Control Register 2		Section 6.7.2.8
16h	ECEINT	Capture Interrupt Enable Register		Section 6.7.2.9
17h	ECFLG	Capture Interrupt Flag Register		Section 6.7.2.10
18h	ECCLR	Capture Interrupt Clear Register		Section 6.7.2.11
19h	ECFRC	Capture Interrupt Force Register		Section 6.7.2.12

Complex bit access types are encoded to fit into small table cells. Table 6-3 shows the codes that are used for access types in this section.

Table 6-3. 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.

6.7.2.1 TSCTR Register (Offset = 0h) [reset = 0h]

Time-Stamp Counter

Figure 6-16. TSCTR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSCTR																															
R/W-0h																															

Table 6-4. 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

6.7.2.2 CTRPHS Register (Offset = 2h) [reset = 0h]

Counter Phase Offset Value Register

Figure 6-17. CTRPHS Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTRPHS																															
R/W-0h																															

Table 6-5. 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 time bases. Reset type: SYSRSn

6.7.2.3 CAP1 Register (Offset = 4h) [reset = 0h]

Capture 1 Register

Figure 6-18. CAP1 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP1																															
R/W-0h																															

Table 6-6. 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

6.7.2.4 CAP2 Register (Offset = 6h) [reset = 0h]

Capture 2 Register

Figure 6-19. CAP2 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP2																															
R/W-0h																															

Table 6-7. 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

6.7.2.5 CAP3 Register (Offset = 8h) [reset = 0h]

Capture 3 Register

Figure 6-20. CAP3 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP3																															
R/W-0h																															

Table 6-8. 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

6.7.2.6 CAP4 Register (Offset = Ah) [reset = 0h]

Capture 4 Register

Figure 6-21. CAP4 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAP4																															
R/W-0h																															

Table 6-9. 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

6.7.2.7 ECCTL1 Register (Offset = 14h) [reset = 0h]

Capture Control Register 1

Figure 6-22. ECCTL1 Register

15	14	13	12	11	10	9	8
FREE_SOFT		PRESCALE					CAPLDEN
R/W-0h		R/W-0h					R/W-0h
7	6	5	4	3	2	1	0
CTRRST4	CAP4POL	CTRRST3	CAP3POL	CTRRST2	CAP2POL	CTRRST1	CAP1POL
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 6-10. ECCTL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	FREE_SOFT	R/W	0h	Emulation Control Reset type: SYSRSn 0h (R/W) = TSCTR counter stops immediately on emulation suspend 1h (R/W) = TSCTR counter runs until = 0 2h (R/W) = TSCTR counter is unaffected by emulation suspend (Run Free) 3h (R/W) = TSCTR counter is unaffected by emulation suspend (Run Free)
13-9	PRESCALE	R/W	0h	Event Filter prescale select Reset type: SYSRSn 0h (R/W) = Divide by 1 (that is, no prescale, by-pass the prescaler) 1h (R/W) = Divide by 2 2h (R/W) = Divide by 4 3h (R/W) = Divide by 6 4h (R/W) = Divide by 8 5h (R/W) = Divide by 10 1Eh (R/W) = Divide by 60 1Fh (R/W) = Divide by 62
8	CAPLDEN	R/W	0h	Enable Loading of CAP1-4 registers on a capture event. Note that this bit does not disable CEVTn events from being generated. Reset type: SYSRSn 0h (R/W) = Disable CAP1-4 register loads at capture event time. 1h (R/W) = Enable CAP1-4 register loads at capture event time.
7	CTRRST4	R/W	0h	Counter Reset on Capture Event 4 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 4 (absolute time stamp operation) 1h (R/W) = Reset counter after Capture Event 4 time-stamp has been captured (used in difference mode operation)
6	CAP4POL	R/W	0h	Capture Event 4 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 4 triggered on a rising edge (RE) 1h (R/W) = Capture Event 4 triggered on a falling edge (FE)
5	CTRRST3	R/W	0h	Counter Reset on Capture Event 3 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 3 (absolute time stamp) 1h (R/W) = Reset counter after Event 3 time-stamp has been captured (used in difference mode operation)

Table 6-10. ECCTL1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	CAP3POL	R/W	0h	Capture Event 3 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 3 triggered on a rising edge (RE) 1h (R/W) = Capture Event 3 triggered on a falling edge (FE)
3	CTRRST2	R/W	0h	Counter Reset on Capture Event 2 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 2 (absolute time stamp) 1h (R/W) = Reset counter after Event 2 time-stamp has been captured (used in difference mode operation)
2	CAP2POL	R/W	0h	Capture Event 2 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 2 triggered on a rising edge (RE) 1h (R/W) = Capture Event 2 triggered on a falling edge (FE)
1	CTRRST1	R/W	0h	Counter Reset on Capture Event 1 Reset type: SYSRSn 0h (R/W) = Do not reset counter on Capture Event 1 (absolute time stamp) 1h (R/W) = Reset counter after Event 1 time-stamp has been captured (used in difference mode operation)
0	CAP1POL	R/W	0h	Capture Event 1 Polarity select Reset type: SYSRSn 0h (R/W) = Capture Event 1 triggered on a rising edge (RE) 1h (R/W) = Capture Event 1 triggered on a falling edge (FE)

6.7.2.8 ECCTL2 Register (Offset = 15h) [reset = 6h]

Capture Control Register 2

Figure 6-23. ECCTL2 Register

15	14	13	12	11	10	9	8
RESERVED					APWMPOL	CAP_APWM	SWSYNC
R-0h					R/W-0h	R/W-0h	R-0/W1S-0h
7	6	5	4	3	2	1	0
SYNCO_SEL		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 6-11. 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 6-11. 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

6.7.2.9 ECEINT Register (Offset = 16h) [reset = 0h]

The interrupt enable bits (CEVT1, ...) block any of the selected events from generating an interrupt. Events will still be latched into the flag bit (ECFLG register) and can be forced/cleared using the ECFRC/ECCLR registers. The proper procedure for configuring peripheral modes and interrupts is as follows:

1. Disable global interrupts
2. Stop eCAP counter
3. Disable eCAP interrupts
4. Configure peripheral registers
5. Clear spurious eCAP interrupt flags
6. Enable eCAP interrupts
7. Start eCAP counter
8. Enable global interrupts

Figure 6-24. ECEINT Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_EQ_CMP	CTR_EQ_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	RESERVED
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h

Table 6-12. ECEINT Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_EQ_CMP	R/W	0h	Counter Equal Compare Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Compare Equal as an Interrupt source 1h (R/W) = Enable Compare Equal as an Interrupt source
6	CTR_EQ_PRD	R/W	0h	Counter Equal Period Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Period Equal as an Interrupt source 1h (R/W) = Enable Period Equal as an Interrupt source
5	CTROVF	R/W	0h	Counter Overflow Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disabled counter Overflow as an Interrupt source 1h (R/W) = Enable counter Overflow as an Interrupt source
4	CEVT4	R/W	0h	Capture Event 4 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 4 as an Interrupt source 1h (R/W) = Capture Event 4 Interrupt Enable
3	CEVT3	R/W	0h	Capture Event 3 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 3 as an Interrupt source 1h (R/W) = Enable Capture Event 3 as an Interrupt source
2	CEVT2	R/W	0h	Capture Event 2 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 2 as an Interrupt source 1h (R/W) = Enable Capture Event 2 as an Interrupt source
1	CEVT1	R/W	0h	Capture Event 1 Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disable Capture Event 1 as an Interrupt source 1h (R/W) = Enable Capture Event 1 as an Interrupt source
0	RESERVED	R	0h	Reserved

6.7.2.10 ECFLG Register (Offset = 17h) [reset = 0h]

Capture Interrupt Flag Register

Figure 6-25. ECFLG Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	INT
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

Table 6-13. ECFLG Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	CTR_CMP	R	0h	Compare Equal Compare Status Flag. This flag is active only in APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) reached the compare register value (ACMP)
6	CTR_PRD	R	0h	Counter Equal Period Status Flag. This flag is only active in APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) reached the period register value (APRD) and was reset.
5	CTROVF	R	0h	Counter Overflow Status Flag. This flag is active in CAP and APWM mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the counter (TSCTR) has made the transition from FFFFFFFF " 00000000
4	CEVT4	R	0h	Capture Event 4 Status Flag This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the fourth event occurred at ECAPx pin
3	CEVT3	R	0h	Capture Event 3 Status Flag. This flag is active only in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the third event occurred at ECAPx pin.
2	CEVT2	R	0h	Capture Event 2 Status Flag. This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the second event occurred at ECAPx pin.
1	CEVT1	R	0h	Capture Event 1 Status Flag. This flag is only active in CAP mode. Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates the first event occurred at ECAPx pin.
0	INT	R	0h	Global Interrupt Status Flag Reset type: SYSRSn 0h (R/W) = Indicates no event occurred 1h (R/W) = Indicates that an interrupt was generated.

6.7.2.11 ECCLR Register (Offset = 18h) [reset = 0h]

Capture Interrupt Clear Register

Figure 6-26. ECCLR Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	INT
R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h	R-0/W1C-0h

Table 6-14. 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

6.7.2.12 ECFRC Register (Offset = 19h) [reset = 0h]

Capture Interrupt Force Register

Figure 6-27. ECFRC Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CTR_CMP	CTR_PRD	CTROVF	CEVT4	CEVT3	CEVT2	CEVT1	RESERVED
R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0/W1S-0h	R-0h

Table 6-15. 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

This page intentionally left blank.

Chapter 7

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.

7.1 Introduction

An incremental encoder disk is patterned with a track of slots along its periphery, as shown in Figure 7-1. These slots create an alternating pattern of dark and light lines. The disk count is defined as the number of dark 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

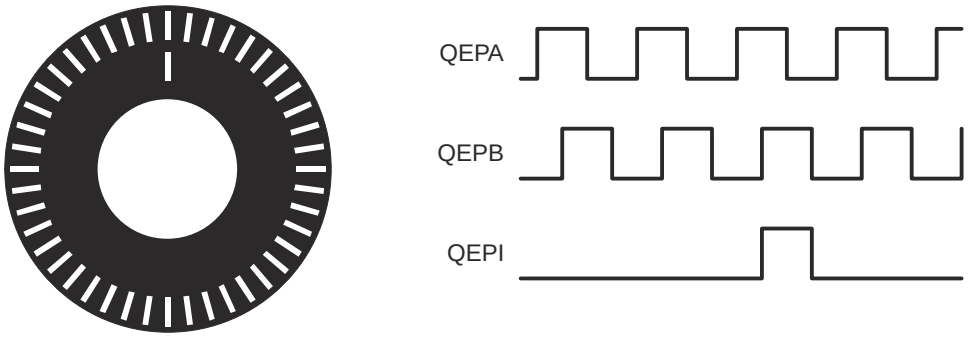
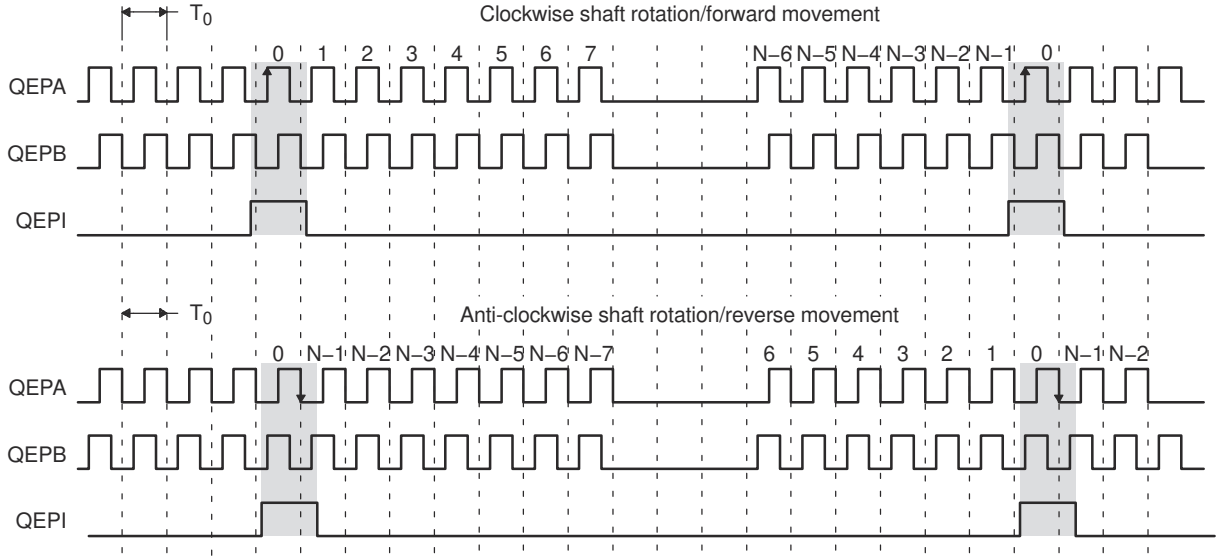


Figure 7-1. Optical Encoder Disk

To derive direction information, the lines on the disk are read out by two different photo-elements that "look" at the disk pattern with a mechanical shift of 1/4 the pitch of a line pair between them. This shift is 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 conversely, as shown in Figure 7-2.



Legend: N = lines per revolution

Figure 7-2. QEP Encoder Output Signal for Forward/Reverse Movement

The encoder wheel typically makes one revolution for every revolution of the motor, or the wheel may be at a geared rotation ratio with respect to the motor. Therefore, the frequency of the digital signal coming from the QEPA and QEPB outputs varies proportionally with the velocity of the motor. For example, a 2000-line encoder directly coupled to a motor running at 5000 revolutions per minute (rpm) results in a frequency of 166.6 kHz, so

by measuring the frequency of either the QEPA or QEPB output, the processor can determine the velocity of the motor.

Quadrature encoders from different manufacturers come with two forms of index pulse (gated index pulse or ungated index pulse) as shown in Figure 7-3. A nonstandard form of index pulse is ungated. In the ungated configuration, the index edges are not necessarily coincident with A and B signals. The gated index pulse is aligned to any of the four quadrature edges and width of the index pulse and can be equal to a quarter, half, or full period of the quadrature signal.

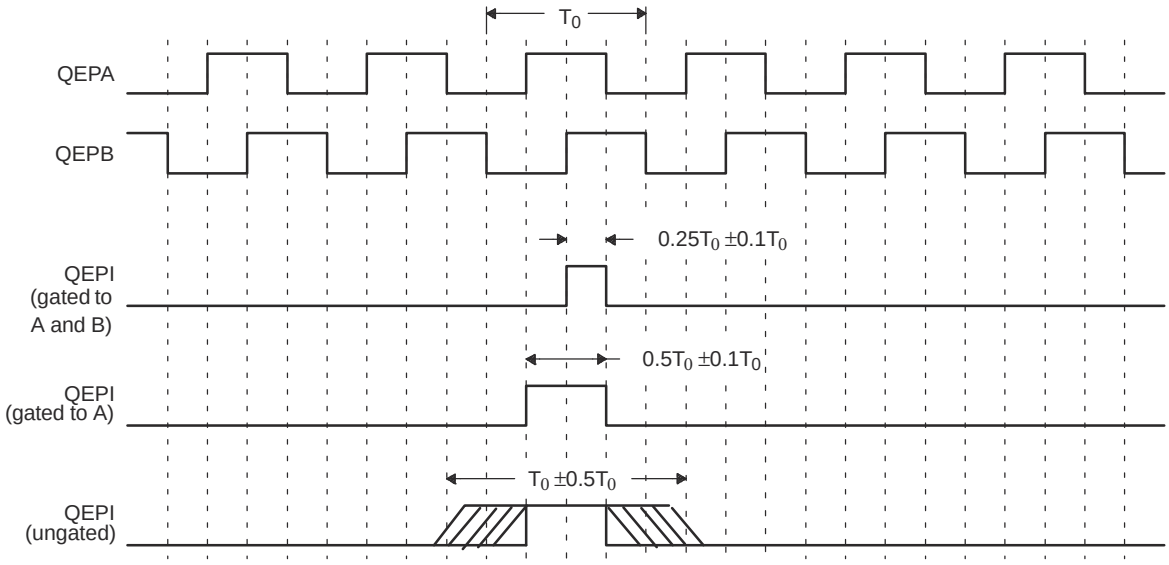


Figure 7-3. Index Pulse Example

Some typical applications of shaft encoders include robotics and 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.

7.1.1 EQEP Related Collateral

Foundational Materials

- *Real-Time Control Reference Guide*
  - Refer to the Encoders section

Getting Started Materials

- *C2000 Position Manager PTO API Reference Guide Application Report*

Expert Materials

- *CW/CCW Support on the C2000 eQEP Module Application Report*

7.2 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.

7.3 Description

This section provides the eQEP inputs, memory map, and functional description.

7.3.1 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 conversely. 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.

7.3.2 Functional Description

The eQEP peripheral contains the following major functional units (as shown in Figure 7-4):

- Programmable input qualification for each pin (part of the GPIO MUX)
- Quadrature decoder unit (QDU)
- Position counter and control unit for position measurement (PCCU)
- Quadrature edge-capture unit for low-speed measurement (QCAP)
- Unit time base for speed/frequency measurement (UTIME)
- Watchdog timer for detecting stalls (QWDOG)

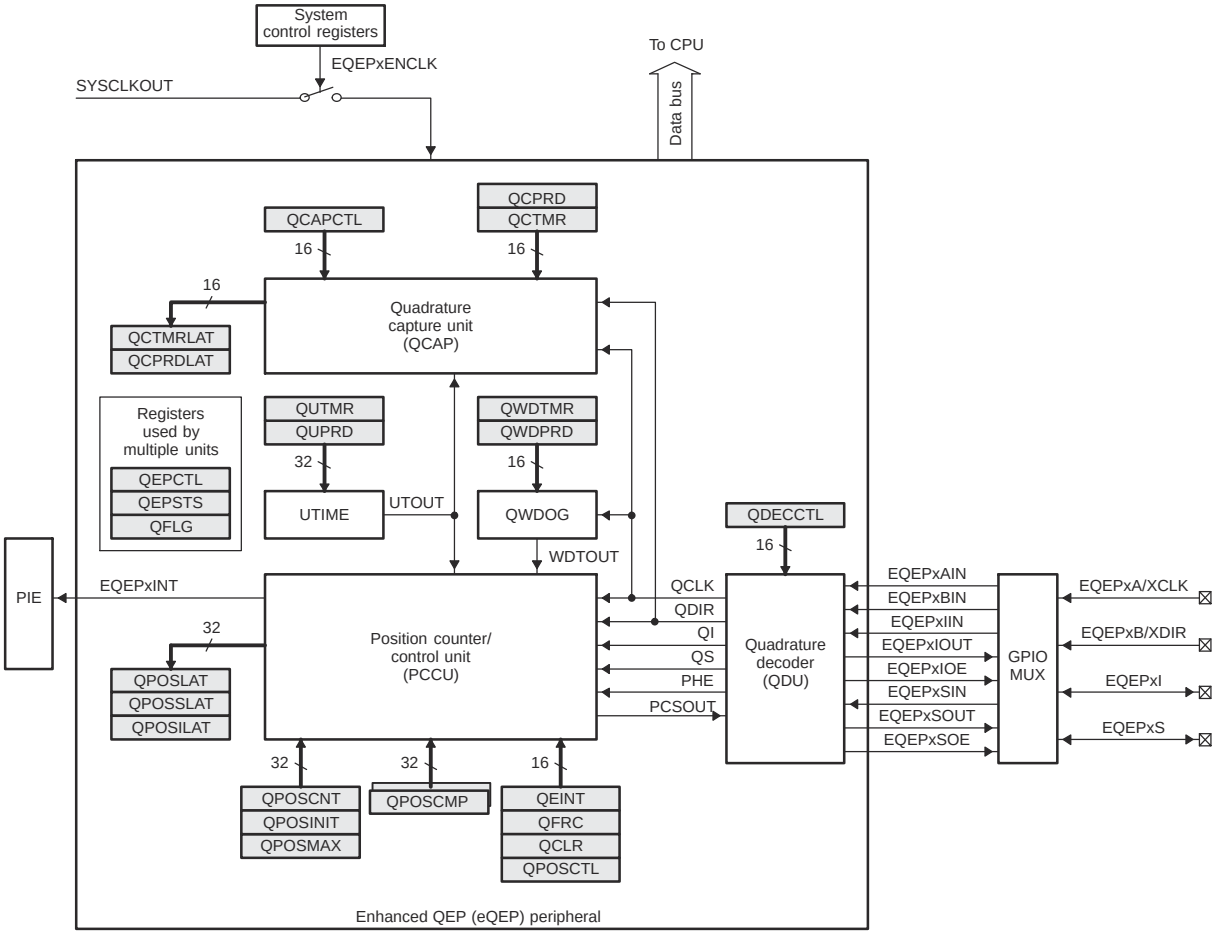


Figure 7-4. Functional Block Diagram of the eQEP Peripheral

7.3.3 eQEP Memory Map

Table 7-1 lists the registers with their memory locations, sizes, and reset values.

Table 7-1. EQEP Memory Map

Name	Offset	Size(x16)/ #shadow	Reset	Register Description
QPOSCNT	0x00	2/0	0x00000000	eQEP Position Counter
QPOSINIT	0x02	2/0	0x00000000	eQEP Initialization Position Count
QPOSMAX	0x04	2/0	0x00000000	eQEP Maximum Position Count
QPOSCMP	0x06	2/1	0x00000000	eQEP Position-compare
QPOSILAT	0x08	2/0	0x00000000	eQEP Index Position Latch
QPOSSLAT	0x0A	2/0	0x00000000	eQEP Strobe Position Latch
QPOSLAT	0x0C	2/0	0x00000000	eQEP Position Latch
QUTMR	0x0E	2/0	0x00000000	QEP Unit Timer
QUPRD	0x10	2/0	0x00000000	eQEP Unit Period Register
QWDTMR	0x12	1/0	0x0000	eQEP Watchdog Timer
QWDPRD	0x13	1/0	0x0000	eQEP Watchdog Period Register
QDECCTL	0x14	1/0	0x0000	eQEP Decoder Control Register
QEPCTL	0x15	1/0	0x0000	eQEP Control Register
QCAPCTL	0x16	1/0	0x0000	eQEP Capture Control Register
QPOSCTL	0x17	1/0	0x00000	eQEP Position-compare Control Register
QEINT	0x18	1/0	0x0000	eQEP Interrupt Enable Register
QFLG	0x19	1/0	0x0000	eQEP Interrupt Flag Register
QCLR	0x1A	1/0	0x0000	eQEP Interrupt Clear Register
QFRC	0x1B	1/0	0x0000	eQEP Interrupt Force Register
QEPSTS	0x1C	1/0	0x0000	eQEP Status Register
QCTMR	0x1D	1/0	0x0000	eQEP Capture Timer
QCPRD	0x1E	1/0	0x0000	eQEP Capture Period Register
QCTMRLAT	0x1F	1/0	0x0000	eQEP Capture Timer Latch
QCPRDLAT	0x20	1/0	0x0000	eQEP Capture Period Latch
reserved	0x21 to 0x3F	31/0		

7.4 Quadrature Decoder Unit (QDU)

Figure 7-5 shows a functional block diagram of the QDU.

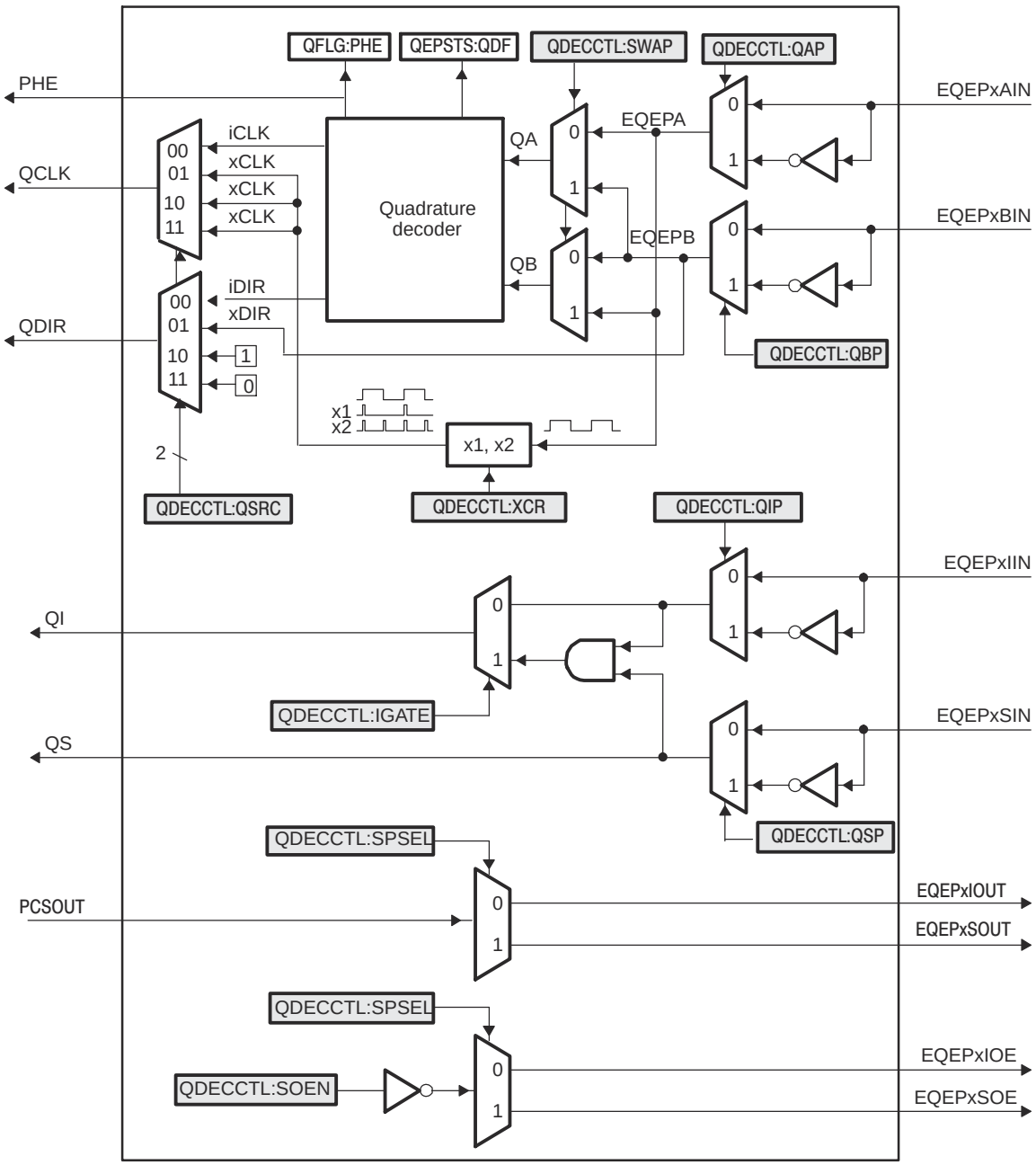


Figure 7-5. Functional Block Diagram of Decoder Unit

7.4.1 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

7.4.1.1 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 7-2 and Figure 7-6 show the direction decoding logic in truth table and state machine form. Both edges of the QEPA and QEPB signals are sensed to generate count pulses for the position counter. Therefore, the frequency of the clock generated by the eQEP logic is four times that of each input sequence. Figure 7-7 shows the direction decoding and clock generation from the eQEP input signals.

Table 7-2. Quadrature Decoder Truth Table

Previous Edge	Present Edge	QDIR	QPOSCNT
QA↑	QB↑	UP	Increment
	QB↓	DOWN	Decrement
	QA↓	TOGGLE	Increment or Decrement
QA↓	QB↓	UP	Increment
	QB↑	DOWN	Decrement
	QA↑	TOGGLE	Increment or Decrement
QB↑	QA↑	DOWN	Decrement
	QA↓	UP	Increment
	QB↓	TOGGLE	Increment or Decrement
QB↓	QA↓	DOWN	Decrement
	QA↑	UP	Increment
	QB↑	TOGGLE	Increment or Decrement

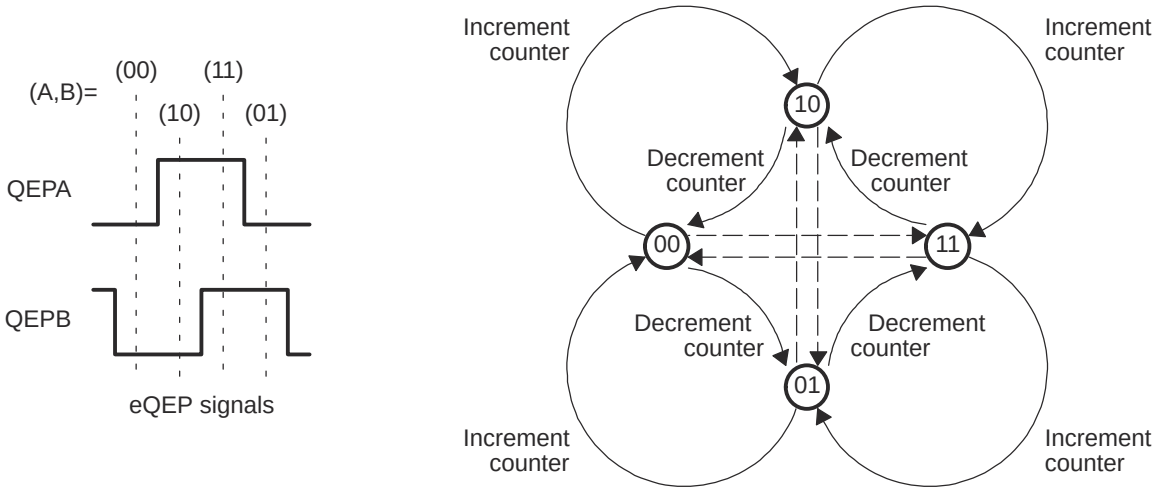


Figure 7-6. Quadrature Decoder State Machine

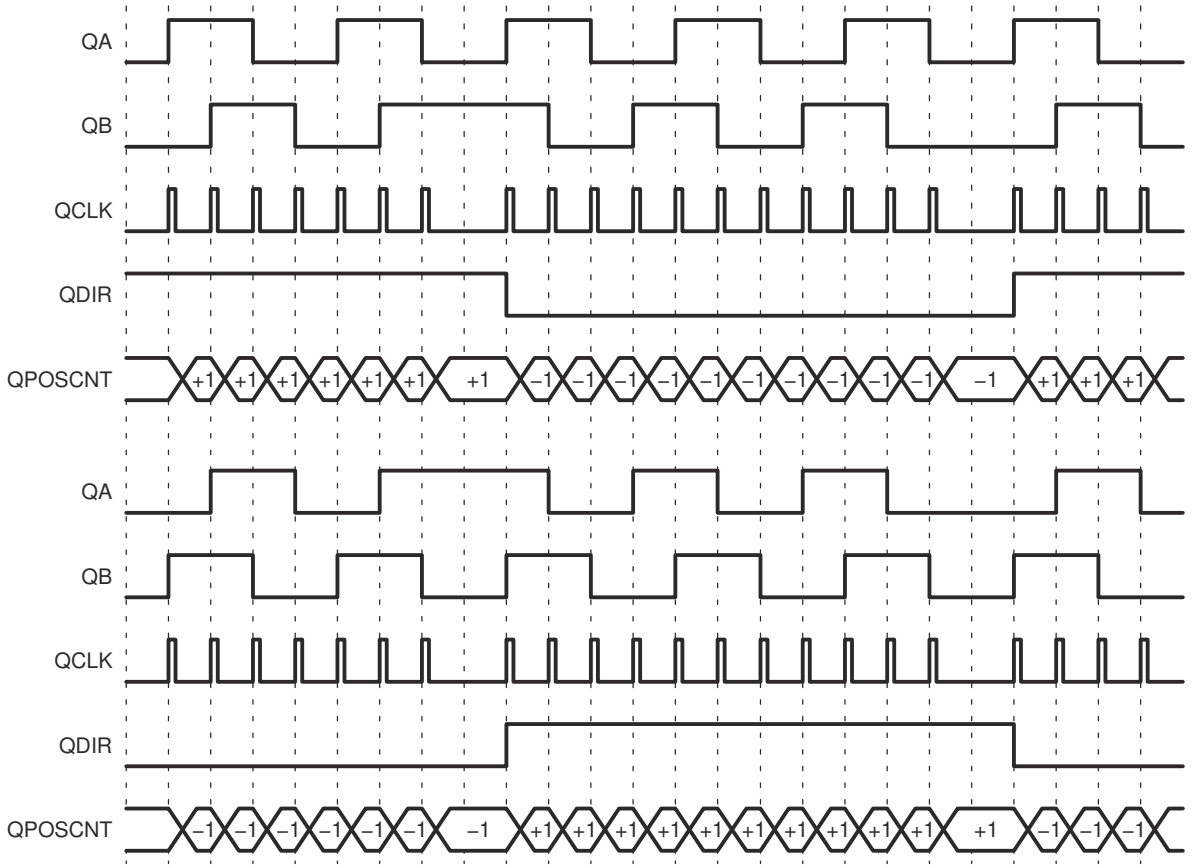


Figure 7-7. Quadrature-clock and Direction Decoding

Phase Error Flag In normal operating conditions, quadrature inputs QEPA and QEPB will be 90 degrees out of phase. The phase error flag (PHE) is set in the QFLG register 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 7-6 are invalid transitions that generate a phase error.

Count Multiplication The eQEP position counter provides 4x times the resolution of an input clock by generating a quadrature-clock (QCLK) on the rising/falling edges of both eQEP input clocks (QEPA and QEPB) as shown in Figure 7-7.

Reverse Count In normal quadrature count operation, QEPA input is fed to the QA input of the quadrature decoder and the QEPB input is fed to the QB input of the quadrature decoder. Reverse counting is enabled by setting the SWAP bit in the QDECCTL register. This will swap the input to the quadrature decoder, thereby reversing the counting direction.

7.4.1.2 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.

7.4.1.3 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.

7.4.1.4 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.

7.4.2 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.

7.4.3 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.

7.5 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.

7.5.1 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.

7.5.1.1 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 7-8.

The position-counter value is latched to the QPOSILAT register and direction information is recorded in the QEPSTS[QDLF] bit on every index event marker. The position-counter error flag (QEPSTS[PCEF]) and error interrupt flag (QFLG[PCE]) are set if the latched value is not equal to 0 or QPOSMAX. The position-counter error flag (QEPSTS[PCEF]) is updated on every index event marker and an interrupt flag (QFLG[PCE]) will be set on error that can be cleared only through software.

The index event latch configuration QEPCTL[IEL] 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.

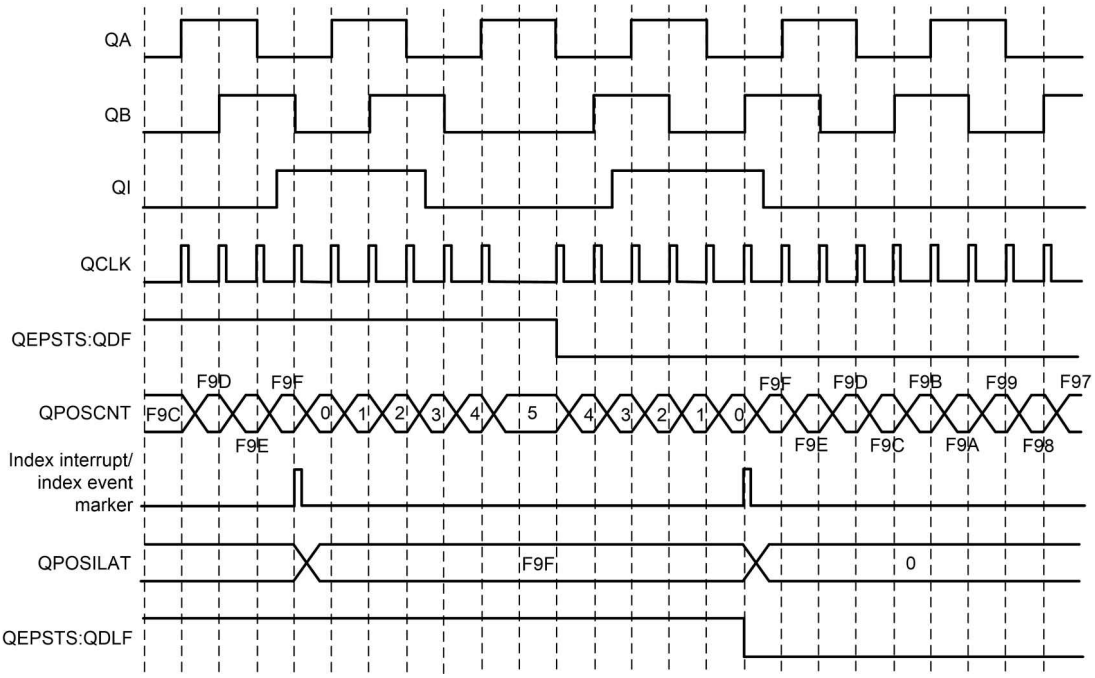


Figure 7-8. Position Counter Reset by Index Pulse for 1000 Line Encoder (QPOSMAX = 3999 or 0xF9F)

Note

In case of a boundary condition where the time period between the Index Event and the 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 the next QCLK edge to occur.

7.5.1.2 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 7-9 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.

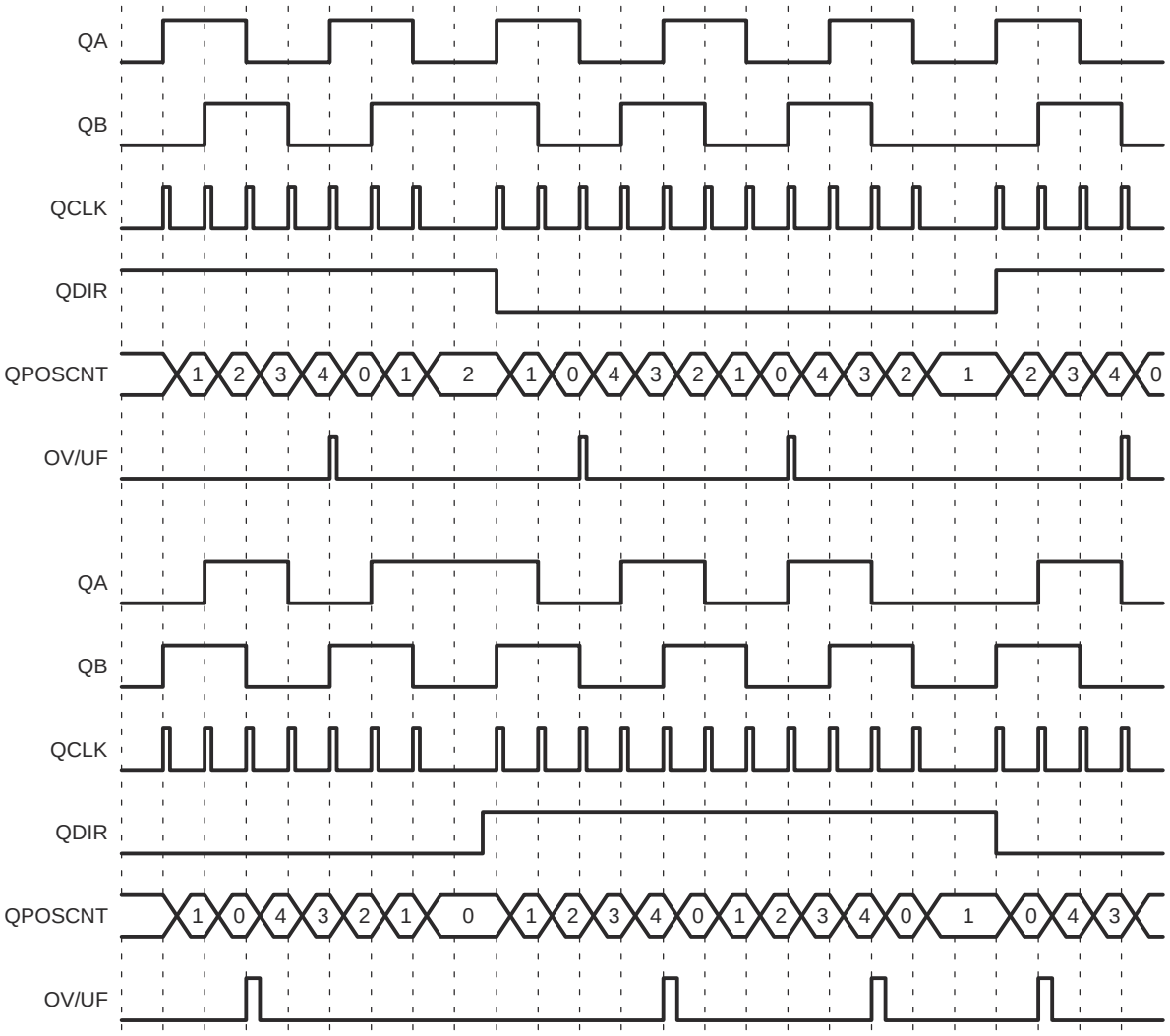


Figure 7-9. Position Counter Underflow/Overflow (QPOSMAX = 4)

7.5.1.3 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 7.5.1.2.

The first index marker fields (QEPSTS[FIDF] and QEPSTS[FIMF]) are not applicable in this mode.

7.5.1.4 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.

7.5.2 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.

7.5.2.1 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 7-10 shows the position counter latch using an index event marker.

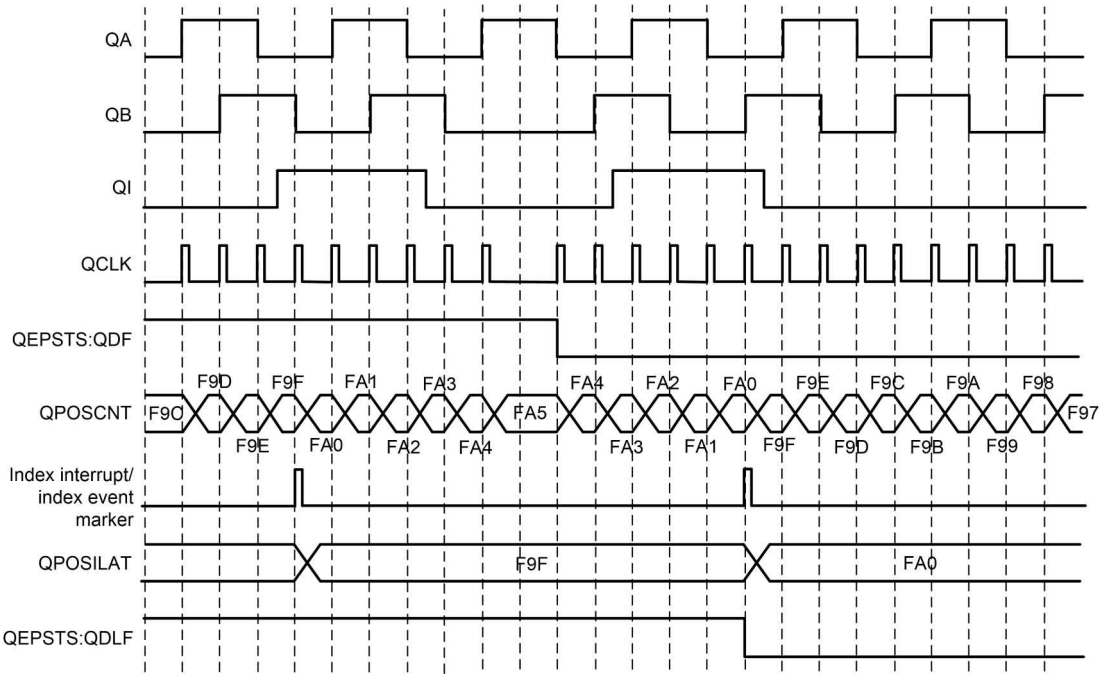


Figure 7-10. Software Index Marker for 1000-line Encoder (QEPCTL[IEL] = 1)

7.5.2.2 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 7-11.

The strobe event latch interrupt flag (QFLG[SEL) is set when the position counter is latched to the QPOSSLAT register.

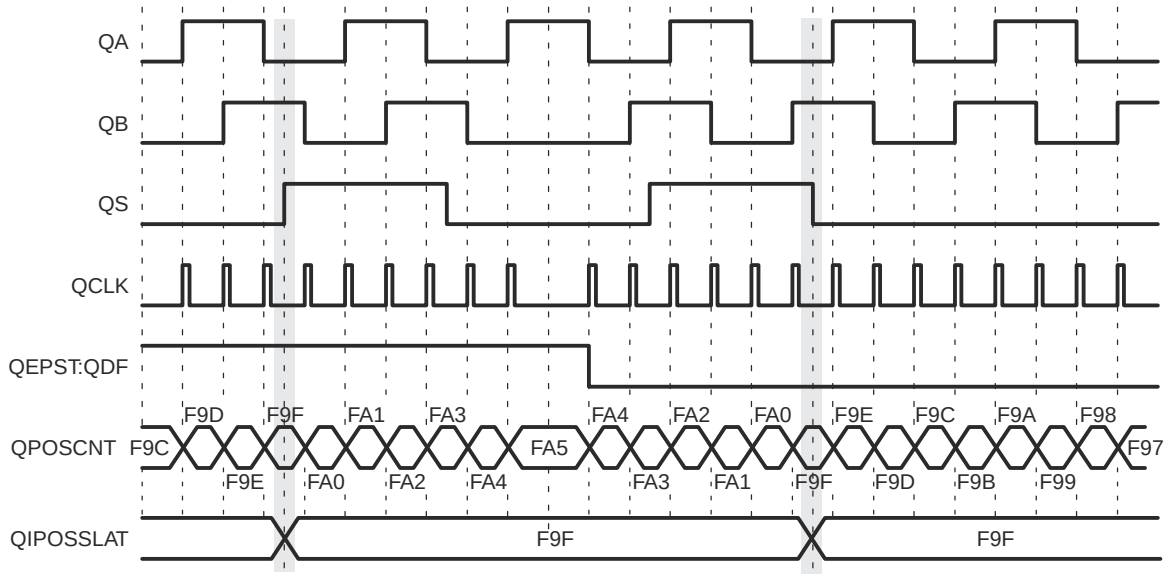


Figure 7-11. Strobe Event Latch (QEPCTL[SEL] = 1)

7.5.3 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.

7.5.4 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 7-12 shows a diagram. The position-compare (QPOSCMP) register is shadowed and shadow mode can be enabled or disabled using the QPOSCTL[PSSHDW] bit. If the shadow mode is not enabled, the CPU writes directly to the active position compare register.

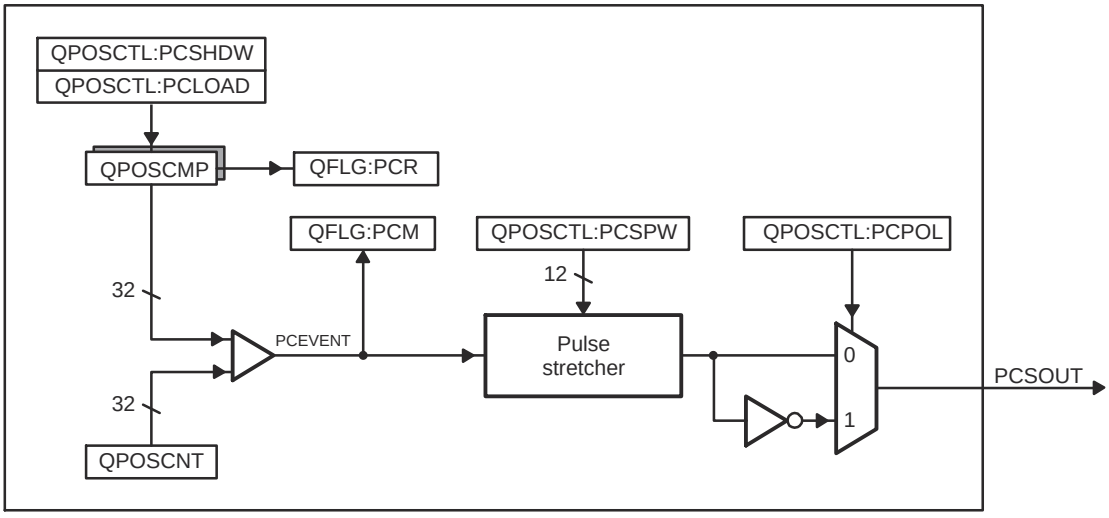


Figure 7-12. eQEP Position-compare Unit

In shadow mode, you can configure the position-compare unit (QPOSCTL[PCLOAD]) to load the shadow register value into the active register on the following events, and to generate the position-compare ready (QFLG[PCR]) interrupt after loading.

- Load on compare match
- Load on position-counter zero event

The position-compare match (QFLG[PCM]) is set when the position-counter value (QPOSCNT) matches with the active position-compare register (QPOSCMP) and the position-compare sync output of the programmable pulse width is generated on compare-match to trigger an external device.

For example, if QPOSCMP = 2, the position-compare unit generates a position-compare event on 1 to 2 transitions of the eQEP position counter for forward counting direction and on 3 to 2 transitions of the eQEP position counter for reverse counting direction (see Figure 7-13).

See the register section for the layout of the eQEP Position-Compare Control Register (QPOSCTL) and description of the QPOSCTL bit fields.

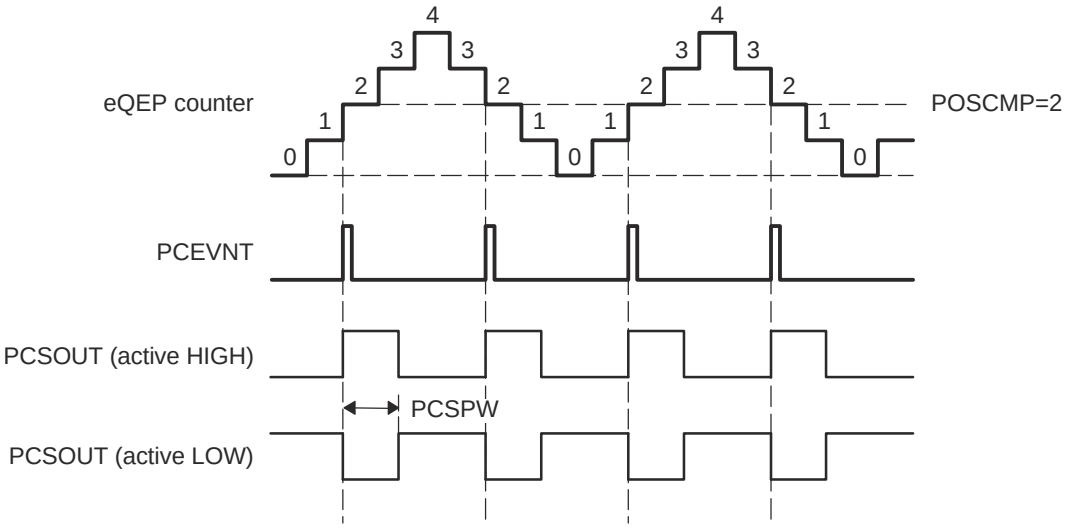


Figure 7-13. eQEP Position-compare Event Generation Points

The pulse stretcher logic in the position-compare unit generates a programmable position-compare sync pulse output on the position-compare match. In the event of a new position-compare match while a previous position-compare pulse is still active, then the pulse stretcher generates a pulse of specified duration from the new position-compare event as shown in Figure 7-14.

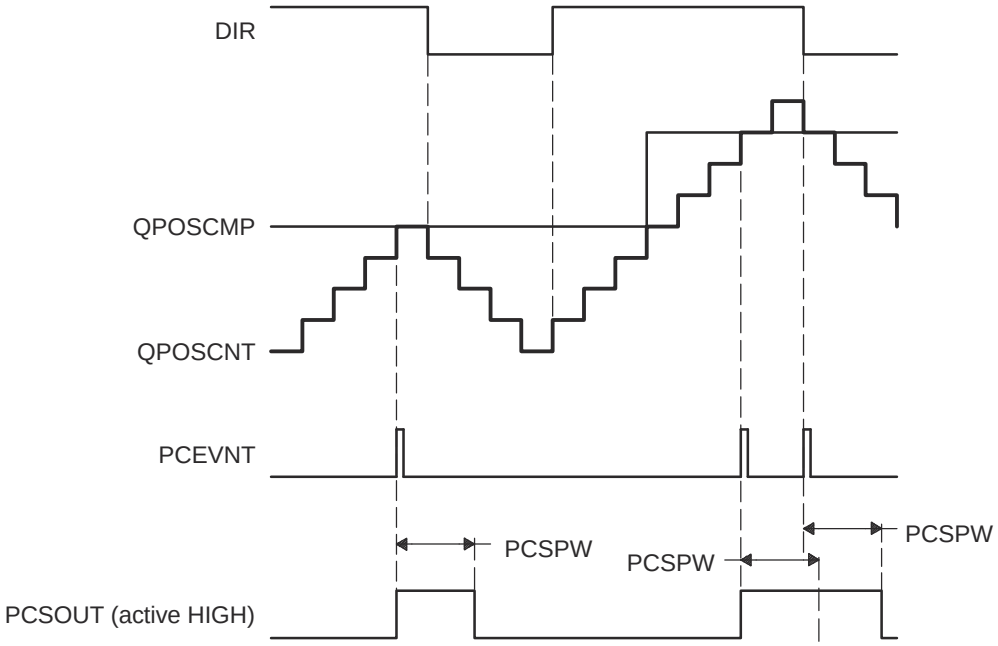


Figure 7-14. eQEP Position-compare Sync Output Pulse Stretcher

7.6 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 7-15. This feature is typically used for low-speed measurement using the following formula:

$$v(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \quad (3)$$

where:

- X = Unit position is defined by integer multiple of quadrature edges (see Figure 7-16)
- ΔT = Elapsed time between unit position events
- v(k) = Velocity at time instant "k"

The eQEP capture timer (QCTMR) runs from prescaled SYSCLKOUT and the prescaler is programmed by the QCAPCTL[CCPS] bits. The capture timer (QCTMR) value is latched into the capture period register (QCPRD) on every unit position event and then the capture timer is reset, a flag is set in QEPSTS:UPEVNT to indicate that new value is latched into the QCPRD register. Software can check this status flag before reading the period register for low speed measurement, and clear the flag by writing 1.

Time measurement (ΔT) between unit position events will be correct if the following conditions are met:

- No more than 65,535 counts have occurred between unit position events.
- No direction change between unit position events.

If the QEP capture timer overflows between unit position events, then it sets the QEP capture overflow flag (QEPSTS[COEF]) in the status register and the QCPRDLAT register is set to 0xFFFF. If direction change occurs between the unit position events, then the error flag is set in the status register (QEPSTS[CDEF]) and the QCPRDLAT register is set to 0xFFFF.

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 7-17 shows the capture unit operation along with the position counter.

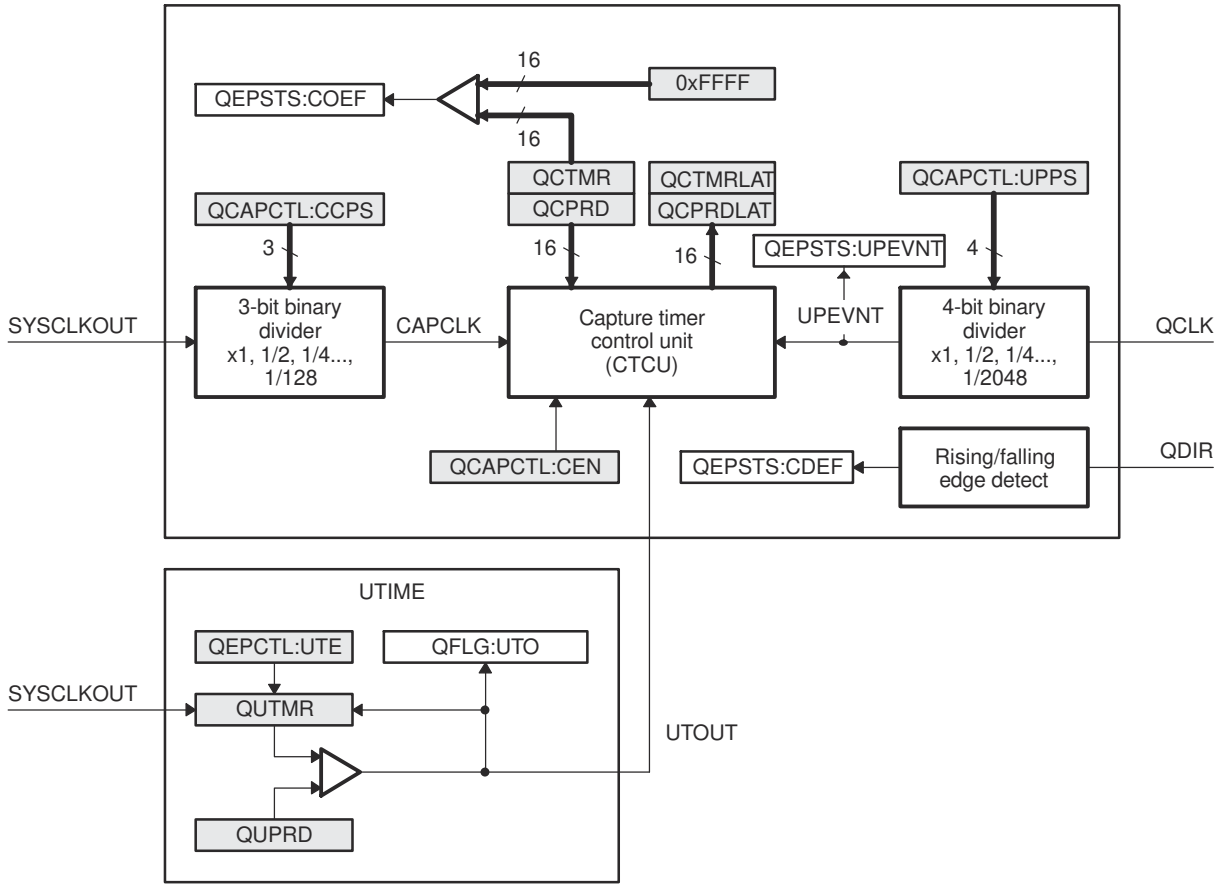
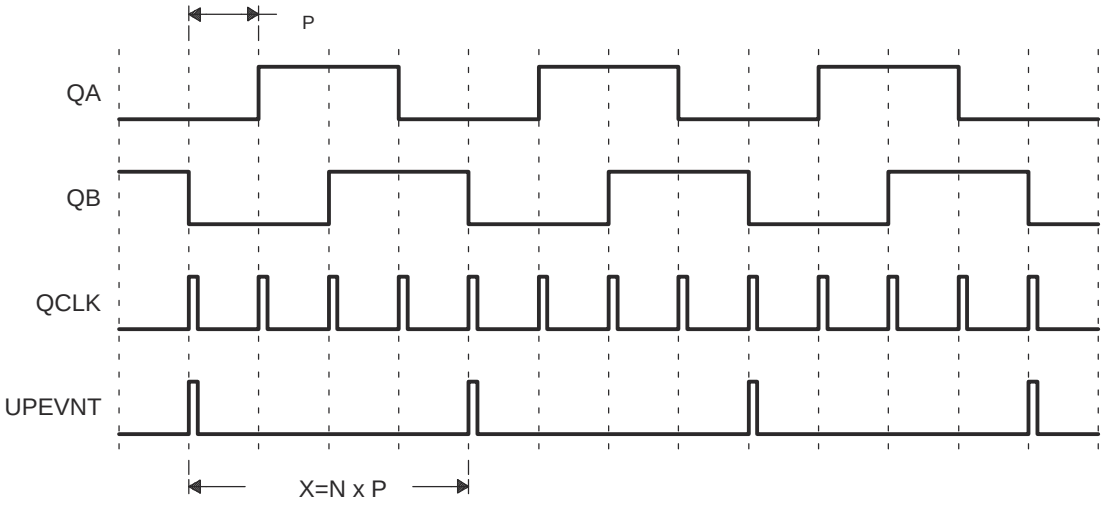


Figure 7-15. eQEP Edge Capture Unit

Note

The QCAPCTL[UPPS] prescaler should not be modified dynamically (such as switching the unit event prescaler from QCLK/4 to QCLK/8). Doing so may result in undefined behavior. The QCAPCTL[CPPS] prescaler can be modified dynamically (such as switching CAPCLK prescaling mode from SYSCLK/4 to SYSCLK/8) only after the capture unit is disabled.



N = Number of quadrature periods selected using QCAPCTL[UPPS] bits

Figure 7-16. Unit Position Event for Low Speed Measurement (QCAPCTL[UPPS] = 0010)

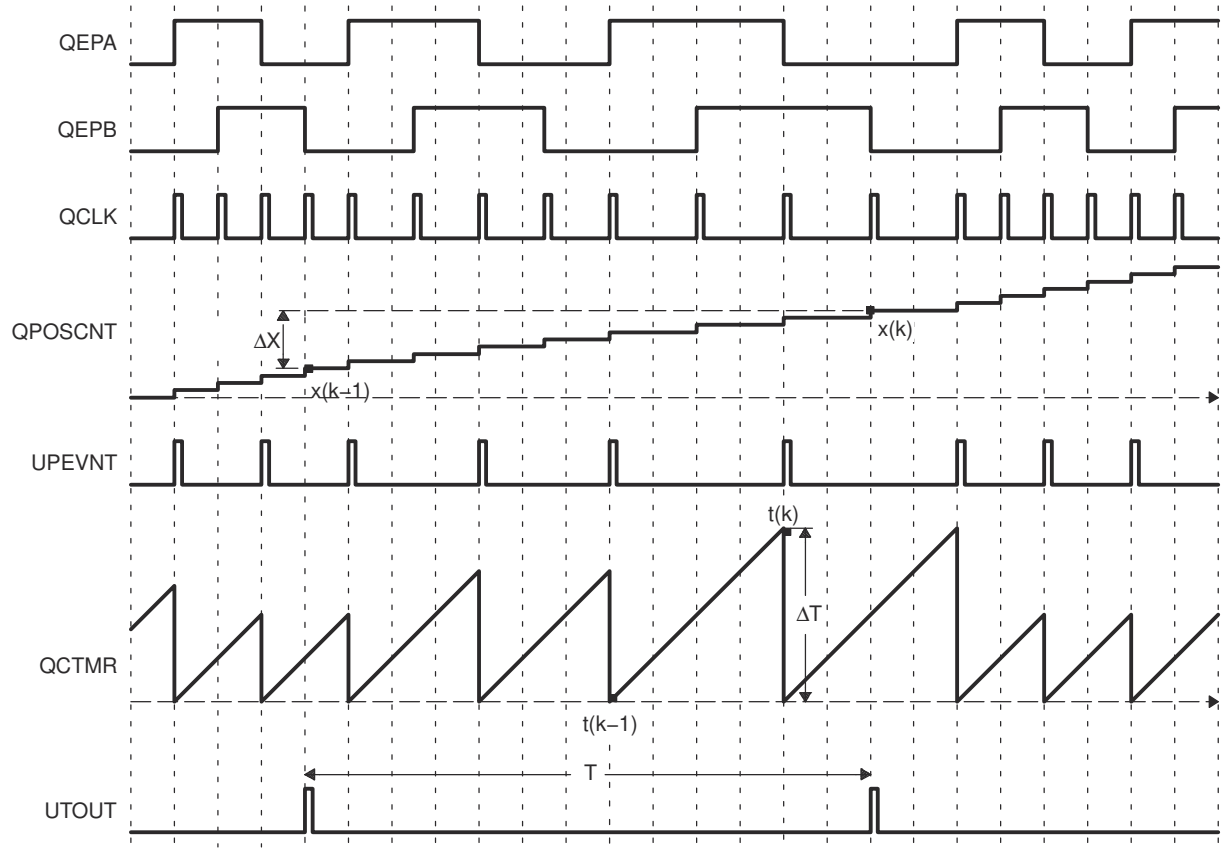


Figure 7-17. eQEP Edge Capture Unit - Timing Details

Velocity calculation equation:

$$v(k) = \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \quad (4)$$

where:

- v(k) = Velocity at time instant k
- x(k) = Position at time instant k
- x(k-1) = Position at time instant k-1
- T = Fixed unit time or inverse of velocity calculation rate
- ΔX = Incremental position movement in unit time
- X = Fixed unit position
- ΔT = Incremental time elapsed for unit position movement
- t(k) = Time instant "k"
- t(k-1) = Time instant "k-1"

Unit time (T) and unit period (X) are configured using the QUPRD and QCAPCTL[UPPS] registers. Incremental position output and incremental time output is available in the QPOSLAT and QCPRDLAT registers.

Parameter	Relevant Register to Configure or Read the Information
T	Unit Period Register (QUPRD)
ΔX	Incremental Position = QPOSLAT(k) - QPOSLAT(K-1)
X	Fixed unit position defined by sensor resolution and ZCAPCTL[UPPS] bits
ΔT	Capture Period Latch (QCPRDLAT)

7.7 eQEP Watchdog

The eQEP peripheral contains a 16-bit watchdog timer (Figure 7-18) 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 quadrature clock event (pulse) resets the watchdog timer. If no quadrature clock event is detected until a period match (QWDPRD = QWDTMR), then the watchdog timer times out and the watchdog interrupt flag is set (QFLG[WTO]). The time-out value is programmable through the watchdog period register (QWDPRD).

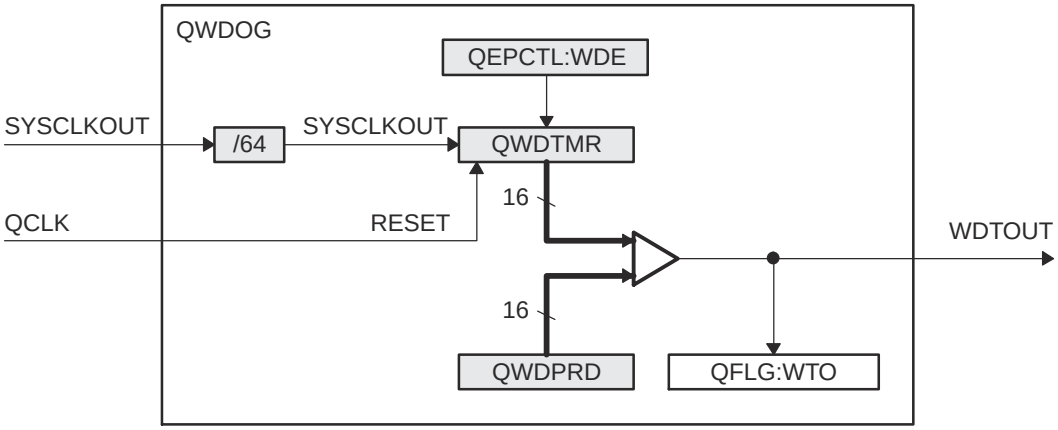


Figure 7-18. eQEP Watchdog Timer

7.8 eQEP Unit Timer Base

The eQEP peripheral includes a 32-bit timer (QUTMR) that is clocked by SYSCLKOUT to generate periodic interrupts for velocity calculations, see Figure 7-19. 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 7.6.

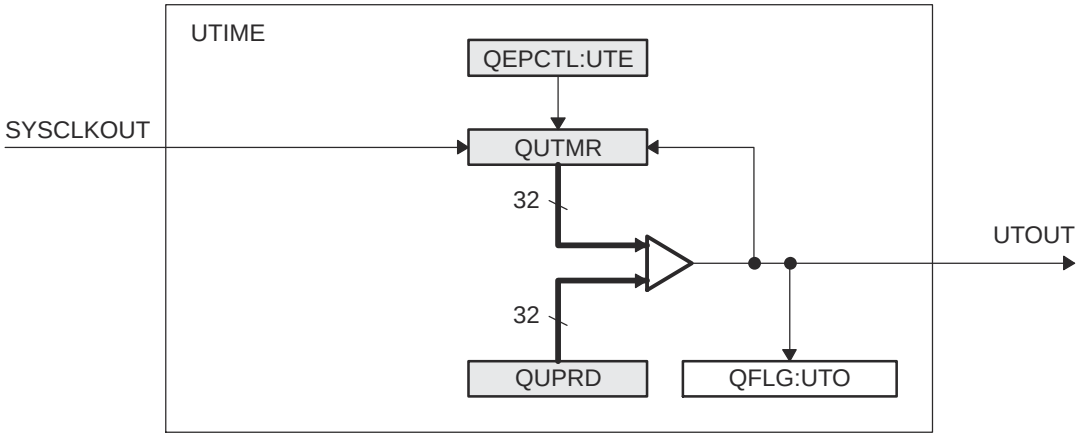


Figure 7-19. eQEP Unit Timer Base

7.9 eQEP Interrupt Structure

Figure 7-20 shows how the interrupt mechanism works in the eQEP module.

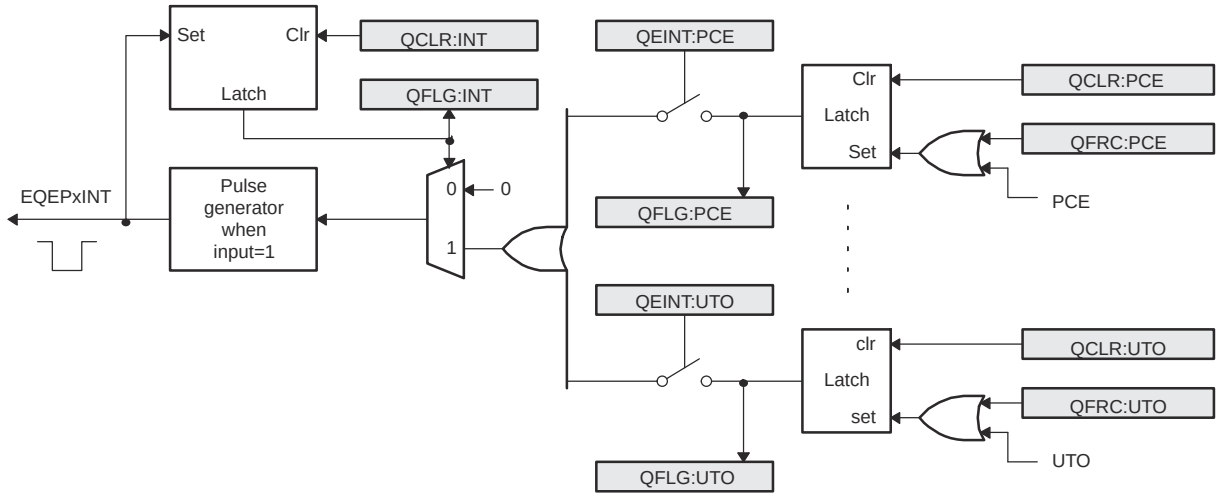


Figure 7-20. eQEP Interrupt Generation

Eleven interrupt events (PCE, PHE, QDC, WTO, PCU, PCO, PCR, PCM, SEL, IEL and UTO) can be generated. The interrupt control register (QEINT) is used to enable/disable individual interrupt event sources. The interrupt flag register (QFLG) indicates if any interrupt event has been latched and contains the global interrupt flag bit (INT).

An interrupt pulse is generated to PIE when:

1. Interrupt is enabled for eQEP event inside QEINT register
2. Interrupt flag for eQEP event inside QFLG register is set, and
3. 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, by way of the interrupt clear register (QCLR), before any other interrupt pulses are generated. If either flags inside the QFLG register are not cleared, further interrupt events will not generate an interrupt to PIE. You can force an interrupt event by way of the interrupt force register (QFRC), which is useful for test purposes.

7.10 eQEP Registers

This section describes the Enhanced Quadrature Encoder Pulse Registers.

7.10.1 eQEP Base Addresses

Table 7-3. eQEP Base Address Table (C28)

Bit Field Name		Base Address
Instance	Structure	
EQep1Regs	EQEP_REGS	0x0000_6B00

7.10.2 EQEP_REGS Registers

Table 7-4 lists the EQEP_REGS registers. All register offset addresses not listed in Table 7-4 should be considered as reserved locations and the register contents should not be modified.

Table 7-4. 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 7-5 shows the codes that are used for access types in this section.

Table 7-5. 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 7-5. 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.

7.10.2.1 QPOSCNT Register (Offset = 0h) [reset = 0h]

QPOSCNT is shown in Figure 7-20 and described in Table 7-6.

Return to the Summary Table.

Position Counter

Figure 7-21. 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 7-6. 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, that is, 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

7.10.2.2 QPOSINIT Register (Offset = 2h) [reset = 0h]

QPOSINIT is shown in Figure 7-21 and described in Table 7-7.

Return to the Summary Table.

Position Counter Init

Figure 7-22. 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 7-7. 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

7.10.2.3 QPOSMAX Register (Offset = 4h) [reset = 0h]

QPOSMAX is shown in Figure 7-22 and described in Table 7-8.

Return to the Summary Table.

Maximum Position Count

Figure 7-23. 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 7-8. 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

7.10.2.4 QPOSCMP Register (Offset = 6h) [reset = 0h]

QPOSCMP is shown in Figure 7-23 and described in Table 7-9.

Return to the Summary Table.

Position Compare

Figure 7-24. 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 7-9. 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

7.10.2.5 QPOSILAT Register (Offset = 8h) [reset = 0h]

QPOSILAT is shown in Figure 7-24 and described in Table 7-10.

Return to the Summary Table.

Index Position Latch

Figure 7-25. 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 7-10. 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

7.10.2.6 QPOSSLAT Register (Offset = Ah) [reset = 0h]

QPOSSLAT is shown in Figure 7-25 and described in Table 7-11.

Return to the Summary Table.

Strobe Position Latch

Figure 7-26. 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 7-11. 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

7.10.2.7 QPOSLAT Register (Offset = Ch) [reset = 0h]

QPOSLAT is shown in Figure 7-26 and described in Table 7-12.

Return to the Summary Table.

Position Latch

Figure 7-27. 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 7-12. 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

7.10.2.8 QUTMR Register (Offset = Eh) [reset = 0h]

QUTMR is shown in Figure 7-27 and described in Table 7-13.

Return to the Summary Table.

QEP Unit Timer

Figure 7-28. 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 7-13. 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

7.10.2.9 QUPRD Register (Offset = 10h) [reset = 0h]

QUPRD is shown in Figure 7-28 and described in Table 7-14.

Return to the Summary Table.

QEP Unit Period

Figure 7-29. 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 7-14. 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

7.10.2.10 QWDTMR Register (Offset = 12h) [reset = 0h]

QWDTMR is shown in Figure 7-29 and described in Table 7-15.

Return to the Summary Table.

QEP Watchdog Timer

Figure 7-30. 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 7-15. 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

7.10.2.11 QWDPRD Register (Offset = 13h) [reset = 0h]

QWDPRD is shown in Figure 7-30 and described in Table 7-16.

Return to the Summary Table.

QEP Watchdog Period

Figure 7-31. 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 7-16. 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

7.10.2.12 QDECCTL Register (Offset = 14h) [reset = 0h]

QDECCTL is shown in Figure 7-31 and described in Table 7-17.

Return to the Summary Table.

Quadrature Decoder Control

Figure 7-32. 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 7-17. 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 7-17. 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

7.10.2.13 QEPCTL Register (Offset = 15h) [reset = 0h]

QEPCTL is shown in Figure 7-32 and described in Table 7-18.

Return to the Summary Table.

QEP Control

Figure 7-33. 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 7-18. 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	Position 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

Table 7-18. QEPCTL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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
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.

Table 7-18. QEPCTL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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
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

7.10.2.14 QCAPCTL Register (Offset = 16h) [reset = 0h]

QCAPCTL is shown in Figure 7-33 and described in Table 7-19.

Return to the Summary Table.

Qaudrature Capture Control

Figure 7-34. 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 7-19. 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

7.10.2.15 QPOSCTL Register (Offset = 17h) [reset = 0h]

QPOSCTL is shown in Figure 7-34 and described in Table 7-20.

Return to the Summary Table.

Position Compare Control

Figure 7-35. 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 7-20. 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

7.10.2.16 QEINT Register (Offset = 18h) [reset = 0h]

QEINT is shown in Figure 7-35 and described in Table 7-21.

Return to the Summary Table.

QEP Interrupt Control

Figure 7-36. 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 7-21. 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 7-21. 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

7.10.2.17 QFLG Register (Offset = 19h) [reset = 0h]

QFLG is shown in Figure 7-36 and described in Table 7-22.

Return to the Summary Table.

QEP Interrupt Flag

Figure 7-37. 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 7-22. 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 7-22. 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

7.10.2.18 QCLR Register (Offset = 1Ah) [reset = 0h]

QCLR is shown in Figure 7-37 and described in Table 7-23.

Return to the Summary Table.

QEP Interrupt Clear

Figure 7-38. 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 7-23. 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 7-23. 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

7.10.2.19 QFRC Register (Offset = 1Bh) [reset = 0h]

QFRC is shown in Figure 7-38 and described in Table 7-24.

Return to the Summary Table.

QEP Interrupt Force

Figure 7-39. 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 7-24. 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 7-24. 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

7.10.2.20 QEPSTS Register (Offset = 1Ch) [reset = 0h]

QEPSTS is shown in Figure 7-39 and described in Table 7-25.

Return to the Summary Table.

QEP Status

Figure 7-40. 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 7-25. 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.

Table 7-25. QEPSTS Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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.
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

7.10.2.21 QCTMR Register (Offset = 1Dh) [reset = 0h]

QCTMR is shown in Figure 7-40 and described in Table 7-26.

Return to the Summary Table.

QEP Capture Timer

Figure 7-41. 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 7-26. 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

7.10.2.22 QCPRD Register (Offset = 1Eh) [reset = 0h]

QCPRD is shown in Figure 7-41 and described in Table 7-27.

Return to the Summary Table.

QEP Capture Period

Figure 7-42. 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 7-27. 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

7.10.2.23 QCTMRLAT Register (Offset = 1Fh) [reset = 0h]

QCTMRLAT is shown in Figure 7-42 and described in Table 7-28.

Return to the Summary Table.

QEP Capture Latch

Figure 7-43. 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 7-28. 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

7.10.2.24 QCPRDLAT Register (Offset = 20h) [reset = 0h]

QCPRDLAT is shown in Figure 7-43 and described in Table 7-29.

Return to the Summary Table.

QEP Capture Period Latch

Figure 7-44. 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 7-29. 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

This page intentionally left blank.

Chapter 8

Analog-to-Digital Converter (ADC)

The Analog-to-Digital Converter (ADC) module is a Type 3 ADC. See the *C2000 Real-Time Control Peripherals Reference Guide* for a list of all devices with modules of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

8.1 Introduction

The ADC module described in this chapter is a 12-bit recyclic ADC; part SAR, part pipelined. The analog circuits of this converter, referred to as the "core" in this document, include the front-end analog multiplexers (MUXs), sample-and-hold (S+H) circuits, the conversion core, voltage regulators, and other analog supporting circuits. Digital circuits, referred to as the "wrapper" in this document, include programmable conversions, result registers, interface to analog circuits, interface to device peripheral bus, and interface to other on-chip modules.

8.1.1 Features

The core of the ADC contains a single 12-bit converter fed by two sample and hold circuits. The sample and hold circuits can be sampled simultaneously or sequentially. These, in turn, are fed by a total of up to 16 analog input channels. See your device-specific data sheet for the specific number of channels available. The converter can be configured to run with an internal bandgap reference to create true-voltage based conversions or with a pair of external voltage references (VREFHI/LO) to create ratiometric based conversions.

Contrary to previous ADC types, this ADC is not sequencer based. It is easy for the user to create a series of conversions from a single trigger. However, the basic principle of operation is centered around the configurations of individual conversions, called SOCs, or Start-Of-Conversions.

Functions of the ADC module include:

- 12-bit ADC core with built-in dual sample-and-hold (S+H)
- Simultaneous sampling or sequential sampling modes
- Full range analog input: 0 V to 3.3 V fixed, or VREFHI/VREFLO ratiometric
- Up to 16-channel, multiplexed inputs
- 16 SOCs, configurable for trigger, sample window, and channel
- 16 result registers (individually addressable) to store conversion values
- Multiple trigger sources
  - S/W - software immediate start
  - ePWM 1-7
  - GPIO XINT2
  - CPU Timers 0/1/2
  - ADCINT1/2
- 9 flexible PIE interrupts, can configure interrupt request after any conversion

8.1.2 ADC Related Collateral

Foundational Materials

- *ADC Input Circuit Evaluation for C2000 MCUs (TINA-TI) Application Report*
- PSpice for TI design and simulation tool
- *Real-Time Control Reference Guide*
  - Refer to the ADC section
- TI Precision Labs - ADCs (Video)
  - Start with the "Introduction to analog-to-digital converters (ADCs)" section.
- TI Precision Labs - ADCs
- TI Precision Labs: Driving the reference input on a SAR ADC (Video)
- TI Precision Labs: Introduction to analog-to-digital converters (ADCs) (Video)
- TI Precision Labs: SAR ADC input driver design (Video)
- TI e2e: Connecting VDDA to VREFHI
- TI e2e: Topologies for ADC Input Protection
- TI e2e: Why does the ADC Input Voltage drop with sampling?
  - Sampling a high impedance voltage divider with ADC
- *Understanding Data Converters Application Report*

Getting Started Materials

- ADC-PWM Synchronization Using ADC Interrupt
  - NOTE: This is a non-TI (third party) site.
- *C2000 ADC (Type-3) Performance Versus ACQPS Application Report*
- *Hardware Design Guide for F2800x C2000 Real-Time MCU Series*

Expert Materials

- Analog Engineer's Calculator
- *Analog Engineer's Pocket Reference*
- Debugging an integrated ADC in a microcontroller using an oscilloscope
- TI Precision Labs: ADC AC specifications (Video)
- TI Precision Labs: ADC Error sources (Video)
- TI Precision Labs: ADC Noise (Video)
- TI Precision Labs: Analog-to-digital converter (ADC) drive topologies (Video)
- TI Precision Labs: Electrical overstress on data converters (Video)
- TI Precision Labs: High-speed ADC fundamentals (Video)
- TI Precision Labs: SAR & Delta-Sigma: Understanding the Difference (Video)
- TI e2e: ADC Bandwidth Clarification
- TI e2e: ADC Calibration and Total Unadjusted Error
- TI e2e: ADC Reference Driver Options
- TI e2e: ADC Resolution with Oversampling
- TI e2e: ADC configuration for interleaved mode
- TI e2e: Simultaneous Sampling with Single ADC

8.1.3 Block Diagram

Figure 8-1 shows the block diagram of the ADC module.

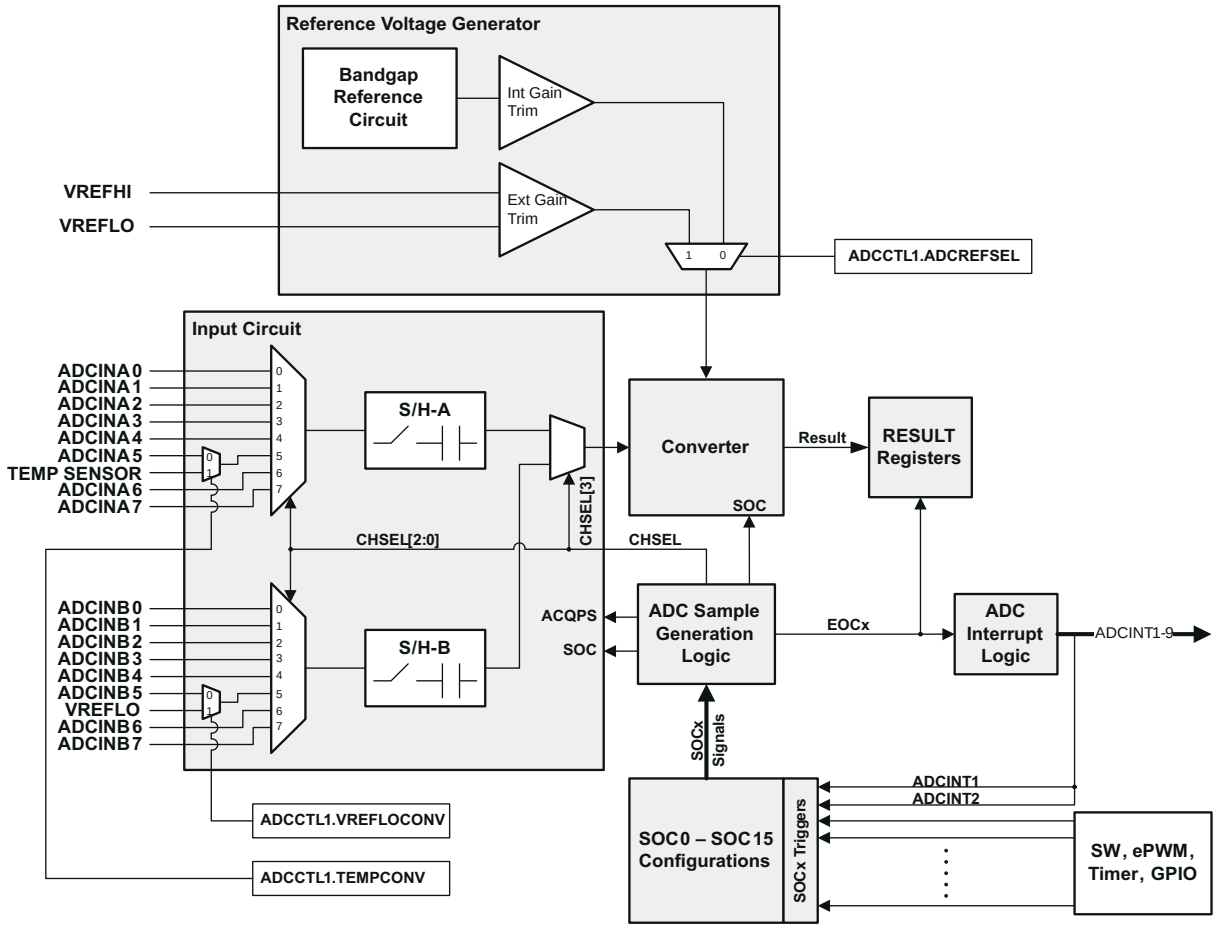


Figure 8-1. ADC Block Diagram

8.2 SOC Principle of Operation

Contrary to previous ADC types, this ADC is not sequencer based. Instead, it is SOC based. The term SOC is configuration set defining the single conversion of a single channel. In that set there are three configurations: the trigger source that starts the conversion, the channel to convert, and the acquisition (sample) window size. Each SOC is independently configured and can have any combination of the trigger, channel, and sample window size available. Multiple SOCs can be configured for the same trigger, channel, and/or acquisition window as desired. This provides a very flexible means of configuring conversions ranging from individual samples of different channels with different triggers, to oversampling the same channel using a single trigger, to creating your own series of conversions of different channels all from a single trigger.

The trigger source for SOCx is configured by a combination of the TRIGSEL field in the ADCSOCxCTL register and the appropriate bits in the ADCINTSOCSEL1 or ADCINTSOCSEL2 register. Software can also force an SOC event with the ADCSOCFRC1 register. The channel and sample window size for SOCx are configured with the CHSEL and ACQPS fields of the ADCSOCxCTL register.

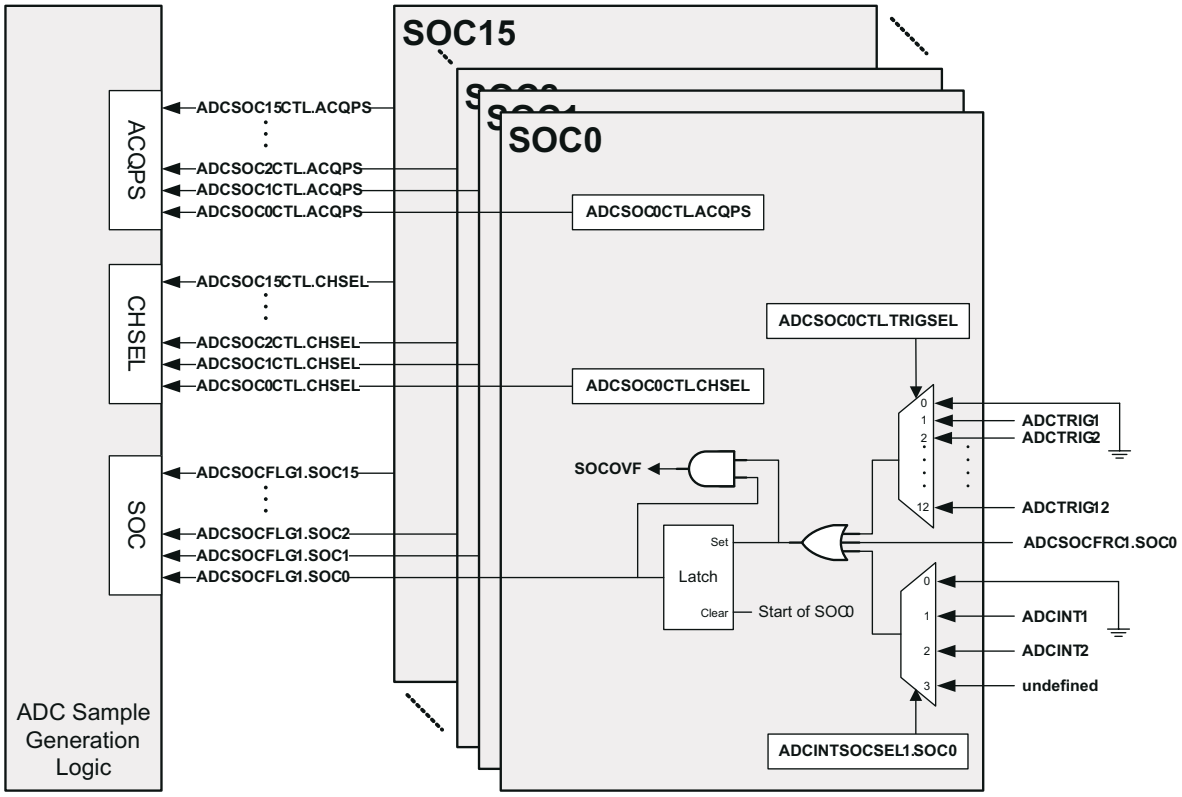


Figure 8-2. SOC Block Diagram

For example, to configure a single conversion on channel ADCINA1 to occur when the ePWM3 timer reaches its period match, you must first setup ePWM3 to output an SOCA or SOCB signal on a period match (see Chapter 3). In this case, we will use SOCA. Then, set up one of the SOCs using its ADCSOCxCTL register. It makes no difference which SOC we choose, so we will use SOC0. The fastest allowable sample window for the ADC is 7 cycles. Choosing the fastest time for the sample window, channel ADCINA1 for the channel to convert, and ePWM3 for the SOC0 trigger, we'll set the ACQPS field to 6, the CHSEL field to 1, and the TRIGSEL field to 9, respectively. The resulting value written into the register is:

```
ADCSOC0CTL = 4846h;                    // (ACQPS=6, CHSEL=1, TRIGSEL=9)
```

When configured as such, a single conversion of ADCINA1 will be started on an ePWM3 SOCA event with the resulting value stored in the ADCRESULT0 register.

If instead ADCINA1 needed to be oversampled by 3X, then SOC1, SOC2, and SOC3 could all be given the same configuration as SOC0.

```
ADCSOC1CTL = 4846h;   // (ACQPS=6, CHSEL=1, TRIGSEL=9)
ADCSOC2CTL = 4846h;   // (ACQPS=6, CHSEL=1, TRIGSEL=9)
ADCSOC3CTL = 4846h;   // (ACQPS=6, CHSEL=1, TRIGSEL=9)
```

When configured as such, four conversions of ADCINA1 will be started in series on an ePWM3 SOCA event with the resulting values stored in the ADCRESULT0 – ADCRESULT3 registers.

Another application may require 3 different signals to be sampled from the same trigger. This can be done by simply changing the CHSEL field for SOC0-SOC2 while leaving the TRIGSEL field unchanged.

```
ADCSOC0CTL = 4846h;   // (ACQPS=6, CHSEL=1, TRIGSEL=9)
ADCSOC1CTL = 4886h;   // (ACQPS=6, CHSEL=2, TRIGSEL=9)
ADCSOC2CTL = 48C6h;   // (ACQPS=6, CHSEL=3, TRIGSEL=9)
```

When configured this way, three conversions will be started in series on an ePWM3 SOCA event. The result of the conversion on channel ADCINA1 will show up in ADCRESULT0. The result of the conversion on channel ADCINA2 will show up in ADCRESULT1. The result of the conversion on channel ADCINA3 will show up in ADCRESULT2. The channel converted and the trigger have no bearing on where the result of the conversion shows up. The RESULT register is associated with the SOC.

Note

These examples are incomplete. Clocks must be enabled via the PCLKCR0 register and the ADC must be powered to work correctly. For a description of the PCLKCR0 register, see the *System Control and Interrupts* chapter. For the power-up sequence of the ADC, see Section 8.8. The CLKDIV2EN bit in the ADCCTL2 register must also be set to a proper value to obtain correct frequency of operation. For more information on the ADCCTL2 register, refer to Section 8.13.

8.2.1 ADC Acquisition (Sample and Hold) Window

External drivers vary in their ability to drive an analog signal quickly and effectively. Some circuits require longer times to properly transfer the charge into the sampling capacitor of an ADC. To address this, the ADC supports control over the sample window length for each individual SOC configuration. Each ADCSOCxCTL register has a 6-bit field, ACQPS, that determines the sample and hold (S+H) window size. The value written to this field is one less than the number of cycles desired for the sampling window for that SOC. Thus, a value of 15 in this field will give 16 clock cycles of sample time. The minimum number of sample cycles allowed is 7 (ACQPS=6). The total sampling time is found by adding the sample window size to the conversion time of the ADC, 13 ADC clocks. Examples of various sample times are shown in Table 8-1.

Table 8-1. Sample Timings with Different Values of ACQPS

ADC Clock	ACQPS	Sample Window	Conversion Time (13 cycles)	Total Time to Process Analog Voltage(1)
40 MHz	6	175.00 ns	325.00 ns	500.00 ns
40 MHz	25	625.00 ns	325.00 ns	950.00 ns
60 MHz	6	116.67 ns	216.67 ns	333.33 ns
60 MHz	25	433.67 ns	216.67 ns	650 ns

(1) The total times are for a single conversion and do not include pipelining effects that increase the average speed over time.

As shown in Figure 8-3, the ADCIN pins can be modeled as an RC circuit. With VREFLO connected to ground, a voltage swing from 0 to 3.3 V on ADCIN yields a typical RC time constant of 2 ns.

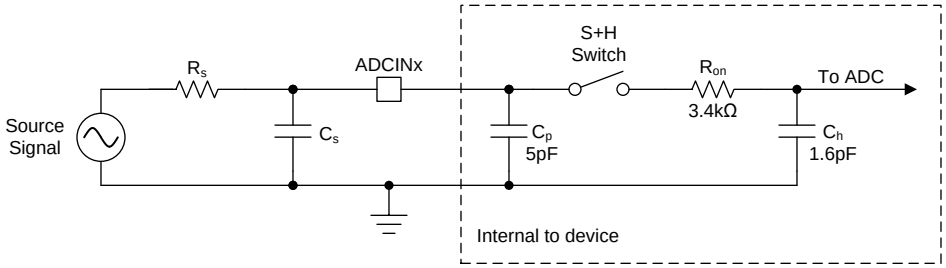


Figure 8-3. ADCINx Input Model

Note

The ADC does not precondition the Ch capacitor voltage before conversions, therefore the following behaviors apply:

1. There is no predetermined ADC conversion value when the ADCIN pin is not connected to a Source Signal.
2. Residual charge will remain on Ch between ADC conversions.
3. Sequential conversions may suffer from cross-talk if the ACQPS window is too short for Ch to settle.

For correct operation, the input signal to the ADC must be allowed adequate time to charge the sample and hold capacitor, C_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.

8.2.1.1 ACQPS Approximation for High Bandwidth Signals

Signals that must be sampled frequently with minimal phase delay (such as feedback sensors used in control-loop calculations) are high bandwidth signals. These signal paths require a small R_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.

8.2.1.1.1 ACQPS Approximation Equations for High Bandwidth Signals

An approximation of the required settling time can be determined using an RC settling model. The time constant (τ) for the model is given by the equation:

$$\tau = (R_s + R_{on})(C_h) + (R_s)(C_s + C_p)$$

And the number of time constants needed is given by the equation:

$$k = \ln\left(\frac{2^n}{settling\ error}\right) - \ln\left(\frac{C_s + C_p}{C_h}\right)$$

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

8.2.1.1.2 ACQPS Approximation Example for High Bandwidth Signals

For example, assuming the following parameters:

- n = 12-bits
- settling error =¼ LSB
- R_{on} = 3400Ω
- C_h = 1.6pF
- C_p = 5pF
- R_s = 56Ω
- C_s = 2.2nF
- f_{ADCCLK} = 30MHz

The time constant would be calculated as:

$$\tau = (56\Omega + 3400\Omega)(1.6pF) + (56\Omega)(2200pF + 5pF) = 5.5ns + 123.5ns = 129ns$$

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.

8.2.1.2 ACQPS Approximation for Low Bandwidth Signals

Signals that are sampled infrequently and are tolerant of low-pass filtering (such as ambient temperature sensors) can be treated as low bandwidth signals. A large C_s that is sized to be much larger than C_h will allow the ADC to sample the R_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.

8.2.1.2.1 ACQPS Approximation Equations for Low Bandwidth Signals

The desired settling accuracy will determine the value of C_s:

$$C_s = C_h \left(\frac{2^n}{settling\ error} \right)$$

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:

$$t_{s+h} = k \cdot \tau$$

Finally, t_{S+H} is used to determine the minimum value to program into the ACQPS field of the ADCSOCxCTL registers:

$$ACQPS = (t_{s+h} \cdot f_{ADCCLK}) - 1$$

Where the following parameters are provided by the ADC input model:

- n = ADC resolution (in bits)
- R_{on} = ADC sampling switch resistance
- C_h = ADC sampling capacitor
- C_p = ADC parasitic pin capacitance for the channel

And the following parameters are dependent on the application design:

- settling error = tolerable settling error (in LSBs)
- $t_{recovery}$ = 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 8-2 shows the relationship between n_τ and the estimated droop error.

Table 8-2. Estimated Droop Error from n_T Value

n_T	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

8.2.1.2.2 ACQPS Approximation Example for Low Bandwidth Signals

For example, assuming the following parameters:

- n = 12-bits
- settling error =¼ LSB
- $t_{recovery}$ = 1 ms
- n_T = 2
- R_{on} = 3400 Ω
- C_h = 1.6 pF
- C_p = 5 pF
- f_{ADCCLK} = 30 MHz

The minimum source capacitance would be calculated and rounded up to a common value:

$$C_s = 1.6pF\left(\frac{2^{12}\ LSB}{0.25\ LSB}\right) = 26.2nF \rightarrow 33nF$$

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.

8.2.2 Trigger Operation

Each SOC can be configured to start on one of many input triggers. Multiple SOCs can be configured for the same channel if desired. Following is a list of the available input triggers:

- Software
- CPU Timers 0/1/2 interrupts
- XINT2 SOC
- ePWM1-7 SOCA and SOCB

See the ADCSOCxCTL register bit definitions for the configuration details of these triggers.

Additionally ADCINT1 and ADCINT2 can be fed back to trigger another conversion. This configuration is controlled in the ADCINTSOCSEL1/2 registers. This mode is useful if a continuous stream of conversions is desired. See Section 8.7 for information on the ADC interrupt signals.

8.2.3 Channel Selection

Each SOC can be configured to convert any of the available ADCIN input channels. When an SOC is configured for sequential sampling mode, the four bit CHSEL field of the ADCSOCxCTL register defines which channel to convert. When an SOC is configured for simultaneous sampling mode, the most significant bit of the CHSEL field is dropped and the lower three bits determine which pair of channels are converted.

ADCINA0 is shared with VREFHI, and therefore cannot be used as a variable input source when using external reference voltage mode. See Section 8.10 for details on this mode.

8.3 ONESHOT Single Conversion Support

This mode will allow you to perform a single conversion on the next triggered SOC in the round robin scheme. The ONESHOT mode is only valid for channels present in the round robin wheel. Channels which are not configured for triggered SOC in the round robin scheme will get priority based on contents of the SOCPRIORITY field in the ADCSOCPRIORITYCTL register.

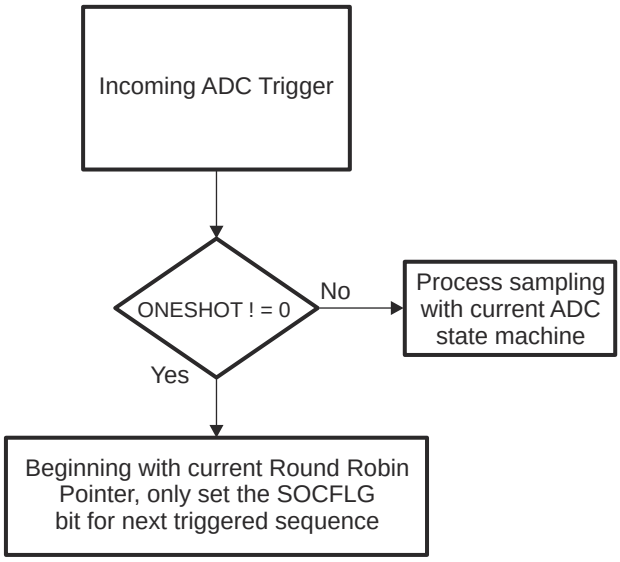


Figure 8-4. ONESHOT Single Conversion

The effect of ONESHOT mode on Sequential Mode and Simultaneous Mode is explained below.

Sequential mode: Only the next active SOC in RR mode (one up from current RR pointer) will be allowed to generate SOC; all other triggers for other SOC slots will be ignored.

Simultaneous mode: If current RR pointer has SOC with simultaneous enabled; active SOC will be incremented by 2 from the current RR pointer. This is because simultaneous mode will create result for SOCx and SOCx+1, and SOCx+1 will never be triggered by the user.

Note

ONESHOT = 1 and SOCPRIORITY = 10h is not a valid combination for above implementation reasons. This should not be a desired mode of operation by the user in any case. The limitation of the above is that the next SOCs must eventually be triggered, or else the ADC will not generate new SOCs for other out-of-order triggers. Any non-orthogonal channels should be placed in the priority mode which is unaffected by ONESHOT mode

8.4 ADC Conversion Priority

When multiple SOC flags are set at the same time, one of two forms of priority determines the order in which they are converted. The default priority method is round robin. In this scheme, no SOC has an inherent higher priority than another. Priority depends on the round robin pointer (RRPOINTER). The RRPOINTER reflected in the ADCSOCPRIORITYCTL register points to the last SOC converted. The highest priority SOC is given to the next value greater than the RRPOINTER value, wrapping around back to SOC0 after SOC15. At reset the value is 16 since 0 indicates a conversion has already occurred. When RRPOINTER equals 16, the highest priority is given to SOC0. The RRPOINTER is reset by a device reset, when the ADCCTL1.RESET bit is set, or when the SOCPRICTL register is written.

An example of the round robin priority method is given in Figure 8-5 .

A After reset, SOC0 is highest priority SOC; SOC7 receives trigger; SOC7 configured channel is converted immediately.

B RRPOINTER changes to point to SOC 7; SOC8 is now highest priority SOC.

C SOC2 & SOC12 triggers rcvd. simultaneously; SOC12 is first on round robin wheel; SOC12 configured channel is converted while SOC2 stays pending.

D RRPOINTER changes to point to SOC 12; SOC2 configured channel is now converted.

E RRPOINTER changes to point to SOC 2; SOC3 is now highest priority SOC.

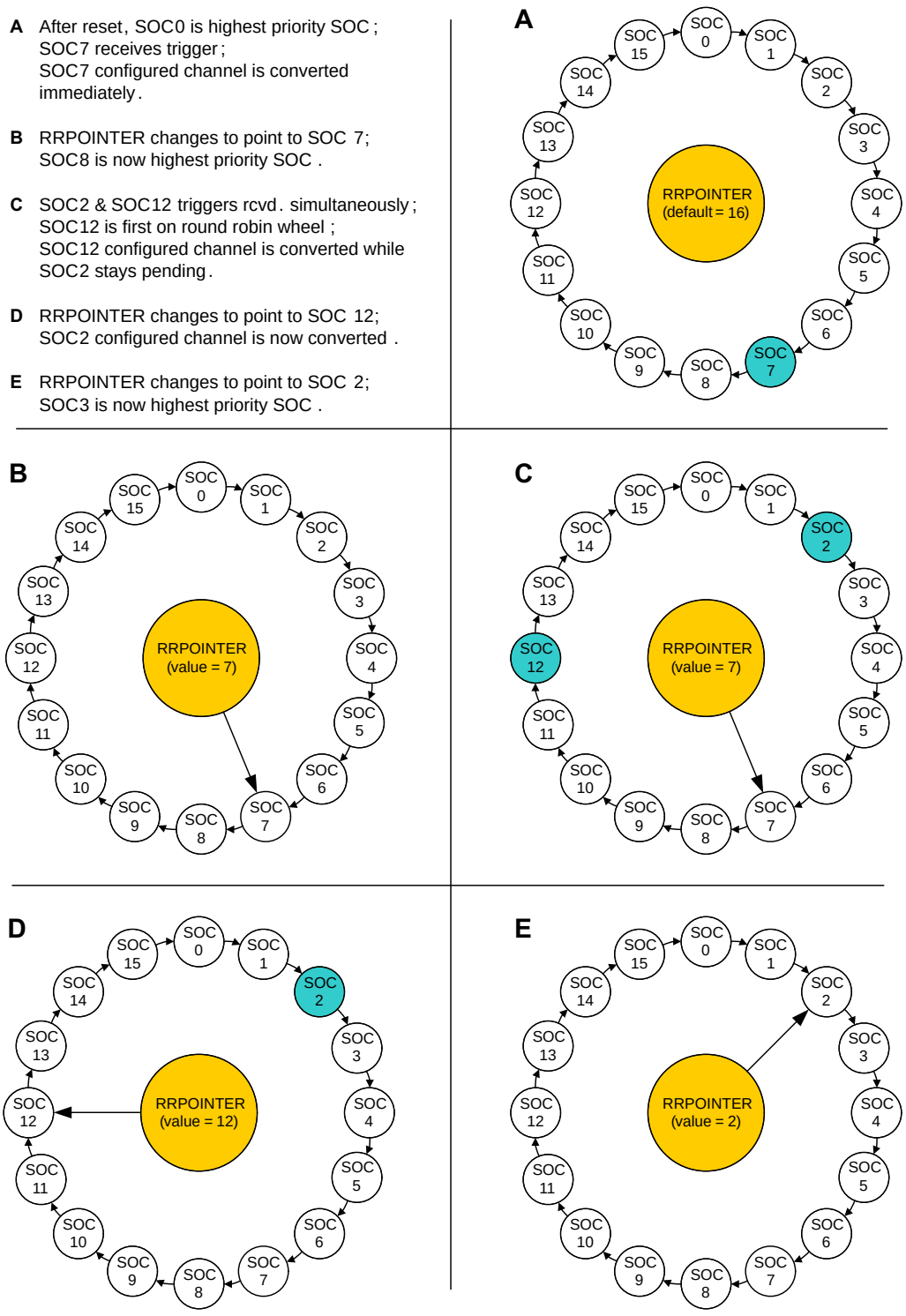


Figure 8-5. Round Robin Priority Example

The SOCPRIORITY field in the ADCSOCPRIORITYCTL register can be used to assign high priority from a single to all of the SOC's. When configured as high priority, an SOC will interrupt the round robin wheel after any current conversion completes and insert itself in as the next conversion. After its conversion completes, the round robin wheel will continue where it was interrupted. If two high priority SOC's are triggered at the same time, the SOC with the lower number will take precedence.

High priority mode is assigned first to SOC0, then in increasing numerical order. The value written in the SOCPRIORITY field defines the first SOC that is not high priority. In other words, if a value of 4 is written into SOCPRIORITY, then SOC0, SOC1, SOC2, and SOC3 are defined as high priority, with SOC0 the highest.

An example using high priority SOC's is given in Figure 8-6 .

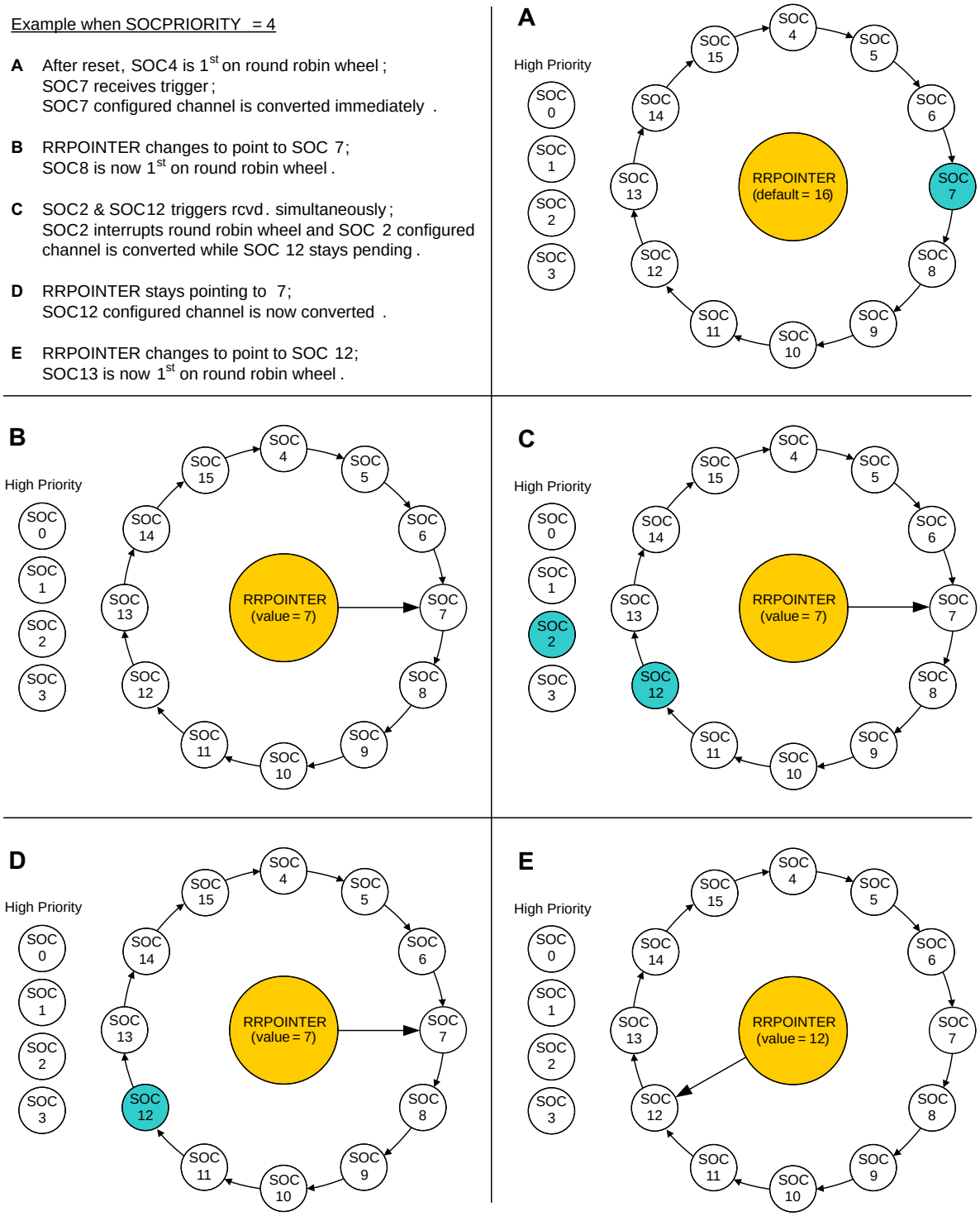


Figure 8-6. High Priority Example

8.5 Sequential Sampling Mode

The default behavior of the ADC is to treat triggered SOCs as single conversions to be processed sequentially. Sequential sampling can convert both A-channels and B-channels without restriction on ordering.

However, a by-product of supporting the Simultaneous Sampling Mode is that the sampling capacitor from the paired simultaneous channel will also be connected to its respective input at the same time as the desired sequential sampling ACQPS window; the ADC will not convert the sample from the paired channel. (The Simultaneous Sampling Mode is described in Section 8.6, and the sampling capacitor is denoted as Ch in Figure 8-3.)

For example, assume that SOC0 is configured to convert ADCINB3, and SOC1 is configured to convert ADCINA5. If SOC0 and SOC1 are triggered together, the following sequence of simplified events would take place:

1. SOC0 Sample: A-channel and B-channel sampling capacitors are connected to ADCINA3 and ADCINB3 for SOC0 ACQPS window
2. SOC0 Convert: ADC converts B-channel sampling capacitor and stores result to ADCRESULT0
3. SOC1 Sample: A-channel and B-channel sampling capacitors are connected to ADCINA5 and ADCINB5 for SOC1 ACQPS window
4. SOC1 Convert: ADC converts A-channel sampling capacitor and stores result to ADCRESULT1

The extraneous sampling capacitor exposure should be taken into consideration for input signals that have a slow recovery or settling time. Typical examples of slow inputs are sensors with high impedance outputs and signals that are conditioned with low-pass filters.

8.6 Simultaneous Sampling Mode

In some applications it is important to keep the delay between the sampling of two signals minimal. The ADC contains dual sample and hold circuits to allow two different channels to be sampled simultaneously. Simultaneous sampling mode is configured for a pair of SOCx's with the ADCSAMPLEMODE register. The even-numbered SOCx and the following odd-numbered SOCx (SOC0 and SOC1) are coupled together with one enable bit (SIMULEN0, in this case). The coupling behavior is as follows:

- Either SOCx's trigger will start a pair of conversions.
- The pair of channels converted will consist of the A-channel and the B-channel corresponding to the value of the CHSEL field of the triggered SOCx. The valid values in this mode are 0-7.
- Both channels will be sampled simultaneously.
- The A channel will always convert first.
- The even EOCx pulse will be generated based off of the A-channel conversion, the odd EOCx pulse will be generated off of the B-channel conversion. See Section 1.6 for an explanation of the EOCx signals.
- The result of the A-channel conversion is placed in the even ADCRESULTx register and the result of the B-channel conversion is written to the odd ADCRESULTx register.

For example, if the ADCSAMPLEMODE.SIMULEN0 bit is set, and SOC0 is configured as follows:

CHSEL = 2 (ADCINA2/ADCINB2 pair)

TRIGSEL = 5 (ADCTRIG5 = ePWM1.ADCSOCA)

When the ePWM1 sends out an ADCSOCA trigger, both ADCINA2 and ADCINB2 will be sampled simultaneously (assuming priority). Immediately after, the ADCINA2 channel will be converted and its value will be stored in the ADCRESULT0 register. Depending on the ADCCTL1.INTPULSEPOS setting, the EOC0 pulse will either occur when the conversion of ADCINA2 begins or completes. Then the ADCINB2 channel will be converted and its value will be stored in the ADCRESULT1 register. Depending on the ADCCTL1.INTPULSEPOS setting, the EOC1 pulse will either occur when the conversion of ADCINB2 begins or completes.

Typically in an application it is expected that only the even SOCx of the pair will be used. However, it is possible to use the odd SOCx instead, or even both. In the latter case, both SOCx triggers will start a conversion.

Therefore, caution is urged as both SOCx's will store their results to the same ADCRESULTx registers, possibly overwriting each other.

The rules of priority for the SOCx's remain the same as in sequential sampling mode.

Section 8.11 shows the timing of simultaneous sampling mode.

8.7 EOC and Interrupt Operation

Just as there are 16 independent SOCx configuration sets, there are 16 EOCx pulses. In sequential sampling mode, the EOCx is associated directly with the SOCx. In simultaneous sampling mode, the even and the following odd EOCx pair are associated with the even and the following odd SOCx pair, as described in Section 8.6. Depending on the ADCCTL1.INTPULSEPOS setting, the EOCx pulse will occur either at the beginning of a conversion or the end. See section 1.11 for exact timings on the EOCx pulses.

The ADC contains 9 interrupts that can be flagged and/or passed on to the PIE. Each of these interrupts can be configured to accept any of the available EOCx signals as its source. The configuration of which EOCx is the source is done in the INTSELxNy registers. Additionally, the ADCINT1 and ADCINT2 signals can be configured to generate an SOCx trigger. This is beneficial to creating a continuous stream of conversions.

Figure 8-7 shows a block diagram of the interrupt structure of the ADC.

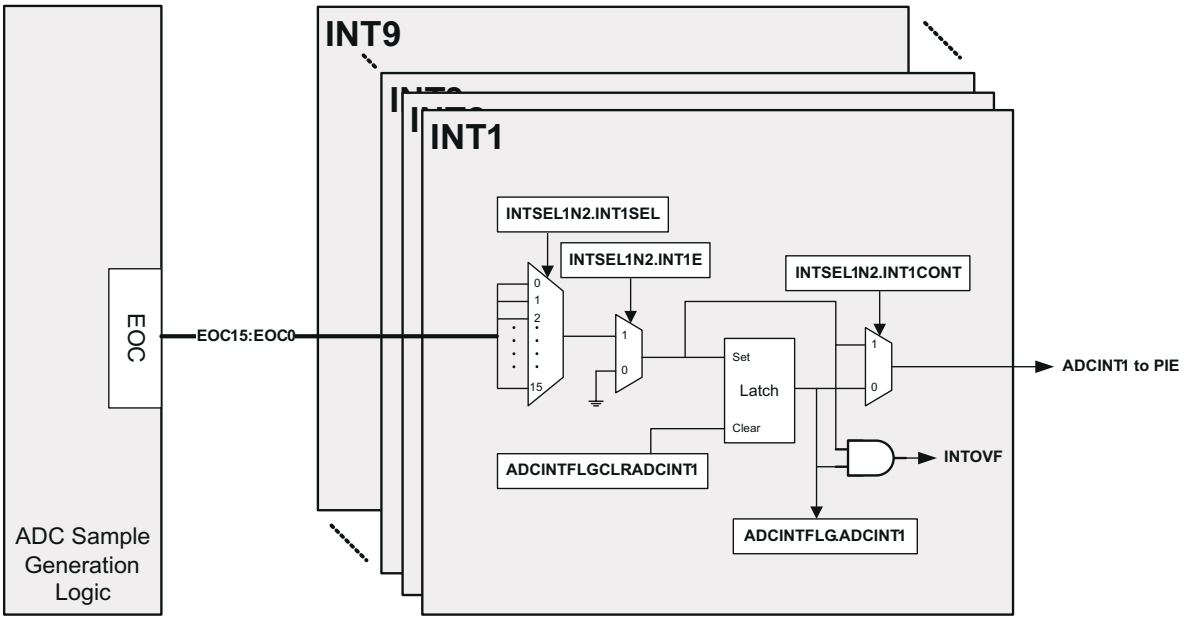


Figure 8-7. Interrupt Structure

Note

Interrupt generation may be disrupted in non-continuous conversion mode when the interrupt overflow bit in ADCINTOVF is set.

8.8 Power-Up Sequence

The ADC resets to the ADC off state. Before writing to any of the ADC registers the ADCENCLK bit in the PCLKCR0 register must be set. For a description of the PCLKCR0 register, see the System Control and Interrupts section in this manual. When powering up the ADC, use the following sequence:

1. If an external reference is desired, enable this mode using bit 3 (ADCREFSEL) in the ADCCTL1 register.
2. Power up the reference, bandgap, and analog circuits together by setting bits 7-5 (ADCPWDN, ADCBGPWD, ADCREFPWD) in the ADCCTL1 register.
3. Enable the ADC by setting bit 14 (ADCENABLE) of the ADCCTL1 register.
4. Before performing the first conversion, a delay of 1 millisecond after step 2 is required.

Alternatively, steps 1 through 3 can be performed simultaneously.

When powering down the ADC, all three bits in step 2 can be cleared simultaneously. The ADC power levels must be controlled via software and they are independent of the state of the device power modes.

Note

This type ADC requires a 1ms delay after all of the circuits are powered up. This differs from the previous type ADC's.

8.9 ADC Calibration

Inherent in any converter is a zero offset error and a full scale gain error. The ADC is factory calibrated at 30-degrees Celsius to correct both of these while allowing the user to modify the offset correction for any application environmental effects, such as the ambient temperature. Except under certain emulation conditions, or unless a modification from the factory settings is desired, the user is not required to perform any specific action. The ADC will be properly calibrated during the device boot process.

Note

If the system is reset or the ADC module is reset using Bit 15 (RESET) from the ADC Control Register 1, the Device_cal() routine must be repeated.

8.9.1 Factory Settings and Calibration Function

During the fabrication and test process Texas Instruments calibrates several ADC settings along with a couple of internal oscillator settings. These settings are embedded into the TI reserved OTP memory as part of a C-callable function named Device_cal(). Called during the startup boot procedure in the Boot ROM this function writes the factory settings into their respective active registers. Until this occurs, the ADC and the internal oscillators will not adhere to their specified parameters. If the boot process is skipped during emulation, the user must ensure the trim settings are written to their respective registers to ensure the ADC and the internal oscillators meet the specifications in the data sheet. This can be done either by calling this function manually or in the application itself, or by a direct write via CCS. A gel function for device calibration is included in CCS when the appropriate .ccxml file is created for the target MCU.

For more information on the Device_cal() function refer to the Boot ROM section in this manual.

Texas Instruments cannot assure the parameters specified in the data sheet, if a value other than the factory settings contained in the TI reserved OTP memory is written into the ADC trim registers.

8.9.2 ADC Zero Offset Calibration

Zero offset error is defined as the resultant digital value that occurs when converting a voltage at VREFLO. This base error affects all conversions of the ADC and together with the full scale gain and linearity specifications, determine the DC accuracy of a converter. The zero offset error can be positive, meaning that a positive digital value is output when VREFLO is presented, or negative, meaning that a voltage higher than a one step above VREFLO still reads as a digital zero value. To correct this error, the two's complement of the error is written into the ADCOFFTRIM register. The value contained in this register will be applied before the results are available in the ADC result registers. This operation is fully contained within the ADC core, so the timing for the results

will not be affected and the full dynamic range of the ADC will be maintained for any trim value. Calling the Device_cal() function writes the ADCOFFTRIM register with the factory calibrated offset error correction, but the user can modify the ADCOFFTRIM register to compensate for additional offset error induced by the application environment. This can be done without sacrificing an ADC channel by using the VREFLOCONV bit in the ADCCTRL1 register.

Use the following procedure to re-calibrate the ADC offset:

1. **Set ADCOFFTRIM to 80 (50h)**. This adds an artificial offset to account for negative offset that may reside in the ADC core.
2. **Set ADCCTL1.VREFLOCONV to 1.** This internally connects VREFLO to input channel B5. See the ADCCTL1 register description.
3. **Perform multiple conversions on B5 (sample VREFLO) and take an average to account for board noise.** See Section 8.2 on how to setup and initiate the ADC to sample B5.
4. **Set ADCOFFTRIM to 80 (50h) minus the average obtained in step 3.** This removes the artificial offset from step 1 and creates a two's compliment of the offset error.
5. **Set ADCCTL1.VREFLOCONV to 0.** This connects B5 back to the external ADCINB5 input pin.

Note

The AdcOffsetSelfCal() function located in F2803x_Adc.c in the common header files performs these steps.

8.9.3 ADC Full Scale Gain Calibration

Gain error occurs as an incremental error as the voltage input is increased. Full scale gain error occurs at the maximum input voltage. As in offset error, gain error can be positive or negative. A positive full scale gain error means that the full scale digital result is reached before the maximum voltage is input. A negative full scale error implies that the full digital result will never be achieved. The calibration function Device_cal() writes a factory trim value to correct the ADC full scale gain error into the ADCREFTRIM register. This register should not be modified after the Device_cal() function is called.

8.9.4 ADC Bias Current Calibration

To further increase the accuracy of the ADC, the calibration function Device_cal() also writes a factory trim value to an ADC register for the ADC bias currents. This register should not be modified after the Device_cal() function is called.

8.10 Internal/External Reference Voltage Selection

8.10.1 Internal Reference Voltage

The ADC can operate in two different reference modes, selected by the ADCCTL1.ADCREFSEL bit. By default the internal bandgap is chosen to generate the reference voltage for the ADC. This will convert the voltage presented according to a fixed scale 0 to 3.3 V range. The equation governing conversions in this mode is:

Digital Value = 0	when Input ≤ 0 V
Digital Value = 4096 [(Input – VREFLO)/3.3 V]	when 0 V < Input < 3.3 V
Digital Value = 4095,	when Input ≥ 3.3 V

*All fractional values are truncated

**VREFLO must be tied to ground in this mode. This is done internally on some devices.

8.10.2 External Reference Voltage

To convert the voltage presented as a ratiometric signal, the external VREFHI/VREFLO pins should be chosen to generate the reference voltage. In contrast with the fixed 0 to 3.3 V input range of the internal bandgap mode, the ratiometric mode has an input range from VREFLO to VREFHI. Converted values will scale to this range. For instance, if VREFLO is set to 0.5 V and VREFHI is 3.0 V, a voltage of 1.75 V will be converted to the digital result of 2048. See the device data sheet for the allowable ranges of VREFLO and VREFHI. On some devices VREFLO is tied to ground internally, and hence limited to 0 V. The equation governing the conversions in this mode is:

Digital Value = 0	when Input ≤ VREFLO
Digital Value = 4096 [(Input – VREFLO)/(VREFHI – VREFLO)]	when VREFLO < Input < VREFHI
Digital Value = 4095,	when Input ≥ VREFHI

*All fractional values are truncated

8.11 ADC Timings

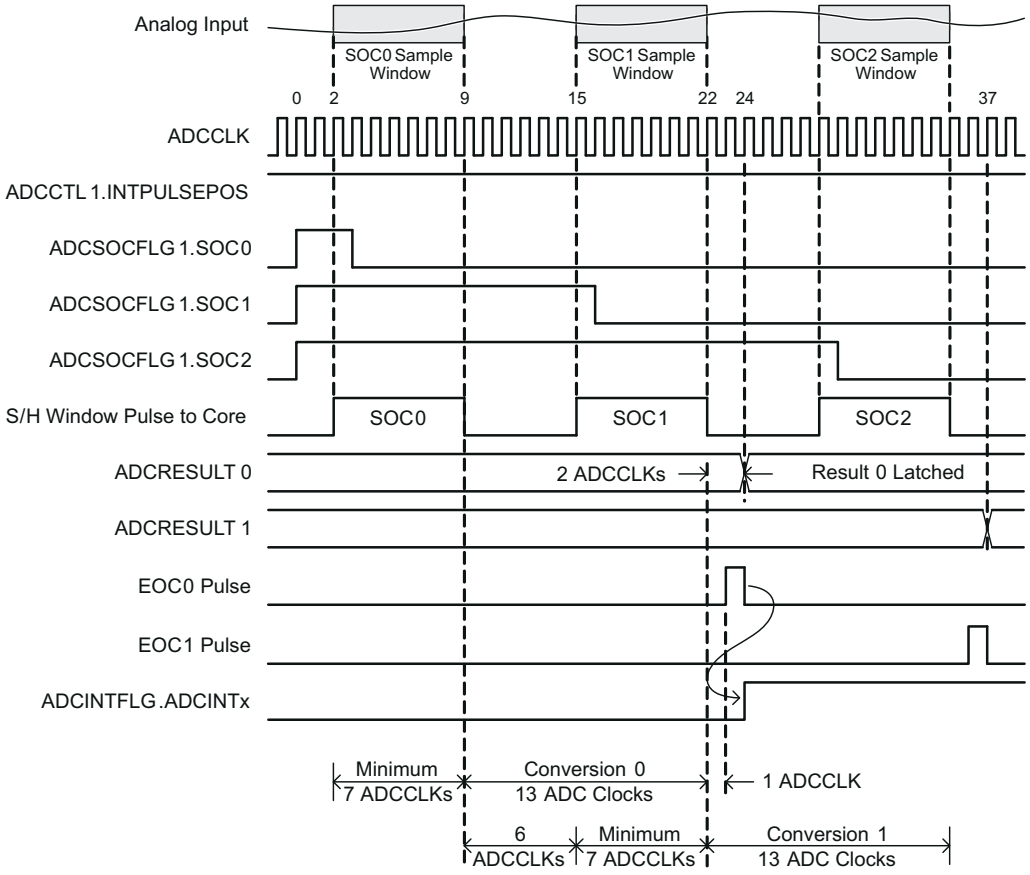


Figure 8-8. Timing Example For Sequential Mode / Late Interrupt Pulse

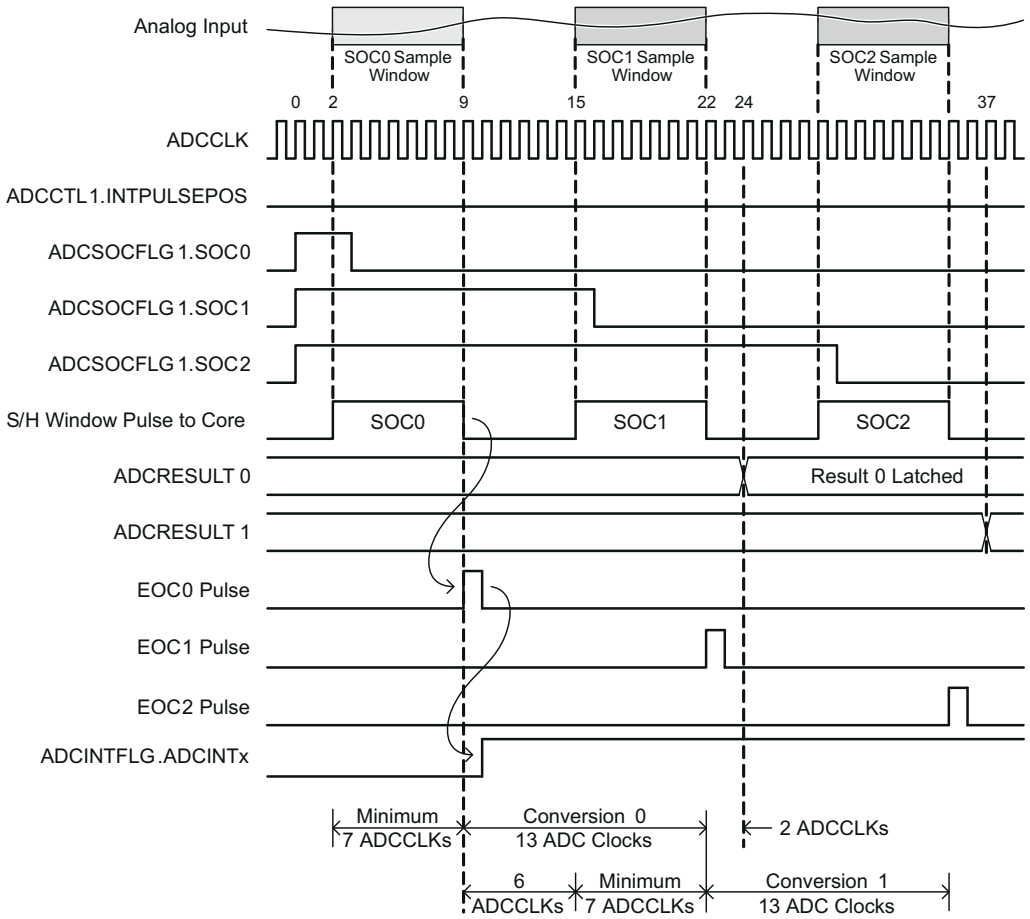


Figure 8-9. Timing Example For Sequential Mode / Early Interrupt Pulse

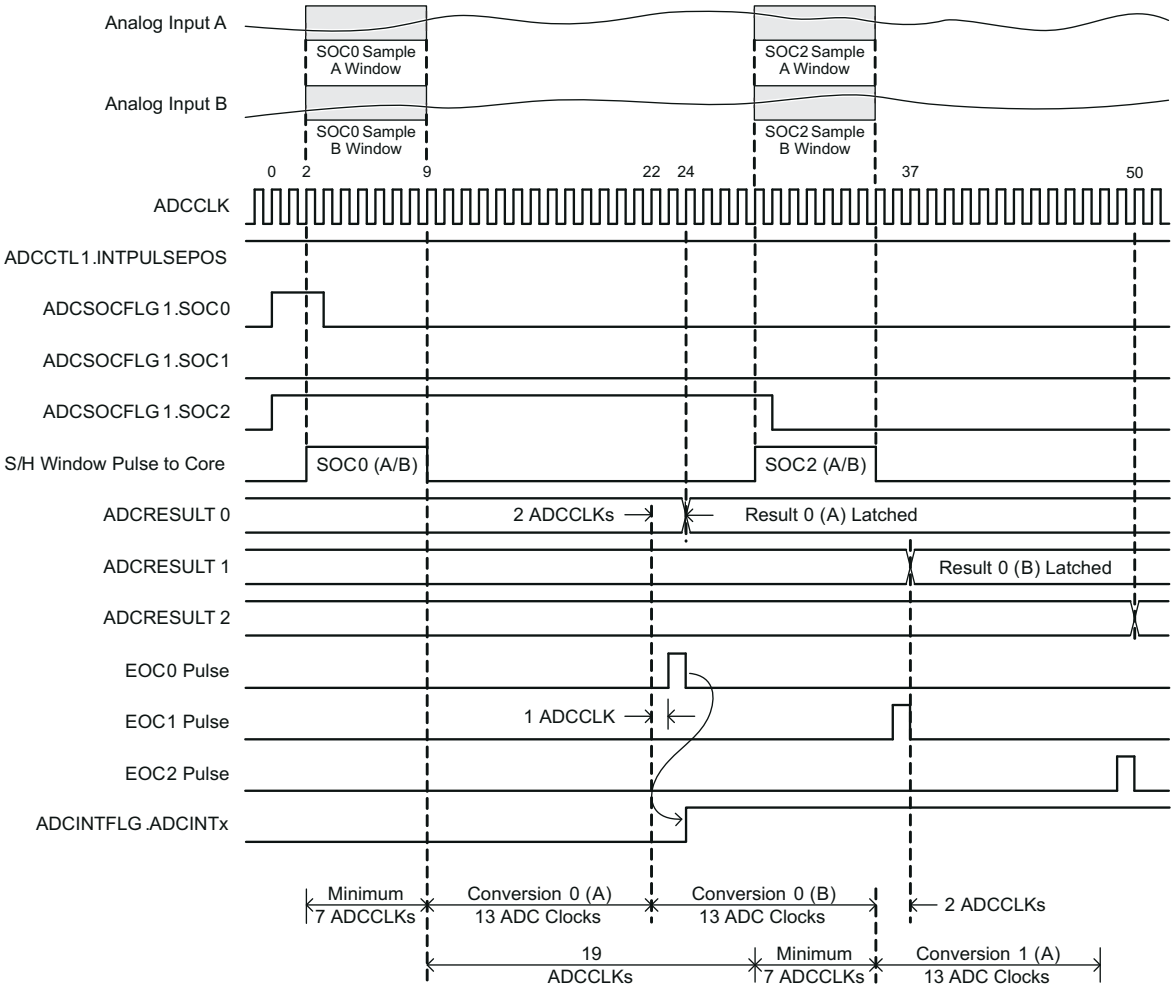


Figure 8-10. Timing Example For Simultaneous Mode / Late Interrupt Pulse

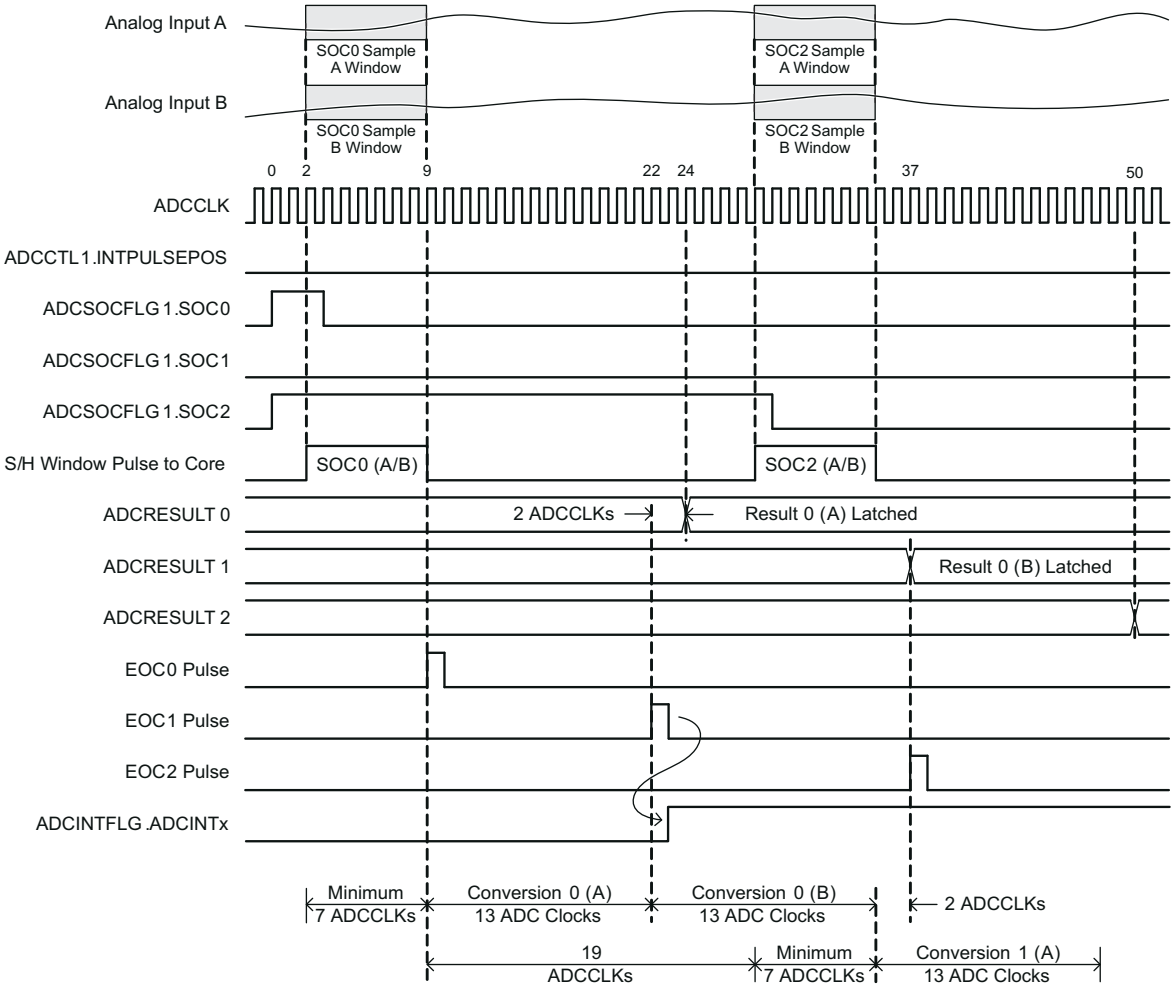


Figure 8-11. Timing Example For Simultaneous Mode / Early Interrupt Pulse

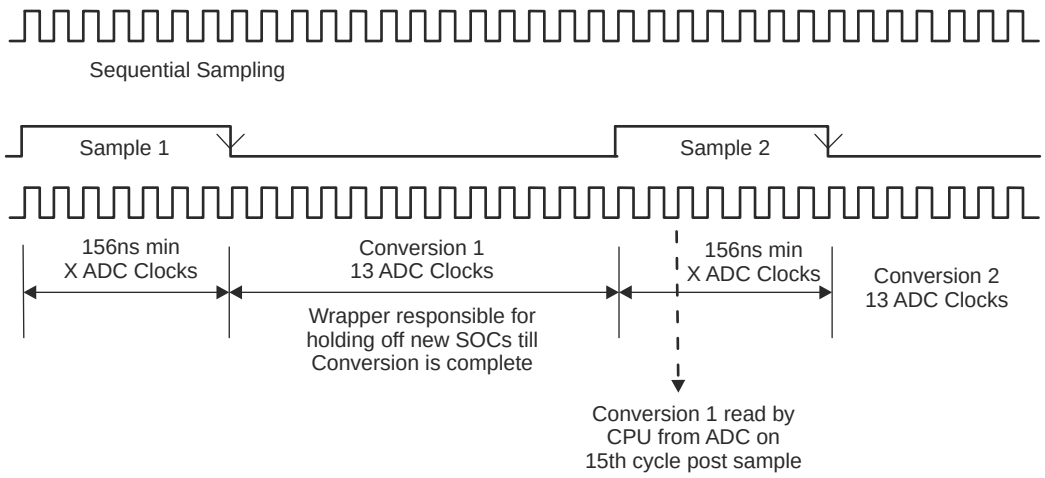


Figure 8-12. Timing Example for NONOVERLAP Mode

Note

The NONOVERLAP bit in the ADCCTL2 register, when enabled, removes the overlap of sampling and conversion stages.

8.12 Internal Temperature Sensor

The internal temperature sensor measures the junction temperature of the device. The sensor output can be sampled with the ADC on channel A5 using a switch controlled by the ADCCTL1.TEMPCONV bit. The switch allows A5 to be used both as an external ADC input pin and the temperature sensor access point. When sampling the temperature sensor, the external circuitry on ADCINA5 has no affect on the sample. Refer to Section 8.13.1 for information about switching between the external ADCINA5 input pin and the internal temperature sensor.

8.12.1 Transfer Function

The temperature sensor output and the resulting ADC values increase with increasing junction temperature. The offset is defined as the 0 °C LSB crossing as illustrated in Figure 8-13. This information can be used to convert the ADC sensor sample into a temperature unit.

The transfer function to determine a temperature is defined as:

Temperature = (*sensor* - Offset) * Slope

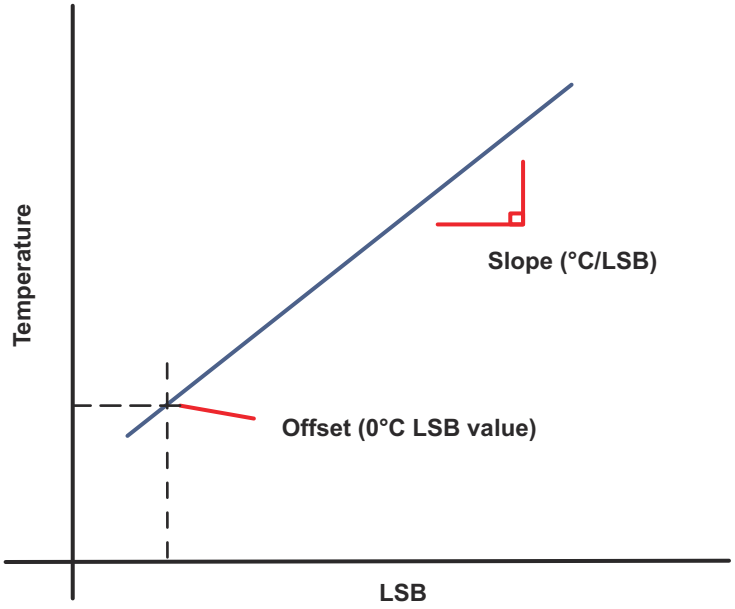


Figure 8-13. Temperature Sensor Transfer Function

Refer to the electrical characteristics section in the *TMS320F2803x Real-Time Microcontrollers Data Manual* for the slope and offset, or use the stored slope and offset calibrated per device in the factory that can be extracted by a function at the following locations.

For F2803x:

- 0x3D7E82 - Slope (°C / LSB, fixed-point Q15 format)
- 0x3D7E85 - Offset (0 °C LSB value)

The values listed are assuming a 3.3-V full-scale range. Using the internal reference mode automatically achieves this fixed range, but if using the external mode, the temperature sensor values must be adjusted accordingly to the external reference voltages.

Example

The header files include an example project to easily sample the temperature sensor and convert the result into two different temperature units. There are three steps to using the temperature sensor:

1. Configure the ADC to sample the temperature sensor
2. Sample the temperature sensor
3. Convert the result into a temperature unit, such as °C.

Here is an example of these steps:

```
// Configure the ADC to sample the temperature sensor EALLOW;
AdcRegs.ADCCTL1.bit.TEMPCONV =  1;  //Connect A5 - temp sensor
AdcRegs.ADCSOC0CTL.bit.CHSEL =  5;  //Set SOC0 to sample A5
AdcRegs.ADCSOC1CTL.bit.CHSEL =  5;  //Set SOC1 to sample A5
AdcRegs.ADCSOC0CTL.bit.ACQPS =  6;  //Set SOC0 ACQPS to 7 ADCCLK
AdcRegs.ADCSOC1CTL.bit.ACQPS =  6;  //Set SOC1 ACQPS to 7 ADCCLK
AdcRegs.INTSEL1N2.bit.INT1SEL = 1;  //Connect ADCINT1 to EOC1
AdcRegs.INTSEL1N2.bit.INT1E =   1;  //Enable ADCINT1
EDIS;

// Sample the temperature sensor
AdcRegs.ADCSOCFRC1.all = 0x03;              //Sample temp sensor
while(AdcRegs.ADCINTFLG.bit.ADCINT1 == 0){} //Wait for ADCINT1
AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;       //Clear ADCINT1
sensorSample = AdcResult.ADCRESULT1;        //Get temp sensor sample result

//Convert raw temperature sensor output to a temperature (degC)
DegreesC = (sensorSample - TempSensorOffset) * TempSensorSlope;

For the F2806x, call the below factory stored slope and offset get functions:
//Slope of temperature sensor (deg. C / ADC code, fixed pt Q15 format)
#define getTempSlope() (*(int (*)(void))0x3D7E82)()

//ADC code corresponding to temperature sensor output at 0-degreesC
#define getTempOffset() (*(int (*)(void))0x3D7E85)()
```

8.13 ADC Registers

This section contains the ADC registers and bit definitions with the registers grouped by function. All of the ADC registers are located in Peripheral Frame 2 except the ADCRESULTx registers, which are found in Peripheral Frame 0. See the device data sheet for specific addresses.

Table 8-3. ADC Configuration and Control Registers (AdcRegs and AdcResult)

Register	Address Offset	Size (x16)	Description	Bit Description
ADCCTL1	0x00	1	Control 1 Register(1)	Section 8.13.1
ADCCTL2	0x01	1	Control 2 Register(1)	Section 8.13.2
ADCINTFLG	0x04	1	Interrupt Flag Register	Section 8.13.3.1
ADCINTFLGCLR	0x05	1	Interrupt Flag Clear Register	Section 8.13.3.2
ADCINTOVF	0x06	1	Interrupt Overflow Register	Section 8.13.3.3
ADCINTOVFCLR	0x07	1	Interrupt Overflow Clear Register	Section 8.13.3.4
INTSEL1N2	0x08	1	Interrupt 1 and 2 Selection Register(1)	Section 8.13.3.5
INTSEL3N4	0x09	1	Interrupt 3 and 4 Selection Register(1)	Section 8.13.3.5
INTSEL5N6	0x0A	1	Interrupt 5 and 6 Selection Register(1)	Section 8.13.3.5
INTSEL7N8	0x0B	1	Interrupt 7 and 8 Selection Register(1)	Section 8.13.3.5
INTSEL9N10	0x0C	1	Interrupt 9 Selection Register (reserved Interrupt 10 Selection)(1)	Section 8.13.3.5
SOCPRICTL	0x10	1	Start of Conversion Priority Control Register(1)	Section 8.13.4
ADCSAMPLEMODE	0x12	1	Sampling Mode Register(1)	Section 8.13.5.1
ADCINTSOCSEL1	0x14	1	Interrupt SOC Selection 1 Register (for 8 channels)(1)	Section 8.13.5.2
ADCINTSOCSEL2	0x15	1	Interrupt SOC Selection 2 Register (for 8 channels)(1)	Section 8.13.5.3
ADCSOCFLG1	0x18	1	SOC Flag 1 Register (for 16 channels)	Section 8.13.5.4
ADCSOCFRC1	0x1A	1	SOC Force 1 Register (for 16 channels)	Section 8.13.5.5
ADCSOCOVF1	0x1C	1	SOC Overflow 1 Register (for 16 channels)	Section 8.13.5.6
ADCSOCOVFCLR1	0x1E	1	SOC Overflow Clear 1 Register (for 16 channels)	Section 8.13.5.7
ADCSOC0CTL - ADCSOC15CTL	0x20 - 0x2F	1	SOC0 Control Register to SOC15 Control Register(1)	Section 8.13.5.8
ADCREFTRIM	0x40	1	Reference Trim Register(1)	Section 8.13.6.1
ADCOFFTRIM	0x41	1	Offset Trim Register(1)	Section 8.13.6.2
COMPHYSTCTL	0x4C	1	Comp Hysteresis Control Register(1)	Section 8.13.7
ADCREV – reserved	0x4F	1	Revision Register	Section 8.13.8
ADCRESULT0 - ADCRESULT15	0x00 - 0x0F(2)	1	ADC Result 0 Register to ADC Result 15 Register	Section 8.13.9

(1) This register is EALLOW protected.
(2) The base address of the ADCRESULT registers differs from the base address of the other ADC registers. In the header files, the ADCRESULT registers are found in the AdcResult register file, not in the AdcRegs.

8.13.1 ADC Control Register 1 (ADCCTL1)

Figure 8-14. ADC Control Register 1 (ADCCTL1)

15	14	13	12				8
RESET	ADCENABLE	ADCBSY	ADCBSYCHN				
R-0/W-1	R-1	R-0	R-0				
7	6	5	4	3	2	1	0
ADCPWN	ADCBGPWD	ADCREFPWD	Reserved	ADCREFSEL	INTPULSEPOS	VREFLO CONV	TEMPCONV
R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; R-0/W-1 = always read as 0, write 1 to set; -*n* = value after reset

Table 8-4. ADC Control Register 1 (ADCCTL1) Field Descriptions

Bit	Field	Value	Description [(1)]
15	RESET		ADC module software reset. This bit causes a master reset on the entire ADC module. All register bits and state machines are reset to the initial state as occurs when the device reset pin is pulled low (or after a power-on reset). This is a one-time-effect bit, meaning this bit is self-cleared immediately after it is set to 1. Read of this bit always returns a 0. Also, the reset of ADC has a latency of two clock cycles (that is, other ADC control register bits should not be modified until two clock cycles after the instruction that resets the ADC.
		0	no effect
		1	Resets entire ADC module (bit is then set back to 0 by ADC logic)
			Note: The ADC module is reset during a system reset. If an ADC module reset is desired at any other time, you can do so by writing a 1 to this bit. After two clock cycles, you can then write the appropriate values to the ADCCTL1 register bits. Assembly code:
			MOV ADCCTL1, #1xxxxxxxxxxxxxxb ; Resets the ADC (RESET = 1)
			NOP ; Delay two cycles
			NOP
			MOV ADCCTL1, #0xxxxxxxxxxxxxxb ; Set to user-desired value
			Note: The second MOV is not required if the default configuration is sufficient. **Note:** If the system is reset or the ADC module is reset using Bit 15 (RESET) from the ADC Control Register 1, the Device_cal() routine must be repeated .
14	ADCENABLE		ADC Enable
		0	ADC disabled (**does not** power down ADC)
		1	ADC Enabled. Must set before an ADC conversion (recommend that it be set directly after setting ADC power-up bits
13	ADCBSY		ADC Busy
			Set when ADC SOC is generated, cleared per below. Used by the ADC state machine to determine if ADC is available to sample.
			Sequential Mode: Cleared 4 ADC clocks after negative edge of S+H pulse
			Simultaneous Mode: Cleared 14 ADC clocks after negative edge of S+H pulse
		0	ADC is available to sample next channel
		1	ADC is busy and cannot sample another channel

Table 8-4. ADC Control Register 1 (ADCCTL1) Field Descriptions (continued)

Bit	Field	Value	Description [1]
12-8	ADCBSYCHN		Set when ADC SOC for current SOC is generated
			When ADCBSY = 0: holds the value of the last converted SOC
			When ADCBSY = 1: reflects SOC currently being processed
		00h	SOC0 is currently processing or was last SOC converted
		01h	SOC1 is currently processing or was last SOC converted
		02h	SOC2 is currently processing or was last SOC converted
		03h	SOC3 is currently processing or was last SOC converted
		04h	SOC4 is currently processing or was last SOC converted
		05h	SOC5 is currently processing or was last SOC converted
		06h	SOC6 is currently processing or was last SOC converted
		07h	SOC7 is currently processing or was last SOC converted
		08h	SOC8 is currently processing or was last SOC converted
		09h	SOC9 is currently processing or was last SOC converted
		0Ah	SOC10 is currently processing or was last SOC converted
		0Bh	SOC11 is currently processing or was last SOC converted
		0Ch	SOC12 is currently processing or was last SOC converted
		0Dh	SOC13 is currently processing or was last SOC converted
		0Eh	SOC14 is currently processing or was last SOC converted
		0Fh	ADCINB15 is currently processing or was last SOC converted
		1xh	Invalid value
7	ADCPWDN		ADC power down (active low).
			This bit controls the power up and power down of all the analog circuitry inside the analog core except the bandgap and reference circuitry
		0	All analog circuitry inside the core except the bandgap and reference circuitry is powered down
		1	The analog circuitry inside the core is powered up
6	ADCBGPWD		Bandgap circuit power down (active low)
		0	Bandgap circuitry is powered down
		1	Bandgap buffer's circuitry inside core is powered up
5	ADCREFPWD		Reference buffers circuit power down (active low)
		0	Reference buffers circuitry is powered down
		1	Reference buffers circuitry inside the core is powered up
4	Reserved	0	Reads return a zero; Writes have no effect.
3	ADCREFSEL		Internal or external reference select
		0	Internal Bandgap used for reference generation
		1	External VREFHI or VREFLO pins used for reference generation. On some devices the VREFHI pin is shared with ADCINA0. In this case ADCINA0 will not be available for conversions in this mode. On some devices the VREFLO pin is shared with VSSA. In this case the VREFLO voltage cannot be varied.
2	INTPULSEPOS		INT Pulse Generation control
		0	INT pulse generation occurs when ADC begins conversion (negative edge of sample pulse of the sampled signal)
		1	INT pulse generation occurs 1 cycle prior to ADC result latching into its result register
1	VREFLOCONV		VREFLO Convert.
			When enabled, internally connects VREFLO to the ADC channel B5 and disconnects the ADCINB5 pin from the ADC. Whether the pin ADCINB5 exists on the device does not affect this function. Any external circuitry on the ADCINB5 pin is unaffected by this mode.
		0	ADCINB5 is passed to the ADC module as normal, VREFLO connection to ADCINB5 is disabled
		1	VREFLO internally connected to the ADC for sampling

Table 8-4. ADC Control Register 1 (ADCCTL1) Field Descriptions (continued)

Bit	Field	Value	Description (1)
0	TEMPCONV		Temperature sensor convert. When enabled internally connects the internal temperature sensor to ADC channel A5 and disconnects the ADCINA5 pin from the ADC. Whether the pin ADCINA5 exists on the device does not affect this function. Any external circuitry on the ADCINA5 pin is unaffected by this mode
		0	ADCINA5 is passed to the ADC module as normal, internal temperature sensor connection to ADCINA5 is disabled.
		1	Temperature sensor is internally connected to the ADC for sampling

(1) This register is EALLOW protected.

8.13.2 ADC Control Register 2 (ADCCTL2)

Figure 8-15. ADC Control Register 2 (ADCCTL2)

15	2	1	0
Reserved		ADCNONOVERLAP	CLKDIV2EN
R-0		R/W-0	R/W-0

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

Table 8-5. ADC Control Register 2 (ADCCTL2) Field Descriptions

Bit	Field	Value	Description (1)
15-2	Reserved	0	Reads return a zero; writes have no effect.
1	ADCNONOVERLAP		ADCNONOVERLAP control bit
		0	Overlap of sample and conversion is allowed
		1	Overlap of sample is not allowed
0	CLKDIV2EN		ADC Clock Prescaler.
		0	ADCCLK = SYSCLK
		1	ADCCLK = SYSCLK / 2

(1) This register is EALLOW protected.

8.13.3 ADC Interrupt Registers

8.13.3.1 ADC Interrupt Flag Register (ADCINTFLG)

Figure 8-16. ADC Interrupt Flag Register (ADCINTFLG)

15						9	8
Reserved							ADCINT9
R-0							R-0

7	6	5	4	3	2	1	0
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 8-6. ADC Interrupt Flag Register (ADCINTFLG) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reads return a zero; Writes have no effect.
8-0	ADCINTx (x = 9 to 1)		ADC Interrupt Flag Bits: Reading this bit indicates if an ADCINT pulse was generated
		0	No ADC interrupt pulse generated
		1	ADC Interrupt pulse generated
			If the ADC interrupt is placed in continuous mode (INTSELxNy register) then further interrupt pulses are generated whenever a selected EOC event occurs even if the flag bit is set.
			If the continuous mode is not enabled, then no further interrupt pulses are generated until the user clears this flag bit using the ADCINTFLGCLR register. The ADCINTOVF flag will be set if EOC events are generated while the ADCINTFLG flag is set. Both ADCINTFLG and ADCINTOVF flags must be cleared before normal interrupt operation can resume in non-continuous mode.

8.13.3.2 ADC Interrupt Flag Clear Register (ADCINTFLGCLR)

Figure 8-17. ADC Interrupt Flag Clear Register (ADCINTFLGCLR)

15							9	8
Reserved								ADCINT9
R-0								R/W-0
7	6	5	4	3	2	1		0
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2		ADCINT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0

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

Table 8-7. ADC Interrupt Flag Clear Register (ADCINTFLGCLR) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reads return a zero; Writes have no effect.
8-0	ADCINTx (x = 9 to 1)		ADC interrupt Flag Clear Bit
		0	No action.
		1	Clears respective flag bit in the ADCINTFLG register. **Boundary condition for clearing or setting flag bits:** If hardware tries to set a flag bit while software tries to clear the flag bit in the same cycle, the following will take place: 1. SW has priority, and will clear the flag 2. HW set will be discarded, no signal will propagate to the PIE form the latch 3. Overflow flag or condition will be generated

8.13.3.3 ADC Interrupt Overflow Register (ADCINTOVF)

Figure 8-18. ADC Interrupt Overflow Register (ADCINTOVF)

15							9	8
Reserved								ADCINT9
R-0								R-0

7	6	5	4	3	2	1	0
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 8-8. ADC Interrupt Overflow Register (ADCINTOVF) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved
8-0	ADCINTx (x = 9 to 1)		ADC Interrupt Overflow Bits. Indicates if an overflow occurred when generating ADCINT pulses. If the respective ADCINTFLG bit is set and a selected additional EOC trigger is generated, then an overflow condition occurs.
		0	No ADC interrupt overflow event detected.
		1	ADC Interrupt overflow event detected.
			The overflow bit does not care about the continuous mode bit state. An overflow condition is generated irrespective of this mode selection. Both ADCINTFLG and ADCINTOVF flags must be cleared before normal interrupt operation can resume in non-continuous mode.

8.13.3.4 ADC Interrupt Overflow Clear Register (ADCINTOVFCLR)

Figure 8-19. ADC Interrupt Overflow Clear Register (ADCINTOVFCLR)

15							9	8
Reserved								ADCINT9
R-0								R-0/W-1

7	6	5	4	3	2	1	0
ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1
R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1	R-0/W-1

LEGEND: R/W = Read/Write; R = Read only; R-0/W-1 =always read 0, write 1 to set; -*n* = value after reset

Table 8-9. ADC Interrupt Overflow Clear Register (ADCINTOVFCLR) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reads return a zero; Writes have no effect.
8-0	ADCINTx (x = 9 to 1)		ADC Interrupt Overflow Clear Bits.
		0	No action.
		1	Clears the respective overflow bit in the ADCINTOVF register. If software tries to set this bit on the same clock cycle that hardware tries to set the overflow bit in the ADCINTOVF register, then hardware has priority and the ADCINTOVF bit will be set.

8.13.3.5 Interrupt Select Registers (INTSELxNy)

Figure 8-20. Interrupt Select 1 and 2 Register (INTSEL1N2)

15	14	13	12	8
Reserved	INT2CONT	INT2E	INT2SEL	
R-0	R/W-0	R/W-0	R/W-0	

7	6	5	4	0
Reserved	INT1CONT	INT1E	INT1SEL	
R-0	R/W-0	R/W-0	R/W-0	

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

Figure 8-21. Interrupt Select 3 and 4 Register (INTSEL3N4)

15	14	13	12	8
Reserved	INT4CONT	INT4E	INT4SEL	
R-0	R/W-0	R/W-0	R/W-0	

7	6	5	4	0
Reserved	INT3CONT	INT3E	INT3SEL	
R-0	R/W-0	R/W-0	R/W-0	

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

Figure 8-22. Interrupt Select 5 and 6 Register (INTSEL5N6)

15	14	13	12	8
Reserved	INT6CONT	INT6E	INT6SEL	
R-0	R/W-0	R/W-0	R/W-0	

7	6	5	4	0
Reserved	INT5CONT	INT5E	INT5SEL	
R-0	R/W-0	R/W-0	R/W-0	

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

Figure 8-23. Interrupt Select 7 and 8 Register (INTSEL7N8)

15	14	13	12	8
Reserved	INT8CONT	INT8E	INT8SEL	
R-0	R/W-0	R/W-0	R/W-0	

7	6	5	4	0
Reserved	INT7CONT	INT7E	INT7SEL	
R-0	R/W-0	R/W-0	R/W-0	

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

Figure 8-24. Interrupt Select 9 and 10 Register (INTSEL9N10)

15	8
Reserved	
R-0	

7	6	5	4	0
Reserved	INT9CONT	INT9E	INT9SEL	
R-0	R/W-0	R/W-0	R/W-0	

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

Table 8-10. Interrupt Select Register (INTSELxNy) Field Descriptions

Bit	Field	Value	Description (1)
15	Reserved	0	Reserved
14	INTyCONT		ADCINTy Continuous Mode Enable
		0	No further ADCINTy pulses are generated until ADCINTy flag (in ADCINTFLG register) is cleared by user.
		1	ADCINTy pulses are generated whenever an EOC pulse is generated irrespective if the flag bit is cleared or not.
13	INTyE		ADCINTy Interrupt Enable
		0	ADCINTy is disabled.
		1	ADCINTy is enabled.

Table 8-10. Interrupt Select Register (INTSELxNy) Field Descriptions (continued)

Bit	Field	Value	Description (1)
12-8	INTySEL		ADCINTy EOC Source Select
		00h	EOC0 is trigger for ADCINTy
		01h	EOC1 is trigger for ADCINTy
		02h	EOC2 is trigger for ADCINTy
		03h	EOC3 is trigger for ADCINTy
		04h	EOC4 is trigger for ADCINTy
		05h	EOC5 is trigger for ADCINTy
		06h	EOC6 is trigger for ADCINTy
		07h	EOC7 is trigger for ADCINTy
		08h	EOC8 is trigger for ADCINTy
		09h	EOC9 is trigger for ADCINTy
		0Ah	EOC10 is trigger for ADCINTy
		0Bh	EOC11 is trigger for ADCINTy
		0Ch	EOC12 is trigger for ADCINTy
		0Dh	EOC13 is trigger for ADCINTy
		0Eh	EOC14 is trigger for ADCINTy
		0Fh	EOC15 is trigger for ADCINTy
		1xh	Invalid value.
7	Reserved	0	Reads return a zero; Writes have no effect.
6	INTxCONT		ADCINTx Continuous Mode Enable.
		0	No further ADCINTx pulses are generated until ADCINTx flag (in ADCINTFLG register) is cleared by user.
		1	ADCINTx pulses are generated whenever an EOC pulse is generated irrespective if the flag bit is cleared or not.
5	INTxE		ADCINTx Interrupt Enable
		0	ADCINTx is disabled.
		1	ADCINTx is enabled .
4-0	INTxSEL		ADCINTx EOC Source Select
		00h	EOC0 is trigger for ADCINTx
		01h	EOC1 is trigger for ADCINTx
		02h	EOC2 is trigger for IADCNTx
		03h	EOC3 is trigger for ADCINTx
		04h	EOC4 is trigger for ADCINTx
		05h	EOC5 is trigger for ADCINTx
		06h	EOC6 is trigger for ADCINTx
		07h	EOC7 is trigger for ADCINTx
		08h	EOC8 is trigger for ADCINTx
		09h	EOC9 is trigger for ADCINTx
		0Ah	EOC10 is trigger for ADCINTx
		0Bh	EOC11 is trigger for ADCINTx
		0Ch	EOC12 is trigger for ADCINTx
		.0Dh	EOC13 is trigger for ADCINTx
		0Eh	EOC14 is trigger for ADCINTx
		0Fh	EOC15 is trigger for ADCINTx
		1xh	Invalid value.

(1) This register is EALLOW protected.

8.13.4 ADC Start of Conversion Priority Control Register (SOCPRICTL)

Figure 8-25. ADC Start of Conversion Priority Control Register (SOCPRICTL)

15	14	11	10	5	4	0
ONESHOT	Reserved		RRPOINTER		SOCPRIORITY	
R/W-0	R-0		R-20h		R/W-0	

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

Table 8-11. SOCPRICTL Register Field Descriptions

Bit	Field	Value	Description [(1)]
15	ONESHOT	0	One shot mode disabled
		1	One shot mode enabled
14-11	Reserved		Reads return a zero; Writes have no effect.
10-5	RRPOINTER		Round Robin Pointer. Holds the value of the last converted round robin SOCx to be used by the round robin scheme to determine order of conversions.
		00h	SOC0 was last round robin SOC to convert. SOC1 is highest round robin priority.
		01h	SOC1 was last round robin SOC to convert. SOC2 is highest round robin priority.
		02h	SOC2 was last round robin SOC to convert. SOC3 is highest round robin priority.
		03h	SOC3 was last round robin SOC to convert. SOC4 is highest round robin priority.
		04h	SOC4 was last round robin SOC to convert. SOC5 is highest round robin priority.
		05h	SOC5 was last round robin SOC to convert. SOC6 is highest round robin priority.
		06h	SOC6 was last round robin SOC to convert. SOC7 is highest round robin priority.
		07h	SOC7 was last round robin SOC to convert. SOC8 is highest round robin priority.
		08h	SOC8 was last round robin SOC to convert. SOC9 is highest round robin priority.
		09h	SOC9 was last round robin SOC to convert. SOC10 is highest round robin priority.
		0Ah	SOC10 was last round robin SOC to convert. SOC11 is highest round robin priority.
		0Bh	SOC11 was last round robin SOC to convert. SOC12 is highest round robin priority.
		0Ch	SOC12 was last round robin SOC to convert. SOC13 is highest round robin priority.
		0Dh	SOC13 was last round robin SOC to convert. SOC14 is highest round robin priority.
		0Eh	SOC14 was last round robin SOC to convert. SOC15 is highest round robin priority.
		0Fh	SOC15 was last round robin SOC to convert. SOC0 is highest round robin priority.
		1xh	Invalid value
		20h	Reset value to indicate no SOC has been converted. SOC0 is highest round robin priority. Set to this value when the device is reset, when the ADCCTL1.RESET bit is set, or when the SOCPRICTL register is written. In the latter case, if a conversion is currently in progress, it will complete and then the new priority will take effect.
		Others	Invalid selection.

Table 8-11. SOCPRICTL Register Field Descriptions (continued)

Bit	Field	Value	Description (1)
4-0	SOCPRIORITY		SOC Priority.
			Determines the cutoff point for priority mode and round robin arbitration for SOCx
		00h	SOC priority is handled in round robin mode for all channels.
		01h	SOC0 is high priority, rest of channels are in round robin mode.
		02h	SOC0-SOC1 are high priority, SOC2-SOC15 are in round robin mode.
		03h	SOC0-SOC2 are high priority, SOC3-SOC15 are in round robin mode.
		04h	SOC0-SOC3 are high priority, SOC4-SOC15 are in round robin mode.
		05h	SOC0-SOC4 are high priority, SOC5-SOC15 are in round robin mode.
		06h	SOC0-SOC5 are high priority, SOC6-SOC15 are in round robin mode.
		07h	SOC0-SOC6 are high priority, SOC7-SOC15 are in round robin mode.
		08h	SOC0-SOC7 are high priority, SOC8-SOC15 are in round robin mode.
		09h	SOC0-SOC8 are high priority, SOC9-SOC15 are in round robin mode.
		0Ah	SOC0-SOC9 are high priority, SOC10-SOC15 are in round robin mode.
		0Bh	SOC0-SOC10 are high priority, SOC11-SOC15 are in round robin mode.
		0Ch	SOC0-SOC11 are high priority, SOC12-SOC15 are in round robin mode.
		0Dh	SOC0-SOC12 are high priority, SOC13-SOC15 are in round robin mode.
		0Eh	SOC0-SOC13 are high priority, SOC14-SOC15 are in round robin mode.
		0Fh	SOC0-SOC14 are high priority, SOC15 is in round robin mode.
		10h	All SOCs are in high priority mode, arbitrated by SOC number
		Others	Invalid selection.

(1) This register is EALLOW protected.

8.13.5 ADC SOC Registers

8.13.5.1 ADC Sample Mode Register (ADCSAMPLEMODE)

Figure 8-26. ADC Sample Mode Register (ADCSAMPLEMODE)

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
SIMULEN14	SIMULEN12	SIMULEN10	SIMULEN8	SIMULEN6	SIMULEN4	SIMULEN2	SIMULEN0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 8-12. ADC Sample Mode Register (ADCSAMPLEMODE) Field Descriptions

Bit	Field	Value	Description [(1)]
15:8	Reserved	0	Reserved
7	SIMULEN14		Simultaneous sampling enable for SOC14/SOC15. Couples SOC14 and SOC15 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC14 or SOC15.
		0	Single sample mode set for SOC14 and SOC15. All bits of CHSEL field define channel to be converted. EOC14 associated with SOC14. EOC15 associated with SOC15. SOC14's result placed in ADCRESULT14 register. SOC15's result placed in ADCRESULT15.
		1	Simultaneous sample for SOC14 and SOC15. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC14 and EOC15 associated with SOC14 and SOC15 pair. SOC14's and SOC15's results will be placed in ADCRESULT14 and ADCRESULT15 registers, respectively.
6	SIMULEN12		Simultaneous sampling enable for SOC12/SOC13. Couples SOC12 and SOC13 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC12 or SOC13.
		0	Single sample mode set for SOC12 and SOC13. All bits of CHSEL field define channel to be converted. EOC12 associated with SOC12. EOC13 associated with SOC13. SOC12's result placed in ADCRESULT12 register. SOC13's result placed in ADCRESULT13.
		1	Simultaneous sample for SOC12 and SOC13. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC12 and EOC13 associated with SOC12 and SOC13 pair. SOC12's and SOC13's results will be placed in ADCRESULT12 and ADCRESULT13 registers, respectively.
5	SIMULEN10		Simultaneous sampling enable for SOC10/SOC11. Couples SOC10 and SOC11 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC10 or SOC11.
		0	Single sample mode set for SOC10 and SOC11. All bits of CHSEL field define channel to be converted. EOC10 associated with SOC10. EOC11 associated with SOC11. SOC10's result placed in ADCRESULT10 register. SOC11's result placed in ADCRESULT11.
		1	Simultaneous sample for SOC10 and SOC11. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC10 and EOC11 associated with SOC10 and SOC11 pair. SOC10's and SOC11's results will be placed in ADCRESULT10 and ADCRESULT11 registers, respectively.
4	SIMULEN8		Simultaneous sampling enable for SOC8/SOC9. Couples SOC8 and SOC9 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC8 or SOC9.
		0	Single sample mode set for SOC8 and SOC9. All bits of CHSEL field define channel to be converted. EOC8 associated with SOC8. EOC9 associated with SOC9. SOC8's result placed in ADCRESULT8 register. SOC9's result placed in ADCRESULT9.
		1	Simultaneous sample for SOC8 and SOC9. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC8 and EOC9 associated with SOC8 and SOC9 pair. SOC8's and SOC9's results will be placed in ADCRESULT8 and ADCRESULT9 registers, respectively.

Table 8-12. ADC Sample Mode Register (ADCSAMPLEMODE) Field Descriptions (continued)

Bit	Field	Value	Description (1)
3	SIMULEN6		Simultaneous sampling enable for SOC6/SOC7. Couples SOC6 and SOC7 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC6 or SOC7.
		0	Single sample mode set for SOC6 and SOC7. All bits of CHSEL field define channel to be converted. EOC6 associated with SOC6. EOC7 associated with SOC7. SOC6's result placed in ADCRESULT6 register. SOC7's result placed in ADCRESULT7.
		1	Simultaneous sample for SOC6 and SOC7. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC6 and EOC7 associated with SOC6 and SOC7 pair. SOC6's and SOC7's results will be placed in ADCRESULT6 and ADCRESULT7 registers, respectively.
2	SIMULEN4		Simultaneous sampling enable for SOC4/SOC5. Couples SOC4 and SOC5 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC4 or SOC5.
		0	Single sample mode set for SOC4 and SOC5. All bits of CHSEL field define channel to be converted. EOC4 associated with SOC4. EOC5 associated with SOC5. SOC4's result placed in ADCRESULT4 register. SOC5's result placed in ADCRESULT5.
		1	Simultaneous sample for SOC4 and SOC5. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC4 and EOC5 associated with SOC4 and SOC5 pair. SOC4's and SOC5's results will be placed in ADCRESULT4 and ADCRESULT5 registers, respectively.
1	SIMULEN2		Simultaneous sampling enable for SOC2/SOC3. Couples SOC2 and SOC3 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC2 or SOC3.
		0	Single sample mode set for SOC2 and SOC3. All bits of CHSEL field define channel to be converted. EOC2 associated with SOC2. EOC3 associated with SOC3. SOC2's result placed in ADCRESULT2 register. SOC3's result placed in ADCRESULT3.
		1	Simultaneous sample for SOC2 and SOC3. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC2 and EOC3 associated with SOC2 and SOC3 pair. SOC2's and SOC3's results will be placed in ADCRESULT2 and ADCRESULT3 registers, respectively.
0	SIMULEN0		Simultaneous sampling enable for SOC0/SOC1. Couples SOC0 and SOC1 in simultaneous sampling mode. See section 1.5 for details. This bit should not be set when the ADC is actively converting SOC0 or SOC1.
		0	Single sample mode set for SOC0 and SOC1. All bits of CHSEL field define channel to be converted. EOC0 associated with SOC0. EOC1 associated with SOC1. SOC0's result placed in ADCRESULT0 register. SOC1's result placed in ADCRESULT1.
		1	Simultaneous sample for SOC0 and SOC1. Lowest three bits of CHSEL field define the pair of channels to be converted. EOC0 and EOC1 associated with SOC0 and SOC1 pair. SOC0's and SOC1's results will be placed in ADCRESULT0 and ADCRESULT1 registers, respectively.

(1) This register is EALLOW protected.

8.13.5.2 ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1)

Figure 8-27. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SOC7		SOC6		SOC5		SOC4		SOC3		SOC2		SOC1		SOC0	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

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

Table 8-13. ADC Interrupt Trigger SOC Select 1 Register (ADCINTSOCSEL1) Register Field Descriptions

Bit	Field	Value	Description (1)
15--0	SOCx (x = 7 to 0)		SOCx ADC Interrupt Trigger Select. Select ADCINT to trigger SOCx. The ADCINT trigger is OR'ed with the trigger selected by the TRIGSEL field in the ADCSOCxCTL register, as well as the software force trigger signal from the ADCSOCFRC1 register.
		00	No ADCINT will trigger SOCx.
		01	ADCINT1 will trigger SOCx.
		10	ADCINT2 will trigger SOCx.
		11	Invalid selection.

(1) This register is EALLOW protected.

8.13.5.3 ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2)

Figure 8-28. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SOC15		SOC14		SOC13		SOC12		SOC11		SOC10		SOC9		SOC8	
R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0		R/W-0	

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

Table 8-14. ADC Interrupt Trigger SOC Select 2 Register (ADCINTSOCSEL2) Field Descriptions

Bit	Field	Value	Description (1)
15-0	SOCx (x = 15 to 8)		SOCx ADC Interrupt Trigger Select. Select ADCINT to trigger SOCx. The ADCINT trigger is OR'ed with the trigger selected by the TRIGSEL field in the ADCSOCxCTL register, as well as the software force trigger signal from the ADCSOCFRC1 register.
		00	No ADCINT will trigger SOCx.
		01	ADCINT1 will trigger SOCx.
		10	ADCINT2 will trigger SOCx.
		11	Invalid selection.

(1) This register is EALLOW protected.

8.13.5.4 ADC SOC Flag 1 Register (ADCSOCFLG1)

Figure 8-29. ADC SOC Flag 1 Register (ADCSOCFLG1)

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 8-15. ADC SOC Flag 1 Register (ADCSOCFLG1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		SOCx Start of Conversion Flag. Indicates the state of individual SOC conversions.
		0	No sample pending for SOCx.
		1	Trigger has been received and sample is pending for SOCx.
			The bit will be automatically cleared when the respective SOCx conversion is started. If contention exists where this bit receives both a request to set **and** a request to clear on the same cycle, regardless of the source of either, this bit will be set and the request to clear will be ignored. In this case the overflow bit in the ADCSOCOVF1 register will not be affected regardless of whether this bit was previously set or not.

8.13.5.5 ADC SOC Force 1 Register (ADCSOCFRC1)

Figure 8-30. ADC SOC Force 1 Register (ADCSOCFRC1)

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 8-16. ADC SOC Force 1 Register (ADCSOCFRC1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		SOCx Force Start of Conversion Flag. Writing a 1 will force to 1 the respective SOCx flag bit in the ADCSOCFLG1 register. This can be used to initiate a software initiated conversion. Writes of 0 are ignored.
		0	No action.
		1	Force SOCx flag bit to 1. This will cause a conversion to start once priority is given to SOCx. If software tries to set this bit on the same clock cycle that hardware tries to clear the SOCx bit in the ADCSOCFLG1 register, then software has priority and the ADCSOCFLG1 bit will be set. In this case the overflow bit in the ADCSOCOVF1 register will not be affected regardless of whether the ADCSOCFLG1 bit was previously set or not.

8.13.5.6 ADC SOC Overflow 1 Register (ADCSOCOVF1)

Figure 8-31. ADC SOC Overflow 1 Register (ADCSOCOVF1)

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 8-17. ADC SOC Overflow 1 Register (ADCSOCOVF1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		SOCx Start of Conversion Overflow Flag. Indicates an SOCx event was generated while an existing SOCx event was already pending.
		0	No SOCx event overflow
		1	SOCx event overflow
			An overflow condition does not stop SOCx events from being processed. It simply is an indication that a trigger was missed.

8.13.5.7 ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1)

Figure 8-32. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1)

15	14	13	12	11	10	9	8
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 8-18. ADC SOC Overflow Clear 1 Register (ADCSOCOVFCLR1) Field Descriptions

Bit	Field	Value	Description
15-0	SOCx (x = 15 to 0)		SOCx Clear Start of Conversion Overflow Flag. Writing a 1 will clear the respective SOCx overflow flag in the ADCSOCOVF1 register. Writes of 0 are ignored.
		0	No action.
		1	Clear SOCx overflow flag.
			If software tries to set this bit on the same clock cycle that hardware tries to set the overflow bit in the ADCSOCOVF1 register, then hardware has priority and the ADCSOCOVF1 bit will be set.

8.13.5.8 ADC SOC0-SOC15 Control Registers (ADCSOCxCTL)

Figure 8-33. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL)

15	11	10	9	6	5	0
TRIGSEL		Reserved	CHSEL		ACQPS	
R/W-0		R-0	R/W-0		R/W-0	

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

Table 8-19. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions

Bit	Field	Value	Description [(1)]
15-11	TRIGSEL		SOCx Trigger Source Select. Configures which trigger will set the respective SOCx flag in the ADCSOCFLG1 register to initiate a conversion to start once priority is given to SOCx. This setting can be overridden by the respective SOCx field in the ADCINTSOCSEL1 or ADCINTSOCSEL2 register.
		00h	ADCTRIG0 - Software only.
		01h	ADCTRIG1 - CPU Timer 0, TINT0n
		02h	ADCTRIG2 - CPU Timer 1, TINT1n
		03h	ADCTRIG3 - CPU Timer 2, TINT2n
		04h	ADCTRIG4 – XINT2, XINT2SOC
		05h	ADCTRIG5 – ePWM1, ADCSOCA
		06h	ADCTRIG6 – ePWM1, ADCSOCB
		07h	ADCTRIG7 – ePWM2, ADCSOCA
		08h	ADCTRIG8 – ePWM2, ADCSOCB
		09h	ADCTRIG9 – ePWM3, ADCSOCA
		0Ah	ADCTRIG10 – ePWM3, ADCSOCB
		0Bh	ADCTRIG11 – ePWM4, ADCSOCA
		0Ch	ADCTRIG12 – ePWM4, ADCSOCB
		0Dh	ADCTRIG13 – ePWM5, ADCSOCA
		0Eh	ADCTRIG14 – ePWM5, ADCSOCB
		0Fh	ADCTRIG15 – ePWM6, ADCSOCA
		10h	ADCTRIG16 – ePWM6, ADCSOCB
		11h	ADCTRIG17 - ePWM7, ADCSOCA
		12h	ADCTRIG18 - ePWM7, ADCSOCB
		Others	Invalid selection.
10	Reserved		Reads return a zero; Writes have no effect.

Table 8-19. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions (continued)

Bit	Field	Value	Description (1)
9-6	CHSEL		SOCx Channel Select. Selects the channel to be converted when SOCx is received by the ADC.
			Sequential Sampling Mode (SIMULENx = 0):
		0h	ADCINA0
		1h	ADCINA1
		2h	ADCINA2
		3h	ADCINA3
		4h	ADCINA4
		5h	ADCINA5
		6h	ADCINA6
		7h	ADCINA7
		8h	ADCINB0
		9h	ADCINB1
		Ah	ADCINB2
		Bh	ADCINB3
		Ch	ADCINB4
		Dh	ADCINB5
		Eh	ADCINB6
		Fh	ADCINB7
			Simultaneous Sampling Mode (SIMULENx = 1):
		0h	ADCINA0/ADCINB0 pair
		1h	ADCINA1/ADCINB1 pair
		2h	ADCINA2/ADCINB2 pair
		3h	ADCINA3/ADCINB3 pair
		4h	ADCINA4/ADCINB4 pair
		5h	ADCINA5/ADCINB5 pair
		6h	ADCINA6/ADCINB6 pair
		7h	ADCINA7/ADCINB7 pair
		8h	Invalid selection.
		9h	Invalid selection.
		Ah	Invalid selection.
		Bh	Invalid selection.
		Ch	Invalid selection.
		Dh	Invalid selection.
		Eh	Invalid selection.
		Fh	Invalid selection.

Table 8-19. ADC SOC0-SOC15 Control Registers (ADCSOCxCTL) Register Field Descriptions (continued)

Bit	Field	Value	Description (1)
5-0	ACQPS		SOCx Acquisition Prescale. Controls the sample and hold window for SOCx.
		00h	Invalid selection.
		01h	Invalid selection.
		02h	Invalid selection.
		03h	Invalid selection.
		04h	Invalid selection.
		05h	Invalid selection.
		06h	Sample window is 7 cycles long (6 + 1 clock cycles).
		07h	Sample window is 8 cycles long (7 + 1 clock cycles).
		08h	Sample window is 9 cycles long (8 + 1 clock cycles).
		09h	Sample window is 10 cycles long (9 + 1 clock cycles).
		...	...
		3Fh	Sample window is 64 cycles long (63 + 1 clock cycles).
Other invalid selections: 10h, 11h, 12h, 13h, 14h, 1Dh, 1Eh, 1Fh, 20h, 21h, 2Ah, 2Bh, 2Ch, 2Dh, 2Eh, 37h, 38h, 39h, 3Ah, 3Bh			

(1) This register is EALLOW protected.

8.13.6 ADC Calibration Registers

8.13.6.1 ADC Reference/Gain Trim Register (ADCREFTRIM)

Figure 8-34. ADC Reference/Gain Trim Register (ADCREFTRIM)

15	14	13	9	8	5	4	0
Reserved		EXTREF_FINE_TRIM		BG_COARSE_TRIM		BG_FINE_TRIM	
R-0		R/W-0		R/W-0		R/W-0	

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

Table 8-20. ADC Reference/Gain Trim Register (ADCREFTRIM) Field Descriptions

Bit	Field	Value	Description (1)
15-14	Reserved		Reads return a zero; Writes have no effect.
13-9	EXTREF_FINE_TRIM		ADC External reference Fine Trim. These bits should not be modified after device boot code loads them with the factory trim setting.
8-5	BG_COARSE_TRIM		ADC Internal Bandgap Fine Trim. These bits should not be modified after device boot code loads them with the factory trim setting.
4-0	BG_FINE_TRIM		ADC Internal Bandgap Coarse Trim. A maximum value of 30 is supported. These bits should not be modified after device boot code loads them with the factory trim setting.

(1) This register is EALLOW protected.

8.13.6.2 ADC Offset Trim Register (ADCOFFTRIM)

Figure 8-35. ADC Offset Trim Register (ADCOFFTRIM)

15	9	8	0
Reserved		OFFTRIM	
R-0		R/W-0	

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

Table 8-21. ADC Offset Trim Register (ADCOFFTRIM) Field Descriptions

Bit	Field	Value	Description (1)
15-9	Reserved		Reads return a zero; Writes have no effect.
8-0	OFFTRIM		ADC Offset Trim. 2's complement of ADC offset. Range is -256 to +255. These bits are loaded by device boot code with a factory trim setting. Modification of this default setting can be made to correct any board induced offset.

(1) This register is EALLOW protected.

8.13.7 Comparator Hysteresis Control Register (COMPHYSTCTL)

Figure 8-36. Comparator Hysteresis Control Register (COMPHYSTCTL)

15	12	11	10	7	6	5	2	1	0
Reserved		COMP3_HYST_DISABLE	Reserved		COMP2_HYST_DISABLE	Reserved		COMP1_HYST_DISABLE	Reserved
R-0		R/W--0	R-0		R/W--0	R-0		R/W--0	R-0

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

Table 8-22. Comparator Hysteresis Control Register (COMPHYSTCTL) Field Descriptions

Bit	Field	Value	Description (1)
15-12	Reserved		Reads return a zero; Writes have no effect.
11	COMP3_HYST_DISABLE		Comparator 3 Hysteresis disable
		0	Hysteresis enabled
		1	Hysteresis disabled
10-7	Reserved		Reserved
6	COMP2_HYST_DISABLE		Comparator 2 Hysteresis disable
		0	Hysteresis enabled
		1	Hysteresis disabled
5-2	Reserved		Reserved
1	COMP1_HYST_DISABLE		Comparator 1 Hysteresis disable
		0	Hysteresis enabled
		1	Hysteresis disabled
0	Reserved		Reserved

(1) This register is EALLOW protected.

8.13.8 ADC Revision Register (ADCREV)

Figure 8-37. ADC Revision Register (ADCREV)

15	8
REV	
R-x	

7	0
TYPE	
R-3h	

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

Table 8-23. ADC Revision Register (ADCREV) Field Descriptions

Bit	Field	Value	Description
15-8	REV		ADC Revision. To allow documentation of differences between revisions. First version is labeled as 00h.
7-0	TYPE	3	ADC Type. Always set to 3 for this type ADC.

8.13.9 ADC RESULT0-RESULT15 Registers (ADCRESULTx)

The ADC Result Registers are found in Peripheral Frame 0 (PF0). In the header files, the ADCRESULTx registers are located in the AdcResult register file, not AdcRegs.

Figure 8-38. ADC RESULT0-RESULT15 Registers (ADCRESULTx)

15	12	11	0
Reserved		RESULT	
R-0		R-0	

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

Table 8-24. ADC RESULT0-ADCRESULT15 Registers (ADCRESULTx) Field Descriptions

Bit	Field	Value	Description
15-12	Reserved		Reads return a zero; Writes have no effect.
11-0	RESULT		12-bit right-justified ADC result Sequential Sampling Mode (SIMULENx = 0): After the ADC completes a conversion of an SOCx, the digital result is placed in the corresponding ADCRESULTx register. For example, if SOC4 is configured to sample ADCINA1, the completed result of that conversion will be placed in ADCRESULT4. Simultaneous Sampling Mode (SIMULENx = 1): After the ADC completes a conversion of a channel pair, the digital results are found in the corresponding ADCRESULTx and ADCRESULTx+1 registers (assuming x is even). For example, for SOC4, the completed results of those conversions will be placed in ADCRESULT4 and ADCRESULT5. See 1.11 for timings of when this register is written.

Chapter 9

Comparator (COMP)

The Comparator (COMP) module described in this chapter is a Type 0 Comparator. See the *C2000 Real-Time Control Peripherals Reference Guide* for a list of all devices with modules of the same type, to determine the differences between the types, and for a list of device-specific differences within a type.

9.1 Introduction

The comparator module is a true analog voltage comparator in the VDDA domain. The core analog circuits include the comparator, its inputs and outputs, and the internal DAC reference. The supporting digital circuits include the DAC controls, interface to other on-chip logic, output qualification block, and the programmable control signals.

9.1.1 Features

The comparator block can monitor two external analog inputs, or monitor one external analog input using the internal DAC reference for the other input. The output of the comparator can be passed asynchronously, or be qualified and synchronized to the system clock period. The comparator output is routed to both the ePWM Trip Zone modules, as well as the GPIO output multiplexer.

9.1.2 Block Diagram

Note

Comparator hysteresis feedback is enabled by default and may interfere with high-impedance input signals. Use the COMPHYSTCTL register from the ADC module to configure the comparator hysteresis.

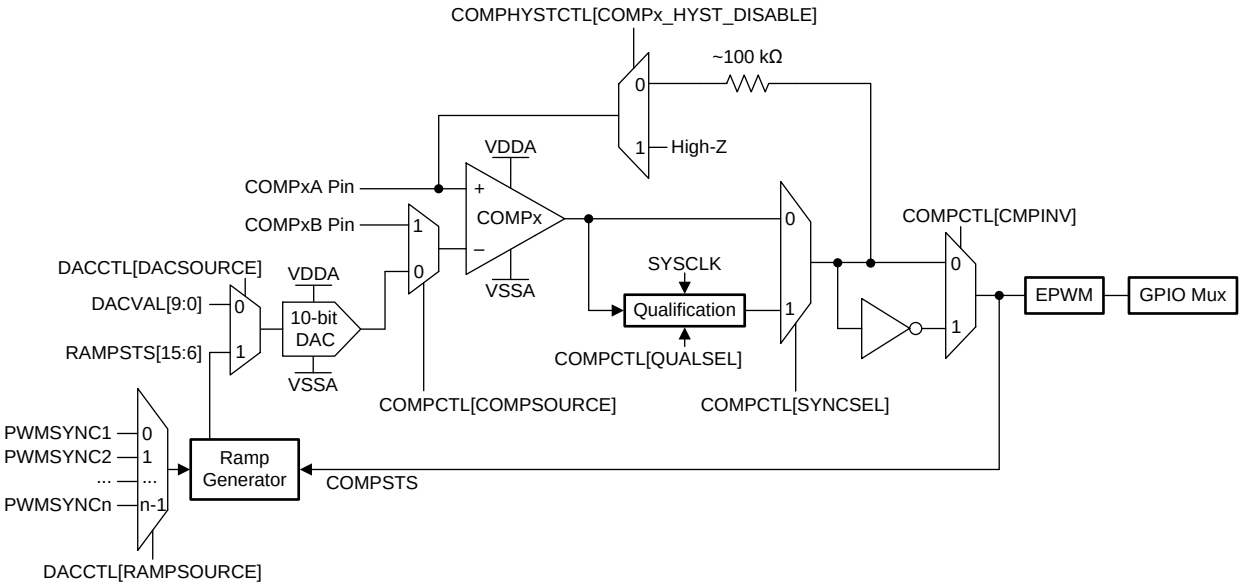


Figure 9-1. Comparator Block Diagram

9.2 Comparator Function

The comparator is an analog comparator module (Figure 9-2) and the output is asynchronous to the system clock. The truth table for the comparator is shown in Table 9-1.

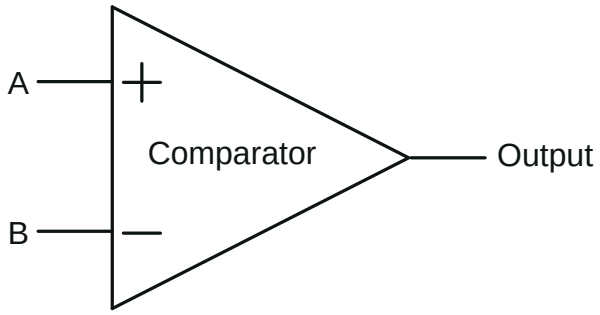


Figure 9-2. Comparator

Table 9-1. Comparator Truth Table

Voltages	Output
Voltage A > Voltage B	1
Voltage B > Voltage A	0

There is no definition for the condition Voltage A = Voltage B, since there is hysteresis in the response of the comparator output. Refer to the device data sheet for the value of this hysteresis. This also limits the sensitivity of the comparator output to noise on the input voltages.

The output state of the comparator, after qualification, is reflected by the COMPSTS bit in the COMPSTS register. The COMPSTS register will not update if the module clock is not enabled.

9.3 DAC Reference

Each comparator block contains an internal 10-bit voltage DAC reference that can used to supply the inverting input (B side input) of the comparator. The voltage output of the DAC is controlled by the DACVAL bit field in the DACVAL register. The output of the DAC is given by the equation:

$$V = \frac{DACVAL * (VDDA-VSSA)}{1023}$$

Since the DAC is also in the analog domain it does not require a clock to maintain its voltage output. A clock is required, however, to modify the digital inputs that control the DAC.

9.4 Ramp Generator Input

When selected, the ramp generator (see Figure 9-3) can produce a falling-ramp DAC output signal. In this mode, the DAC uses the most significant 10-bits of the 16-bit RAMPSTS countdown register as its input.

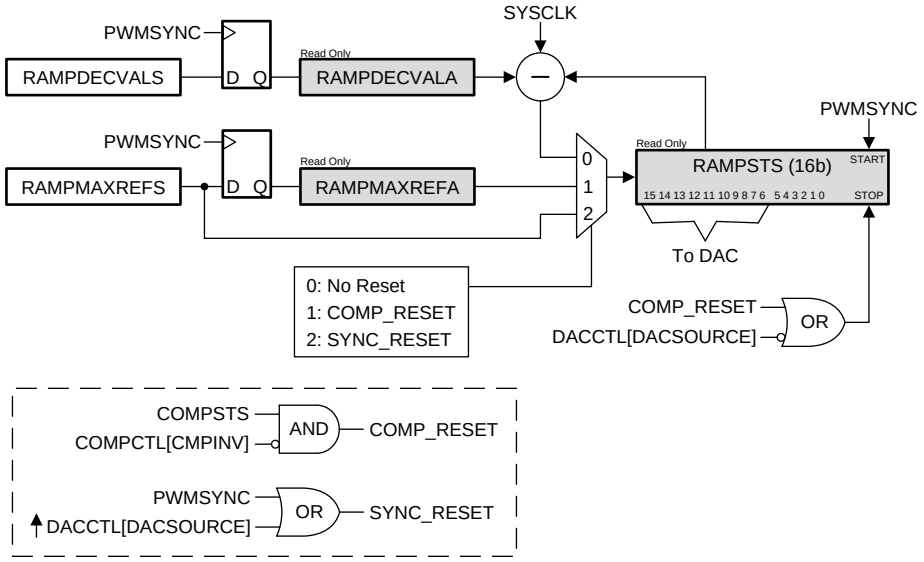


Figure 9-3. Ramp Generator Block Diagram

Note

The PWMSYNC signal for the Ramp Generator is derived from the HRPWM register field HRPCTL[PWMSYNCSEL]. The PWMSYNC signal is not the same as the EPWMSYNCI and EPWMSYNCO signals.

The RAMPSTS register is set to the value of RAMPMAXREF_SHDW when a selected PWMSYNC signal is received, and the value of RAMPDECVAL_ACTIVE is subtracted from RAMPSTS on every SYSCLK cycle thereafter. When the ramp generator is first enabled by setting DACSOURCE = 1, the value of RAMPSTS is loaded from RAMPMAXREF_SHDW, and the register remains static until the first PWMSYNC signal is received.

If the COMPSTS bit is set by the comparator while the ramp generator is active, the RAMPSTS register will reset to the value of RAMPMAXREF_ACTIVE and remain static until the next PWMSYNC signal is received. If the value of RAMPSTS reaches zero, the RAMPSTS register will remain static at zero until the next PWMSYNC signal is received.

To reduce the likelihood of race conditions when updating the ramp generator RAMPMAXREFA and RAMPDECVALA values, only the shadow registers RAMPMAXREF_SHDW and RAMPDECVAL_SHDW have write permissions. The values of the shadow registers are copied to the active registers on the next PWMSYNC signal. User software should take further steps to avoid writing to the shadow registers in the same cycle as a PWMSYNC signal or else the previous shadow register value may be lost.

The PWMSYNC signal width must be greater than SYSCLK to ensure that the ramp generator is able to detect the PWMSYNC signal.

The ramp generator behavior is further illustrated in Figure 9-4.

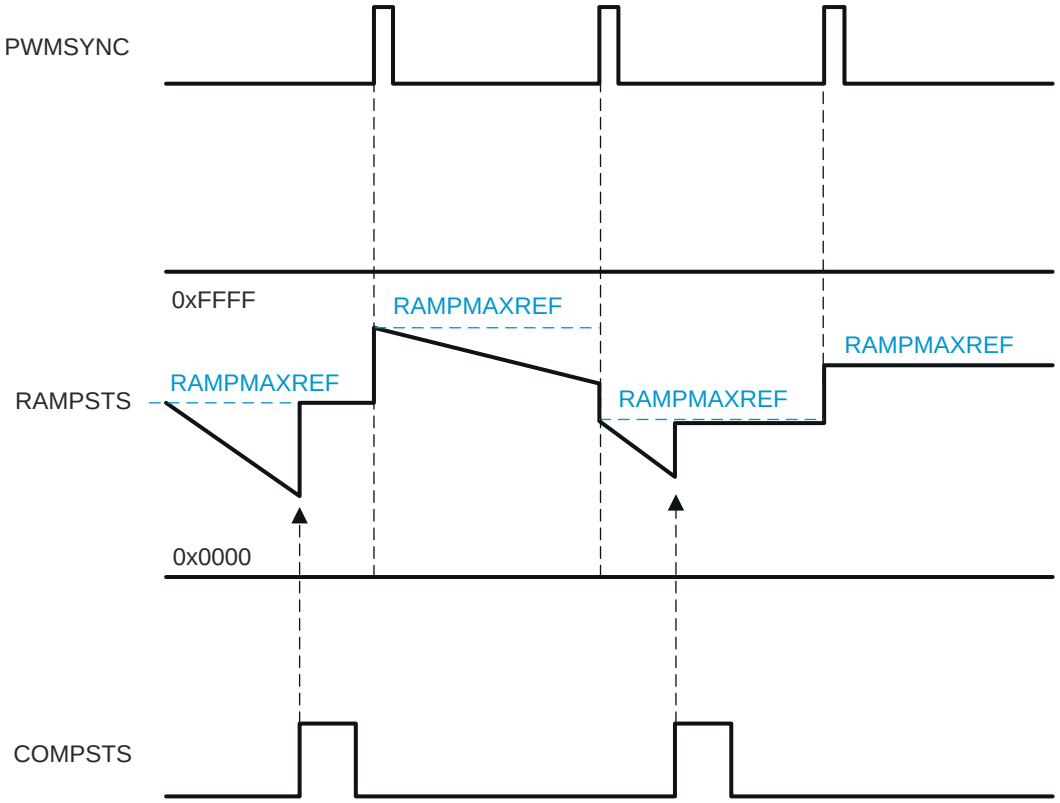


Figure 9-4. Ramp Generator Behavior

9.5 Initialization

There are 2 steps that must be performed prior to using the comparator block:

1. Enable the Band Gap inside the ADC by writing a 1 to the ADCBGPWD bit inside ADCCTL1.
2. Enable the comparator block by writing a 1 to the COMPDACEN bit in the COMPCTL register.

9.6 Digital Domain Manipulation

At the output of the comparator there are two more functional blocks that can be used to influence the behavior of the comparator output. They are:

1. Inverter circuit: Controlled by the CMPINV bit in the COMPCTL register; will apply a logical NOT to the output of the comparator. This function is asynchronous, while its control requires a clock present in order to change its value.
2. Qualification block: Controlled by the QUALSEL bit field in the COMPCTL register, and gated by the SYNCSEL bit in the COMPCTL register. This block can be used as a simple filter to only pass the output of the comparator once it is synchronized to the system clock. and qualified by the number of system clocks defined in QUALSEL bit field.

9.7 Comparator Registers

This device has the comparator modules listed in Table 9-2 and the registers listed in Table 9-3.

Table 9-2. Comparator Modules

Name	Address Range	Size(x16)	Description
COMP1	6400h – 641Fh	1	Comparator
COMP2	6420h – 643Fh	1	Comparator
COMP3	6440h – 645Fh	1	Comparator

Table 9-3. Comparator Module Registers

Name	Address Range (base)	Size (x16)	Description	Bit Description
COMPCTL	0x00	1	Comparator Control(1)	Section 9.7.1
Reserved	0x01	1	Reserved	-
COMPSTS	0x02	1	Compare Output Status	Section 9.7.2
Reserved	0x03	1	Reserved	-
DACCTL	0x04	1	DAC Control(1)	Section 9.7.3
Reserved	0x05	1	Reserved	-
DACVAL	0x06	1	10-bit DAC Value	Section 9.7.4
Reserved	0x07	1	Reserved	-
RAMPMAXREF_ACTIVE	0x08	1	Ramp Generator Maximum Reference (Active)	Section 9.7.5
Reserved	0x09	1	Reserved	-
RAMPMAXREF_SHDW	0x0A	1	Ramp Generator Maximum Reference (Shadow)	Section 9.7.6
Reserved	0x0B	1	Reserved	-
RAMPDECVAL_ACTIVE	0x0C	1	Ramp Generator Decrement Value (Active)	Section 9.7.7
Reserved	0x0D	1	Reserved	-
RAMPDECVAL_SHDW	0x0E	1	Ramp Generator Decrement Value (Shadow)	Section 9.7.8
Reserved	0x0F	1	Reserved	-
RAMPSTS	0x10	1	Ramp Generator Status	Section 9.7.9
Reserved	0x11-0x1F	15	Reserved	-

(1) This register is EALLOW protected.

9.7.1 Comparator Control (COMPCTL) Register

Figure 9-5. Comparator Control (COMPCTL) Register

15	9	8
Reserved		SYNCSEL
R-0		R/W-0

7	3	2	1	0
QUALSEL		CMPINV	COMPSOURCE	COMPDACEN
R/W-0		R/W-0	R/W-0	R/W-0

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

Table 9-4. Comparator Control (COMPCTL) Register Field Descriptions

Bit	Field	Value	Description (1)
15-9	Reserved		Reads return a 0; Writes have no effect.
8	SYNCSEL		Synchronization select for output of the comparator before being passed to EPWM/GPIO blocks
		0	Asynchronous version of Comparator output is passed
		1	Synchronous version of comparator output is passed
7-3	QUALSEL		Qualification Period for synchronized output of the comparator
		0h	Synchronized value of comparator is passed through
		1h	Input to the block must be consistent for 2 consecutive clocks before output of Qual block can change
		2h	Input to the block must be consistent for 3 consecutive clocks before output of Qual block can change
		...	...
		1Fh	Input to the block must be consistent for 32 consecutive clocks before output of Qual block can change
2	CMPINV		Invert select for Comparator
		0	Output of comparator is passed
		1	Inverted output of comparator is passed
1	COMPSOURCE		Source select for comparator inverting input
		0	Inverting input of comparator connected to internal DAC
		1	Inverting input connected to external pin
0	COMPDACEN		Comparator/DAC Enable
		0	Comparator/DAC logic is powered down.
		1	Comparator/DAC logic is powered up.

(1) This register is EALLOW protected.

9.7.2 Compare Output Status (COMPSTS) Register

Figure 9-6. Compare Output Status (COMPSTS) Register

15	1	0
Reserved		COMPSTS
R-0		R-0

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

Table 9-5. Compare Output Status (COMPSTS) Register Field Descriptions

Bit	Field	Value	Description
15-1	Reserved		Reads return zero and writes have no effect.
0	COMPSTS		Logical latched value of the comparator.

9.7.3 DAC Control (DACCTL) Register

Figure 9-7. DAC Control (DACCTL) Register

15	14	13	8
FREE:SOFT		Reserved	
R/W-0		R-0	

7	5	4	1	0
Reserved		RAMPSOURCE		DACSOURCE
R-0		R/W-0		R/W-0

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

Table 9-6. DAC Control (DACCTL) Register Field Descriptions

Bit	Field	Value	Description (1)
15-14	FREE:SOFT		Emulation mode behavior. Selects ramp generator behavior during emulation suspend.
		0h	Stop immediately
		1h	Complete current ramp, and stop on the next PWMSYNC signal
		2h-3h	Run free
13-5	Reserved		Reads return a 0; Writes have no effect.
4-1	RAMPSOURCE		Ramp generator source sync select. PWMSYNC is derived from the HRPWM register field HRPCTL[PWMSYNCSEL].
		0h	PWMSYNC1 is the source sync
		1h	PWMSYNC2 is the source sync
		2h	PWMSYNC3 is the source sync
		...	...
		n-1	PWMSYNCn is the source sync
0	DACSOURCE		DAC source control. Select DACVAL or ramp generator to control the DAC.
		0	DAC controlled by DACVAL
		1	DAC controlled by ramp generator

(1) This register is EALLOW protected.

9.7.4 DAC Value (DACVAL) Register

Figure 9-8. DAC Value (DACVAL) Register

15	10	9	0
Reserved		DACVAL	
R-0		R/W-0	

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

Table 9-7. DAC Value (DACVAL) Register Field Descriptions

Bit	Field	Value	Description
15-10	Reserved		Reads return zero and writes have no effect.
9-0	DACVAL	0-3FFh	DAC Value bits, scales the output of the DAC from 0 – 1023.

9.7.5 Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register

Figure 9-9. Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register

15	0
RAMPMAXREFA	
R-0	

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

Table 9-8. Ramp Generator Maximum Reference Active (RAMPMAXREF_ACTIVE) Register Field Descriptions

Bit	Field	Value	Description
15-0	RAMPMAXREFA	0-FFFFh	16-bit maximum reference active value for down ramp generator. This value is loaded from RAMPMAXREF_SHDW when the PWMSYNC signal is received.

9.7.6 Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register

Figure 9-10. Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register

15	0
RAMPMAXREFS	
R/W-0	

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

Table 9-9. Ramp Generator Maximum Reference Shadow (RAMPMAXREF_SHDW) Register Field Descriptions

Bit	Field	Value	Description
15-0	RAMPMAXREFS	0-FFFFh	16-bit maximum reference shadow value for down ramp generator.

9.7.7 Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register

Figure 9-11. Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register

15	0
RAMPDECVALA	
R-0	

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

Table 9-10. Ramp Generator Decrement Value Active (RAMPDECVAL_ACTIVE) Register Field Descriptions

Bit	Field	Value	Description
15-0	RAMPDECVALA	0-FFFFh	16-bit decrement active value for down ramp generator. This value is loaded from RAMPDECVAL_SHDW when the PWMSYNC signal is received.

9.7.8 Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register

Figure 9-12. Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register

15	0
RAMPDECVALS	
R/W-0	

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

Table 9-11. Ramp Generator Decrement Value Shadow (RAMPDECVAL_SHDW) Register Field Descriptions

Bit	Field	Value	Description
15-0	RAMPDECVALS	0-FFFFh	16-bit decrement shadow value for down ramp generator.

9.7.9 Ramp Generator Status (RAMPSTS) Register

Figure 9-13. Ramp Generator Status (RAMPSTS) Register

15	0
RAMPVALUE	
R-0	

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

Table 9-12. Ramp Generator Status (RAMPSTS) Register Field Descriptions

Bit	Field	Value	Description
15-0	RAMPVALUE	0-FFFFh	16-bit value of down ramp generator.

Chapter 10

Control Law Accelerator (CLA)

The Control Law Accelerator (CLA) Type-0 is an independent, fully-programmable, 32-bit floating-point math processor that brings concurrent control-loop execution to the C28x family. The low interrupt latency of the CLA allows it to read ADC samples "just-in-time." This significantly reduces the ADC sample to output delay to enable faster system response and higher MHz control loops. By using the CLA to service time-critical control loops, the main CPU is free to perform other system tasks such as communications and diagnostics. This chapter provides an overview of the architectural structure and components of the control law accelerator.

10.1 Introduction

The Control Law Accelerator extends the capabilities of the C28x CPU by adding parallel processing. Time-critical control loops serviced by the CLA can achieve low ADC sample to output delay. Thus, the CLA enables faster system response and higher frequency control loops. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently.

10.1.1 Features

The following is a list of major features of the CLA:

- C compilers are available for CLA software development.
- Clocked at the same rate as the main CPU (SYSCLKOUT).
- An independent architecture allowing CLA algorithm execution independent of the main C28x CPU.
  - Complete bus architecture:
    - Program Address Bus (PAB) and Program Data Bus (PDB)
    - Data Read Address Bus (DRAB), Data Read Data Bus (DRDB), Data Write Address Bus (DWAB), and Data Write Data Bus (DWDB)
  - Independent eight stage pipeline.
  - 12-bit program counter (MPC)
  - Four 32-bit result registers (MR0-MR3)
  - Two 16-bit auxiliary registers (MAR0, MAR1)
  - Status register (MSTF)
- Instruction set includes:
  - IEEE single-precision (32-bit) floating-point math operations
  - Floating-point math with parallel load or store
  - Floating-point multiply with parallel add or subtract
  - 1/X and 1/sqrt(X) estimations
  - Data type conversions.
  - Conditional branch and call
  - Data load/store operations
- The CLA program code can consist of up to eight tasks or interrupt service routines
  - The start address of each task is specified by the MVECT registers.
  - No limit on task size as long as the tasks fit within the configurable CLA program memory space.
  - One task is serviced at a time until its completion. There is no nesting of tasks.
  - Upon task completion a task-specific interrupt is flagged within the PIE.
  - When a task finishes the next highest-priority pending task is automatically started.
- Task trigger mechanisms:
  - C28x CPU via the IACK instruction
  - Task1 to Task8: trigger sources from peripherals connected to the shared bus on which the CLA assumes secondary ownership.
- Memory and Shared Peripherals:
  - Two dedicated message RAMs for communication between the CLA and the main CPU.
  - The C28x CPU can map CLA program and data memory to the main CPU space or CLA space.

10.1.2 CLA Related Collateral

Foundational Materials

- C2000 CLA C Compiler Series (Video)
- CLA Hands On Workshop (Video)
- CLA usage in Valley Switching Boost Power Factor Correction (PFC) Reference Design (Video)
- Control Law Accelerator Debug in CCS 4 (Video)
- Control Law Accelerator Example Framework (Video)
- *Enhancing the Computational Performance of the C2000 Microcontroller Family Application Report*

Getting Started Materials

- CLA Software Development Guide
- *Software Examples to Showcase Unique Capabilities of TI's C2000 CLA Application Report*

Expert Materials

- *Digital Control of Two Phase Interleaved PFC and Motor Drive Using MCU With CLA Application Report*
- *Sensorless Field Oriented Control:3-Phase Perm.Magnet Synch. Motors With CLA Application Report*

10.1.3 Block Diagram

Figure 10-1 is a block diagram of the CLA.

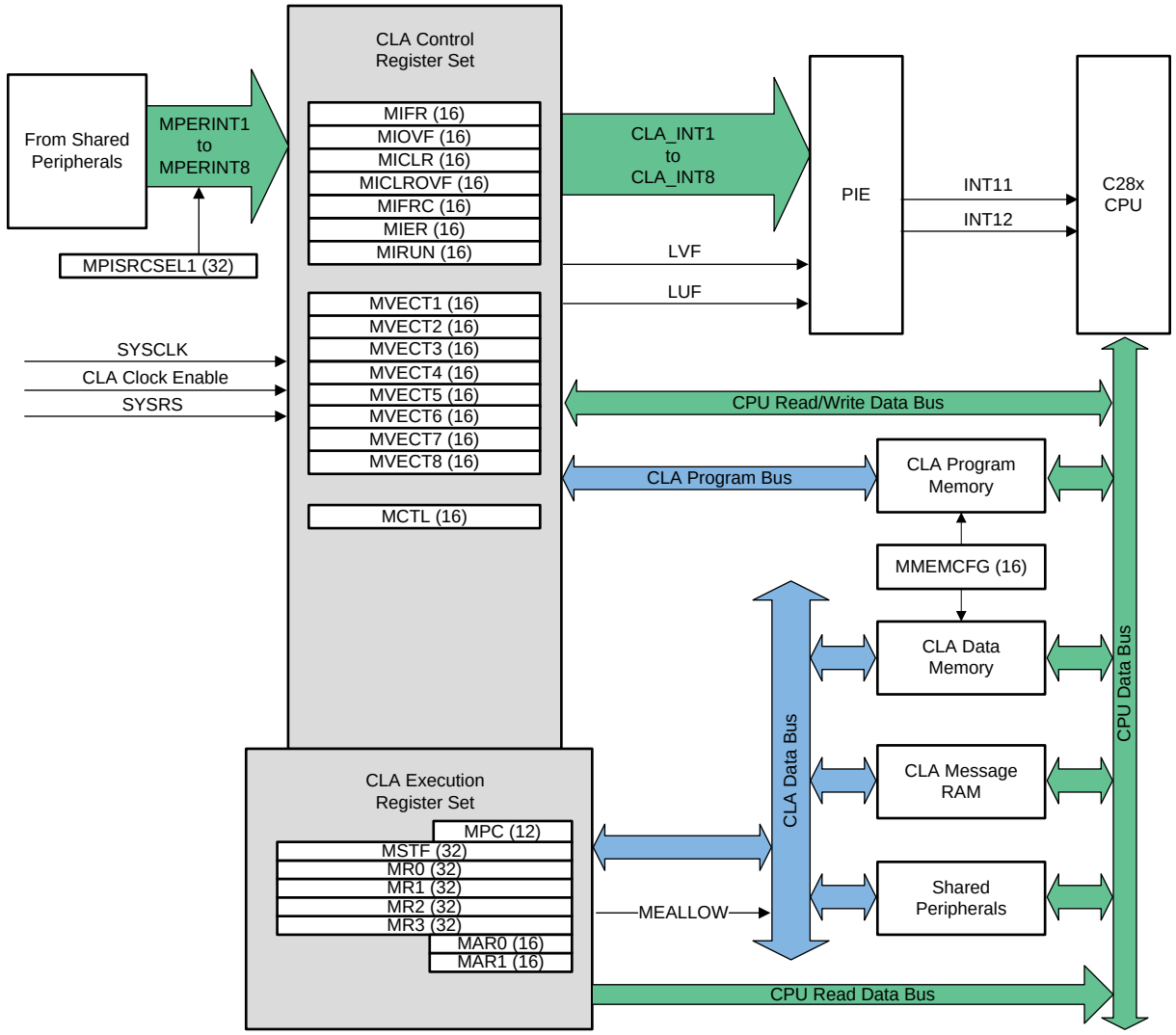


Figure 10-1. CLA (Type 0) Block Diagram

10.2 CLA Interface

This section describes how the C28x main CPU can interface to the CLA and conversely.

10.2.1 CLA Memory

The CLA can access three types of memory: program, data and message RAMs. The behavior and arbitration for each type of memory is described in this chapter. The CLA RAMs are protected by the CSM module. Refer to the Code Security Module (CSM) section of the *System Control and Interrupts* chapter for more details on the security scheme.

- **CLA Program Memory**

  The CLA program can be loaded to a designated memory block. At reset, all memory blocks are mapped to the CPU. While mapped to the CPU space, the CPU can copy the CLA program code into the memory. During debug, the memory can also be loaded directly by the Code Composer Studio™ IDE.

  Once the memory is initialized with CLA code, the CPU maps it to the CLA program space by setting the MMEMCFG[PROGE] bit.

  When a memory block is configured as CLA program memory, debug accesses are allowed only on cycles where the CLA is not fetching a new instruction. A detailed explanation of the memory configurations and access arbitration (CPU, CLA, and DEBUG) process can be found in the Memory Controller Module section of the *System Control and Interrupts* chapter.

  All CLA program fetches are performed as 32-bit read operations and all opcodes must be aligned to an even address. Since all CLA opcodes are 32-bits, this alignment occurs naturally.

- **CLA Data Memory**

  Designated memory locations can serve as data memory blocks to the CLA. At reset, all blocks are mapped to the CPU memory space, whereby the CPU can initialize the memory with data tables, coefficients, and so on, for the CLA to use.

  Once the memory is initialized with CLA data, the CPU maps it to the CLA data space by setting the respective MMEMCFG[RAMxE] bit.

  When a memory block is configured as CLA data memory, CLA read and write accesses are arbitrated along with CPU accesses. The user has the option of turning on CPU fetch or write protection to the memory by writing to the appropriate MMEMCFG[RAMnCPUE] bits . A detailed explanation of the memory configurations and access arbitration (CPU, CLA, and DEBUG) process can be found in the Memory Controller Module section of the *System Control and Interrupts* chapter.
- **CLA Shared Message RAMs**

  There are two memory blocks for data sharing and communication between the CLA and the CPU . The message RAMs are always mapped to both CPU and CLA memory spaces, and only data access is allowed; no program fetches can be performed.
  - **CLA to CPU Message RAM:** The CLA can use this block to pass data to the CPU. This block is both readable and writable by the CLA. This block is also readable by the CPU but writes by the CPU are ignored.
  - **CPU to CLA Message RAM:** The CPU can use this block to pass data and messages to the CLA. This message RAM is both readable and writable by the CPU. The CLA can perform reads but writes by the CLA are ignored.

10.2.2 CLA Memory Bus

The CLA has dedicated bus architecture similar to that of the C28x CPU where there are separate program read, data read, and data write buses. Thus, there can be simultaneous instruction fetch, data read, and data write in a single cycle. Like the C28x CPU, the CLA expects memory logic to align any 32-bit read or write to an even address. If the address-generation logic generates an odd address, the CLA will begin reading or writing at the previous even address. This alignment does not affect the address values generated by the address-generation logic.

- **CLA Program Bus**

  The CLA program bus has an access range of 2048 32-bit instructions. Since all CLA instructions are 32 bits, this bus always fetches 32 bits at a time and the opcodes must be even-word aligned. The amount of program space available for the CLA is limited to the number of allocated memory blocks. This number is device-dependent and will be described in the device-specific data manual.

- **CLA Data Read Bus**

  The CLA data read bus has a 64K x 16 address range. The bus can perform 16 or 32-bit reads and will automatically stall if there are memory access conflicts. The data read bus has access to both the message RAMs, CLA data memory, and the shared peripherals.

- **CLA Data Write Bus**

  The CLA data write bus has a 64K x 16 address range. This bus can perform 16 or 32-bit writes. The bus will automatically stall if there are memory access conflicts. The data write bus has access to the CLA to CPU message RAM, CLA data memory, and the shared peripherals.

10.2.3 Shared Peripherals and EALLOW Protection

The CPU and CLA share access to some peripherals. Section 10.3 describes the arbitration between the CPU and CLA.

Refer to the device data sheet for the list of peripherals connected to the bus.

Several peripheral control registers are protected from spurious 28x CPU writes by the EALLOW protection mechanism. These same registers are also protected from spurious CLA writes. The EALLOW bit in the CPU status register 1 (ST1) indicates the state of protection for the CPU. Likewise the MEALLOW bit in the CLA status register (MSTF) indicates the state of write protection for the CLA. The MEALLOW CLA instruction enables write access by the CLA to EALLOW protected registers. Likewise the MEDIS CLA instruction will disable write access. This way the CLA can enable/disable write access independent of the CPU.

The ADC offers the option to generate an early interrupt pulse at the start of a sample conversion. If this option is used to start an ADC-triggered CLA task, the user may use the intervening cycles, until the completion of the conversion, to perform preliminary calculations or loads and stores before finally reading the ADC value. The CLA pipeline activity for this scenario is shown in Section 10.5.

10.2.4 CLA Tasks and Interrupt Vectors

The CLA program code is divided up into tasks or interrupt service routines. Tasks do not have a fixed starting location or length. The CLA program memory can be divided up as desired. The CLA uses the contents of the interrupt vectors (MVECT1 to MVECT8) to determine where a task begins; tasks are terminated by the MSTOP instruction.

The CLA supports eight tasks. Task 1 has the highest priority and task 8 has the lowest priority.

A task can be requested by a peripheral interrupt or by software:

- **Peripheral interrupt trigger**

  Each task can be triggered by software-selectable interrupt sources. The trigger for each task is defined by writing an appropriate value to the MPISRCSEL1[PERINTnSEL] bit field. Each option specifies an interrupt source from a specific peripheral on the shared bus. The peripheral interrupt triggers are listed in Section 10.7.3.3.

  For example, Task 1 (MVECT1) can be set to trigger on EPWM1_INT by writing 2 to MPISRCSEL1[PERINT1SEL]. To disable the triggering of a task by a peripheral, the user must configure the MPISRCSEL1[PERINTxSEL] bit field to a "No interrupt source" selection.

- **Software Trigger**

  CPU software can trigger tasks by writing to the MIFRC register or by the IACK instruction. Using the IACK instruction is more efficient because it does not require you to issue an EALLOW to set MIFR bits. Set the MCTL[IACKE] bit to enable the IACK feature. Each bit in the operand of the IACK instruction corresponds to a task. For example IACK #0x0001 will set bit 0 in the MIFR register to start task 1. Likewise IACK #0x0003 will set bits 0 and 1 in the MIFR register to start task 1 and task 2.

The CLA has its own fetch mechanism and can run and execute a task independent of the CPU. Only one task is serviced at a time; there is no nesting of tasks. The task currently running is indicated in the MIRUN register. Interrupts that have been received but not yet serviced are indicated in the flag register (MIFR). If an interrupt request from a peripheral is received and that same task is already flagged, then the overflow flag bit is set. Overflow flags will remain set until they are cleared by the CPU.

If the CLA is idle (no task is currently running) then the highest priority interrupt request that is both flagged (MIFR) and enabled (MIER) will start.

The flow is as follows:

1. The associated RUN register bit is set (MIRUN) and the flag bit (MIFR) is cleared.
2. The CLA begins execution at the location indicated by the associated interrupt vector (MVECTx). MVECT contains the absolute 16-bit address of the task in the lower 64K memory space.
3. The CLA executes instructions until the MSTOP instruction is found. This indicates the end of the task.
4. The MIRUN bit is cleared.
5. The task-specific interrupt to the PIE is issued. This informs the main CPU that the task has completed.
6. The CLA returns to idle.

Once a task completes the next highest-priority pending task is automatically serviced and this sequence repeats.

10.3 CLA and CPU Arbitration

The CLA typically operates independently of the CPU. Under circumstance where both the CLA and the CPU are attempting to currently access memory or a peripheral register, an arbitration procedure will occur. The one exception is the ADC result registers which do not create a conflict when read by both the CPU and the CLA simultaneously.

The interfaces that can experience access conflicts are:

- CLA Message RAMs
- CLA Program Memory
- CLA Data RAMs

10.3.1 CLA Message RAMs

Message RAMs consist of two blocks. These blocks are useful for passing data between the CPU and CLA. No opcode fetches are allowed from the message RAMs. The message RAMs have the following characteristics:

- **CLA to CPU Message RAM:**

  The following accesses are allowed:
  - CPU reads
  - CLA reads and writes
  - CPU debug reads and writes

  The following accesses are ignored
  - CPU writes

  Priority of accesses are (highest priority first):
  1. CLA write
  2. CPU debug write
  3. CPU data read, program read, CPU debug read
  4. CLA data read

- **CPU to CLA Message RAM:**

  The following accesses are allowed:
  - CPU reads and writes
  - CLA reads
  - CPU debug reads and writes

  The following accesses are ignored
  - CLA writes

  Priority of accesses are (highest priority first):
  1. CLA read
  2. CPU data write, program write, CPU debug write
  3. CPU data read, CPU debug read
  4. CPU program read

10.3.2 CLA Program Memory

The behavior of the program memory depends on the state of the MMEMCFG[PROGE] bit. This bit controls whether the memory is mapped to CLA space or CPU space.

- **MMEMCFG[PROGE] == 0**

  In this case the memory is mapped to the CPU. The CLA will be halted and no tasks should be incoming.
  - Any CLA fetch will be treated as an illegal opcode condition as described in Section 10.4.4. This condition will not occur if the proper procedure is followed to map the program memory.
  - CLA reads and writes cannot occur
  - The memory block behaves as any normal RAM block mapped to CPU memory space.

  Priority of accesses are (highest priority first):
  1. CPU data write, program write, debug write
  2. CPU data read, program read, debug read
  3. CPU fetch, program read

- **MMEMCFG[PROGE] == 1**

  In this case the memory block is mapped to CLA space. The CPU can only make debug accesses.
  - CLA reads and writes cannot occur
  - CLA fetches are allowed
  - CPU fetches return 0 which is an illegal opcode and will cause an ITRAP interrupt.
  - CPU data reads and program reads return 0
  - CPU data writes and program writes are ignored

  Priority of accesses are (highest priority first):
  1. CLA fetch
  2. CPU debug write
  3. CPU debug read

Note

Because the CLA fetch has higher priority than CPU debug reads, it is possible for the CLA to permanently block debug accesses if the CLA is executing in a loop. This might occur when initially developing CLA code due to a bug. To avoid this issue, the program memory will return all 0x0000 for CPU debug reads (ignore writes) when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access can be performed.

10.3.3 CLA Data Memory

There are two independent data memory blocks. The behavior of the data memory depends on the state of the MMEMCFG[RAM0E] and MMEMCFG[RAM1E] bits. These bits determine whether the memory blocks are mapped to CLA space or CPU space.

- **MMEMCFG[RAMxE] == 0**

  In this case the memory block is mapped to the CPU.
  - CLA fetches cannot occur to this block.
  - CLA reads return 0.
  - CLA writes are ignored.
  - The memory block behaves as any normal RAM block mapped to the CPU memory space.

Priority of accesses are (highest priority first):

1. CPU data write/program write/debug access write
2. CPU data read/debug access read
3. CPU fetch/program read

- **MMEMCFG[RAMxE] == 1**

  In this case the memory block is mapped to CLA space. The CPU can make only debug accesses.
  - CLA fetches cannot occur to this block.
  - CLA read and CLA writes are allowed.
  - CPU fetches return 0
  - CPU data reads and program reads return 0.
  - CPU data writes and program writes are ignored.

  Priority of accesses are (highest priority first):

  1. CLA data write
  2. CPU debug write
  3. CPU debug read
  4. CLA read

10.3.4 Peripheral Registers (ePWM, HRPWM, Comparator)

Accesses to the registers follow these rules:

- If both the CPU and CLA request access at the same time, then the CLA will have priority and the main CPU is stalled.
- If a CPU access is in progress and another CPU access is pending, then the CLA will have priority over the pending CPU access. In this case the CLA access will begin when the current CPU access completes.
- While a CPU access is in progress any incoming CLA access will be stalled.
- While a CLA access is in progress any incoming CPU access will be stalled.
- A CPU write operation has priority over a CPU read operation.
- A CLA write operation has priority over a CLA read operation.
- If the CPU is performing a read-modify-write operation and the CLA performs a write to the same location, the CLA write may be lost if the operation occurs in-between the CPU read and write. For this reason, you should not mix CPU and CLA accesses to same location.

10.4 CLA Configuration and Debug

This section discusses the steps necessary to configure and debug the CLA.

10.4.1 Building a CLA Application

The control law accelerator can be programmed in either CLA assembly code, using the instructions described in Section 10.6, or a reduced subset of the C language. CLA assembly code resides in the same project with C28x code. The only restriction is the CLA code must be in its own assembly section. This can be easily done using the **.sect** assembly directive. This does not prevent CLA and C28x code from being linked into the same memory region in the linker command file.

System and CLA initialization are performed by the main CPU. This would typically be done in C or C++ but can also include C28x assembly code. The main CPU will also copy the CLA code to the program memory and, if needed, initialize the CLA data RAM(s). Once system initialization is complete and the application begins, the CLA will service its interrupts using the CLA assembly code (or tasks). The main CPU can perform other tasks concurrently with CLA program execution.

The CLA Type 0 requires Codegen V5.2.0 or later with the compiler switch: --cla_support=cla0.

10.4.2 Typical CLA Initialization Sequence

A typical CLA initialization sequence is performed by the main CPU as described in this section.

1. **Copy CLA code into the CLA program RAM**

   The source for the CLA code can initially reside in the Flash or a data stream from a communications peripheral or anywhere the main CPU can access it. The debugger can also be used to load code directly to the CLA program RAM during development.
2. **Initialize CLA data RAM, if necessary**

   Populate the CLA data RAM with any required data coefficients or constants.
3. **Configure the CLA registers**

   Configure the CLA registers, but keep interrupts disabled until later (leave MIER = 0):
   - **Enable the CLA peripheral clock using the assigned PCLKCRn register**

     The peripheral clock control (PCLKCRn) registers are defined in the *System Control and Interrupts* chapter.
   - **Populate the CLA task interrupt vectors**
     - MVECT1 to MVECT8

       Each vector needs to be initialized with the start address of the task to be executed when the CLA receives the associated interrupt. The address is an offset from the base address of the assigned CLA Program memory block.
   - **Select the task interrupt sources**

     For each task select the interrupt source in the CLA1TASKSRCSELx register. If a task is software triggered, select no interrupt.
   - **Enable IACK to start a task from software, if desired**

     To enable the IACK instruction to start a task set the MCTL[IACKE] bit. Using the IACK instruction avoids having to set and clear the EALLOW bit.
   - **Map CLA data RAM to CLA space, if necessary**

     Map the data RAM to the CLA space by writing a 1 to the MMEMCFG[RAMxE] bit. The CPU will have restricted access to the memory block once the MMEMCFG[RAMxE] bit is set.
   - **Map CLA program RAM to CLA space**

     Map the CLA program RAM to CLA space by setting the MMEMCFG[PROGE] bit. The CPU will only have debug access to program RAM once the MMEMCFG[PROGE] bit is set. Allow two SYSCLK cycles for MMEMCFG updates to take effect.

4. **Initialize the PIE vector table and registers**

   When a CLA task completes, the associated interrupt in the PIE will be flagged. The CLA overflow and underflow flags also have associated interrupts within the PIE.
5. **Enable CLA tasks/interrupts**

   Set appropriate bits in the interrupt enable register (MIER) to allow the CLA to service interrupts. It should be noted that a CLA task only triggers on a level transition (a falling edge) of the configured interrupt source. If a peripheral is enabled and an interrupt fires before the CLA is configured, then the CLA will not see the interrupt edge and will not respond. To avoid this, configure the CLA before the peripherals or clear any pending peripheral interrupts before setting bits in the MIER register.
6. **Initialize other peripherals**

   Initialize any peripherals (such as ePWM, ADC, and others) that will generate interrupt triggers for enabled CLA tasks.

   The CLA is now ready to service interrupts and the message RAMs can be used to pass data between the CPU and the CLA. Mapping of the CLA program and data RAMs typically occurs only during the initialization process. If the RAM mapping needs to be changed after initialization, the CLA interrupts must be disabled and all tasks must be completed (by checking the MIRUN register) prior to modifying the RAM ownership.

10.4.3 Debugging CLA Code

Debugging the CLA code is a simple process that occurs independently of the main CPU. .

10.4.3.1 Breakpoint Support (MDEBUGSTOP)

1. **Insert a breakpoint in CLA code**

   Insert a CLA breakpoint (MDEBUGSTOP instruction) into the code where you want the CLA to halt, then rebuild and reload the code. Because the CLA does not flush its pipeline when you single-step, the MDEBUGSTOP instruction must be inserted as part of the code. The debugger cannot insert it as needed.

   If CLA breakpoints are not enabled, then the MDEBUGSTOP will be ignored and is treated as a MNOP. The MDEBUGSTOP instruction can be placed anywhere in the CLA code as long as it is not within three instructions of a MBCNDD, MCCNDD, or MRCNDD instruction. When programming in C, the user can use the __mdebugstop() intrinsic instead; the compiler will ensure that the placement of the MDEBUSTOP instruction in the generated assembly does not violate any of the pipeline restrictions.
2. **Enable CLA breakpoints**

   Enable the CLA breakpoints in the debugger. In the Code Composer Studio™ IDE, this is done by connecting to the CLA core (or tap) from the debug perspective. Breakpoints are disabled when the core is disconnected.
3. **Start the task**

   There are three ways to start the task:
   a. The peripheral can assert an interrupt,
   b. The main CPU can execute an IACK instruction, or
   c. The user can manually write to the MIFRC register in the debugger window

   When the task starts, the CLA will execute instructions until the MDEBUGSTOP is in the D2 phase of the pipeline. At this point, the CLA will halt and the pipeline will be frozen. The MPC register will reflect the address of the MDEBUGSTOP instruction.

4. **Single-step the CLA code**

   Once halted, the user can single-step the CLA code. The behavior of a CLA single-step is different than the main C28x. When issuing a CLA single-step, the pipeline is clocked only one cycle and then again frozen. On the C28x CPU, the pipeline is flushed for each single-step.

   You can also run to the next MDEBUGSTOP or to the end of the task. If another task is pending, it will automatically start when you run to the end of the task.

   Note

   A CLA fetch has higher priority than CPU debug reads. For this reason, it is possible for the CLA to permanently block CPU debug accesses if the CLA is executing in a loop. This might occur when initially developing CLA code due to a bug that causes an infinite loop. To avoid locking up the main CPU, the program memory will return all 0x0000 for CPU debug reads when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access to CLA program memory can be performed.

   If the CLA gets caught in an infinite loop, you can use a soft or hard reset to exit the condition. A debugger reset will also exit the condition.

   There are special cases that can occur when single-stepping a task such that the program counter, MPC, reaches the MSTOP instruction at the end of the task.

   - **MPC halts at or after the MSTOP with a task already pending**

     If you are single-stepping or halted in "task A" and "task B" comes in before the MPC reaches the MSTOP, then "task B" will start if you continue to step through the MSTOP instruction. Basically if "task B" is pending before the MPC reaches MSTOP in "task A" then there is no issue in "task B" starting and no special action is required.
   - **MPC halts at or after the MSTOP with no task pending**

     In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. If "task B" comes in at this point, it will be flagged in the MIFR register but it may or may not start if you continue to single-step through the MSTOP instruction of "task A."

     It depends on exactly when the new task comes in. To reliably start "task B" perform a soft reset and reconfigure the MIER bits. Once this is done, you can start single-stepping "task B."

     This case can be handled slightly differently if there is control over when "task B" comes in (for example, using the IACK instruction to start the task). In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. Before forcing "task B," run free to force the CLA out of the debug state. Once this is done you can force "task B" and continue debugging.

5. **Disable CLA breakpoints, if desired**

   In the Code Composer Studio™ IDE, you can disable the CLA breakpoints by disconnecting the CLA core in the debug perspective. Make sure to first issue a run or reset; otherwise, the CLA will be halted and no other tasks will start.

10.4.4 CLA Illegal Opcode Behavior

If the CLA fetches an opcode that does not correspond to a legal instruction, it will behave as follows:

- The CLA will halt with the illegal opcode in the D2 phase of the pipeline as if it were a breakpoint. This will occur whether CLA breakpoints are enabled or not.
- The CLA will issue the task-specific interrupt to the PIE.
- The MIRUN bit for the task will remain set.

Further single-stepping is ignored once execution halts due to an illegal op-code. To exit this situation, issue either a soft or hard reset of the CLA as described in Section 10.4.5.

10.4.5 Resetting the CLA

There may be times when you need to reset the CLA. For example, during code debug the CLA may enter an infinite loop due to a code bug. The CLA has two types of resets: hard and soft. Both of these resets can be performed by the debugger or by the main CPU.

- **Hard Reset**

  Writing a 1 to the MCTL[HARDRESET] bit will perform a hard reset of the CLA. The behavior of a hard reset is the same as a system reset (via $\overline{\text{XRS}}$ or the debugger). In this case all CLA configuration and execution registers will be set to their default state and CLA execution will halt.
- **Soft Reset**

  Writing a 1 to the MCTL[SOFTRESET] bit performs a soft reset of the CLA. If a task is executing it will halt and the associated MIRUN bit will be cleared. All bits within the interrupt enable (MIER) register will also be cleared so that no new tasks start.

10.5 Pipeline

This section describes the CLA pipeline stages and presents cases where pipeline alignment must be considered.

10.5.1 Pipeline Overview

The CLA pipeline is very similar to the C28x pipeline with eight stages:

1. **Fetch 1 (F1):** During the F1 stage the program read address is placed on the CLA program address bus.
2. **Fetch 2 (F2):** During the F2 stage the instruction is read using the CLA program data bus.
3. **Decode 1 (D1):** During D1 the instruction is decoded.
4. **Decode 2 (D2):** Generate the data read address. Changes to MAR0 and MAR1 due to post-increment using indirect addressing takes place in the D2 phase. Conditional branch decisions are also made at this stage based on the MSTF register flags.
5. **Read 1 (R1):** Place the data read address on the CLA data-read address bus. If a memory conflict exists, the R1 stage will be stalled.
6. **Read 2 (R2):** Read the data value using the CLA data read data bus.
7. **Execute (EXE):** Execute the operation. Changes to MAR0 and MAR1 due to loading an immediate value or value from memory take place in this stage.
8. **Write (W):** Place the write address and write data on the CLA write data bus. If a memory conflict exists, the W stage will be stalled.

10.5.2 CLA Pipeline Alignment

The majority of the CLA instructions do not require any special pipeline considerations. This section lists the few operations that do require special consideration.

- **Write Followed by Read**

  In both the C28x pipeline and the CLA pipeline, the read operation occurs before the write. This means that if a read operation immediately follows a write, then the read will complete first as shown in Table 10-1. In most cases this does not cause a problem since the contents of one memory location does not depend on the state of another. For accesses to peripherals where a write to one location can affect the value in another location, the code must wait for the write to complete before issuing the read as shown in Table 10-2.

  This behavior is different for the C28x CPU. For the C28x CPU, any write followed by read to the same location is protected by what is called write-followed-by-read protection. This protection automatically stalls the pipeline so that the write will complete before the read. In addition, some peripheral frames are protected such that a C28x CPU write to one location within the frame will always complete before a read to the frame. The CLA does not have this protection mechanism. Instead the code must wait to perform the read.

Table 10-1. Write Followed by Read - Read Occurs First

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1 MMOV16 @Reg1, MR3	I1							
I2 MMOV16 MR2, @Reg2	I2	I1						
		I2	I1					
			I2	I1				
				I2	I1			
					I2	I1		
						I2	I1	
							I2	I1

Table 10-2. Write Followed by Read - Write Occurs First

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1 MMOV16 @Reg1, MR3	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
I5 MMOV16 MR2, @Reg2	I5	I4	I3	I2	I1			
		I5	I4	I3	I2	I1		
			I5	I4	I3	I2	I1	
				I5	I4	I3	I2	I1
					I5	I4	I3	
						I5	I4	
							I5	

- **Delayed Conditional instructions: MBCNDD, MCCNDD, and MRCNDD**

  Referring to Example 10-1, the following applies to delayed conditional instructions:

  – **I1**

    I1 is the last instruction that can effect the CNDF flags for the branch, call or return instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MBCNDD, MCCNDD, or MRCNDD is in the D2 phase.

  – **I2, I3, and I4**

    The three instructions preceding MBCNDD can change MSTF flags but will have no effect on whether the MBCNDD instruction branches or not. This is because the flag modification will occur after the D2 phase of the branch, call or return instruction. These three instructions must not be a MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD, or MRCNDD.

  – **I5, I6, and I7**

    The three instructions following a branch, call or return are always executed irrespective of whether the condition is true or not. These instructions must not be MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

  For a more detailed description refer to the functional description for MBCNDD, MCCNDD, and MRCNDD.

Example 10-1. Code Fragment For MBCNDD, MCCNDD, or MRCNDD

```
<Instruction 1>    ; I1 Last instruction that can affect flags for
                   ;     the branch, call or return operation
<Instruction 2>    ; I2 Cannot be stop, branch, call or return
<Instruction 3>    ; I3 Cannot be stop, branch, call or return
<Instruction 4>    ; I4 Cannot be stop, branch, call or return
<branch/call/ret>  ; MBCNDD, MCCNDD or MRCNDD
                   ; I5-I7: Three instructions after are always
                   ; executed whether the branch/call or return is
                   ; taken or not
<Instruction 5>    ; I5 Cannot be stop, branch, call or return
<Instruction 6>    ; I6 Cannot be stop, branch, call or return
<Instruction 7>    ; I7 Cannot be stop, branch, call or return
<Instruction 8>    ; I8
<Instruction 9>    ; I9
....
```

- **Stop or Halting a Task: MSTOP and MDEBUGSTOP**

  The MSTOP and MDEBUGSTOP instructions cannot be placed three instructions before or after a conditional branch, call or return instruction (MBCNDD, MCCNDD, or MRCNDD). Refer to Example 10-1. To single-step through a branch/call or return, insert the MDEBUGSTOP at least four instructions back and step from there.

- **Loading MAR0 or MAR1**

  A load of auxiliary register MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Referring to Example 10-2, the following applies when loading the auxiliary registers:

  – **I1 and I2**

    The two instructions following the load instruction will use the value in MAR0 or MAR1 before the update occurs.

  – **I3**

    Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win and the auxiliary register will not be updated with #_X.

  – **I4**

    Starting with the 4th instruction MAR0 or MAR1 will have the new value.

Example 10-2. Code Fragment for Loading MAR0 or MAR1

```
; Assume MAR0 is 50 and #_X is 20

   MMOVI16 MAR0, #_X ; Load MAR0 with address of X (20)
   <Instruction 1>   ; I1 Will use the old value of MAR0 (50)
   <Instruction 2>   ; I2 Will use the old value of MAR0 (50)
   <Instruction 3>   ; I3 Cannot use MAR0
   <Instruction 4>   ; I4 Will use the new value of MAR0 (20)
   <Instruction 5>   ; I5 Will use the new value of MAR0 (20
   ....
```

10.5.2.1 ADC Early Interrupt to CLA Response

The ADC can be configured to generate an early interrupt pulse before the ADC conversion completes. If this option is used to start a CLA task, the CLA will be able to read the result as soon as the conversion result is available in the ADC result register. This combination of just-in-time sampling along with the low interrupt response of the CLA enable faster system response and higher frequency control loops. The CLA task trigger to first instruction fetch interrupt latency is 4 cycles.

Timings for ADC conversions are shown in the timing diagrams of the ADC chapter. If the ADCCLK is a divided down version of the SYSCLK, the user will have to account for the conversion time in SYSCLK cycles.

For example, if using the ADC with ADCCLK at SYSCLK / 2, it would take 13 ADCCLK x 2 SYSCLK = 26 SYSCLK cycles to complete a conversion.

From a CLA perspective, the pipeline activity is shown in Table 10-3 for an N-cycle (SYSCLK) ADC conversion. The N-2 instruction will arrive in the R2 phase just in time to read the result register. While the prior instructions will enter the R2 phase of the pipeline too soon to read the conversion, they can be efficiently used for pre-processing calculations needed by the task.

Table 10-3. ADC to CLA Early Interrupt Response

ADC Activity	CLA Activity	F1	F2	D1	D2	R1	R2	E	W
Sample									
Sample									
...									
Sample									
$Conversion_{(Cycle\ 1)}$	Interrupt Received								
$Conversion_{(Cycle\ 2)}$	Task Startup								
$Conversion_{(Cycle\ 3)}$	Task Startup								
$Conversion_{(Cycle\ 4)}$	$I_{(Cycle\ 4)}$	$I_{(Cycle\ 4)}$							
$Conversion_{(Cycle\ 5)}$	$I_{(Cycle\ 5)}$	$I_{(Cycle\ 5)}$	$I_{(Cycle\ 4)}$						
$Conversion_{(...)}$	...	...	...	...	...	...	...		
$Conversion_{(Cycle\ N-6)}$	$I_{(Cycle\ N-6)}$	$I_{(Cycle\ N-6)}$	$I_{(Cycle\ N-7)}$	$I_{(Cycle\ N-8)}$	$I_{(Cycle\ N-9)}$	$I_{(Cycle\ N-10)}$	$I_{(Cycle\ N-11)}$		
$Conversion_{(Cycle\ N-5)}$	$I_{(Cycle\ N-5)}$	$I_{(Cycle\ N-5)}$	$I_{(Cycle\ N-6)}$	$I_{(Cycle\ N-7)}$	$I_{(Cycle\ N-8)}$	$I_{(Cycle\ N-9)}$	$I_{(Cycle\ N-10)}$		
$Conversion_{(Cycle\ N-4)}$	$I_{(Cycle\ N-4)}$	$I_{(Cycle\ N-4)}$	$I_{(Cycle\ N-5)}$	$I_{(Cycle\ N-6)}$	$I_{(Cycle\ N-7)}$	$I_{(Cycle\ N-8)}$	$I_{(Cycle\ N-9)}$		
$Conversion_{(Cycle\ N-3)}$	$I_{(Cycle\ N-3)}$	$I_{(Cycle\ N-3)}$	$I_{(Cycle\ N-4)}$	$I_{(Cycle\ N-5)}$	$I_{(Cycle\ N-6)}$	$I_{(Cycle\ N-7)}$	$I_{(Cycle\ N-8)}$		
$Conversion_{(Cycle\ N-2)}$	**Read RESULT**	**Read RESULT**	$I_{(Cycle\ N-3)}$	$I_{(Cycle\ N-4)}$	$I_{(Cycle\ N-5)}$	$I_{(Cycle\ N-6)}$	$I_{(Cycle\ N-7)}$		
$Conversion_{(Cycle\ N-1)}$			**Read RESULT**	$I_{(Cycle\ N-3)}$	$I_{(Cycle\ N-4)}$	$I_{(Cycle\ N-5)}$	$I_{(Cycle\ N-6)}$		
$Conversion_{(Cycle\ N-0)}$				**Read RESULT**	$I_{(Cycle\ N-3)}$	$I_{(Cycle\ N-4)}$	$I_{(Cycle\ N-5)}$		
Conversion Complete					**Read RESULT**	$I_{(Cycle\ N-3)}$	$I_{(Cycle\ N-4)}$		
RESULT Latched						**Read RESULT**	$I_{(Cycle\ N-3)}$		
RESULT Available							**Read RESULT**		

10.5.3 Parallel Instructions

Parallel instructions are single opcodes that perform two operations in parallel. The following types of parallel instructions are available: math operation in parallel with a move operation, or two math operations in parallel. Both operations complete in a single cycle and there are no special pipeline alignment requirements.

Example 10-3. Math Operation with Parallel Load

```
;  MADDF32 || MMOV32 instruction: 32-bit floating-point add with parallel move
;  MADDF32 is a 1 cycle operation
;  MMOV32 is a 1 cycle operation
         MADDF32    MR0,  MR1, #2    ; MR0 = MR1 + 2,
      || MMOV32   MR1,  @Val         ; MR1 gets the contents of Val
                                     ; <-- MMOV32 completes here (MR1 is valid)
                                     ; <-- DDF32 completes here (MR0 is valid)
         MMPYF32 MR0, MR0, MR1       ; Any instruction, can use MR1 and/or MR0
```

Example 10-4. Multiply with Parallel Add

```
;  MMPYF32 || MADDF32 instruction: 32-bit floating-point multiply with parallel add
;  MMPYF32 is a 1 cycle operation
;  MADDF32 is a 1 cycle operation
     MMPYF32 MR0, MR1, MR3        ; MR0 = MR1 * MR3
 ||  MADDF32 MR1, MR2, MR0        ; MR1 = MR2 + MR0 (Uses value of MR0 before MMPYF32)
                                  ; <-- MMPYF32 and MADDF32 complete here (MR0 and MR1 are valid)
     MMPYF32 MR1, MR1, MR0        ; Any instruction, can use MR1 and/or MR0
```

10.6 Instruction Set

This section describes the assembly language instructions of the control law accelerator. Also described are parallel operations, conditional operations, resource constraints, and addressing modes. The instructions listed here are independent from C28x and C28x+FPU instruction sets.

10.6.1 Instruction Descriptions

This section gives detailed information on the instruction set. Each instruction may present the following information:

- Operands
- Opcode
- Description
- Exceptions
- Pipeline
- Examples
- See also

The example INSTRUCTION is shown to familiarize you with the way each instruction is described. The example describes the kind of information you will find in each part of the individual instruction description and where to obtain more information. CLA instructions follow the same format as the C28x; the source operand(s) are always on the right and the destination operand(s) are on the left.

The explanations for the syntax of the operands used in the instruction descriptions for the C28x CLA are given in Table 10-4.

Table 10-4. Operand Nomenclature

Symbol	Description
#16FHi	16-bit immediate (hex or float) value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FHiHex	16-bit immediate hex value that represents the upper 16-bits of an IEEE 32-bit floating-point value. Lower 16-bits of the mantissa are assumed to be zero.
#16FLoHex	A 16-bit immediate hex value that represents the lower 16-bits of an IEEE 32-bit floating-point value
#32Fhex	32-bit immediate value that represents an IEEE 32-bit floating-point value
#32F	Immediate float value represented in floating-point representation
#0.0	Immediate zero
#SHIFT	Immediate value of 1 to 32 used for arithmetic and logical shifts.
addr	Opcode field indicating the addressing mode
CNDF	Condition to test the flags in the MSTF register
FLAG	Selected flags from MSTF register (OR) 8 bit mask indicating which floating-point status flags to change
MAR0	auxiliary register 0
MAR1	auxiliary register 1
MARx	Either MAR0 or MAR1
mem16	16-bit memory location accessed using direct, indirect, or offset addressing modes
mem32	32-bit memory location accessed using direct, indirect, or offset addressing modes
MRa	MR0 to MR3 registers
MRb	MR0 to MR3 registers
MRc	MR0 to MR3 registers
MRd	MR0 to MR3 registers
MRe	MR0 to MR3 registers
MRf	MR0 to MR3 registers
MSTF	CLA Floating-point Status Register
shift	Opcode field indicating the number of bits to shift.

Table 10-4. Operand Nomenclature (continued)

Symbol	Description
VALUE	Flag value of 0 or 1 for selected flag (OR) 8 bit mask indicating the flag value; 0 or 1

Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

Table 10-5. INSTRUCTION dest, source1, source2 Short Description

	Description
dest1	Description for the 1st operand for the instruction
source1	Description for the 2nd operand for the instruction
source2	Description for the 3rd operand for the instruction
Opcode	This section shows the opcode for the instruction
Description	Detailed description of the instruction execution is described. Any constraints on the operands imposed by the processor or the assembler are discussed.
Restrictions	Any constraints on the operands or use of the instruction imposed by the processor are discussed.
Pipeline	This section describes the instruction in terms of pipeline cycles as described in Section 10.5
Example	Examples of instruction execution. If applicable, register and memory values are given before and after instruction execution. Some examples are code fragments while other examples are full tasks that assume the CLA is correctly configured and the main CPU has passed it data.
Operands	Each instruction has a table that gives a list of the operands and a short description. Instructions always have their destination operand(s) first followed by the source operand(s).

10.6.2 Addressing Modes and Encoding

The CLA uses the same address to access data and registers as the main CPU. For example if the main CPU accesses an ePWM register at address 0x00 6800, then the CLA will access it using address 0x6800. Since all CLA accessible memory and registers are within the low 64k x 16 of memory, only the low 16-bits of the address are used by the CLA.

To address the CLA data memory, message RAMs and shared peripherals, the CLA supports two addressing modes:

- Direct addressing mode: Uses the address of the variable or register directly.
- Indirect addressing with 16-bit post increment. This mode uses either XAR0 or XAR1.

The CLA does not use a data page pointer or a stack pointer. The two addressing modes are encoded as shown Table 10-6.

Table 10-6. Addressing Modes

Addressing Mode	'addr' Opcode Field Encode(1)	Description
@dir	**0000**	**Direct Addressing Mode** Example 1: MMOV32 MR1, @_VarA Example 2: MMOV32 MR1, @_EPwm1Regs.CMPA.all In this case the 'mmmm mmmm mmmm mmmm' opcode field will be populated with the 16-bit address of the variable. This is the low 16-bits of the address that you would use to access the variable using the main CPU. For example @_VarA will populate the address of the variable VarA. and @_EPwm1Regs.CMPA.all will populate the address of the CMPA register.
***MAR0[#imm16]++**	**0001**	**MAR0 Indirect Addressing with 16-bit Immediate Post Increment**
***MAR1[#imm16]++**	**0010**	**MAR1 Indirect Addressing with 16-bit Immediate Post Increment** addr = MAR0 (or MAR1) Access memory using the address stored in MAR0 (or MAR1). MAR0 (or MAR1) += #imm16 Then post increment MAR0 (or MAR1) by #imm16. Example 1: MMOV32 MR0, *MAR0[2]++ Example 2: MMOV32 MR1, *MAR1[-2]++ For a post increment of 0 the assembler will accept both *MAR0 and *MAR0[0]++. The 'mmmm mmmm mmmm mmmm' opcode field will be populated with the signed 16-bit pointer offset. For example if #imm16 is 2, then the opcode field will be 0x0002. Likewise if #imm16 is -2, then the opcode field will be 0xFFFE. If addition of the 16-bit immediate causes overflow, then the value will wrap around on a 16-bit boundary.

(1) Values not shown are reserved.

Encoding for the shift fields in the MASR32, MLSR32 and MLSL32 instructions is shown in Table 10-7.

Table 10-7. Shift Field Encoding

Shift Value	'shift' Opcode Field Encode
1	0000
2	0001
3	0010
....	....
32	1111

For instructions that use MRx (where x could be 'a' through 'f') as operands, the trailing alphabet appears in the opcode as a two-bit field. For example:

```
MMPYF32 MRa, MRb, MRc ||
MADDF32 MRd, MRe, MRf
```

whose opcode is,

```
LSW: 0000 ffee ddcc bbaa
MSW: 0111 1010 0000 0000
```

The two-bit field specifies one of four working registers according to Table 10-8.

Table 10-8. Operand Encoding

Two-Bit Field	Working Register
00	MR0
01	MR1
10	MR2
11	MR3

Table 10-9 shows the condition field encoding for conditional instructions such as MNEGF, MSWAPF, MBCNDD, MCCNDD, and MRCNDD.

Table 10-9. Condition Field Encoding

Encode(1)	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF(2)	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

10.6.3 Instructions

The instructions are listed alphabetically.

Instruction Set Summary

MABSF32 MRa, MRb

32-Bit Floating-Point Absolute Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0010 0000
```

Description

The absolute value of MRb is loaded into MRa. Only the sign bit of the operand is modified by the MABSF32 instruction.

```
if (MRb < 0) {MRa = -MRb};
        else {MRa =  MRb};
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
NF = 0;
ZF = 0;
if ( MRa(30:23) == 0) ZF = 1;
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ MR0, #-2.0 ; MR0 = -2.0 (0xC0000000)
MABSF32 MR0, MR0  ; MR0 = 2.0 (0x40000000), ZF = NF = 0
MMOVIZ MR0, #5.0  ; MR0 = 5.0 (0x40A00000)
MABSF32 MR0, MR0  ; MR0 = 5.0 (0x40A00000), ZF = NF = 0
MMOVIZ MR0, #0.0  ; MR0 = 0.0
MABSF32 MR0, MR0  ; MR0 = 0.0 ZF = 1, NF = 0
```

See also

MNEGF32 MRa, MRb {, CNDF}

MADD32 MRa, MRb, MRc

32-Bit Integer Add

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point destination register (MR0 to MR3)
MRc	CLA floating-point destination register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 000cc bbaa
MSW: 0111 1110 1100 0000
```

Description

32-bit integer addition of MRb and MRc.

```
MRa(31:0) = MRb(31:0) + MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; };
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
; Calculate Y2 = A + B + C
;
_Cla1Task1:
       MMOV32 MR0, @_A      ; MR0 = 1 (0x00000001)
       MMOV32 MR1, @_B      ; MR1 = 2 (0x00000002)
       MMOV32 MR2, @_C      ; MR2 = -7 (0xFFFFFFF9)
       MADD32 MR3, MR0, MR1 ; A + B
       MADD32 MR3, MR2, MR3 ; A + B + C = -4 (0xFFFFFFFC)
       MMOV32 @_y2, MR3     ; Store y2
       MSTOP                ; end of task
```

See also

MAND32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MADDF32 MRa, #16FHi, MRb

32-Bit Floating-Point Addition

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1100 bbaa
```

Description

Add MRb to the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb + #16FHi:0;
```

This instruction can also be written as MADDF32 MRa, MRb, #16FHi.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Add to MR1 the value 2.0 in 32-bit floating-point format
; Store the result in MR0
  MADDF32 MR0, #2.0, MR1    ; MR0 = 2.0 + MR1
; Add to MR3 the value -2.5 in 32-bit floating-point format
; Store the result in MR2
  MADDF32 MR2, #-2.5, MR3   ; MR2 = -2.5 + MR3
; Add to MR3 the value 0x3FC00000 (1.5)
; Store the result in MR3
  MADDF32 MR3, #0x3FC0, MR3 ; MR3 = 1.5 + MR3
```

See also

MADDF32 MRa, MRb, #16FHi
MADDF32 MRa, MRb, MRc
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MADDF32 MRa, MRb, #16FHi

32-Bit Floating-Point Addition

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1100 bbaa
```

Description

Add MRb to the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb + #16FHi:0;
```

This instruction can also be written as MADDF32 MRa, #16FHi, MRb.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

MADDF32 MRa, MRb, #16FHi (continued)

32-Bit Floating-Point Addition

Example 1

```
; X is an array of 32-bit floating-point values
; Find the maximum value in an array X
; and store it in Result
;
_Cla1Task1:
    MMOVI16    MAR1,#_X          ; Start address
    MUI16TOF32 MR0, @_len        ; Length of the array
    MNOP                         ; delay for MAR1 load
    MNOP                         ; delay for MAR1 load
    MMOV32     MR1, *MAR1[2]++   ; MR1 = X0
LOOP
    MMOV32     MR2, *MAR1[2]++   ; MR2 = next element
    MMAXF32    MR1,  MR2         ; MR1 = MAX(MR1, MR2)
    MADDF32    MR0, MR0, #-1.0   ; Decrement the counter
    MCMPF32    MR0 #0.0          ; Set/clear flags for MBCNDD
    MNOP
    MNOP
    MNOP
    MBCNDD LOOP, NEQ             ; Branch if not equal to zero
    MMOV32 @_Result, MR1         ; Always executed
    MNOP                         ; Always executed
    MNOP                         ; Always executed
    MSTOP                        ; End of task
```

Example 2

```
; Show the basic operation of MADDF32
;
; Add to MR1 the value 2.0 in 32-bit floating-point format
; Store the result in MR0
    MADDF32 MR0, MR1, #2.0    ; MR0 = MR1 + 2.0
; Add to MR3 the value -2.5 in 32-bit floating-point format
; Store the result in MR2
    MADDF32 MR2, MR3, #-2.5   ; MR2 = MR3 + (-2.5)
; Add to MR0 the value 0x3FC00000 (1.5)
; Store the result in MR0
   MADDF32 MR0, MR0, #0x3FC0  ; MR0 = MR0 + 1.5
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, MRc
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MADDF32 MRa, MRb, MRc

32-Bit Floating-Point Addition

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 000 0000 00cc bbaa
MSW: 0111 1100 0010 0000
```

Description

Add the contents of MRc to the contents of MRb and load the result into MRa.

```
MRa = MRb + MRc;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Given M1, X1 and B1 are 32-bit floating point numbers
; Calculate Y1 = M1*X1+B1
;
_Cla1Task1:
   MMOV32 MR0,@M1       ; Load MR0 with M1
   MMOV32 MR1,@X1       ; Load MR1 with X1
   MMPYF32 MR1,MR1,MR0  ; Multiply M1*X1
|| MMOV32 MR0,@B1       ; and in parallel load MR0 with B1
   MADDF32 MR1,MR1,MR0  ; Add M*X1 to B1 and store in MR1
   MMOV32 @Y1,MR1       ; Store the result
   MSTOP                ; end of task
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, #16FHi
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MADDF32 MRd, MRe, MRf||MMOV32 mem32, MRa

32-Bit Floating-Point Addition with Parallel Move

Operands

MRd	CLA floating-point destination register for the MADDF32 (MR0 to MR3)
MRe	CLA floating-point source register for the MADDF32 (MR0 to MR3)
MRf	CLA floating-point source register for the MADDF32 (MR0 to MR3)
mem32	32-bit memory location accessed using one of the available addressing modes. This will be the destination of the MMOV32.
MRa	CLA floating-point source register for the MMOV32 (MR0 to MR3)

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0101 ffee ddaa addr
```

Description

Perform an MADDF32 and a MMOV32 in parallel. Add MRf to the contents of MRe and store the result in MRd. In parallel move the contents of MRa to the 32-bit location mem32.

```
MRd = MRe + MRf;
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

Pipeline

Both MADDF32 and MMOV32 complete in a single cycle.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A * B) + C
;
_Cla1Task2:
   MMOV32   MR0, @_A         ; Load MR0 with A
   MMOV32   MR1, @_B         ; Load MR1 with B
   MMPYF32  MR1, MR1, MR0    ; Multiply A*B
|| MMOV32   MR0, @_C         ;     and in parallel load MR0 with C
   MADDF32  MR1, MR1, MR0    ; Add (A*B) to C
|| MMOV32   @_Y2, MR1        ;     and in parallel store A*B
   MMOV32   @_Y3, MR1        ; Store the A*B + C
   MSTOP                     ; end of task
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, #16FHi
MADDF32 MRa, MRb, MRc
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf
MADDF32 MRd, MRe, MRf || MMOV32 MRa, mem32

MADDF32 MRd, MRe, MRf ||MMOV32 MRa, mem32

32-Bit Floating-Point Addition with Parallel Move

Operands

MRd	CLA floating-point destination register for the MADDF32 (MR0 to MR3). MRd cannot be the same register as MRa.
MRe	CLA floating-point source register for the MADDF32 (MR0 to MR3)
MRf	CLA floating-point source register for the MADDF32 (MR0 to MR3)
MRa	CLA floating-point destination register for the MMOV32 (MR0 to MR3). MRa cannot be the same register as MRd.
mem32	32-bit memory location accessed using one of the available addressing modes. This is the source for the MMOV32.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0001 ffee ddaa addr
```

Description

Perform an MADDF32 and a MMOV32 operation in parallel. Add MRf to the contents of MRe and store the result in MRd. In parallel move the contents of the 32-bit location mem32 to MRa.

```
MRd = MRe + MRf;
MRa = [mem32];
```

Restrictions

The destination register for the MADDF32 and the MMOV32 must be unique. That is, MRa and MRd cannot be the same register.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MADDF32 generates an underflow condition.
- LVF = 1 if MADDF32 generates an overflow condition.

The MMOV32 Instruction will set the NF and ZF flags as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; };
```

Pipeline

The MADDF32 and the MMOV32 both complete in a single cycle.

MADDF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 (continued)

32-Bit Floating-Point Addition with Parallel Move

Example 1

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y1 = A + 4B
;           Y2 = A + C
;
_Cla1Task1:
    MMOV32 MR0, @A          ; Load MR0 with A
    MMOV32 MR1, @B          ; Load MR1 with B
    MMPYF32 MR1, MR1, #4.0 ; Multiply 4 * B
||  MMOV32 MR2, @C            and in parallel load C
    MADDF32 MR3, MR0, MR1   ; Add A + 4B
    MADDF32 MR3, MR0, MR2   ; Add A + C
||  MMOV32 @Y1, MR3         ; and in parallel store A+4B
    MMOV32 @Y2, MR3         ; store A + C MSTOP
                            ; end of task
```

Example 2

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y3 = (A + B)
;           Y4 = (A + B) * C
;
_Cla1Task2:
     MMOV32 MR0, @A          ; Load MR0 with A
     MMOV32 MR1, @B          ; Load MR1 with B
     MADDF32 MR1, MR1, MR0   ; Add A+B
||   MMOV32 MR0, @C          ; and in parallel load MR0 with C
     MMPYF32 MR1, MR1, MR0   ; Multiply (A+B) by C
||   MMOV32 @Y3, MR1         ; and in parallel store A+B
     MMOV32 @Y4, MR1         ; Store the (A+B) * C
     MSTOP                   ; end of task
```

See also

MADDF32 MRa, #16FHi, MRb
MADDF32 MRa, MRb, #16FHi
MADDF32 MRa, MRb, MRc
MADDF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MAND32 MRa, MRb, MRc

Bitwise AND

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 0110 0000
```

Description

Bitwise AND of MRb with MRc.

```
MRa(31:0) = MRb(31:0) AND MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ   MR0,    #0x5555  ; MR0 = 0x5555AAAA
MMOVXI   MR0,    #0xAAAA
MMOVIZ   MR1,    #0x5432  ; MR1 = 0x5432FEDC
MMOVXI   MR1,    #0xFEDC
; 0101 AND 0101 = 0101 (5)
; 0101 AND 0100 = 0100 (4)
; 0101 AND 0011 = 0001 (1)
; 0101 AND 0010 = 0000 (0)
; 1010 AND 1111 = 1010 (A)
; 1010 AND 1110 = 1010 (A)
; 1010 AND 1101 = 1000 (8)
; 1010 AND 1100 = 1000 (8)
MAND32 MR2, MR1, MR0      ; MR3 = 0x5410AA88
```

See also

MADD32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MASR32 MRa, #SHIFT

Arithmetic Shift Right

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#SHIFT	Number of bits to shift (1 to 32)

Opcode

```
LSW: 0000 0000 0shi ftaa
MSW: 0111 1011 0100 0000
```

Description

Arithmetic shift right of MRa by the number of bits indicated. The number of bits can be 1 to 32.

```
MARa(31:0) = Arithmetic Shift(MARa(31:0) by #SHIFT bits);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given m2 = (int32)32
;       x2 = (int32)64
;       b2 = (int32)-128
;
; Calculate
;       m2 = m2/2
;       x2 = x2/4
;       b2 = b2/8
;
_Cla1Task2:
   MMOV32 MR0, @_m2 ; MR0 = 32 (0x00000020)
   MMOV32 MR1, @_x2 ; MR1 = 64 (0x00000040)
   MMOV32 MR2, @_b2 ; MR2 = -128 (0xFFFFFF80)
   MASR32 MR0, #1   ; MR0 = 16 (0x00000010)
   MASR32 MR1, #2   ; MR1 = 16 (0x00000010)
   MASR32 MR2, #3   ; MR2 = -16 (0xFFFFFFF0)
   MMOV32 @_m2, MR0 ; store results
   MMOV32 @_x2, MR1
   MMOV32 @_b2, MR2
   MSTOP ; end of task
```

See also

MADD32 MRa, MRb, MRc
MAND32 MRa, MRb, MRc
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MBCNDD 16BitDest {, CNDF}

Branch Conditional Delayed

Operands

16BitDest	16-bit destination if condition is true
CNDF	Optional condition tested

Opcode

```
LSW: dest dest dest dest
MSW: 0111 1001 1000 cndf
```

Description

If the specified condition is true, then branch by adding the signed 16BitDest value to the MPC value. Otherwise, continue without branching. If the address overflows, it wraps around. Therefore a value of "0xFFFE" will put the MPC back to the MBCNDD instruction.

Please refer to the pipeline section for important information regarding this instruction.

```
if (CNDF == TRUE) MPC += 16BitDest;
```

CNDF is one of the following conditions:

Encode(1)	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF(2)	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation, if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Restrictions

The MBCNDD instruction is not allowed three instructions before or after a MBCNDD, MCCNDD or MRCNDD instruction. Refer to the pipeline section for more information.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

MBCNDD 16BitDest {, CNDF} (continued)

Branch Conditional Delayed

Pipeline

The MBCNDD instruction by itself is a single-cycle instruction. As shown in Table 10-10 for each branch 6 instruction slots are executed; three before the branch instruction (I2-I4) and three after the branch instruction (I5-I7). The total number of cycles for a branch taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a branch can, therefore, range from 1 to 7 cycles. The number of cycles for a branch taken may not be the same as for a branch not taken.

Referring to Table 10-10 and Table 10-11, the instructions before and after MBCNDD have the following properties:

- **I1**
  - I1 is the last instruction that can effect the CNDF flags for the MBCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MBCNDD is in the D2 phase.
  - There are no restrictions on the type of instruction for I1.
- **I2, I3 and I4**
  - The three instructions proceeding MBCNDD can change MSTF flags but will have no effect on whether the MBCNDD instruction branches or not. This is because the flag modification will occur after the D2 phase of the MBCNDD instruction.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.
- **I5, I6 and I7**
  - The three instructions following MBCNDD are always executed irrespective of whether the branch is taken or not.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

```
<Instruction 1>   ; I1 Last instruction that can affect flags for
                  ; the MBCNDD operation
<Instruction 2>   ; I2 Cannot be stop, branch, call or return
<Instruction 3>   ; I3 Cannot be stop, branch, call or return
<Instruction 4>   ; I4 Cannot be stop, branch, call or return
MBCNDD _Skip, NEQ ; Branch to Skip if not eqal to zero
                  ; Three instructions after MBCNDD are always
                  ; executed whether the branch is taken or not
<Instruction 5>   ; I5 Cannot be stop, branch, call or return
<Instruction 6>   ; I6 Cannot be stop, branch, call or return
<Instruction 7>   ; I7 Cannot be stop, branch, call or return
<Instruction 8>   ; I8
<Instruction 9>   ; I9
....
_Skip:
 <Destination 1>  ; d1 Can be any instruction
 <Destination 2>  ; d2
 <Destination 3>  ; d3
....
....
MSTOP
....
```

MBCNDD 16BitDest {, CNDF} (continued)

Branch Conditional Delayed

Table 10-10. Pipeline Activity For MBCNDD, Branch Not Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MBCNDD	MBCNDD	I4	I3	I2	I1			
I5	I5	MBCNDD	I4	I3	I2	I1		
I6	I6	I5	MBCNDD	I4	I3	I2	I1	
I7	I7	I6	I5	MBCNDD	I4	I3	I2	
I8	I8	I7	I6	I5	-	I4	I3	
I9	I9	I8	I7	I6	I5	-	I4	
I10	I10	I9	I8	I7	I6	I5	-	
		I10	I9	I8	I7	I6	I5	
			I10	I9	I8	I7	I6	
				I10	I9	I8	I7	
					I10	I9	I8	
						I10	I9	
							I10	

Table 10-11. Pipeline Activity For MBCNDD, Branch Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MBCNDD	MBCNDD	I4	I3	I2	I1			
I5	I5	MBCNDD	I4	I3	I2	I1		
I6	I6	I5	MBCNDD	I4	I3	I2	I1	
I7	I7	I6	I5	MBCNDD	I4	I3	I2	
d1	d1	I7	I6	I5	-	I4	I3	
d2	d2	d1	I7	I6	I5	-	I4	
d3	d3	d2	d1	I7	I6	I5	-	
		d3	d2	d1	I7	I6	I5	
			d3	d2	d1	I7	I6	
				d3	d2	d1	I7	
					d3	d2	d1	
						d3	d2	
							d3	

MBCNDD 16BitDest {, CNDF} (continued)

Branch Conditional Delayed

Example 1

```
; if (State == 0.1)
; RampState = RampState || RAMPMASK
; else if (State == 0.01)
; CoastState = CoastState || COASTMASK
; else
; SteadyState = SteadyState || STEADYMASK
;
_Cla1Task1:
 MMOV32 MR0, @State
 MCMPF32 MR0, #0.1        ; Affects flags for 1st MBCNDD (A)
 MNOP
 MNOP
 MNOP
 MBCNDD Skip1, NEQ        ; (A) If State != 0.1, go to Skip1
 MNOP ; Always executed
 MNOP ; Always executed
 MNOP ; Always executed
 MMOV32 MR1, @RampState   ; Execute if (A) branch not taken
 MMOVXI MR2, #RAMPMASK    ; Execute if (A) branch not taken
 MOR32 MR1, MR2           ; Execute if (A) branch not taken
 MMOV32 @RampState, MR1   ; Execute if (A) branch not taken
 MSTOP                    ; end of task if (A) branch not taken
Skip1:
 MCMPF32 MR0,#0.01        ; Affects flags for 2nd MBCNDD (B)
 MNOP
 MNOP
 MNOP
 MBCNDD Skip2,NEQ         ; (B) If State != 0.01, go to Skip2
 MNOP ; Always executed
 MNOP ; Always executed
 MNOP ; Always executed
 MMOV32 MR1, @CoastState  ; Execute if (B) branch not taken
 MMOVXI MR2, #COASTMASK   ; Execute if (B) branch not taken
 MOR32 MR1, MR2           ; Execute if (B) branch not taken
 MMOV32 @CoastState, MR1  ; Execute if (B) branch not taken
 MSTOP
Skip2:
 MMOV32 MR3, @SteadyState ; Executed if (B) branch taken
 MMOVXI MR2, #STEADYMASK  ; Executed if (B) branch taken
 MOR32 MR3, MR2           ; Executed if (B) branch taken
 MMOV32 @SteadyState, MR3 ; Executed if (B) branch taken
 MSTOP
```

MBCNDD 16BitDest {, CNDF} (continued)

Branch Conditional Delayed

Example 2

```
; This example is the same as Example 1, except
; the code is optimized to take advantage of delay slots
;
; if (State == 0.1)
; RampState = RampState || RAMPMASK
; else if (State == 0.01)
; CoastState = CoastState || COASTMASK
; else
; SteadyState = SteadyState || STEADYMASK
;
_Cla1Task2:
 MMOV32 MR0, @State
 MCMPF32 MR0, #0.1          ; Affects flags for 1st MBCNDD (A)
 MCMPF32 MR0, #0.01         ; Check used by 2nd MBCNDD (B)
 MTESTTF EQ                 ; Store EQ flag in TF for 2nd MBCNDD (B)
 MNOP
 MBCNDD Skip1, NEQ          ; (A) If State != 0.1, go to Skip1
 MMOV32 MR1, @RampState     ; Always executed
 MMOVXI MR2, #RAMPMASK      ; Always executed
 MOR32 MR1, MR2             ; Always executed
 MMOV32 @RampState, MR1     ; Execute if (A) branch not taken
 MSTOP                      ; end of task if (A) branch not taken
Skip1:
 MMOV32 MR3, @SteadyState
 MMOVXI MR2, #STEADYMASK
 MOR32 MR3, MR2
 MBCNDD Skip2, NTF          ; (B) if State != .01, go to Skip2
 MMOV32 MR1, @CoastState    ; Always executed
 MMOVXI MR2, #COASTMASK     ; Always executed
 MOR32 MR1, MR2             ; Always executed
 MMOV32 @CoastState, MR1    ; Execute if (B) branch not taken
 MSTOP                      ; end of task if (B) branch not taken
Skip2:
 MMOV32 @SteadyState, MR3   ; Executed if (B) branch taken
 MSTOP
```

See also

MCCNDD 16BitDest, CNDF
MRCNDD CNDF

MCCNDD 16BitDest {, CNDF}

Call Conditional Delayed

Operands

16BitDest	16-bit destination if condition is true
CNDF	Optional condition to be tested

Opcode

```
LSW: dest dest dest dest
MSW: 0111 1001 1001 cndf
```

Description

If the specified condition is true, then store the return address in the RPC field of MSTF and make the call by adding the signed 16BitDest value to the MPC value. Otherwise, continue code execution without making the call. If the address overflows, it wraps around. Therefore a value of "0xFFFE" will put the MPC back to the MCCNDD instruction.

Please refer to the pipeline section for important information regarding this instruction.

```
if (CNDF == TRUE)
{
    RPC = return address;
    MPC += 16BitDest;
};
```

CNDF is one of the following conditions:

Encode[1]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF[2]	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Restrictions

The MCCNDD instruction is not allowed three instructions before or after a MBCNDD, MCCNDD, or MRCNDD instruction. Refer to the Pipeline section for more details.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

MCCNDD 16BitDest {, CNDF} (continued)

Call Conditional Delayed

Pipeline

The MCCNDD instruction by itself is a single-cycle instruction. As shown in Table 10-12, for each call 6 instruction slots are executed; three before the call instruction (I2-I4) and three after the call instruction (I5-I7). The total number of cycles for a call taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a call can, therefore, range from 1 to 7 cycles. The number of cycles for a call taken may not be the same as for a call not taken.

Referring to the following code fragment and the pipeline diagrams in Table 10-12 and Table 10-13, the instructions before and after MCCNDD have the following properties:

- **I1**
  - I1 is the last instruction that can effect the CNDF flags for the MCCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to branch or not when MCCNDD is in the D2 phase.
  - There are no restrictions on the type of instruction for I1.
- **I2, I3 and I4**
  - The three instructions proceeding MCCNDD can change MSTF flags but will have no effect on whether the MCCNDD instruction makes the call or not. This is because the flag modification will occur after the D2 phase of the MCCNDD instruction.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.
- **I5, I6 and I7**
  - The three instructions following MBCNDD are always executed irrespective of whether the branch is taken or not.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

```
<Instruction 1>   ; I1 Last instruction that can affect flags for
                  ;    the MCCNDD operation
<Instruction 2>   ; I2 Cannot be stop, branch, call or return
<Instruction 3>   ; I3 Cannot be stop, branch, call or return
<Instruction 4>   ; I4 Cannot be stop, branch, call or return
MCCNDD _func, NEQ ; Call to func if not eqal to zero
                  ; Three instructions after MCCNDD are always
                  ; executed whether the call is taken or not
<Instruction 5>   ; I5 Cannot be stop, branch, call or return
<Instruction 6>   ; I6 Cannot be stop, branch, call or return
<Instruction 7>   ; I7 Cannot be stop, branch, call or return
<Instruction 8>   ; I8 The address of this instruction is saved
                  ;    in the RPC field of the MSTF register.
                  ;    Upon return this value is loaded into MPC
                  ;    and fetching continues from this point.
<Instruction 9>   ; I9
....
_func:
<Destination 1>   ; d1 Can be any instruction
<Destination 2>   ; d2
<Destination 3>   ; d3
<Destination 4>   ; d4 Last instruction that can affect flags for
                  ;    the MRCNDD operation
<Destination 5>   ; d5 Cannot be stop, branch, call or return
<Destination 6>   ; d6 Cannot be stop, branch, call or return
<Destination 7>   ; d7 Cannot be stop, branch, call or return
MRCNDD UNC        ; Return to <Instruction 8>, unconditional
                  ; Three instructions after MRCNDD are always
                  ; executed whether the return is taken or not
<Destination 8>   ; d8 Cannot be stop, branch, call or return
<Destination 9>   ; d9 Cannot be stop, branch, call or return
```

MCCNDD 16BitDest {, CNDF} (continued)

Call Conditional Delayed

```
<Destination 10>  ; d10 Cannot be stop, branch, call or return
<Destination 11>  ; d11
....
MSTOP
```

Table 10-12. Pipeline Activity For MCCNDD, Call Not Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MCCNDD	MCCNDD	I4	I3	I2	I1			
I5	I5	MCCNDD	I4	I3	I2	I1		
I6	I6	I5	MCCNDD	I4	I3	I2	I1	
I7	I7	I6	I5	MCCNDD	I4	I3	I2	
I8	I8	I7	I6	I5	-	I4	I3	
I9	I9	I8	I7	I6	I5	-	I4	
I10	I10	I9	I8	I7	I6	I5	-	
etc		I10	I9	I8	I7	I6	I5	
....			I10	I9	I8	I7	I6	
....				I10	I9	I8	I7	
....					I10	I9	I8	
						I10	I9	
							I10	

Table 10-13. Pipeline Activity For MCCNDD, Call Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
I4	I4	I3	I2	I1				
MCCNDD	MCCNDD	I4	I3	I2	I1			
I5	I5	MCCNDD	I4	I3	I2	I1		
I6	I6	I5	MCCNDD	I4	I3	I2	I1	
I7 (1)	I7	I6	I5	MCCNDD	I4	I3	I2	
d1	d1	I7	I6	I5	-	I4	I3	
d2	d2	d1	I7	I6	I5	-	I4	
d3	d3	d2	d1	I7	I6	I5	-	
etc		d3	d2	d1	I7	I6	I5	
....			d3	d2	d1	I7	I6	
....				d3	d2	d1	I7	
....					d3	d2	d1	
						d3	d2	
							d3	

(1) The RPC value in the MSTF register will point to the instruction following I7 (instruction I8).

See also

MBCNDD #16BitDest, CNDF
MMOV32 mem32, MSTF
MMOV32 MSTF, mem32

MCCNDD 16BitDest {, CNDF} (continued)

Call Conditional Delayed

MRCNDD CNDF

MCMP32 MRa, MRb

32-Bit Integer Compare for Equal, Less Than or Greater Than

Operands

MRa	CLA floating-point source register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1111 0010 0000
```

Description

Set ZF and NF flags on the result of MRa - MRb where MRa and MRb are 32-bit integers. For a floating point compare refer to MCMPF32.

Note

A known hardware issue exists in the MCMP32 instruction. Signed integer comparisons using MCMP32 by itself will set the status bits in a way that is not useful for comparison when the difference between the two operands is too large, such as when the inputs have opposite sign and are near the extreme 32-bit signed values. This affects both signed and unsigned integer comparisons.

The compiler (version 18.1.5.LTS or higher) has implemented a workaround for this issue. The compiler checks the upper bits of the operands by performing a floating point comparison before proceeding to do the integer comparison or subtraction.

The compiler flag --cla_signed_compare_workaround enables this workaround.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
If(MRa ==    MRb) {ZF=1; NF=0;}
If(MRa >  MRb) {ZF=0; NF=0;}
If(MRa <  MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
; Behavior of ZF and NF flags for different comparisons
;
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
    MMOV32 MR0, @_A ; MR0 = 1 (0x00000001)
    MMOV32 MR1, @_B ; MR1 = 2 (0x00000002)
    MMOV32 MR2, @_C ; MR2 = -7 (0xFFFFFFF9)
    MCMP32 MR2, MR2 ; NF = 0, ZF = 1
    MCMP32 MR0, MR1 ; NF = 1, ZF = 0
    MCMP32 MR1, MR0 ; NF = 0, ZF = 0
```

See also

MADD32 MRa, MRb, MRc

MCMP32 MRa, MRb (continued)

32-Bit Integer Compare for Equal, Less Than or Greater Than

MSUB32 MRa, MRb, MRc

MCMPF32 MRa, MRb

32-Bit Floating-Point Compare for Equal, Less Than or Greater Than

Operands

MRa	CLA floating-point source register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0000 0000
```

Description

Set ZF and NF flags on the result of MRa - MRb. The MCMPF32 instruction is performed as a logical compare operation. This is possible because of the IEEE format offsetting the exponent. Basically the bigger the binary number, the bigger the floating-point value.

Special cases for inputs:

- Negative zero will be treated as positive zero.
- A denormalized value will be treated as positive zero.
- Not-a-Number (NaN) will be treated as infinity.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
If(MRa == MRb)   {ZF=1; NF=0;}
If(MRa > MRb) {ZF=0; NF=0;}
If(MRa < MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
; Behavior of ZF and NF flags for different comparisons
    MMOVIZ    MR1, #-2.0  ; MR1 = -2.0 (0xC0000000)
    MMOVIZ    MR0, #5.0   ; MR0 = 5.0 (0x40A00000)
    MCMPF32   MR1, MR0    ; ZF = 0, NF = 1
    MCMPF32   MR0, MR1    ; ZF = 0, NF = 0
    MCMPF32   MR0, MR0    ; ZF = 1, NF = 0
```

See also

MCMPF32 MRa, #16FHi
MMAXF32 MRa, #16FHi
MMAXF32 MRa, MRb
MMINF32 MRa, #16FHi
MMINF32 MRa, MRb

MCMPF32 MRa, #16FHi

32-Bit Floating-Point Compare for Equal, Less Than or Greater Than

Operands

MRa	CLA floating-point source register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 1100 00aa
```

Description

Compare the value in MRa with the floating-point value represented by the immediate operand. Set the ZF and NF flags on (MRa - #16FHi:0).

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

The MCMPF32 instruction is performed as a logical compare operation. This is possible because of the IEEE floating-point format offsets the exponent. Basically the bigger the binary number, the bigger the floating-point value.

Special cases for inputs:

- Negative zero will be treated as positive zero.
- Denormalized value will be treated as positive zero.
- Not-a-Number (NaN) will be treated as infinity.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
If(MRa == #16FHi:0)   {ZF=1, NF=0;}
If(MRa > #16FHi:0) {ZF=0, NF=0;}
If(MRa < #16FHi:0) {ZF=0, NF=1;}
```

Pipeline

This is a single-cycle instruction

Example 1

```
; Behavior of ZF and NF flags for different comparisons
  MMOVIZ      MR1, #-2.0  ; MR1 = -2.0 (0xC0000000)
  MMOVIZ      MR0, #5.0   ; MR0 = 5.0 (0x40A00000)
  MCMPF32     MR1, #-2.2  ; ZF = 0, NF = 0
  MCMPF32     MR0, #6.5   ; ZF = 0, NF = 1
  MCMPF32     MR0, #5.0   ; ZF = 1, NF = 0
```

MCMPF32 MRa, #16FHi (continued)

32-Bit Floating-Point Compare for Equal, Less Than or Greater Than

Example 2

```
; X is an array of 32-bit floating-point values
; and has len elements. Find the maximum value in
; the array and store it in Result
;
; Note: MCMPF32 and MSWAPF can be replaced with MMAXF32
;
_Cla1Task1:
  MMOVI16 MAR1,#_X         ; Start address
  MUI16TOF32 MR0, @_len    ; Length of the array
  MNOP                     ; delay for MAR1 load
  MNOP                     ; delay for MAR1 load
  MMOV32 MR1, *MAR1[2]++   ; MR1 = X0
LOOP
  MMOV32 MR2, *MAR1[2]++   ; MR2 = next element
  MCMPF32 MR2, MR1         ; Compare MR2 with MR1
  MSWAPF MR1, MR2, GT      ; MR1 = MAX(MR1, MR2)
  MADDF32 MR0, MR0, #-1.0 ; Decrememt the counter
  MCMPF32 MR0 #0.0         ; Set/clear flags for MBCNDD
  MNOP
  MNOP
  MNOP
  MBCNDD LOOP, NEQ         ; Branch if not equal to zero
  MMOV32 @_Result, MR1     ; Always executed
  MNOP                     ; Always executed
  MNOP                     ; Always executed
  MSTOP                    ; End of task
```

See also

MCMPF32 MRa, MRb
MMAXF32 MRa, #16FHi
MMAXF32 MRa, MRb
MMINF32 MRa, #16FHi
MMINF32 MRa, MRb

MDEBUGSTOP

Debug Stop Task

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 0110 0000
```

Description

When CLA breakpoints are enabled, the MDEBUGSTOP instruction is used to halt a task so that it can be debugged. That is, MDEBUGSTOP is the CLA breakpoint. If CLA breakpoints are not enabled, the MDEBUGSTOP instruction behaves like a MNOP. Unlike the MSTOP, the MIRUN flag is not cleared and an interrupt is not issued. A single-step or run operation will continue execution of the task.

Restrictions

The MDEBUGSTOP instruction cannot be placed 3 instructions before or after a MBCNDD, MCCNDD, or MRCNDD instruction.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

See also

MSTOP

MEALLOW

Enable CLA Write Access to EALLOW Protected Registers

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1001 0000
```

Description

This instruction sets the MEALLOW bit in the CLA status register MSTF. When this bit is set, the CLA is allowed write access to EALLOW protected registers. To again protect against CLA writes to protected registers, use the MEDIS instruction.

MEALLOW and MEDIS only control CLA write access; reads are allowed even if MEALLOW has not been executed. MEALLOW and MEDIS are also independent from the main CPU's EALLOW/EDIS. This instruction does not modify the EALLOW bit in the main CPU's status register. The MEALLOW bit in MSTF only controls access for the CLA while the EALLOW bit in the ST1 register only controls access for the main CPU.

As with EALLOW, the MEALLOW bit is overridden via the JTAG port, allowing full control of register accesses during debug from Code Composer Studio.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; C header file including definition of
; the EPwm1Regs structure
;
; The ePWM TZSEL register is EALLOW protected
;
   .cdecls C,LIST,"CLAShared.h"
   ...
_Cla1Task1:
   ...
   MEALLOW                          ; Allow CLA write access
   MMOV16 @_EPwm1Regs.TZSEL.all, MR3 ; Write to TZSEL
   MEDIS                            ; Disallow CLA write access
   ...
   ...
   MSTOP
```

See also

MEDIS

MEDIS

Disable CLA Write Access to EALLOW Protected Registers

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1011 0000
```

Description

This instruction clears the MEALLOW bit in the CLA status register MSTF. When this bit is clear, the CLA is not allowed write access to EALLOW-protected registers. To enable CLA writes to protected registers, use the MEALLOW instruction.

MEALLOW and MEDIS only control CLA write access; reads are allowed even if MEALLOW has not been executed. MEALLOW and MEDIS are also independent from the main CPU's EALLOW/EDIS. This instruction does not modify the EALLOW bit in the main CPU's status register. The MEALLOW bit in MSTF only controls access for the CLA while the EALLOW bit in the ST1 register only controls access for the main CPU.

As with EALLOW, the MEALLOW bit is overridden via the JTAG port, allowing full control of register accesses during debug from the Code Composer Studio™ IDE.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; C header file including definition of
; the EPwm1Regs structure
;
; The ePWM TZSEL register is EALLOW protected
;
   .cdecls C,LIST,"CLAShared.h"
   ...
_Cla1Task1:
   ...
   MEALLOW                          ; Allow CLA write access
   MMOV16 @_EPwm1Regs.TZSEL.all, MR3 ; Write to TZSEL
   MEDIS                            ; Disallow CLA write access
   ...
   ...
   MSTOP
```

See also

MEALLOW

MEINVF32 MRa, MRb

32-Bit Floating-Point Reciprocal Approximation

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1111 0000 0000
```

Description

This operation generates an estimate of 1/X in 32-bit floating-point format accurate to approximately 8 bits. This value can be used in a Newton-Raphson algorithm to get a more accurate answer. That is:

```
Ye = Estimate(1/X);
Ye = Ye*(2.0 - Ye*X);
Ye = Ye*(2.0 - Ye*X);
```

After two iterations of the Newton-Raphson algorithm, you will get an exact answer accurate to the 32-bit floating-point format. On each iteration the mantissa bit accuracy approximately doubles. The MEINVF32 operation will not generate a negative zero, DeNorm or NaN value.

```
MRa = Estimate of 1/MRb;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MEINVF32 generates an underflow condition.
- LVF = 1 if MEINVF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Calculate Num/Den using a Newton-Raphson algorithum for 1/Den
; Ye = Estimate(1/X)
; Ye = Ye*(2.0 - Ye*X)
; Ye = Ye*(2.0 - Ye*X)
;
_Cla1Task1:
    MMOV32 MR1, @_Den       ; MR1 = Den
    MEINVF32 MR2, MR1       ; MR2 = Ye = Estimate(1/Den)
    MMPYF32 MR3, MR2, MR1   ; MR3 = Ye*Den
    MSUBF32 MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32 MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
    MMPYF32 MR3, MR2, MR1   ; MR3 = Ye*Den
 || MMOV32 MR0, @_Num       ; MR0 = Num
    MSUBF32 MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32 MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
 || MMOV32 MR1, @_Den       ; Reload Den To Set Sign
    MNEGF32 MR0, MR0, EQ    ; if(Den == 0.0) Change Sign Of Num
    MMPYF32 MR0, MR2, MR0   ; MR0 = Y = Ye*Num
    MMOV32 @_Dest, MR0      ; Store result
    MSTOP                   ; end of task
```

See also

MEISQRTF32 MRa, MRb

MEISQRTF32 MRa, MRb

32-Bit Floating-Point Square-Root Reciprocal Approximation

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0100 0000
```

Description

This operation generates an estimate of 1/sqrt(X) in 32-bit floating-point format accurate to approximately 8 bits. This value can be used in a Newton-Raphson algorithm to get a more accurate answer. That is:

```
Ye = Estimate(1/sqrt(X));
Ye = Ye*(1.5 - Ye*Ye*X/2.0);
Ye = Ye*(1.5 - Ye*Ye*X/2.0);
```

After 2 iterations of the Newton-Raphson algorithm, you will get an exact answer accurate to the 32-bit floating-point format. On each iteration the mantissa bit accuracy approximately doubles. The MEISQRTF32 operation will not generate a negative zero, DeNorm or NaN value.

```
MRa = Estimate of 1/sqrt (MRb);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MEISQRTF32 generates an underflow condition.
- LVF = 1 if MEISQRTF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Y = sqrt(X)
; Ye = Estimate(1/sqrt(X));
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Y = X*Ye
;
_Cla1Task3:
    MMOV32 MR0, @_x          ; MR0 = X
    MEISQRTF32 MR1, MR0      ; MR1 = Ye = Estimate(1/sqrt(X))
    MMOV32 MR1, @_x, EQ      ; if(X == 0.0) Ye = 0.0
    MMPYF32 MR3, MR0, #0.5   ; MR3 = X*0.5
    MMPYF32 MR2, MR1, MR3    ; MR2 = Ye*X*0.5
    MMPYF32 MR2, MR1, MR2    ; MR2 = Ye*Ye*X*0.5
    MSUBF32 MR2, #1.5, MR2   ; MR2 = 1.5 - Ye*Ye*X*0.5
    MMPYF32 MR1, MR1, MR2    ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
    MMPYF32 MR2, MR1, MR3    ; MR2 = Ye*X*0.5
    MMPYF32 MR2, MR1, MR2    ; MR2 = Ye*Ye*X*0.5
    MSUBF32 MR2, #1.5, MR2   ; MR2 = 1.5 - Ye*Ye*X*0.5
    MMPYF32 MR1, MR1, MR2    ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
    MMPYF32 MR0, MR1, MR0    ; MR0 = Y = Ye*X
    MMOV32 @_y, MR0          ; Store Y = sqrt(X)
    MSTOP                    ; end of task
```

MEISQRTF32 MRa, MRb (continued)

32-Bit Floating-Point Square-Root Reciprocal Approximation

See also MEINVF32 MRa, MRb

MF32TOI16 MRa, MRb

Convert 32-Bit Floating-Point Value to 16-Bit Integer

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1110 0000
```

Description

Convert a 32-bit floating point value in MRb to a 16-bit integer and truncate. The result will be stored in MRa.

```
MRa(15:0) = F32TOI16(MRb);
MRa(31:16) = sign extension of MRa(15);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #5.0   ; MR0        = 5.0 (0x40A00000)
MF32TOI16   MR1, MR0    ; MR1(15:0)  = MF32TOI16(MR0) = 0x0005
                        ; MR1(31:16) = Sign extension of MR1(15) = 0x0000
MMOVIZ      MR2, #-5.0  ; MR2        = -5.0 (0xC0A00000)
MF32TOI16   MR3, MR2    ; MR3(15:0)  = MF32TOI16(MR2) = -5 (0xFFFB)
                        ; MR3(31:16) = Sign extension of MR3(15) = 0xFFFF
```

See also

MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOI16R MRa, MRb

Convert 32-Bit Floating-Point Value to 16-Bit Integer and Round

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0110 0000
```

Description

Convert the 32-bit floating point value in MRb to a 16-bit integer and round to the nearest even value. The result is stored in MRa.

```
MRa(15:0) = F32TOI16round(MRb);
MRa(31:16) = sign extension of MRa(15);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ MR0, #0x3FD9    ; MR0(31:16) = 0x3FD9
MMOVXI MR0, #0x999A    ; MR0(15:0) = 0x999A
                       ; MR0 = 1.7 (0x3FD9999A)
MF32TOI16R MR1, MR0    ; MR1(15:0) = MF32TOI16round (MR0) = 2 (0x0002)
                       ; MR1(31:16) = Sign extension of MR1(15) = 0x0000
MMOVF32 MR2, #-1.7     ; MR2 = -1.7 (0xBFD9999A)
MF32TOI16R MR3, MR2    ; MR3(15:0) = MF32TOI16round (MR2) = -2 (0xFFFE)
                       ; MR3(31:16) = Sign extension of MR2(15) = 0xFFFF
```

See also

MF32TOI16 MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOI32 MRa, MRb

Convert 32-Bit Floating-Point Value to 32-Bit Integer

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0110 0000
```

Description

Convert the 32-bit floating-point value in MRb to a 32-bit integer value and truncate. Store the result in MRa.

```
MRa = F32TOI32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example 1

```
MMOVF32    MR2, #11204005.0   ; MR2 = 11204005.0 (0x4B2AF5A5)
MF32TOI32  MR3, MR2           ; MR3 = MF32TOI32(MR2) = 11204005 (0x00AAF5A5)
MMOVF32    MR0, #-11204005.0  ; MR0 = -11204005.0 (0xCB2AF5A5)
MF32TOI32  MR1, MR0           ; MR1 = MF32TOI32(MR0) = -11204005 (0xFF550A5B)
```

Example 2

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
; Convert M, X and B from IQ24 to float
;
_Cla1Task2:
   MI32TOF32 MR0, @_M          ; MR0 = 0x4BC00000
   MI32TOF32 MR1, @_X          ; MR1 = 0x4C200000
   MI32TOF32 MR2, @_B          ; MR2 = 0xCB000000
   MMPYF32   MR0, MR0, #0x3380 ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
   MMPYF32   MR1, MR1, #0x3380 ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
   MMPYF32   MR2, MR2, #0x3380 ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
   MMPYF32   MR3, MR0, MR1     ; M*X
   MADDF32   MR2, MR2, MR3     ; Y=MX+B = 3.25 (0x40500000)
; Convert Y from float32 to IQ24
   MMPYF32 MR2, MR2, #0x4B80   ; Y * 1*2^24
   MF32TOI32 MR2, MR2          ; IQ24(Y) = 0x03400000
   MMOV32 @_Y, MR2             ; store result
   MSTOP                       ; end of task
```

See also

MF32TOUI32 MRa, MRb
MI32TOF32 MRa, MRb
MI32TOF32 MRa, mem32
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MF32TOUI16 MRa, MRb

Convert 32-Bit Floating-Point Value to 16-bit Unsigned Integer

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1010 0000
```

Description

Convert the 32-bit floating point value in MRb to an unsigned 16-bit integer value and truncate to zero. The result will be stored in MRa. To instead round the integer to the nearest even value use the MF32TOUI16R instruction.

```
MRa(15:0) = F32TOUI16(MRb);
MRa(31:16) = 0x0000;
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #9.0     ; MR0 = 9.0 (0x41100000)
MF32TOUI16  MR1, MR0      ; MR1(15:0) = MF32TOUI16(MR0) = 9 (0x0009)
                          ; MR1(31:16) = 0x0000
MMOVIZ      MR2, #-9.0    ; MR2 = -9.0 (0xC1100000)
MF32TOUI16  MR3, MR2      ; MR3(15:0) = MF32TOUI16(MR2) = 0 (0x0000)
                          ; MR3(31:16) = 0x0000
```

See also

MF32TOI16 MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOUI16R MRa, MRb

Convert 32-Bit Floating-Point Value to 16-bit Unsigned Integer and Round

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1100 0000
```

Description

Convert the 32-bit floating-point value in MRb to an unsigned 16-bit integer and round to the closest even value. The result will be stored in MRa. To instead truncate the converted value, use the MF32TOUI16 instruction.

```
MRa(15:0)  = MF32TOUI16round(MRb);
MRa(31:16) = 0x0000;
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ       MR0, #0x412C    ; MR0 = 0x412C
MMOVXI       MR0, #0xCCCD    ; MR0 = 0xCCCD ; MR0 = 10.8 (0x412CCCCD)
MF32TOUI16R  MR1, MR0        ; MR1(15:0) = MF32TOUI16round(MR0) = 11 (0x000B)
                             ; MR1(31:16) = 0x0000
MMOVF32      MR2, #-10.8     ; MR2 = -10.8 (0x0xC12CCCCD)
MF32TOUI16R  MR3, MR2        ; MR3(15:0) = MF32TOUI16round(MR2) = 0 (0x0000)
                             ; MR3(31:16) = 0x0000
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MF32TOUI32 MRa, MRb

Convert 32-Bit Floating-Point Value to 32-Bit Unsigned Integer

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1010 0000
```

Description

Convert the 32-bit floating-point value in MRb to an unsigned 32-bit integer and store the result in MRa.

```
MRa = F32TOUI32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #12.5   ; MR0 = 12.5 (0x41480000)
MF32TOUI32  MR0, MR0     ; MR0 = MF32TOUI32 (MR0) = 12 (0x0000000C)
MMOVIZ      MR1, #-6.5   ; MR1 = -6.5 (0xC0D00000)
MF32TOUI32  MR2, MR1     ; MR2 = MF32TOUI32 (MR1) = 0.0 (0x00000000)
```

See also

MF32TOI32 MRa, MRb
MI32TOF32 MRa, MRb
MI32TOF32 MRa, mem32
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MFRACF32 MRa, MRb

Fractional Portion of a 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 0000 0000
```

Description Returns in MRa the fractional portion of the 32-bit floating-point value in MRb

Flags This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline This is a single-cycle instruction.

Example

```
MMOVIZ    MR2, #19.625 ; MR2 = 19.625 (0x419D0000)
MFRACF32  MR3, MR2     ; MR3 = MFRACF32(MR2) = 0.625 (0x3F200000)0)
```

MI16TOF32 MRa, MRb

Convert 16-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1000 0000
```

Description

Convert the 16-bit signed integer in MRb to a 32-bit floating point value and store the result in MRa.

```
MRa = MI16TOF32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ      MR0, #0x0000    ; MR0(31:16) = 0.0 (0x0000)
MMOVXI      MR0, #0x0004    ; MR0(15:0) = 4.0 (0x0004)
MI16TOF32   MR1, MR0        ; MR1 = MI16TOF32 (MR0) = 4.0 (0x40800000)
MMOVIZ      MR2, #0x0000    ; MR2(31:16) = 0.0 (0x0000)
MMOVXI      MR2, #0xFFFC    ; MR2(15:0) = -4.0 (0xFFFC)
MI16TOF32   MR3, MR2        ; MR3 = MI16TOF32 (MR2) = -4.0 (0xC0800000)
MSTOP
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MI16TOF32 MRa, mem16

Convert 16-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem16	16-bit source memory location to be converted

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 00aa addr
```

Description

Convert the 16-bit signed integer indicated by the mem16 pointer to a 32-bit floating-point value and store the result in MRa.

```
MRa = MI16TOF32[mem16];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction:

Example

```
; Assume A = 4 (0x0004)
;        B = -4 (0xFFFC)
   MI16TOF32 MR0, @_A ; MR0 = MI16TOF32(A) = 4.0 (0x40800000)
   MI16TOF32 MR1, @_B ; MR1 = MI16TOF32(B) = -4.0 (0xC0800000
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MUI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MI32TOF32 MRa, mem32

Convert 32-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem32	32-bit memory source for the MMOV32 operation.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 01aa addr
```

Description

Convert the 32-bit signed integer indicated by mem32 to a 32-bit floating point value and store the result in MRa.

```
MRa = MI32TOF32[mem32];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
; Convert M, X and B from IQ24 to float
;
_Cla1Task3:
   MI32TOF32 MR0, @_M          ; MR0 = 0x4BC00000
   MI32TOF32 MR1, @_X          ; MR1 = 0x4C200000
   MI32TOF32 MR2, @_B          ; MR2 = 0xCB000000
   MMPYF32 MR0, MR0, #0x3380   ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
   MMPYF32 MR1, MR1, #0x3380   ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
   MMPYF32 MR2, MR2, #0x3380   ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
   MMPYF32 MR3, MR0, MR1       ; M*X
   MADDF32 MR2, MR2, MR3       ; Y=MX+B = 3.25 (0x40500000)
; Convert Y from float32 to IQ24
   MMPYF32 MR2, MR2, #0x4B80   ; Y * 1*2^24
   MF32TOI32 MR2, MR2          ; IQ24(Y) = 0x03400000
   MMOV32 @_Y, MR2             ; store result
   MSTOP                       ; end of task
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, MRb
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MI32TOF32 MRa, MRb

Convert 32-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1000 0000
```

Description

Convert the signed 32-bit integer in MRb to a 32-bit floating-point value and store the result in MRa.

```
MRa = MI32TOF32(MRb);
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ    MR2, #0x1111 ; MR2(31:16) = 4369 (0x1111)
MMOVXI    MR2, #0x1111 ; MR2(15:0) = 4369 (0x1111)
                       ; MR2 = +286331153 (0x11111111)
MI32TOF32 MR3, MR2     ; MR3 = MI32TOF32 (MR2) = 286331153.0 (0x4D888888)
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, mem32
MUI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MLSL32 MRa, #SHIFT

Logical Shift Left

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#SHIFT	Number of bits to shift (1 to 32)

Opcode

```
LSW: 0000 0000 0shi ftaa
MSW: 0111 1011 1100 0000
```

Description

Logical shift left of MRa by the number of bits indicated. The number of bits can be 1 to 32.

```
MARa(31:0) = Logical Shift Left(MARa(31:0) by #SHIFT bits);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given m2 = (int32)32
;       x2 = (int32)64
;       b2 = (int32)-128
;
; Calculate:
;       m2 = m2*2
;       x2 = x2*4
;       b2 = b2*8
;
_Cla1Task3:
   MMOV32 MR0, @_m2    ; MR0 = 32 (0x00000020)
   MMOV32 MR1, @_x2    ; MR1 = 64 (0x00000040)
   MMOV32 MR2, @_b2    ; MR2 = -128 (0xFFFFFF80)
   MLSL32 MR0, #1      ; MR0 = 64 (0x00000040)
   MLSL32 MR1, #2      ; MR1 = 256 (0x00000100)
   MLSL32 MR2, #3      ; MR2 = -1024 (0xFFFFFC00)
   MMOV32 @_m2, MR0    ; Store results
   MMOV32 @_x2, MR1
   MMOV32 @_b2, MR2
   MSTOP               ; end of task
```

See also

MADD32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MAND32 MRa, MRb, MRc
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MLSR32 MRa, #SHIFT

Logical Shift Right

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#SHIFT	Number of bits to shift (1 to 32)

Opcode

```
LSW: 0000 0000 0shi ftaa
MSW: 0111 1011 1000 0000
```

Description

Logical shift right of MRa by the number of bits indicated. The number of bits can be 1 to 32. Unlike the arithmetic shift (MASR32), the logical shift does not preserve the number's sign bit. Every bit in the operand is moved the specified number of bit positions, and the vacant bit-positions are filled in with zeros

```
MARa(31:0) = Logical Shift Right(MARa(31:0) by #SHIFT bits);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
; Illustrate the difference between MASR32 and MLSR32
MMOVIZ MR0, #0xAAAA   ; MR0 = 0xAAAA5555
MMOVXI MR0, #0x5555
MMOV32 MR1, MR0       ; MR1 = 0xAAAA5555
MMOV32 MR2, MR0       ; MR2 = 0xAAAA5555
MASR32 MR1, #1        ; MR1 = 0xD5552AAA
MLSR32 MR2, #1        ; MR2 = 0x55552AAA
MASR32 MR1, #1        ; MR1 = 0xEAAA9555
MLSR32 MR2, #1        ; MR2 = 0x2AAA9555
MASR32 MR1, #6        ; MR1 = 0xFFAAAA55
MLSR32 MR2, #6        ; MR2 = 0x00AAAA55
```

See also

MADD32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MAND32 MRa, MRb, MRc
MLSL32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc
MSUB32 MRa, MRb, MRc

MMACF32 MR3, MR2, MRd, MRe, MRf ||MMOV32 MRa, mem32

32-Bit Floating-Point Multiply and Accumulate with Parallel Move

Operands

MR3	floating-point destination/source register MR3 for the add operation
MR2	CLA floating-point source register MR2 for the add operation
MRd	CLA floating-point destination register (MR0 to MR3) for the multiply operation MRd cannot be the same register as MRa
MRe	CLA floating-point source register (MR0 to MR3) for the multiply operation
MRf	CLA floating-point source register (MR0 to MR3) for the multiply operation
MRa	CLA floating-point destination register for the MMOV32 operation (MR0 to MR3). MRa cannot be MR3 or the same register as MRd.
mem32	32-bit source for the MMOV32 operation

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0011 ffee ddaa addr
```

Description

Multiply and accumulate the contents of floating-point registers and move from register to memory. The destination register for the MMOV32 cannot be the same as the destination registers for the MMACF32.

```
MR3 = MR3 + MR2;
MRd = MRe * MRf;
MRa = [mem32];
```

Restrictions

The destination registers for the MMACF32 and the MMOV32 must be unique. That is, MRa cannot be MR3 and MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMACF32 (add or multiply) generates an underflow condition.
- LVF = 1 if MMACF32 (add or multiply) generates an overflow condition.

MMOV32 sets the NF and ZF flags as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
```

Pipeline

MMACF32 and MMOV32 complete in a single cycle.

MMACF32 MR3, MR2, MRd, MRe, MRf ||MMOV32 MRa, mem32 (continued)

32-Bit Floating-Point Multiply and Accumulate with Parallel Move

Example 1

```
; Perform 5 multiply and accumulate operations:
;
; X and Y are 32-bit floating point arrays
;
; 1st multiply: A = X0 * Y0
; 2nd multiply: B = X1 * Y1
; 3rd multiply: C = X2 * Y2
; 4th multiply: D = X3 * Y3
; 5th multiply: E = X3 * Y3
;
; Result = A + B + C + D + E
;
_Cla1Task1:
   MMOVI16 MAR0, #_X               ; MAR0 points to X array
   MMOVI16 MAR1, #_Y               ; MAR1 points to Y array
   MNOP                            ; Delay for MAR0, MAR1 load
   MNOP                            ; Delay for MAR0, MAR1 load
                                   ; <-- MAR0 valid
   MMOV32 MR0, *MAR0[2]++          ; MR0 = X0, MAR0 += 2
                                   ; <-- MAR1 valid
   MMOV32 MR1, *MAR1[2]++          ; MR1 = Y0, MAR1 += 2
   MMPYF32 MR2, MR0, MR1           ; MR2 = A = X0 * Y0
|| MMOV32 MR0, *MAR0[2]++          ; In parallel MR0 = X1, MAR0 += 2
   MMOV32 MR1, *MAR1[2]++          ; MR1 = Y1, MAR1 += 2
   MMPYF32 MR3, MR0, MR1           ; MR3 = B = X1 * Y1
|| MMOV32 MR0, *MAR0[2]++          ; In parallel MR0 = X2, MAR0 += 2
   MMOV32 MR1, *MAR1[2]++          ; MR1 = Y2, MAR2 += 2
   MMACF32 MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2
|| MMOV32 MR0, *MAR0[2]++          ; In parallel MR0 = X3
   MMOV32 MR1, *MAR1[2]++          ; MR1 = Y3 M
   MACF32 MR3, MR2, MR2, MR0, MR1  ; MR3 = (A + B) + C, MR2 = D = X3 * Y3
|| MMOV32 MR0, *MAR0               ; In parallel MR0 = X4
   MMOV32 MR1, *MAR1               ; MR1 = Y4
   MMPYF32 MR2, MR0, MR1           ; MR2 = E = X4 * Y4
|| MADDF32 MR3, MR3, MR2           ; in parallel MR3 = (A + B + C) + D
   MADDF32 MR3, MR3, MR2           ; MR3 = (A + B + C + D) + E
   MMOV32 @_Result, MR3            ; Store the result
   MSTOP                           ; end of task
```

Example 2

```
; sum = X0*B0 + X1*B1 + X2*B2 + Y1*A1 + Y2*B2
;
;       X2 = X1
;       X1 = X0
;       Y2 = Y1 ; Y1 = sum
;
_ClaTask2:
     MMOV32     MR0, @_B2       ; MR0 = B2
     MMOV32     MR1, @_X2       ; MR1 = X2
     MMPYF32    MR2, MR1, MR0   ; MR2 = X2*B2
  || MMOV32     MR0, @_B1       ; MR0 = B1
     MMOVD32    MR1, @_X1       ; MR1 = X1, X2 = X1
     MMPYF32    MR3, MR1, MR0   ; MR3 = X1*B1
  || MMOV32     MR0, @_B0       ; MR0 = B0
     MMOVD32    MR1, @_X0       ; MR1 = X0, X1 = X0
;  MR3 = X1*B1 + X2*B2, MR2 = X0*B0
;  MR0 = A2
     MMACF32 MR3, MR2, MR2, MR1, MR0
  || MMOV32 MR0, @_A2 M

     MOV32 MR1, @_Y2            ; MR1 = Y2
;  MR3 = X0*B0 + X1*B1 + X2*B2, MR2 = Y2*A2
;  MR0 = A1
     MMACF32 MR3, MR2, MR2, MR1, MR0
  || MMOV32 MR0, @_A1
     MMOVD32 MR1,@_Y1           ; MR1 = Y1, Y2 = Y1
     MADDF32 MR3, MR3, MR2      ; MR3 = Y2*A2 + X0*B0 + X1*B1 + X2*B2
  || MMPYF32 MR2, MR1, MR0      ; MR2 = Y1*A1
     MADDF32 MR3, MR3, MR2      ; MR3 = Y1*A1 + Y2*A2 + X0*B0 + X1*B1 + X2*B2
```

MMACF32 MR3, MR2, MRd, MRe, MRf ||MMOV32 MRa, mem32 (continued)

32-Bit Floating-Point Multiply and Accumulate with Parallel Move

```
    MMOV32 @_Y1, MR3          ; Y1 = MR3
    MSTOP                     ; end of task
```

See also MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MMAXF32 MRa, MRb

32-Bit Floating-Point Maximum

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0010 0000
```

Description

```
if(MRa < MRb) MRa = MRb;
```

Special cases for the output from the MMAXF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == MRb)  {ZF=1; NF=0;}
if(MRa > MRb) {ZF=0; NF=0;}
if(MRa < MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example 1

```
MMOVIZ   MR0, #5.0  ; MR0 =  5.0  (0x40A00000)
MMOVIZ   MR1, #-2.0 ; MR1 = -2.0  (0xC0000000)
MMOVIZ   MR2, #-1.5 ; MR2 = -1.5  (0xBFC00000)
MMAXF32   MR2, MR1  ; MR2 = -1.5, ZF = NF = 0
MMAXF32   MR1, MR2  ; MR1 = -1.5, ZF = 0, NF = 1
MMAXF32   MR2, MR0  ; MR2 =  5.0, ZF = 0, NF = 1
MAXF32   MR0, MR2   ; MR2 =  5.0, ZF = 1, NF = 0
```

MMAXF32 MRa, MRb (continued)

32-Bit Floating-Point Maximum

Example 2

```
; X is an array of 32-bit floating-point values
; Find the maximum value in an array X
; and store it in Result
;
_Cla1Task1:
  MMOVI16    MAR1,#_X          ; Start address
  MUI16TOF32 MR0, @_len        ; Length of the array
  MNOP                         ; delay for MAR1 load
  MNOP                         ; delay for MAR1 load
  MMOV32     MR1, *MAR1[2]++   ; MR1 = X0
LOOP
  MMOV32     MR2, *MAR1[2]++   ; MR2 = next element
  MMAXF32    MR1, MR2          ; MR1 = MAX(MR1, MR2)
  MADDF32    MR0, MR0, #-1.0   ; Decrememt the counter
  MCMPF32    MR0 #0.0          ; Set/clear flags for MBCNDD
  MNOP
  MNOP
  MNOP
  MBCNDD     LOOP, NEQ         ; Branch if not equal to zero
  MMOV32     @_Result, MR1     ; Always executed
  MNOP                         ; Always executed
  MNOP                         ; Always executed
  MSTOP                        ; End of task
```

See also

MCMPF32 MRa, MRb
MCMPF32 MRa, #16FHi
MMAXF32 MRa, #16FHi
MMINF32 MRa, MRb
MMINF32 MRa, #16FHi

MMAXF32 MRa, #16FHi

32-Bit Floating-Point Maximum

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1001 0000 00aa
```

Description

Compare MRa with the floating-point value represented by the immediate operand. If the immediate value is larger, then load it into MRa.

```
if(MRa < #16FHi:0) MRa = #16FHi:0;
```

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

Special cases for the output from the MMAXF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == #16FHi:0)  {ZF=1; NF=0;}
if(MRa > #16FHi:0) {ZF=0; NF=0;}
if(MRa < #16FHi:0) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ    MR0, #5.0  ; MR0 =  5.0  (0x40A00000)
MMOVIZ    MR1, #4.0  ; MR1 =  4.0  (0x40800000)
MMOVIZ    MR2, #-1.5 ; MR2 = -1.5  (0xBFC00000)
MMAXF32   MR0, #5.5  ; MR0 =  5.5, ZF = 0, NF = 1
MMAXF32   MR1, #2.5  ; MR1 =  4.0, ZF = 0, NF = 0
MMAXF32   MR2, #-1.0 ; MR2 = -1.0, ZF = 0, NF = 1
MMAXF32   MR2, #-1.0 ; MR2 = -1.5, ZF = 1, NF = 0
```

See also

MMAXF32 MRa, MRb
MMINF32 MRa, MRb
MMINF32 MRa, #16FHi

MMINF32 MRa, MRb

32-Bit Floating-Point Minimum

Operands

MRa	CLA floating-point source/destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 0100 0000
```

Description

```
if(MRa > MRb) MRa = MRb;
```

Special cases for the output from the MMINF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == MRb)  {ZF=1; NF=0;}
if(MRa > MRb) {ZF=0; NF=0;}
if(MRa < MRb) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example 1

```
MMOVIZ MR0, #5.0  ; MR0 = 5.0 (0x40A00000)
MMOVIZ MR1, #4.0  ; MR1 = 4.0 (0x40800000)
MMOVIZ MR2, #-1.5 ; MR2 = -1.5 (0xBFC00000)
MMINF32 MR0, MR1  ; MR0 = 4.0, ZF = 0, NF = 0
MMINF32 MR1, MR2  ; MR1 = -1.5, ZF = 0, NF = 0
MMINF32 MR2, MR1  ; MR2 = -1.5, ZF = 1, NF = 0
MMINF32 MR1, MR0  ; MR2 = -1.5, ZF = 0, NF = 1
```

MMINF32 MRa, MRb (continued)

32-Bit Floating-Point Minimum

Example 2

```
;
; X is an array of 32-bit floating-point values
; Find the minimum value in an array X
; and store it in Result
;
_Cla1Task1:
MMOVI16    MAR1,#_X          ; Start address
MUI16TOF32 MR0, @_len        ; Length of the array
MNOP                         ; delay for MAR1 load
MNOP                         ; delay for MAR1 load
MMOV32     MR1, *MAR1[2]++   ; MR1 = X0
LOOP
MMOV32     MR2, *MAR1[2]++   ; MR2 = next element
MMINF32    MR1, MR2          ; MR1 = MAX(MR1, MR2)
MADDF32    MR0, MR0, #-1.0   ; Decrememt the counter
MCMPF32    MR0 #0.0          ; Set/clear flags for MBCNDD
MNOP
MNOP
MNOP
MBCNDD     LOOP, NEQ         ; Branch if not equal to zero
MMOV32     @_Result, MR1     ; Always executed
MNOP                         ; Always executed
MNOP                         ; Always executed
MSTOP                        ; End of task
```

See also

MMAXF32 MRa, MRb
MMAXF32 MRa, #16FHi
MMINF32 MRa, #16FHi

MMINF32 MRa, #16FHi

32-Bit Floating-Point Minimum

Operands

MRa	floating-point source/destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1001 0100 00aa
```

Description

Compare MRa with the floating-point value represented by the immediate operand. If the immediate value is smaller, then load it into MRa.

```
if(MRa > #16FHi:0) MRa = #16FHi:0;
```

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. This addressing mode is most useful for constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

Special cases for the output from the MMINF32 operation:

- NaN output will be converted to infinity
- A denormalized output will be converted to positive zero.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The ZF and NF flags are configured on the result of the operation, not the result stored in the destination register.

```
if(MRa == #16FHi:0)   {ZF=1; NF=0;}
if(MRa > #16FHi:0) {ZF=0; NF=0;}
if(MRa < #16FHi:0) {ZF=0; NF=1;}
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ   MR0, #5.0   ; MR0 =  5.0  (0x40A00000)
MMOVIZ   MR1, #4.0   ; MR1 =  4.0  (0x40800000)
MMOVIZ   MR2, #-1.5  ; MR2 = -1.5  (0xBFC00000)
MMINF32  MR0, #5.5   ; MR0 =  5.0, ZF = 0, NF = 1
MMINF32  MR1, #2.5   ; MR1 =  2.5, ZF = 0, NF = 0
MMINF32  MR2, #-1.0  ; MR2 = -1.5, ZF = 0, NF = 1
MMINF32  MR2, #-1.5  ; MR2 = -1.5, ZF = 1, NF = 0
```

See also

MMAXF32 MRa, #16FHi
MMAXF32 MRa, MRb
MMINF32 MRa, MRb

MMOV16 MARx, MRa, #16I

Load the Auxiliary Register with MRa + 16-bit Immediate Value

Operands

MARx	Auxiliary register MAR0 or MAR1
MRa	CLA Floating-point register (MR0 to MR3)
#16I	16-bit immediate value

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOV16 MAR0, MRa, #16I)
MSW: 0111 1111 1101 00AA
LSW: IIII IIII IIII IIII (opcode of MMOV16 MAR1, MRa, #16I)
MSW: 0111 1111 1111 00AA
```

Description

Load the auxiliary register, MAR0 or MAR1, with MRa(15:0) + 16-bit immediate value. Refer to the pipeline section for important information regarding this instruction.

```
MARx = MRa(15:0) + #16I;
```

Flags

This instruction does not modify flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction. The load of MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Therefore the following applies when loading the auxiliary registers:

- **I1 and I2**

  The two instructions following MMOV16 will use MAR0/MAR1 before the update occurs. Thus these two instructions will use the old value of MAR0 or MAR1.
- **I3**

  Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win and the auxiliary register will not be updated with #_X.
- **I4**

  Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOVI16.

```
; Assume MAR0 is 50, MR0 is 10, and #_X is 20
  MMOV16 MAR0, MR0, #_X        ; Load MAR0 with address of X (20) + MR0 (10)
  <Instruction 1>         ; I1 Will use the old value of MAR0 (50)
  <Instruction 2>         ; I2 Will use the old value of MAR0 (50)
  <Instruction 3>         ; I3 Cannot use MAR0
  <Instruction 4>         ; I4 Will use the new value of MAR0 (30)
  <Instruction 5>         ; I5
```

MMOV16 MARx, MRa, #16I (continued)

Load the Auxiliary Register with MRa + 16-bit Immediate Value

Table 10-14. Pipeline Activity For MMOV16 MARx, MRa , #16I

Instruction	F1	F2	D1	D2	R1	R2	E	W
MMOV16 MAR0, MR0, #_X	MMOV16							
I1	I1	MMOV16						
I2	I2	I1	MMOV16					
I3	I3	I2	I1	MMOV16				
I4	I4	I3	I2	I1	MMOV16			
I5	I5	I4	I3	I2	I1	MMOV16		
I6	I6	I5	I4	I3	I2	I1	MMOV16	

Example 1

```
; Calculate an offset into a sin/cos table
;
_Cla1Task1:
    MMOV32 MR0,@_rad                ; MR0 = rad
    MMOV32 MR1,@_TABLE_SIZEDivTwoPi ; MR1 = TABLE_SIZE/(2*Pi)
    MMPYF32 MR1,MR0,MR1             ; MR1 = rad* TABLE_SIZE/(2*Pi)
||  MMOV32 MR2,@_TABLE_MASK         ; MR2 = TABLE_MASK
    MF32TOI32 MR3,MR1               ; MR3 = K=int(rad*TABLE_SIZE/(2*Pi))
    MAND32 MR3,MR3,MR2              ; MR3 = K & TABLE_MASK
    MLSL32 MR3,#1                   ; MR3 = K * 2
    MMOV16 MAR0,MR3,#_Cos0          ; MAR0 K*2+addr of table.Cos0
    MFRACF32 MR1,MR1                ; I1
    MMOV32 MR0,@_TwoPiDivTABLE_SIZE ; I2
    MMPYF32 MR1,MR1,MR0             ; I3
||  MMOV32 MR0,@_Coef3
    MMOV32 MR2,*MAR0[#-64]++        ; MR2 = *MAR0, MAR0 += (-64)
    ...
    ...
    MSTOP ; end of task
```

MMOV16 MARx, MRa, #16I (continued)

Load the Auxiliary Register with MRa + 16-bit Immediate Value

Example 2

```
; This task logs the last NUM_DATA_POINTS
; ADCRESULT1 values in the array VoltageCLA
;
; When the last element in the array has been
; filled, the task will go back to the
; the first element.
;
; Before starting the ADC conversions, force
; Task 8 to initialize the ConversionCount to zero
;
; The ADC is set to sample (acquire) for 15 SYSCLK cycles
; or 75ns. After the capacitor has captured the analog
; value, the ADC will trigger this task early.
; It takes 10.5 ADCCLKs to complete a conversion,
; the ADCCLK being SYSCLK/4
;   T_sys = 1/200MHz = 5ns
;   T_adc = 4*T_sys  = 20ns
; The ADC will take 10.5 * 4 or 42 SYSCLK cycles to complete
; a conversion. The ADC result register may be read on the
; 36th instruction after the task begins.
;
_Cla1Task2:
    .asg       0, N
    .loop
    MNOP                                    ;I1 - I28 Wait till I36 to read
result
    .eval      N + 1, N
    .break     N = 28
    .endloop
    MMOVZ16    MR0, @_ConversionCount           ;I29 Current Conversion
    MMOV16     MAR1, MR0, #_VoltageCLA          ;I30 Next array location
    MUI16TOF32 MR0, MR0                         ;I31 Convert count to float32
    MADDF32    MR0, MR0, #1.0                   ;I32 Add 1 to conversion count
    MCMPF32    MR0, #NUM_DATA_POINTS.0          ;I33 Compare count to max
    MF32TOUI16 MR0, MR0                         ;I34 Convert count to Uint16
    MNOP                                        ;I35 Wait till I36 to read
result
    MMOVZ16    MR2, @_AdcaResultRegs.ADCRESULT1 ;I36 Read ADCRESULT1
    MMOV16     *MAR1, MR2                       ; Store ADCRESULT1
    MBCNDD     _RestartCount, GEQ               ; If count >= NUM_DATA_POINTS
    MMOVIZ     MR1, #0.0                        ; Always executed: MR1=0
    MNOP
    MNOP
    MMOV16     @_ConversionCount, MR0           ; If branch not taken
    MSTOP                                       ; store current count
_RestartCount
    MMOV16     @_ConversionCount, MR1           ; If branch taken, restart
count
    MSTOP                                       ; end of task
; This task initializes the ConversionCount
; to zero
;
_Cla1Task8:
    MMOVIZ     MR0, #0.0
    MMOV16     @_ConversionCount, MR0
    MSTOP
_ClaT8End:
```

MMOV16 MARx, mem16

Load MAR1 with 16-bit Value

Operands

MARx	CLA auxiliary register MAR0 or MAR1
mem16	16-bit destination memory accessed using one of the available addressing modes

Opcode

```
LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 MAR0, mem16)
MSW: 0111 0110 0000 addr
LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 MAR1, mem16)
MSW: 0111 0110 0100 addr
```

Description

Load MAR0 or MAR1 with the 16-bit value pointed to by mem16. Refer to the pipeline section for important information regarding this instruction.

```
MAR1 = [mem16];
```

Flags

No flags MSTF flags are affected.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction. The load of MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Therefore the following applies when loading the auxiliary registers:

- **I1 and I2**

  The two instructions following MMOV16 will use MAR0/MAR1 before the update occurs. Thus these two instructions will use the old value of MAR0 or MAR1.
- **I3**

  Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win send the auxiliary register will not be updated with #_X.
- **I4**

  Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOV16.

```
; Assume MAR0 is 50 and @_X is 20
MMOV16 MAR0, @_X    ; Load MAR0 with the contents of X (20)
<Instruction 1>     ; I1 Will use the old value of MAR0 (50)
<Instruction 2>     ; I2 Will use the old value of MAR0 (50)
<Instruction 3>     ; I3 Cannot use MAR0
<Instruction 4>     ; I4 Will use the new value of MAR0 (20)
<Instruction 5>     ; I5
....
```

MMOV16 MARx, mem16 (continued)

Load MAR1 with 16-bit Value

Table 10-15. Pipeline Activity For MMOV16 MAR0/MAR1, mem16

Instruction	F1	F2	D1	D2	R1	R2	E	W
MMOV16 MAR0, @_X	MMOV16							
I1	I1	MMOV16						
I2	I2	I1	MMOV16					
I3	I3	I2	I1	MMOV16				
I4	I4	I3	I2	I1	MMOV16			
I5	I5	I4	I3	I2	I1	MMOV16		
I6	I6	I5	I4	I3	I2	I1	MMOV16	

Example

```
; This task logs the last NUM_DATA_POINTS
; ADCRESULT1 values in the array VoltageCLA
;
; When the last element in the array has been
; filled, the task will go back to the
; the first element.
;
; Before starting the ADC conversions, force
; Task 8 to initialize the ConversionCount to zero
;
; The ADC is set to sample (acquire) for 15 SYSCLK cycles
; or 75ns. After the capacitor has captured the analog
; value, the ADC will trigger this task early.
; It takes 10.5 ADCCLKs to complete a conversion,
; the ADCCLK being SYSCLK/4
;   T_sys = 1/200MHz = 5ns
;   T_adc = 4*T_sys  = 20ns
; The ADC will take 10.5 * 4 or 42 SYSCLK cycles to complete
; a conversion. The ADC result register may be read on the
; 36th instruction after the task begins.
;
_Cla1Task2:
    .asg       0, N
    .loop
    MNOP
;I1 - I28 Wait till I36 to read result
    .eval      N + 1, N
    .break     N = 28
    .endloop
    MMOVZ16    MR0, @_ConversionCount             ;I29 Current Conversion
    MMOV16     MAR1, MR0, #_VoltageCLA            ;I30 Next array location
    MUI16TOF32 MR0, MR0                           ;I31 Convert count to float32
    MADDF32    MR0, MR0, #1.0                     ;I32 Add 1 to conversion count
    MCMPF32    MR0, #NUM_DATA_POINTS.0            ;I33 Compare count to max
    MF32TOUI16 MR0, MR0                           ;I34 Convert count to Uint16
    MNOP                                          ;I35 Wait till I36 to read
result
    MMOVZ16    MR2, @_AdcaResultRegs.ADCRESULT1   ;I36 Read ADCRESULT1
    MMOV16     *MAR1, MR2                         ; Store ADCRESULT1
    MBCNDD     _RestartCount, GEQ                 ; If count >= NUM_DATA_POINTS
    MMOVIZ     MR1, #0.0                          ; Always executed: MR1=0
    MNOP
    MNOP
    MMOV16     @_ConversionCount, MR0             ; If branch not taken
    MSTOP                                         ; store current count
_RestartCount
    MMOV16     @_ConversionCount, MR1             ; If branch taken, restart
count
    MSTOP                                         ; end of task
; This task initializes the ConversionCount
; to zero
;
_Cla1Task8:
```

MMOV16 MARx, mem16 (continued)

Load MAR1 with 16-bit Value

```
    MMOVIZ    MR0, #0.0
    MMOV16    @_ConversionCount, MR0
    MSTOP
_ClaT8End:
```

MMOV16 mem16, MARx

Move 16-Bit Auxiliary Register Contents to Memory

Operands

mem16	16-bit destination memory accessed using one of the available addressing modes
MARx	CLA auxiliary register MAR0 or MAR1

Opcode

```
LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 mem16, MAR0)
MSW: 0111 0110 1000 addr
LSW: mmmm mmmm mmmm mmmm (Opcode for MMOV16 mem16, MAR1)
MSW: 0111 0110 1100 addr
```

Description

Store the contents of MAR0 or MAR1 in the 16-bit memory location pointed to by mem16.

```
[mem16] = MAR0;
```

Flags

No flags MSTF flags are affected.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

MMOV16 mem16, MRa

Move 16-Bit Floating-Point Register Contents to Memory

Operands

mem16	16-bit destination memory accessed using one of the available addressing modes
MRa	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 11aa addr
```

Description

Move 16-bit value from the lower 16-bits of the floating-point register (MRa(15:0)) to the location pointed to by mem16.

```
[mem16] = MRa(15:0);
```

Flags

No flags MSTF flags are affected.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; This task logs the last NUM_DATA_POINTS
; ADCRESULT1 values in the array VoltageCLA
;
; When the last element in the array has been
; filled, the task will go back to the
; the first element.
;
; Before starting the ADC conversions, force
; Task 8 to initialize the ConversionCount to zero
;
; The ADC is set to sample (acquire) for 15 SYSCLK cycles
; or 75ns. After the capacitor has captured the analog
; value, the ADC will trigger this task early.
; It takes 10.5 ADCCLKs to complete a conversion,
; the ADCCLK being SYSCLK/4
;   T_sys = 1/200MHz = 5ns
;   T_adc = 4*T_sys  = 20ns
; The ADC will take 10.5 * 4 or 42 SYSCLK cycles to complete
; a conversion. The ADC result register may be read on the
; 36th instruction after the task begins.
;
_Cla1Task2:
    .asg       0, N
    .loop
    MNOP
;I1 - I28 Wait till I36 to read result
    .eval      N + 1, N
    .break     N = 28
    .endloop
    MMOVZ16   MR0, @_ConversionCount            ;I29 Current Conversion
    MMOV16    MAR1, MR0, #_VoltageCLA           ;I30 Next array location
    MUI16TOF32 MR0, MR0                         ;I31 Convert count to float32
    MADDF32   MR0, MR0, #1.0                    ;I32 Add 1 to conversion count
    MCMPF32   MR0, #NUM_DATA_POINTS.0           ;I33 Compare count to max
    MF32TOUI16 MR0, MR0                         ;I34 Convert count to Uint16
    MNOP                                        ;I35 Wait till I36 to read
result
    MMOVZ16   MR2, @_AdcaResultRegs.ADCRESULT1  ;I36 Read ADCRESULT1
    MMOV16    *MAR1, MR2                        ; Store ADCRESULT1
    MBCNDD    _RestartCount, GEQ                ; If count >= NUM_DATA_POINTS
    MMOVIZ    MR1, #0.0                         ; Always executed: MR1=0
```

MMOV16 mem16, MRa (continued)

Move 16-Bit Floating-Point Register Contents to Memory

```
    MNOP
    MNOP
    MMOV16    @_ConversionCount, MR0            ; If branch not taken
    MSTOP                                       ; store current count
_RestartCount
    MMOV16    @_ConversionCount, MR1            ; If branch taken, restart
count
    MSTOP                                       ; end of task
; This task initializes the ConversionCount
; to zero
;
_Cla1Task8:
    MMOVIZ    MR0, #0.0
    MMOV16    @_ConversionCount, MR0
    MSTOP
_ClaT8End:
```

See also

MMOVIZ MRa, #16FHi
MMOVXI MRa, #16FLoHex

MMOV32 mem32, MRa

Move 32-Bit Floating-Point Register Contents to Memory

Operands

MRa	floating-point register (MR0 to MR3)
mem32	32-bit destination memory accessed using one of the available addressing modes

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 11aa addr
```

Description

Move from MRa to 32-bit memory location indicated by mem32.

```
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

No flags affected.

Pipeline

This is a single-cycle instruction.

Example

```
; Perform 5 multiply and accumulate operations:
;
; X and Y are 32-bit floating point arrays;
; 1st multiply: A = X0 * Y0
; 2nd multiply: B = X1 * Y1
; 3rd multiply: C = X2 * Y2
; 4th multiply: D = X3 * Y3
; 5th multiply: E = X3 * Y3;
; Result = A + B + C + D + E
;
_Cla1Task1:
   MMOVI16       MAR0, #_X               ; MAR0 points to X array
   MMOVI16       MAR1, #_Y               ; MAR1 points to Y array
   MNOP                                  ; Delay for MAR0, MAR1 load
   MNOP                                  ; Delay for MAR0, MAR1 load
                                         ; <-- MAR0 valid
   MMOV32        MR0, *MAR0[2]++         ; MR0 = X0, MAR0 += 2
                                         ; <-- MAR1 valid
   MMOV32        MR1, *MAR1[2]++         ; MR1 = Y0, MAR1 += 2
   MMPYF32       MR2,  MR0, MR1          ; MR2 = A = X0 * Y0
|| MMOV32        MR0, *MAR0[2]++         ; In parallel MR0 = X1, MAR0 += 2
   MMOV32        MR1, *MAR1[2]++         ; MR1 = Y1, MAR1 += 2
   MMPYF32       MR3, MR0, MR1           ; MR3 = B = X1 * Y1
|| MMOV32        MR0, *MAR0[2]++         ; In parallel MR0 = X2, MAR0 += 2
   MMOV32        MR1, *MAR1[2]++         ; MR1 = Y2, MAR2 += 2
   MMACF32       MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2
|| MMOV32        MR0, *MAR0[2]++         ; In parallel MR0 = X3
   MMOV32        MR1, *MAR1[2]++         ; MR1 = Y3
   MMACF32       MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3
|| MMOV32        MR0, *MAR0              ; In parallel MR0 = X4
   MMOV32        MR1, *MAR1              ; MR1 = Y4
   MMPYF32       MR2, MR0, MR1           ; MR2 = E = X4 * Y4
|| MADDF32       MR3, MR3, MR2           ; in parallel MR3 = (A + B + C) + D
   MADDF32       MR3, MR3, MR2           ; MR3 = (A + B + C + D) + E
   MMOV32        @_Result, MR3           ; Store the result MSTOP ; end of task
```

See also

MMOV32 mem32, MSTF

MMOV32 mem32, MSTF

Move 32-Bit MSTF Register to Memory

Operands

MSTF	floating-point status register
mem32	32-bit destination memory

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0111 0100 addr
```

Description

Copy the CLA's floating-point status register, MSTF, to memory.

```
[mem32] = MSTF;
```

Flags

This instruction does not modify flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

One of the uses of this instruction is to save off the return PC (RPC) prior to calling a function. The decision to jump to a function is made when the MCCNDD is in the decode2 (D2) phase of the pipeline; the RPC is also updated in this phase. The actual jump occurs three cycles later when MCCNDD enters its execution (E) phase. The user must, therefore, save the old RPC before MCCNDD updates it in the D2 phase, that is, it must save MSTF three instructions prior to the function call.

Example

The following example illustrates the pipeline flow for the context save (of the flags and RPC) prior to a function call. The first column in the comments shows the pipeline stages for the MMOV32 instruction while the second column pertains to the MCCNDD instruction.

```
 MMOV32 @_temp, MSTF ; D2|   |
MNOP                 ; R1|F1| MCCNDD is fetched
MNOP                 ; R2|F2|
MNOP                 ; E |D1|
MCCNDD _bar, UNC     ; W |D2| old RPC written to memory,
                     ;   |  | RPC updated with MPC+1
MNOP                 ;   |R1|
MNOP                 ;   |R2|
MNOP                 ;   |E | execution branches to _bar
```

See also

MMOV32 mem32, MRa

MMOV32 MRa, mem32 {, CNDF}

Conditional 32-Bit Move

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem32	32-bit memory location accessed using one of the available addressing modes
CNDF	optional condition.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 00cn dfaa addr
```

Description

If the condition is true, then move the 32-bit value referenced by mem32 to the floating-point register indicated by MRa.

```
if (CNDF == TRUE) MRa = [mem32];
```

CNDF is one of the following conditions:

Encode(1)	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF(2)	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

```
if(CNDF == UNCF)
{
  NF = MRa(31);
  ZF = 0;
  if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
}
  else No flags modified;
```

Pipeline

This is a single-cycle instruction.

MMOV32 MRa, mem32 {, CNDF} (continued)

Conditional 32-Bit Move

Example

```
; Given A, B, X, M1 and M2 are 32-bit floating-point
; numbers
;
; if(A == B) calculate Y = X*M1
; if(A! = B) calculate Y = X*M2
;
_Cla1Task5:
    MMOV32    MR0, @_A
    MMOV32    MR1, @_B
    MCMPF32   MR0, MR1
    MMOV32    MR2, @_M1, EQ  ; if A == B, MR2 = M1
                             ;     Y = M1*X
    MMOV32    MR2, @_M2, NEQ ; if A! = B, MR2 = M2
                             ;     Y = M2*X
    MMOV32    MR3, @_X
    MMPYF32   MR3, MR2, MR3  ; Calculate Y
    MMOV32    @_Y, MR3       ; Store Y
    MSTOP                    ; end of task
```

See also

MMOV32 MRa, MRb {, CNDF}
MMOVD32 MRa, mem32

MMOV32 MRa, MRb {, CNDF}

Conditional 32-Bit Move

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
CNDF	optional condition.

Opcode

```
LSW: 0000 0000 cndf bbaa
MSW: 0111 1010 1100 0000
```

Description

If the condition is true, then move the 32-bit value in MRb to the floating-point register indicated by MRa.

```
if (CNDF == TRUE) MRa = MRb;
```

CNDF is one of the following conditions:

Encode(1)	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF(2)	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF, and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

```
if(CNDF == UNCF)
 {
   NF = MRa(31); ZF = 0;
   if(MRa(30:23) == 0) {ZF = 1; NF = 0;}
}
else No flags modified;
```

Pipeline

This is a single-cycle instruction.

MMOV32 MRa, MRb {, CNDF} (continued)

Conditional 32-Bit Move

Example

```
; Given: X = 8.0
;        Y = 7.0
;        A = 2.0
;        B = 5.0
; _ClaTask1
    MMOV32   MR3, @_X      ; MR3 = X = 8.0
    MMOV32   MR0, @_Y      ; MR0 = Y = 7.0
    MMAXF32  MR3, MR0      ; ZF = 0, NF = 0, MR3 = 8.0
    MMOV32   MR1, @_A, GT  ; true, MR1 = A = 2.0
    MMOV32   MR1, @_B, LT  ; false, does not load MR1
    MMOV32   MR2, MR1, GT  ; true, MR2 = MR1 = 2.0
    MMOV32   MR2, MR0, LT  ; false, does not load MR2
    MSTOP
```

See also MMOV32 MRa, mem32 {,CNDF}

MMOV32 MSTF, mem32

Move 32-Bit Value from Memory to the MSTF Register

Operands

MSTF	CLA status register
mem32	32-bit source memory location

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0111 0000 addr
```

Description

Move from memory to the CLA's status register MSTF. This instruction is most useful when nesting function calls (via MCCNDD).

```
MSTF = [mem32];
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	Yes	Yes	Yes	Yes	Yes

Loading the status register will overwrite all flags and the RPC field. The MEALLOW field is not affected.

Pipeline

This is a single-cycle instruction.

See also

MMOV32 mem32, MSTF

MMOVD32 MRa, mem32

Move 32-Bit Value from Memory with Data Copy

Operands

MRa	CLA floating-point register (MR0 to MR3)
mem32	32-bit memory location accessed using one of the available addressing modes

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 00aa addr
```

Description

Move the 32-bit value referenced by mem32 to the floating-point register indicated by MRa.

```
MRa = [mem32];
[mem32+2] = [mem32];
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0){ ZF = 1; NF = 0; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; sum = X0*B0 + X1*B1 + X2*B2 + Y1*A1 + Y2*B2
;
;       X2 = X1
;       X1 = X0
;       Y2 = Y1
;       Y1 = sum
;
_Cla1Task2:
    MMOV32 MR0, @_B2        ; MR0 = B2
    MMOV32 MR1, @_X2        ; MR1 = X2
    MMPYF32 MR2, MR1, MR0   ; MR2 = X2*B2
||  MMOV32 MR0, @_B1        ; MR0 = B1
    MMOVD32 MR1, @_X1       ; MR1 = X1, X2 = X1
    MMPYF32 MR3, MR1, MR0   ; MR3 = X1*B1
||  MMOV32 MR0, @_B0        ; MR0 = B0
    MMOVD32 MR1, @_X0       ; MR1 = X0, X1 = X0
; MR3 = X1*B1 + X2*B2, MR2 = X0*B0
; MR0 = A2
    MMACF32 MR3, MR2, MR2, MR1, MR0
||  MMOV32 MR0, @_A2

    MMOV32 MR1, @_Y2        ; MR1 = Y2
; MR3 = X0*B0 + X1*B1 + X2*B2, MR2 = Y2*A2
; MR0 = A1
    MMACF32 MR3, MR2, MR2, MR1, MR0
||  MMOV32 MR0, @_A1
    MMOVD32 MR1,@_Y1        ; MR1 = Y1, Y2 = Y1
    MADDF32 MR3, MR3, MR2   ; MR3 = Y2*A2 + X0*B0 + X1*B1 + X2*B2
||  MMPYF32 MR2, MR1, MR0   ; MR2 = Y1*A1
    MADDF32 MR3, MR3, MR2   ; MR3 = Y1*A1 + Y2*A2 + X0*B0 + X1*B1 + X2*B2
    MMOV32 @_Y1, MR3        ; Y1 = MR3
    MSTOP                   ; end of task
```

MMOVD32 MRa, mem32 (continued)

	Move 32-Bit Value from Memory with Data Copy
See also	MMOV32 MRa, mem32 {,CNDF}

MMOVF32 MRa, #32F

Load the 32-Bits of a 32-Bit Floating-Point Register

Operands

This instruction is an alias for MMOVIZ and MMOVXI instructions. The second operand is translated by the assembler such that the instruction becomes:

```
MMOVIZ MRa, #16FHiHex MMOVXI MRa, #16FLoHex
```

MRa	CLA floating-point destination register (MR0 to MR3)
#32F	immediate float value represented in floating-point representation

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOVIZ MRa, #16FHiHex)
MSW: 0111 1000 0100 00aa
LSW: IIII IIII IIII IIII (opcode of MMOVXI MRa, #16FLoHex)
MSW: 0111 1000 1000 00aa
```

Description

Note: This instruction accepts the immediate operand only in floating-point representation. To specify the immediate value as a hex value (IEEE 32-bit floating- point format) use the MOVI32 MRa, #32FHex instruction.

Load the 32-bits of MRa with the immediate float value represented by #32F.

#32F is a float value represented in floating-point representation. The assembler will only accept a float value represented in floating-point representation. That is, 3.0 can only be represented as #3.0. #0x40400000 will result in an error.

```
MRa = #32F;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

Depending on #32FH, this instruction takes one or two cycles. If all of the lower 16-bits of the IEEE 32-bit floating-point format of #32F are zeros, then the assembler will convert MMOVF32 into only MMOVIZ instruction. If the lower 16-bits of the IEEE 32-bit floating-point format of #32F are not zeros, then the assembler will convert MMOVF32 into MMOVIZ and MMOVXI instructions.

Example

```
MMOVF32 MR1, #3.0     ; MR1 = 3.0 (0x40400000)
                      ; Assembler converts this instruction as
                      ; MMOVIZ MR1, #0x4040
MMOVF32 MR2, #0.0     ; MR2 = 0.0 (0x00000000)
                      ; Assembler converts this instruction as
                      ; MMOVIZ MR2, #0x0
MMOVF32 MR3, #12.265  ; MR3 = 12.625 (0x41443D71)
                      ; Assembler converts this instruction as
                      ; MMOVIZ MR3, #0x4144
                      ; MMOVXI MR3, #0x3D71
```

See also

MMOVIZ MRa, #16FHi
MMOVXI MRa, #16FLoHex
MMOVI32 MRa, #32FHex

MMOVI16 MARx, #16I

Load the Auxiliary Register with the 16-Bit Immediate Value

Operands

MARx	Auxiliary register MAR0 or MAR1
#16I	16-bit immediate value

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOVI16 MAR0, #16I)
MSW: 0111 1111 1100 0000
LSW: IIII IIII IIII IIII (opcode of MMOVI16 MAR1, #16I)
MSW: 0111 1111 1110 0000
```

Description

Load the auxiliary register, MAR0 or MAR1, with a 16-bit immediate value. Refer to the pipeline section for important information regarding this instruction.

```
MARx = #16I;
```

Flags

This instruction does not modify flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction. The immediate load of MAR0 or MAR1 will occur in the EXE phase of the pipeline. Any post increment of MAR0 or MAR1 using indirect addressing will occur in the D2 phase of the pipeline. Therefore the following applies when loading the auxiliary registers:

- **I1 and I2**

  The two instructions following MMOVI16 will use MAR0/MAR1 before the update occurs. Thus these two instructions will use the old value of MAR0 or MAR1.
- **I3**

  Loading of an auxiliary register occurs in the EXE phase while updates due to post-increment addressing occur in the D2 phase. Thus I3 cannot use the auxiliary register or there will be a conflict. In the case of a conflict, the update due to address-mode post increment will win snd the auxiliary register will not be updated with #_X.
- **I4**

  Starting with the 4th instruction MAR0 or MAR1 will be the new value loaded with MMOVI16.

```
;  Assume MAR0 is 50 and #_X is 20
MMOVI16 MAR0, #_X             ; Load MAR0 with address of X (20)
<Instruction 1>          ; I1 Will use the old value of MAR0 (50)
<Instruction 2>          ; I2 Will use the old value of MAR0 (50)
<Instruction 3>          ; I3 Cannot use MAR0
<Instruction 4>          ; I4 Will use the new value of MAR0 (20)
<Instruction 5>          ; I5
....
```

MMOVI16 MARx, #16I (continued)

Load the Auxiliary Register with the 16-Bit Immediate Value

Table 10-16. Pipeline Activity For MMOVI16 MAR0/MAR1, #16I

Instruction	F1	F2	D1	D2	R1	R2	E	W
MMOVI16 MAR0, #_X	MMOVI16							
I1	I1	MMOVI16						
I2	I2	I1	MMOVI16					
I3	I3	I2	I1	MMOVI16				
I4	I4	I3	I2	I1	MMOVI16			
I5	I5	I4	I3	I2	I1	MMOVI16		
I6	I6	I5	I4	I3	I2	I1	MMOVI16	

MMOVI32 MRa, #32FHex

Load the 32-Bits of a 32-Bit Floating-Point Register with the Immediate

Operands

MRa	floating-point register (MR0 to MR3)
#32FHex	A 32-bit immediate value that represents an IEEE 32-bit floating-point value.

This instruction is an alias for MMOVIZ and MMOVXI instructions. The second operand is translated by the assembler such that the instruction becomes:

```
MMOVIZ MRa, #16FHiHex
MMOVXI MRa, #16FLoHex
```

Opcode

```
LSW: IIII IIII IIII IIII (opcode of MMOVIZ MRa, #16FHiHex)
MSW: 0111 1000 0100 00aa
LSW: IIII IIII IIII IIII (opcode of MMOVXI MRa, #16FLoHex)
MSW: 0111 1000 1000 00aa
```

Description

Note: This instruction only accepts a hex value as the immediate operand. To specify the immediate value with a floating-point representation use the MMOVF32 MRa, #32F instruction.

Load the 32-bits of MRa with the immediate 32-bit hex value represented by #32Fhex.

#32Fhex is a 32-bit immediate hex value that represents the IEEE 32-bit floating-point value of a floating-point number. The assembler will only accept a hex immediate value. That is, 3.0 can only be represented as #0x40400000. #3.0 will result in an error.

```
MRa = #32FHex;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

Depending on #32FHex, this instruction takes one or two cycles. If all of the lower 16-bits of #32FHex are zeros, then assembler will convert MOVI32 to the MMOVIZ instruction. If the lower 16-bits of #32FHex are not zeros, then assembler will convert MOVI32 to a MMOVIZ and a MMOVXI instruction.

Example

```
MOVI32   MR1, #0x40400000 ; MR1 = 0x40400000
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR1, #0x4040
MOVI32  MR2, #0x00000000  ; MR2 = 0x00000000
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR2, #0x0
MOVI32  MR3, #0x40004001  ; MR3 = 0x40004001
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR3, #0x4000
                          ; MMOVXI MR3, #0x4001
MOVI32  MR0, #0x00004040  ; MR0 = 0x00004040
                          ; Assembler converts this instruction as
                          ; MMOVIZ MR0, #0x0000
                          ; MMOVXI MR0, #0x4040
```

MMOVI32 MRa, #32FHex (continued)

Load the 32-Bits of a 32-Bit Floating-Point Register with the Immediate

See also

MMOVIZ MRa, #16FHi
MMOVXI MRa, #16FLoHex
MMOVF32 MRa, #32F

MMOVIZ MRa, #16FHi

Load the Upper 16-Bits of a 32-Bit Floating-Point Register

Operands

MRa	floating-point register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 0100 00aa
```

Description

Load the upper 16-bits of MRa with the immediate value #16FHi and clear the low 16-bits of MRa.

#16FHiHex is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. The assembler will only accept a decimal or hex immediate value. That is, -1.5 can be represented as #-1.5 or #0xBFC0.

By itself, MMOVIZ is useful for loading a floating-point register with a constant in which the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). If a constant requires all 32-bits of a floating-point register to be initialized, then use MMOVIZ along with the MMOVXI instruction.

```
MRa(31:16) = #16FHi;
MRa(15:0)  = 0;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Load MR0 and MR1 with -1.5 (0xBFC00000)
     MMOVIZ    MR0, #0xBFC0      ; MR0 = 0xBFC00000 (1.5)
     MMOVIZ    MR1, #-1.5        ; MR1 = -1.5 (0xBFC00000)
; Load MR2 with pi = 3.141593 (0x40490FDB)
     MMOVIZ    MR2, #0x4049      ; MR2 = 0x40490000
     MMOVXI    MR2, #0x0FDB      ; MR2 = 0x40490FDB
```

See also

MMOVF32 MRa, #32F
MMOVI32 MRa, #32FHex
MMOVXI MRa, #16FLoHex

MMOVZ16 MRa, mem16

Load MRx With 16-bit Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem16	16-bit source memory location

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 10aa addr
```

Description

Move the 16-bit value referenced by mem16 to the floating-point register indicated by MRa.

```
MRa(31:16) = 0;
MRa(15:0) = [mem16];
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = 0;
if (MRa(31:0)== 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

MMOVXI MRa, #16FLoHex

Move Immediate to the Low 16-Bits of a Floating-Point Register

Operands

MRa	CLA floating-point register (MR0 to MR3)
#16FLoHex	A 16-bit immediate hex value that represents the lower 16-bits of an IEEE 32-bit floating-point value. The upper 16-bits will not be modified.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 1000 00aa
```

Description

Load the low 16-bits of MRa with the immediate value #16FLoHex. #16FLoHex represents the lower 16-bits of an IEEE 32-bit floating-point value. The upper 16-bits of MRa will not be modified. MMOVXI can be combined with the MMOVIZ instruction to initialize all 32-bits of a MRa register.

```
MRa(15:0) = #16FLoHex;
MRa(31:16) = Unchanged;
```

Flags

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Load MR0 with pi = 3.141593 (0x40490FDB)
    MMOVIZ      MR0,#0x4049   ; MR0 = 0x40490000
    MMOVXI      MR0,#0x0FDB   ; MR0 = 0x40490FDB
```

See also

MMOVIZ MRa, #16FHi

MMPYF32 MRa, MRb, MRc

32-Bit Floating-Point Multiply

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 0000 0000
```

Description

Multiply the contents of two floating-point registers.

```
MRa = MRb * MRc;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Calculate Num/Den using a Newton-Raphson algorithum for 1/Den
; Ye = Estimate(1/X)
; Ye = Ye*(2.0 - Ye*X)
; Ye = Ye*(2.0 - Ye*X)
;
_Cla1Task1:
    MMOV32      MR1, @_Den      ; MR1 = Den
    MEINVF32    MR2, MR1        ; MR2 = Ye = Estimate(1/Den)
    MMPYF32     MR3, MR2, MR1   ; MR3 = Ye*Den
    MSUBF32     MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32     MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
    MMPYF32     MR3, MR2, MR1   ; MR3 = Ye*Den
||  MMOV32      MR0, @_Num      ; MR0 = Num
    MSUBF32     MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32     MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
||  MMOV32      MR1, @_Den      ; Reload Den To Set Sign
    MNEGF32     MR0, MR0, EQ    ; if(Den == 0.0) Change Sign Of Num
    MMPYF32     MR0, MR2, MR0   ; MR0 = Y = Ye*Num
    MMOV32      @_Dest, MR0     ; Store result
    MSTOP                       ; end of task
```

See also

MMPYF32 MRa, #16FHi, MRb
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf
MMPYF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MMPYF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf
MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRa, #16FHi, MRb

32-Bit Floating-Point Multiply

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1000 bbaa
```

Description

Multiply MRb with the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb * #16FHi:0;
```

This instruction can also be written as MMPYF32 MRa, MRb, #16FHi.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example 1

```
; Same as example 2 but #16FHi is represented in float
    MMOVIZ    MR3, #2.0       ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, #3.0, MR3 ; MR0 = 3.0 * MR3 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0        ; Save the result in variable X
```

Example 2

```
; Same as example 1 but #16FHi is represented in Hex
    MMOVIZ    MR3, #2.0          ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, #0x4040, MR3 ; MR0 = 0x4040 * MR3 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0           ; Save the result in variable X
```

MMPYF32 MRa, #16FHi, MRb (continued)

32-Bit Floating-Point Multiply

Example 3

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
;
_Cla1Task2:
;
; Convert M, X and B from IQ24 to float
     MI32TOF32   MR0, @_M          ; MR0 = 0x4BC00000
     MI32TOF32   MR1, @_X          ; MR1 = 0x4C200000
     MI32TOF32   MR2, @_B          ; MR2 = 0xCB000000
     MMPYF32     MR0, MR0, #0x3380 ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
     MMPYF32     MR1, MR1, #0x3380 ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
     MMPYF32     MR2, MR2, #0x3380 ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
     MMPYF32     MR3, MR0, MR1     ; M*X
     MADDF32     MR2, MR2, MR3     ; Y=MX+B = 3.25 (0x40500000)
; Convert Y from float32 to IQ24
     MMPYF32     MR2, MR2, #0x4B80 ; Y * 1*2^24
     MF32TOI32   MR2, MR2          ; IQ24(Y) = 0x03400000
     MMOV32 @_Y, MR2               ; store result
     MSTOP                         ; end of task
```

See also

MMPYF32 MRa, MRb, #16FHi
MMPYF32 MRa, MRb, MRc
MMPYF32 MRa, MRb, MRc || MADDF32 MRd, MRe, MRf

MMPYF32 MRa, MRb, #16FHi

32-Bit Floating-Point Multiply

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 0111 1000 bbaa
```

Description

Multiply MRb with the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = MRb * #16FHi:0;
```

This instruction can also be written as MMPYF32 MRa, #16FHi, MRb.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example 1

```
;Same as example 2 but #16FHi is represented in float
    MMOVIZ    MR3, #2.0       ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, MR3, #3.0  ; MR0 = MR3 * 3.0 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0        ; Save the result in variable X
```

Example 2

```
;Same as example 1 but #16FHi is represented in Hex
    MMOVIZ    MR3, #2.0          ; MR3 = 2.0 (0x40000000)
    MMPYF32   MR0, MR3, #0x4040  ; MR0 = MR3 * 0x4040 = 6.0 (0x40C00000)
    MMOV32    @_X, MR0           ; Save the result in variable X
```

MMPYF32 MRa, MRb, #16FHi (continued)

32-Bit Floating-Point Multiply

Example 3

```
; Given X, M and B are IQ24 numbers:
; X = IQ24(+2.5) = 0x02800000
; M = IQ24(+1.5) = 0x01800000
; B = IQ24(-0.5) = 0xFF800000
;
; Calculate Y = X * M + B
;
_Cla1Task2:
;
; Convert M, X and B from IQ24 to float
     MI32TOF32   MR0, @_M          ; MR0 = 0x4BC00000
     MI32TOF32   MR1, @_X          ; MR1 = 0x4C200000
     MI32TOF32   MR2, @_B          ; MR2 = 0xCB000000
     MMPYF32     MR0, #0x3380, MR0 ; M = 1/(1*2^24) * iqm = 1.5 (0x3FC00000)
     MMPYF32     MR1, #0x3380, MR1 ; X = 1/(1*2^24) * iqx = 2.5 (0x40200000)
     MMPYF32     MR2, #0x3380, MR2 ; B = 1/(1*2^24) * iqb = -.5 (0xBF000000)
     MMPYF32     MR3, MR0, MR1     ; M*X
     MADDF32     MR2, MR2, MR3     ; Y=MX+B = 3.25 (0x40500000)
; Convert Y from float32 to IQ24
     MMPYF32     MR2, #0x4B80, MR2 ; Y * 1*2^24
     MF32TOI32   MR2, MR2          ; IQ24(Y) = 0x03400000
     MMOV32      @_Y, MR2          ; store result
     MSTOP                         ; end of task
```

See also

MMPYF32 MRa, #16FHi, MRb
MMPYF32 MRa, MRb, MRc

MMPYF32 MRa, MRb, MRc||MADDF32 MRd, MRe, MRf

32-Bit Floating-Point Multiply with Parallel Add

Operands

MRa	CLA floating-point destination register for MMPYF32 (MR0 to MR3) MRa cannot be the same register as MRd
MRb	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRc	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRd	CLA floating-point destination register for MADDF32 (MR0 to MR3) MRd cannot be the same register as MRa
MRe	CLA floating-point source register for MADDF32 (MR0 to MR3)
MRf	CLA floating-point source register for MADDF32 (MR0 to MR3)

Opcode

```
LSW: 0000 ffee ddcc bbaa
MSW: 0111 1010 0000 0000
```

Description

Multiply the contents of two floating-point registers with parallel addition of two registers.

```
MRa = MRb * MRc;
MRd = MRe + MRf;
```

Restrictions

The destination register for the MMPYF32 and the MADDF32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 or MADDF32 generates an underflow condition.
- LVF = 1 if MMPYF32 or MADDF32 generates an overflow condition.

Pipeline

Both MMPYF32 and MADDF32 complete in a single cycle.

MMPYF32 MRa, MRb, MRc||MADDF32 MRd, MRe, MRf (continued)

32-Bit Floating-Point Multiply with Parallel Add

Example

```
; Perform 5 multiply and accumulate operations:
;
; X and Y are 32-bit floating point arrays
;
; 1st multiply: A = X0 * Y0
; 2nd multiply: B = X1 * Y1
; 3rd multiply: C = X2 * Y2
; 4th multiply: D = X3 * Y3
; 5th multiply: E = X3 * Y3
;
; Result = A + B + C + D + E
;
_Cla1Task1:
    MMOVI16    MAR0, #_X               ; MAR0 points to X array
    MMOVI16    MAR1, #_Y               ; MAR1 points to Y array
    MNOP                               ; Delay for MAR0, MAR1 load
    MNOP                               ; Delay for MAR0, MAR1 load
                                       ; <-- MAR0 valid
    MMOV32     MR0, *MAR0[2]++         ; MR0 = X0, MAR0 += 2
                                       ; <-- MAR1 valid
    MMOV32     MR1, *MAR1[2]++         ; MR1 = Y0, MAR1 += 2
    MMPYF32    MR2, MR0, MR1           ; MR2 = A = X0 * Y0
||  MMOV32     MR0, *MAR0[2]++         ; In parallel MR0 = X1, MAR0 += 2
    MMOV32     MR1, *MAR1[2]++         ; MR1 = Y1, MAR1 += 2
    MMPYF32    MR3, MR0, MR1           ; MR3 = B = X1 * Y1
||  MMOV32     MR0, *MAR0[2]++         ; In parallel MR0 = X2, MAR0 += 2
    MMOV32     MR1, *MAR1[2]++         ; MR1 = Y2, MAR2 += 2
    MMACF32    MR3, MR2, MR2, MR0, MR1 ; MR3 = A + B, MR2 = C = X2 * Y2
||  MMOV32     MR0, *MAR0[2]++         ; In parallel MR0 = X3
    MMOV32     MR1, *MAR1[2]++         ; MR1 = Y3
    MMACF32    MR3, MR2, MR2, MR0, MR1 ; MR3 = (A + B) + C, MR2 = D = X3 * Y3
||  MMOV32     MR0, *MAR0              ; In parallel MR0 = X4
    MMOV32     MR1, *MAR1              ; MR1 = Y4
    MMPYF32    MR2, MR0, MR1           ; MR2 = E = X4 * Y4
||  MADDF32    MR3, MR3, MR2           ; in parallel MR3 = (A + B + C) + D

    MADDF32    MR3, MR3, MR2           ; MR3 = (A + B + C + D) + E
    MMOV32     @_Result, MR3           ; Store the result
    MSTOP                              ; end of task
```

See also MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRd, MRe, MRf ||MMOV32 MRa, mem32

32-Bit Floating-Point Multiply with Parallel Move

Operands

MRd	CLA floating-point destination register for MMPYF32 (MR0 to MR3) MRd cannot be the same register as MRa
MRe	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRf	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRa	CLA floating-point destination register for MMOV32 (MR0 to MR3) MRa cannot be the same register as MRd
mem32	32-bit memory location accessed using one of the available addressing modes. This will be the source of MMOV32.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0000 ffee ddaa addr
```

Description

Multiply the contents of two floating-point registers and load another.

```
MRd = MRe * MRf;
MRa = [mem32];
```

Restrictions

The destination register for the MMPYF32 and the MMOV32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

The MMOV32 Instruction will set the NF and ZF flags as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
```

Pipeline

Both MMPYF32 and MMOV32 complete in a single cycle.

Example 1

```
; Given M1, X1 and B1 are 32-bit floating point
; Calculate Y1 = M1*X1+B1
;
_Cla1Task1:
    MMOV32     MR0, @M1         ; Load MR0 with M1
    MMOV32     MR1, @X1         ; Load MR1 with X1
    MMPYF32    MR1, MR1, MR0    ; Multiply M1*X1
||  MMOV32     MR0, @B1         ; and in parallel load MR0 with B1
    MADDF32    MR1, MR1, MR0    ; Add M*X1 to B1 and store in MR1
    MMOV32     @Y1, MR1         ; Store the result
    MSTOP                       ; end of task
```

MMPYF32 MRd, MRe, MRf ||MMOV32 MRa, mem32 (continued)

32-Bit Floating-Point Multiply with Parallel Move

Example 2

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A * B) * C
;
_Cla1Task2:
    MMOV32    MR0, @A         ; Load MR0 with A
    MMOV32    MR1, @B         ; Load MR1 with B
    MMPYF32   MR1, MR1, MR0   ; Multiply A*B
||  MMOV32    MR0, @C         ; and in parallel load MR0 with C
    MMPYF32   MR1, MR1, MR0   ; Multiply (A*B) by C
||  MMOV32    @Y2, MR1        ; and in parallel store A*B
    MMOV32    @Y3, MR1        ; Store the result
    MSTOP                     ; end of task
```

See also

MMPYF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRd, MRe, MRf ||MMOV32 mem32, MRa

32-Bit Floating-Point Multiply with Parallel Move

Operands

MRd	CLA floating-point destination register for MMPYF32 (MR0 to MR3)
MRe	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRf	CLA floating-point source register for MMPYF32 (MR0 to MR3)
mem32	32-bit memory location accessed using one of the available addressing modes. This will be the destination of MMOV32.
MRa	CLA floating-point source register for MMOV32 (MR0 to MR3)

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0100 ffee ddaa addr
```

Description

Multiply the contents of two floating-point registers and move from memory to register.

```
MRd = MRe * MRf;
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 generates an underflow condition.
- LVF = 1 if MMPYF32 generates an overflow condition.

Pipeline

MMPYF32 and MMOV32 both complete in a single cycle.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A * B) * C
;
_Cla1Task2:
     MMOV32    MR0, @A         ; Load MR0 with A
     MMOV32    MR1, @B         ; Load MR1 with B
     MMPYF32   MR1, MR1, MR0   ; Multiply A*B
||   MMOV32    MR0, @C         ; and in parallel load MR0 with C
     MMPYF32   MR1, MR1, MR0   ; Multiply (A*B) by C
||   MMOV32    @Y2, MR1        ; and in parallel store A*B
     MMOV32    @Y3, MR1        ; Store the result
     MSTOP                     ; end of task
```

See also

MMPYF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MMACF32 MR3, MR2, MRd, MRe, MRf || MMOV32 MRa, mem32

MMPYF32 MRa, MRb, MRc ||MSUBF32 MRd, MRe, MRf

32-Bit Floating-Point Multiply with Parallel Subtract

Operands

MRa	CLA floating-point destination register for MMPYF32 (MR0 to MR3) MRa cannot be the same register as MRd
MRb	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRc	CLA floating-point source register for MMPYF32 (MR0 to MR3)
MRd	CLA floating-point destination register for MSUBF32 (MR0 to MR3) MRd cannot be the same register as MRa
MRe	CLA floating-point source register for MSUBF32 (MR0 to MR3)
MRf	CLA floating-point source register for MSUBF32 (MR0 to MR3)

Opcode

```
LSW: 0000 ffee ddcc bbaa
MSW: 0111 1010 0100 0000
```

Description

Multiply the contents of two floating-point registers with parallel subtraction of two registers.

```
MRa = MRb * MRc;
MRd = MRe - MRf;
```

Restrictions

The destination register for the MMPYF32 and the MSUBF32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MMPYF32 or MSUBF32 generates an underflow condition.
- LVF = 1 if MMPYF32 or MSUBF32 generates an overflow condition.

Pipeline

MMPYF32 and MSUBF32 both complete in a single cycle.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = (A * B)
;           Y3 = (A - B)
;
_Cla1Task2:
     MMOV32   MR0, @A          ; Load MR0 with A
     MMOV32   MR1, @B          ; Load MR1 with B
     MMPYF32  MR2, MR0, MR1    ; Multiply (A*B)
||   MSUBF32  MR3, MR0, MR1    ; and in parallel Sub (A-B)
     MMOV32   @Y2, MR2         ; Store A*B
     MMOV32   @Y3, MR3         ; Store A-B
     MSTOP                     ; end of task
```

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MSUBF32 MRd, MRe, MRf || MMOV32 mem32, MRa

MNEGF32 MRa, MRb{, CNDF}

Conditional Negation

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
CNDF	condition tested

Opcode

```
LSW: 0000 0000 cndf bbaa
MSW: 0111 1010 1000 0000
```

Description

```
if (CNDF == true) {MRa = - MRb; }
else {MRa = MRb; }
```

CNDF is one of the following conditions:

Encode[1]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF[2]	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF, and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

Pipeline

This is a single-cycle instruction.

MNEGF32 MRa, MRb{, CNDF} (continued)

Conditional Negation

Example 1

```
; Show the basic operation of MNEGF32
;
     MMOVIZ    MR0, #5.0       ; MR0 = 5.0 (0x40A00000)
     MMOVIZ    MR1, #4.0       ; MR1 = 4.0 (0x40800000)
     MMOVIZ    MR2, #-1.5      ; MR2 = -1.5 (0xBFC00000)
     MMPYF32   MR3, MR1, MR2   ; MR3 = -6.0
     MMPYF32   MR0, MR0, MR1   ; MR0 = 20.0
     MMOVIZ    MR1, #0.0
     MCMPF32   MR3, MR1        ; NF = 1
     MNEGF32   MR3, MR3, LT    ; if NF = 1, MR3 = 6.0
     MCMPF32   MR0, MR1        ; NF = 0
     MNEGF32   MR0, MR0, GEQ   ; if NF = 0, MR0 = -20.0
```

Example 2

```
; Calculate Num/Den using a Newton-Raphson algorithum for 1/Den
; Ye = Estimate(1/X)
; Ye = Ye*(2.0 - Ye*X)
; Ye = Ye*(2.0 - Ye*X)
;
_Cla1Task1:
    MMOV32    MR1, @_Den      ; MR1 = Den
    MEINVF32  MR2, MR1        ; MR2 = Ye = Estimate(1/Den)
    MMPYF32   MR3, MR2, MR1   ; MR3 = Ye*Den
    MSUBF32   MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32   MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
    MMPYF32   MR3, MR2, MR1   ; MR3 = Ye*Den
||  MMOV32    MR0, @_Num      ; MR0 = Num
    MSUBF32   MR3, #2.0, MR3  ; MR3 = 2.0 - Ye*Den
    MMPYF32   MR2, MR2, MR3   ; MR2 = Ye = Ye*(2.0 - Ye*Den)
||  MMOV32    MR1, @_Den      ; Reload Den To Set Sign
    MNEGF32   MR0, MR0, EQ    ; if(Den == 0.0) Change Sign Of Num
    MMPYF32   MR0, MR2, MR0   ; MR0 = Y = Ye*Num
    MMOV32    @_Dest, MR0     ; Store result
    MSTOP                     ; end of task
```

See also MABSF32 MRa, MRb

MNOP

No Operation

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1010 0000
```

Description

Do nothing. This instruction is used to fill required pipeline delay slots when other instructions are not available to fill the slots.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; X is an array of 32-bit floating-point values
; Find the maximum value in an array X
; and store it in Result
;
_Cla1Task1:
    MMOVI16    MAR1,#_X          ; Start address
    MUI16TOF32 MR0, @_len        ; Length of the array
    MNOP                         ; delay for MAR1 load
    MNOP                         ; delay for MAR1 load
    MMOV32     MR1, *MAR1[2]++   ; MR1 = X0
LOOP
    MMOV32     MR2, *MAR1[2]++   ; MR2 = next element
    MMAXF32    MR1, MR2          ; MR1 = MAX(MR1, MR2)
    MADDF32    MR0, MR0, #-1.0   ; Decrememt the counter
    MCMPF32    MR0 #0.0          ; Set/clear flags for MBCNDD
    MNOP                         ; Too late to affect MBCNDD
    MNOP                         ; Too late to affect MBCNDD
    MNOP                         ; Too late to affect MBCNDD
    MBCNDD    LOOP, NEQ          ; Branch if not equal to zero
    MMOV32    @_Result, MR1      ; Always executed
    MNOP                         ; Pad to seperate MBCNDD and MSTOP
    MNOP                         ; Pad to seperate MBCNDD and MSTOP
    MSTOP                        ; End of task
```

MOR32 MRa, MRb, MRc

Bitwise OR

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 1000 0000
```

Description

Bitwise OR of MRb with MRc.

```
MARa(31:0) = MARb(31:0) OR MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ    MR0, #0x5555 ; MR0 = 0x5555AAAA
MMOVXI    MR0, #0xAAAA
MMOVIZ    MR1, #0x5432 ; MR1 = 0x5432FEDC
MMOVXI    MR1, #0xFEDC
                       ; 0101 OR 0101 = 0101 (5)
                       ; 0101 OR 0100 = 0101 (5)
                       ; 0101 OR 0011 = 0111 (7)
                       ; 0101 OR 0010 = 0111 (7)
                       ; 1010 OR 1111 = 1111 (F)
                       ; 1010 OR 1110 = 1110 (E)
                       ; 1010 OR 1101 = 1111 (F)
                       ; 1010 OR 1100 = 1110 (E)
MOR32 MR2, MR1, MR0    ; MR3 = 0x5555FEFE
```

See also

MAND32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc

MRCNDD {CNDF}

Return Conditional Delayed

Operands

CNDF	optional condition.

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1001 1010 cndf
```

Description

If the specified condition is true, then the RPC field of MSTF is loaded into MPC and fetching continues from that location. Otherwise program fetches will continue without the return.

Please refer to the pipeline section for important information regarding this instruction.

```
if (CNDF == TRUE) MPC = RPC;
```

CNDF is one of the following conditions:

Encode(1)	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF(2)	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

The MRCNDD instruction by itself is a single-cycle instruction. As shown in Table 10-17, for each return 6 instruction slots are executed; three before the return instruction (d5-d7) and three after the return instruction (d8-d10). The total number of cycles for a return taken or not taken depends on the usage of these slots. That is, the number of cycles depends on how many slots are filled with a MNOP as well as which slots are filled. The effective number of cycles for a return can, therefore, range from 1 to 7 cycles. The number of cycles for a return taken may not be the same as for a return not taken.

MRCNDD {CNDF} (continued)

Return Conditional Delayed

Referring to the following code fragment and the pipeline diagrams in Table 10-17 and Table 10-18, the instructions before and after MRCNDD have the following properties:

```
;
;
<Instruction 1>     ; I1 Last instruction that can affect flags for
                    ; the MCCNDD operation
<Instruction 2>     ; I2 Cannot be stop, branch, call or return
<Instruction 3>     ; I3 Cannot be stop, branch, call or return
<Instruction 4>     ; I4 Cannot be stop, branch, call or return
MCCNDD _func, NEQ   ; Call to func if not eqal to zero
                    ; Three instructions after MCCNDD are always
                    ; executed whether the call is taken or not
<Instruction 5>     ; I5 Cannot be stop, branch, call or return
<Instruction 6>     ; I6 Cannot be stop, branch, call or return
<Instruction 7>     ; I7 Cannot be stop, branch, call or return
<Instruction 8>     ; I8 The address of this instruction is saved
                    ; in the RPC field of the MSTF register.
                    ; Upon return this value is loaded into MPC
                    ; and fetching continues from this point.
<Instruction 9>     ; I9
<Instruction 10>    ; I10
....
....
_func:
<Destination 1>     ; d1 Can be any instruction
<Destination 2>     ; d2
<Destination 3>     ; d3
<Destination 4>     ; d4 Last instruction that can affect flags for
                    ; the MRCNDD operation
<Destination 5>     ; d5 Cannot be stop, branch, call or return
<Destination 6>     ; d6 Cannot be stop, branch, call or return
<Destination 7>     ; d7 Cannot be stop, branch, call or return
MRCNDD NEQ          ; Return to <Instruction 8> if not equal to zero
                    ; Three instructions after MRCNDD are always
                    ; executed whether the return is taken or not
<Destination 8>     ; d8 Cannot be stop, branch, call or return
<Destination 9>     ; d9 Cannot be stop, branch, call or return
<Destination 10>    ; d10 Cannot be stop, branch, call or return
<Destination 11>    ; d11
<Destination 12>    ; d12
....
....
MSTOP
....
```

- **d4**
  - d4 is the last instruction that can effect the CNDF flags for the MRCNDD instruction. The CNDF flags are tested in the D2 phase of the pipeline. That is, a decision is made whether to return or not when MRCNDD is in the D2 phase.
  - There are no restrictions on the type of instruction for d4.
- **d5, d6 and d7**
  - The three instructions proceeding MRCNDD can change MSTF flags but will have no effect on whether the MRCNDD instruction makes the return or not. This is because the flag modification will occur after the D2 phase of the MRCNDD instruction.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.
- **d8, d9 and d10**
  - The three instructions following MRCNDD are always executed irrespective of whether the return is taken or not.
  - These instructions must not be the following: MSTOP, MDEBUGSTOP, MBCNDD, MCCNDD or MRCNDD.

MRCNDD {CNDF} (continued)

Return Conditional Delayed

Table 10-17. Pipeline Activity For MRCNDD, Return Not Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
d4	d4	d3	d2	d1	I7	I6	I5	
d5	d5	d4	d3	d2	d1	I7	I6	
d6	d6	d5	d4	d3	d2	d1	i7	
d7	d7	d6	d5	d4	d3	d2	d1	
MRCNDD	MRCNDD	d7	d6	d5	d4	d3	d2	
d8	d8	MRCNDD	d7	d6	d5	d4	d3	
d9	d9	d8	MRCNDD	d7	d6	d5	d4	
d10	d10	d9	d8	MRCNDD	d7	d6	d5	
d11	d11	d10	d9	d8	-	d7	d6	
d12	d12	d11	d10	d9	d8	-	d7	
etc....	....	d12	d11	d10	d9	d8	-	
....	....	....	d12	d11	d10	d9	d8	
....	....	....	....	d12	d11	d10	d9	
					d12	d11	d10	
						d12	d11	
							d12	

Table 10-18. Pipeline Activity For MRCNDD, Return Taken

Instruction	F1	F2	D1	D2	R1	R2	E	W
d4	d4	d3	d2	d1	I7	I6	I5	
d5	d5	d4	d3	d2	d1	I7	I6	
d6	d6	d5	d4	d3	d2	d1	i7	
d7	d7	d6	d5	d4	d3	d2	d1	
MRCNDD	MRCNDD	d7	d6	d5	d4	d3	d2	
d8	d8	MRCNDD	d7	d6	d5	d4	d3	
d9	d9	d8	MRCNDD	d7	d6	d5	d4	
d10	d10	d9	d8	MRCNDD	d7	d6	d5	
I8	I8	d10	d9	d8	-	d7	d6	
I9	I9	I8	d10	d9	d8	-	d7	
I10	I10	I9	I8	d10	d9	d8	-	
etc....	....	I10	I9	I8	d10	d9	d8	
....	....		I10	I9	I8	d10	d9	
....	....			I10	I9	I8	d10	
					I10	I9	I8	
						I10	I9	
							I10	

See also

MBCNDD #16BitDest, CNDF
MCCNDD 16BitDest, CNDF
MMOV32 mem32, MSTF
MMOV32 MSTF, mem32

MSETFLG FLAG, VALUE

Set or Clear Selected Floating-Point Status Flags

Operands

FLAG	8-bit mask indicating which floating-point status flags to change.
VALUE	8-bit mask indicating the flag value: 0 or 1.

Opcode

```
LSW: FFFF FFFF VVVV VVVV
MSW: 0111 1001 1100 0000
```

Description

The MSETFLG instruction is used to set or clear selected floating-point status flags in the MSTF register. The FLAG field is an 11-bit value that indicates which flags will be changed. That is, if a FLAG bit is set to 1 it indicates that flag will be changed; all other flags will not be modified. The bit mapping of the FLAG field is:

9	8	7	6	5	4	3	2	1	0
RNDF32	Reserved		TF	Reserved		ZF	NF	LUF	LVF

The VALUE field indicates the value the flag should be set to: 0 or 1.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	Yes	Yes	Yes	Yes	Yes

Any flag can be modified by this instruction. The MEALLOW and RPC fields cannot be modified with this instruction.

Pipeline

This is a single-cycle instruction.

Example

To make it easier and legible, the assembler accepts a FLAG=VALUE syntax for the MSTFLG operation as shown below:

```
MSETFLG RNDF32=0, TF=0, NF=1; FLAG = 11000100; VALUE = 00XXX1XX;
```

See also

MMOV32 mem32, MSTF
MMOV32 MSTF, mem32

MSTOP

Stop Task

Operands

none	This instruction does not have any operands

Opcode

```
LSW: 0000 0000 0000 0000
MSW: 0111 1111 1000 0000
```

Description

The MSTOP instruction must be placed to indicate the end of each task. In addition, placing MSTOP in unused memory locations within the CLA program RAM can be useful for debugging and preventing run away CLA code. When MSTOP enters the D2 phase of the pipeline, the MIRUN flag for the task is cleared and the associated interrupt is flagged in the PIE vector table.

There are three special cases that can occur when single-stepping a task such that the MPC reaches the MSTOP instruction.

1. If you are single-stepping or halted in "task A" and "task B" comes in before the MPC reaches the MSTOP, then "task B" will start if you continue to step through the MSTOP instruction. Basically if "task B" is pending before the MPC reaches MSTOP in "task A" then there is no issue in "task B" starting and no special action is required.
2. In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. If "task B" comes in at this point, it will be flagged in the MIFR register but it may or may not start if you continue to single-step through the MSTOP instruction of "task A". It depends on exactly when the new task comes in. To reliably start "task B" perform a soft reset and reconfigure the MIER bits. Once this is done, you can start single-stepping "task B".
3. Case 2 can be handled slightly differently if there is control over when "task B" comes in (for example using the IACK instruction to start the task). In this case you have single-stepped or halted in "task A" and the MPC has reached the MSTOP with no tasks pending. Before forcing "task B", run free to force the CLA out of the debug state. Once this is done you can force "task B" and continue debugging.

Restrictions

The MSTOP instruction cannot be placed 3 instructions before or after a MBCNDD, MCCNDD, or MRCNDD instruction.

Flags

This instruction does not modify flags in the MSTF register.

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

MSTOP (continued)

Stop Task

Pipeline

This is a single-cycle instruction. Table 10-19 shows the pipeline behavior of the MSTOP instruction. The MSTOP instruction cannot be placed with 3 instructions of a MBCNDD, MCCNDD, or MRCNDD instruction.

Table 10-19. Pipeline Activity For MSTOP

Instruction	F1	F2	D1	D2	R1	R2	E	W
I1	I1							
I2	I2	I1						
I3	I3	I2	I1					
MSTOP	MSTOP	I3	I2	I1				
I4	I4	MSTOP	I3	I2	I1			
I5	I5	I4	MSTOP	I3	I2	I1		
I6	I6	I5	I4	MSTOP	I3	I2	I1	
New Task Arbitrated and Prioritized	-	-	-	-	-	I3	I2	
New Task Arbitrated and Prioritized	-	-	-	-	-	-	I3	
I1	I1	-	-	-	-	-	-	
I2	I2	I1	-	-	-	-	-	
I3	I3	I2	I1	-	-	-	-	
I4	I4	I3	I2	I1	-	-	-	
I5	I5	I4	I3	I2	I1	-	-	
I6	I6	I5	I4	I3	I2	I1	-	
I7	I7	I6	I5	I4	I3	I2	I1	
etc								

Example

```
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
; Calculate Y2 = A - B - C
_Cla1Task3:
   MMOV32   MR0, @_A      ; MR0 = 1 (0x00000001)
   MMOV32   MR1, @_B      ; MR1 = 2 (0x00000002)
   MMOV32   MR2, @_C      ; MR2 = -7 (0xFFFFFFF9)
   MSUB32   MR3, MR0, MR1 ; A + B
   MSUB32   MR3, MR3, MR2 ; A + B + C = 6 (0x0000006)
   MMOV32   @_y2, MR3     ; Store y2
   MSTOP                  ; End of task
```

See also

MDEBUGSTOP

MSUB32 MRa, MRb, MRc

32-Bit Integer Subtraction

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point destination register (MR0 to MR3)
MRc	CLA floating-point destination register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 1110 0000
```

Description

32-bit integer addition of MRb and MRc.

```
MARa(31:0) = MARb(31:0) - MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified as follows:

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
; Given A = (int32)1
;       B = (int32)2
;       C = (int32)-7
;
;
Calculate Y2 = A - B - C
;
_Cla1Task3:
   MMOV32   MR0, @_A          ; MR0 = 1 (0x00000001)
   MMOV32   MR1, @_B          ; MR1 = 2 (0x00000002)
   MMOV32   MR2, @_C          ; MR2 = -7 (0xFFFFFFF9)
   MSUB32   MR3, MR0, MR1     ; A + B
   MSUB32   MR3, MR3, MR2     ; A + B + C = 6 (0x0000006)
   MMOV32   @_y2, MR3         ; Store y2
   MSTOP                      ; End of task
```

See also

MADD32 MRa, MRb, MRc
MAND32 MRa, MRb, MRc
MASR32 MRa, #SHIFT
MLSL32 MRa, #SHIFT
MLSR32 MRa, #SHIFT
MOR32 MRa, MRb, MRc
MXOR32 MRa, MRb, MRc

MSUBF32 MRa, MRb, MRc

32-Bit Floating-Point Subtraction

Operands

MRa	CLA floating-point destination register (MR0 to R1)
MRb	CLA floating-point source register (MR0 to R1)
MRc	CLA floating-point source register (MR0 to R1)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 0100 0000
```

Description

Subtract the contents of two floating-point registers

```
MRa = MRb - MRc;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Given A, B and C are 32-bit floating-point numbers
; Calculate Y2 = A + B - C
;
_Cla1Task5:
   MMOV32    MR0, @_A      ; Load MR0 with A
   MMOV32    MR1, @_B      ; Load MR1 with B
   MADDF32   MR0, MR1, MR0 ; Add A + B
|| MMOV32    MR1, @_C      ;   and in parallel load C
   MSUBF32   MR0, MR0, MR1 ; Subtract C from (A + B)
   MMOV32    @Y,  MR0      ; (A+B) - C
   MSTOP                   ; end of task
```

See also

MSUBF32 MRa, #16FHi, MRb
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MSUBF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSUBF32 MRa, #16FHi, MRb

32-Bit Floating-Point Subtraction

Operands

MRa	CLA floating-point destination register (MR0 to R1)
#16FHi	A 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0.
MRb	CLA floating-point source register (MR0 to R1)

Opcode

```
LSW: IIII IIII IIII IIII
MSW: 0111 1000 0000 baaa
```

Description

Subtract MRb from the floating-point value represented by the immediate operand. Store the result of the addition in MRa.

#16FHi is a 16-bit immediate value that represents the upper 16-bits of an IEEE 32-bit floating-point value. The low 16-bits of the mantissa are assumed to be all 0. #16FHi is most useful for representing constants where the lowest 16-bits of the mantissa are 0. Some examples are 2.0 (0x40000000), 4.0 (0x40800000), 0.5 (0x3F000000), and -1.5 (0xBFC00000). The assembler will accept either a hex or float as the immediate value. That is, the value -1.5 can be represented as #-1.5 or #0xBFC0.

```
MRa = #16FHi:0 - MRb;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

Pipeline

This is a single-cycle instruction.

Example

```
; Y = sqrt(X)
; Ye = Estimate(1/sqrt(X));
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Ye = Ye*(1.5 - Ye*Ye*X*0.5)
; Y = X*Ye
;
_Cla1Task3:
   MMOV32     MR0, @_x           ; MR0 = X
   MEISQRTF32 MR1, MR0           ; MR1 = Ye = Estimate(1/sqrt(X))
   MMOV32     MR1, @_x, EQ       ; if(X == 0.0) Ye = 0.0
   MMPYF32    MR3, MR0, #0.5     ; MR3 = X*0.5
   MMPYF32    MR2, MR1, MR3      ; MR2 = Ye*X*0.5
   MMPYF32    MR2, MR1, MR2      ; MR2 = Ye*Ye*X*0.5
   MSUBF32    MR2, #1.5, MR2     ; MR2 = 1.5 - Ye*Ye*X*0.5
   MMPYF32    MR1, MR1, MR2      ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
   MMPYF32    MR2, MR1, MR3      ; MR2 = Ye*X*0.5
   MMPYF32    MR2, MR1, MR2      ; MR2 = Ye*Ye*X*0.5
   MSUBF32    MR2, #1.5, MR2     ; MR2 = 1.5 - Ye*Ye*X*0.5
   MMPYF32    MR1, MR1, MR2      ; MR1 = Ye = Ye*(1.5 - Ye*Ye*X*0.5)
   MMPYF32    MR0, MR1, MR0      ; MR0 = Y = Ye*X
   MMOV32     @_y, MR0           ; Store Y = sqrt(X)
   MSTOP                         ; end of task
```

MSUBF32 MRa, #16FHi, MRb (continued)

32-Bit Floating-Point Subtraction

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MSUBF32 MRd, MRe, MRf || MMOV32 mem32, MRa
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSUBF32 MRd, MRe, MRf ||MMOV32 MRa, mem32

32-Bit Floating-Point Subtraction with Parallel Move

Operands

MRd	CLA floating-point destination register (MR0 to MR3) for the MSUBF32 operation MRd cannot be the same register as MRa
MRe	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
MRf	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
MRa	CLA floating-point destination register (MR0 to MR3) for the MMOV32 operation MRa cannot be the same register as MRd
mem32	32-bit memory location accessed using one of the available addressing modes. Source for the MMOV32 operation.

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0010 ffee ddaa addr
```

Description

Subtract the contents of two floating-point registers and move from memory to a floating-point register.

```
MRd = MRe - MRf;
MRa = [mem32];
```

Restrictions

The destination register for the MSUBF32 and the MMOV32 must be unique. That is, MRa cannot be the same register as MRd.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

The MMOV32 Instruction will set the NF and ZF flags.

Pipeline

Both MSUBF32 and MMOV32 complete in a single cycle.

Example

```
NF = MRa(31);
ZF = 0;
if(MRa(30:23) == 0) { ZF = 1; NF = 0; }
```

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRa, #16FHi, MRb
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSUBF32 MRd, MRe, MRf ||MMOV32 mem32, MRa

32-Bit Floating-Point Subtraction with Parallel Move

Operands

MRd	CLA floating-point destination register (MR0 to MR3) for the MSUBF32 operation
MRe	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
MRf	CLA floating-point source register (MR0 to MR3) for the MSUBF32 operation
mem32	32-bit destination memory location for the MMOV32 operation
MRa	CLA floating-point source register (MR0 to MR3) for the MMOV32 operation

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0110 ffee ddaa addr
```

Description

Subtract the contents of two floating-point registers and move from a floating-point register to memory.

```
MRd = MRe - MRf;
[mem32] = MRa;
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	Yes	Yes

The MSTF register flags are modified as follows:

- LUF = 1 if MSUBF32 generates an underflow condition.
- LVF = 1 if MSUBF32 generates an overflow condition.

Pipeline

Both MSUBF32 and MMOV32 complete in a single cycle.

See also

MSUBF32 MRa, MRb, MRc
MSUBF32 MRa, #16FHi, MRb
MSUBF32 MRd, MRe, MRf || MMOV32 MRa, mem32
MMPYF32 MRa, MRb, MRc || MSUBF32 MRd, MRe, MRf

MSWAPF MRa, MRb {, CNDF}

Conditional Swap

Operands

MRa	CLA floating-point register (MR0 to MR3)
MRb	CLA floating-point register (MR0 to MR3)
CNDF	Optional condition tested based on the MSTF flags

Opcode

```
LSW: 0000 0000 CNDF bbaa
MSW: 0111 1011 0000 0000
```

Description

Conditional swap of MRa and MRb.

```
if (CNDF == true) swap MRa and MRb;
```

CNDF is one of the following conditions:

Encode[1]	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF[2]	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

No flags affected

Pipeline

This is a single-cycle instruction.

MSWAPF MRa, MRb {, CNDF} (continued)

Conditional Swap

Example

```
; X is an array of 32-bit floating-point values
; and has len elements. Find the maximum value in
; the array and store it in Result
;
; Note: MCMPF32 and MSWAPF can be replaced by MMAXF32
;
_Cla1Task1:
    MMOVI16    MAR1,#_X          ; Start address
    MUI16TOF32 MR0, @_len        ; Length of the array
    MNOP                         ; delay for MAR1 load
    MNOP                         ; delay for MAR1 load
    MMOV32     MR1, *MAR1[2]++   ; MR1 = X0
LOOP
    MMOV32     MR2, *MAR1[2]++   ; MR2 = next element
    MCMPF32    MR2, MR1          ; Compare MR2 with MR1
    MSWAPF     MR1, MR2, GT      ; MR1 = MAX(MR1, MR2)
    MADDF32    MR0, MR0, #-1.0   ; Decrememt the counter
    MCMPF32    MR0 #0.0          ; Set/clear flags for MBCNDD
    MNOP
    MNOP
    MNOP
    MBCNDD     LOOP, NEQ         ; Branch if not equal to zero
    MMOV32     @_Result, MR1     ; Always executed
    MNOP                         ; Always executed
    MNOP                         ; Always executed
    MSTOP                        ; End of task
```

MTESTTF CNDF

Test MSTF Register Flag Condition

Operands

CNDF	condition to test based on MSTF flags

Opcode

```
LSW: 0000 0000 0000 cndf
MSW: 0111 1111 0100 0000
```

Description

Test the CLA floating-point condition and if true, set the MSTF[TF] flag. If the condition is false, clear the MSTF[TF] flag. This is useful for temporarily storing a condition for later use.

```
if (CNDF == true) TF = 1;
else TF = 0;
```

CNDF is one of the following conditions:

Encode(1)	CNDF	Description	MSTF Flags Tested
0000	NEQ	Not equal to zero	ZF == 0
0001	EQ	Equal to zero	ZF == 1
0010	GT	Greater than zero	ZF == 0 AND NF == 0
0011	GEQ	Greater than or equal to zero	NF == 0
0100	LT	Less than zero	NF == 1
0101	LEQ	Less than or equal to zero	ZF == 1 OR NF == 1
1010	TF	Test flag set	TF == 1
1011	NTF	Test flag not set	TF == 0
1100	LU	Latched underflow	LUF == 1
1101	LV	Latched overflow	LVF == 1
1110	UNC	Unconditional	None
1111	UNCF(2)	Unconditional with flag modification	None

(1) Values not shown are reserved.
(2) This is the default operation if no CNDF field is specified. This condition will allow the ZF and NF flags to be modified when a conditional operation is executed. All other conditions will not modify these flags.

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	Yes	No	No	No	No

```
TF = 0;
if (CNDF == true) TF = 1;
```

Note: If (CNDF == UNC or UNCF), the TF flag will be set to 1.

Pipeline

This is a single-cycle instruction.

MTESTTF CNDF (continued)

Test MSTF Register Flag Condition

Example

```
; if (State == 0.1)
;    RampState = RampState || RAMPMASK
; else if (State == 0.01)
;    CoastState = CoastState || COASTMASK
; else
;    SteadyState = SteadyState || STEADYMASK
;
_Cla1Task2:
   MMOV32    MR0, @_State
   MCMPF32   MR0, #0.1         ; Affects flags for 1st MBCNDD (A)
   MCMPF32   MR0, #0.01        ; Check used by 2nd MBCNDD (B)
   MTESTTF   EQ                ; Store EQ flag in TF for 2nd MBCNDD (B)
   MNOP
   MBCNDD    _Skip1, NEQ       ; (A) If State != 0.1, go to Skip1
   MMOV32    MR1, @_RampState  ; Always executed
   MMOVXI    MR2, #RAMPMASK    ; Always executed
   MOR32     MR1, MR2          ; Always executed
   MMOV32    @_RampState, MR1  ; Execute if (A) branch not taken
   MSTOP                       ; end of task if (A) branch not taken
_Skip1:
   MMOV32    MR3, @_SteadyState
   MMOVXI    MR2, #STEADYMASK
   MOR32     MR3, MR2
   MBCNDD    _Skip2, NTF       ; (B) if State != .01, go to Skip2
   MMOV32    MR1, @_CoastState ; Always executed
   MMOVXI    MR2, #COASTMASK   ; Always executed
   MOR32     MR1, MR2          ; Always executed
   MMOV32    @_CoastState, MR1 ; Execute if (B) branch not taken
   MSTOP                       ; end of task if (B) branch not taken
_Skip2:
   MMOV32 @_SteadyState, MR3   ; Executed if (B) branch taken
   MSTOP
```

MUI16TOF32 MRa, mem16

Convert Unsigned 16-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem16	16-bit source memory location

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0101 01aa addr
```

Description

When converting F32 to I16/UI16 data format, the MF32TOI16/UI16 operation truncates to zero while the MF32TOI16R/UI16R operation will round to nearest (even) value.

```
MRa = UI16TOF32[mem16];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, MRb

MUI16TOF32 MRa, MRb

Convert Unsigned 16-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1110 1110 0000
```

Description

Convert an unsigned 16-bit integer to a 32-bit floating-point value. When converting float32 to I16/UI16 data format, the MF32TOI16/UI16 operation truncates to zero while the MF32TOI16R/UI16R operation will round to nearest (even) value.

```
MRa = UI16TOF32[MRb];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVXI MR1, #0x800F ; MR1(15:0) = 32783 (0x800F)
MUI16TOF32 MR0, MR1 ; MR0 = UI16TOF32 (MR1(15:0))
                    ; = 32783.0 (0x47000F00)
```

See also

MF32TOI16 MRa, MRb
MF32TOI16R MRa, MRb
MF32TOUI16 MRa, MRb
MF32TOUI16R MRa, MRb
MI16TOF32 MRa, MRb
MI16TOF32 MRa, mem16
MUI16TOF32 MRa, mem16

MUI32TOF32 MRa, mem32

Convert Unsigned 32-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
mem32	32-bit memory location accessed using one of the available addressing modes

Opcode

```
LSW: mmmm mmmm mmmm mmmm
MSW: 0111 0100 10aa addr
```

Description

```
MRa = UI32TOF32[mem32];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
; Given x2, m2 and b2 are Uint32 numbers:
;
; x2 = Uint32(2) = 0x00000002
; m2 = Uint32(1) = 0x00000001
; b2 = Uint32(3) = 0x00000003
;
; Calculate y2 = x2 * m2 + b2
;
_Cla1Task1:
   MUI32TOF32  MR0, @_m2      ; MR0 = 1.0 (0x3F800000)
   MUI32TOF32  MR1, @_x2      ; MR1 = 2.0 (0x40000000)
   MUI32TOF32  MR2, @_b2      ; MR2 = 3.0 (0x40400000)
   MMPYF32     MR3, MR0, MR1  ; M*X
   MADDF32     MR3, MR2, MR3  ; Y=MX+B = 5.0 (0x40A00000)
   MF32TOUI32  MR3, MR3       ; Y = Uint32(5.0) = 0x00000005
   MMOV32      @_y2, MR3      ; store result
   MSTOP                      ; end of task
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, mem32
MI32TOF32 MRa, MRb
MUI32TOF32 MRa, MRb

MUI32TOF32 MRa, MRb

Convert Unsigned 32-Bit Integer to 32-Bit Floating-Point Value

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 0000 bbaa
MSW: 0111 1101 1100 0000
```

Description

```
MRa = UI32TOF32 [MRb];
```

Flags

This instruction does not affect any flags:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	No	No	No	No

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ     MR3, #0x8000 ; MR3(31:16) = 0x8000
MMOVXI     MR3, #0x1111 ; MR3(15:0) = 0x1111
                        ; MR3 = 2147488017
MUI32TOF32 MR3, MR3     ; MR3 = MUI32TOF32 (MR3) = 2147488017.0 (0x4F000011)
```

See also

MF32TOI32 MRa, MRb
MF32TOUI32 MRa, MRb
MI32TOF32 MRa, mem32
MI32TOF32 MRa, MRb
MUI32TOF32 MRa, mem32

MXOR32 MRa, MRb, MRc

Bitwise Exclusive Or

Operands

MRa	CLA floating-point destination register (MR0 to MR3)
MRb	CLA floating-point source register (MR0 to MR3)
MRc	CLA floating-point source register (MR0 to MR3)

Opcode

```
LSW: 0000 0000 00cc bbaa
MSW: 0111 1100 1010 0000
```

Description

Bitwise XOR of MRb with MRc.

```
MARa(31:0) = MARb(31:0) XOR MRc(31:0);
```

Flags

This instruction modifies the following flags in the MSTF register:

Flag	TF	ZF	NF	LUF	LVF
Modified	No	Yes	Yes	No	No

The MSTF register flags are modified based on the integer results of the operation.

```
NF = MRa(31);
ZF = 0;
if(MRa(31:0) == 0) { ZF = 1; }
```

Pipeline

This is a single-cycle instruction.

Example

```
MMOVIZ MR0, #0x5555   ; MR0 = 0x5555AAAA
  MMOVXI MR0, #0xAAAA
  MMOVIZ MR1, #0x5432   ; MR1 = 0x5432FEDC
  MMOVXI MR1, #0xFEDC
  ; 0101 XOR 0101 = 0000 (0)
  ; 0101 XOR 0100 = 0001 (1)
  ; 0101 XOR 0011 = 0110 (6)
  ; 0101 XOR 0010 = 0111 (7)
  ; 1010 XOR 1111 = 0101 (5)
  ; 1010 XOR 1110 = 0100 (4)
  ; 1010 XOR 1101 = 0111 (7)
  ; 1010 XOR 1100 = 0110 (6)
  MXOR32 MR2, MR1, MR0  ; MR3 = 0x01675476
```

See also

MAND32 MRa, MRb, MRc
MOR32 MRa, MRb, MRc

10.7 CLA Registers

The CLA register set is independent from that of the main CPU. This section describes the Control Law Accelerator registers.

10.7.1 Register Memory Mapping

Table 10-20 lists the CLA module control and status register set.

Table 10-20. CLA Module Control and Status Register Set

Name	Offset	Size (x16)	EALLOW	CSM Protected	Description	Section
Task Interrupt Vectors						
MVECT1	0x0000	1	Yes	Yes	Task 1 Interrupt Vector	Section 10.7.2.1
MVECT2	0x0001	1	Yes	Yes	Task 2 Interrupt Vector	Section 10.7.2.1
MVECT3	0x0002	1	Yes	Yes	Task 3 Interrupt Vector	Section 10.7.2.1
MVECT4	0x0003	1	Yes	Yes	Task 4 Interrupt Vector	Section 10.7.2.1
MVECT5	0x0004	1	Yes	Yes	Task 5 Interrupt Vector	Section 10.7.2.1
MVECT6	0x0005	1	Yes	Yes	Task 6 Interrupt Vector	Section 10.7.2.1
MVECT7	0x0006	1	Yes	Yes	Task 7 Interrupt Vector	Section 10.7.2.1
MVECT8	0x0007	1	Yes	Yes	Task 8 Interrupt Vector	Section 10.7.2.1
Configuration Registers						
MCTL	0x0010	1	Yes	Yes	Control Register	Section 10.7.3.1
MMEMCFG	0x0011	1	Yes	Yes	Memory Configuration Register	Section 10.7.3.2
MPISRCSEL1	0x0014	2	Yes	Yes	Peripheral Interrupt Source Select 1 Register	Section 10.7.3.3
MIFR	0x0020	1	Yes	Yes	Interrupt Flag Register	Section 10.7.3.4
MIOVF	0x0021	1	Yes	Yes	Interrupt Overflow Flag Register	Section 10.7.3.5
MIFRC	0x0022	1	Yes	Yes	Interrupt Force Register	Section 10.7.3.6
MICLR	0x0023	1	Yes	Yes	Interrupt Flag Clear Register	Section 10.7.3.7
MICLROVF	0x0024	1	Yes	Yes	Interrupt Overflow Flag Clear Register	Section 10.7.3.8
MIER	0x0025	1	Yes	Yes	Interrupt Enable Register	Section 10.7.3.9
MIRUN	0x0026	1	Yes	Yes	Interrupt Run Status Register	Section 10.7.3.10
Execution Registers (1)						
MPC	0x0028	1	-	Yes	CLA Program Counter	Section 10.7.4.1
MAR0	0x002A	1	-	Yes	CLA Auxiliary Register 0	-
MAR1	0x002B	1	-	Yes	CLA Auxiliary Register 1	-
MSTF	0x002E	2	-	Yes	CLA Floating-Point Status Register	Section 10.7.4.2
MR0	0x0030	2	-	Yes	CLA Floating-Point Result Register 0	-
MR1	0x0034	2	-	Yes	CLA Floating-Point Result Register 1	-
MR2	0x0038	2	-	Yes	CLA Floating-Point Result Register 2	-
MR3	0x003C	2	-	Yes	CLA Floating-Point Result Register 3	-

(1) The main C28x CPU only has read access to the CLA execution registers for debug purposes. The main CPU cannot perform CPU or debugger writes to these registers.

10.7.2 Task Interrupt Vector Registers

Each CLA interrupt has its own interrupt vector (MVECT1 to MVECT8). This interrupt vector points to the first instruction of the associated task. When a task begins, the CLA will start fetching instructions at the location indicated by the appropriate MVECT register .

10.7.2.1 Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Register

The task interrupt vector registers (MVECT1/2/3/4/5/6/7/8) are is shown in Section 10.7.2.1 and described in Figure 10-2.

Figure 10-2. Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Register

15	12	11	0
Reserved		MVECT	
R-0		R-0	

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

Table 10-21. Task Interrupt Vector (MVECT1/2/3/4/5/6/7/8) Field Descriptions

Bits	Name	Value	Description (1)
15-12	Reserved		Any writes to these bit(s) must always have a value of 0.
11-0	MVECT	0000 - 0FFF	Offset of the first instruction in the associated task from the start of CLA program space. The CLA will begin instruction fetches from this location when the specific task begins. For example: If CLA program memory begins at CPU address 0x009000 and the code for task 5 begins at CPU address 0x009120, then MVECT5 should be initialized with 0x0120. There is one MVECT register per task. Interrupt 1 uses MVECT1, interrupt 2 uses MVECT2 and so forth.

(1) These registers are protected by EALLOW and the code security module.

10.7.3 Configuration Registers

The configuration registers are described here.

10.7.3.1 Control Register (MCTL)

The configuration control register (MCTL) is shown in Figure 10-3 and described in Table 10-22.

Figure 10-3. Control Register (MCTL)

15	8
Reserved	
R -0	

7	3	2	1	0
Reserved		IACKE	SOFTRESET	HARDRESET
R/W-0		R/W-0	R/W-0	R/W-0

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

Table 10-22. Control Register (MCTL) Field Descriptions

Bits	Name	Value	Description (1)
15-3	Reserved		Any writes to these bit(s) must always have a value of 0.
2	IACKE		IACK enable
		0	The CLA ignores the IACK instruction. (default)
		1	Enable the main CPU to use the IACK #16bit instruction to set MIFR bits in the same manner as writing to the MIFRC register. Each bit in the operand, #16bit, corresponds to a bit in the MIFRC register. Using IACK has the advantage of not having to first set the EALLOW bit. This allows the main CPU to efficiently trigger a CLA task through software.
			Examples `IACK #0x0001` Write a 1 to MIFRC bit 0 to force task 1
			`IACK #0x0003` Write a 1 to MIFRC bit 0 and 1 to force task 1 and task 2
1	SOFTRESET		Soft Reset
		0	This bit always reads back 0 and writes of 0 are ignored.
		1	Writing a 1 will cause a soft reset of the CLA. This will stop the current task, clear the MIRUN flag and clear all bits in the MIER register. After a soft reset you must wait at least 1 SYSCLKOUT cycle before reconfiguring the MIER bits. If these two operations are done back-to-back then the MIER bits will not get set.
0	HARDRESET		Hard Reset
		0	This bit always reads back 0 and writes of 0 are ignored.
		1	Writing a 1 will cause a hard reset of the CLA. This will set all CLA registers to their default state.

(1) This register is protected by EALLOW and the code security module.

10.7.3.2 Memory Configuration Register (MMEMCFG)

The MMEMCFG register is used to map the CLA program and data RAMs to either the CPU or the CLA memory space.

Figure 10-4. Memory Configuration Register (MMEMCFG)

15					11	10	9	8
Reserved						Reserved	Reserved	Reserved
R-0						R-0	R-0	R-0

7	6	5	4	3		1	0
Reserved	Reserved	RAM1E	RAM0E	Reserved			PROGE
R-0	R-0	R/W-0	R/W-0	R-0			R/W-0

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

Table 10-23. Memory Configuration Register (MMEMCFG) Field Descriptions

Bits	Field	Value	Description
15-6	Reserved		Any writes to these bits must always have a value of 0
5	RAM1E		CLA Data RAM 1 Enable
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory
		0	CLA data RAM block 1 is mapped to the main CPU program and data space. CLA reads will return zero. (default)
		1	CLA data RAM block 1 is mapped to the CLA space. The RAM1CPUE bit determines the CPU access to this memory
4	RAM0E		CLA Data RAM 0 Enable
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory
		0	CLA data RAM block 0 is mapped to the main CPU program and data space. CLA reads will return zero. (default)
		1	CLA data RAM block 0 is mapped to the CLA space. The RAM0CPUE bit determines the CPU access to this memory
3 - 1	Reserved		Any writes to these bit(s) must always have a value of 0
0	PROGE		CLA Program Space Enable
			Allow two SYSCLKOUT cycles between changing this bit and accessing the memory
		0	CLA program RAM is mapped to the main CPU program and data space. If the CLA attempts a program fetch the result will be the same as an illegal opcode fetch as described in Section 3.4.(default)
		1	CLA program RAM is mapped to the CLA program space. The main CPU can only make debug accesses to this block
			In this state the CLA has higher priority than CPU debug reads. It is, therefore, possible for the CLA to permanently block debug accesses if the CLA is executing in a loop. This might occur when if the CLA code has a bug. To avoid this issue, the program memory will return 0x0000 for CPU debug reads (ignore writes) when the CLA is running. When the CLA is halted or idle then normal CPU debug read and write access can be performed

10.7.3.3 CLA Peripheral Interrupt Source Select 1 Register (MPISRCSEL1)

Each task has specific peripherals that can start it. For example, Task2 can be started by ADCINT2 or EPWM2_INT. To configure which of the possible peripherals will start a task configure the MPISRCSEL1 register shown in Figure 10-5. Choosing the option "no interrupt source" means that only the main CPU software will be able to start the given task.

Figure 10-5. CLA Peripheral Interrupt Source Select 1 Register (MPISRCSEL1)

31–28	27–24	23–20	19–16
PERINT8SEL	PERINT7SEL	PERINT6SEL	PERINT5SEL
R/W-0	R/W-0	R/W-0	R/W-0

15–12	11–8	7–4	3–0
PERINT4SEL	PERINT3SEL	PERINT2SEL	PERINT1SEL
R/W-0	R/W-0	R/W-0	R/W-0

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

Table 10-24. Peripheral Interrupt Source Select 1 (MPISRCSEL1) Register Field Descriptions

Bits	Field	Value [1]	Description[2]
31-28	PERINT8SEL		Task 8 Peripheral Interrupt Input Select
		0000	ADCINT8 is the input for interrupt task 8. (default)
		0010	CPU Timer 0 is the input for interrupt task 8. (TINT0)
		Other	No interrupt source for task 8.
27-24	PERINT7SEL		Task 7 Peripheral Interrupt Input Select
		0000	ADCINT7 is the input for interrupt task 7. (default)
		0010	ePWM7 is the input for interrupt task 7. (EPWM7_INT)
		Other	No interrupt source for task 7.
23-20	PERINT6SEL		Task 6 Peripheral Interrupt Input Select
		0000	ADCINT6 is the input for interrupt task 6. (default)
		0010	ePWM6 is the input for interrupt task 6. (EPWM6_INT)
		Other	No interrupt source for task 6.
19-16	PERINT5SEL		Task 5 Peripheral Interrupt Input Select
		0000	ADCINT5 is the input for interrupt task 5. (default)
		0010	ePWM5 is the input for interrupt task 5. (EPWM5_INT)
		Other	No interrupt source for task 5.
15-12	PERINT4SEL		Task 4 Peripheral Interrupt Input Select
		0000	ADCINT4 is the input for interrupt task 4. (default)
		0010	ePWM4 is the input for interrupt task 4. (EPWM4_INT)
		Other	No interrupt source for task 4.
11-8	PERINT3SEL		Task 3 Peripheral Interrupt Input Select
		0000	ADCINT3 is the input for interrupt task 3. (default)
		0010	ePWM3 is the input for interrupt task 3. (EPWM3_INT)
		xxx1	No interrupt source for task 3.
7-4	PERINT2SEL		Task 2 Peripheral Interrupt Input Select
		0000	ADCINT2 is the input for interrupt task 2. (default)
		0010	ePWM2 is the input for interrupt task 2. (EPWM2_INT)
		xxx1	No interrupt source for task 2.
3-0	PERINT1SEL		Task 1 Peripheral Interrupt Input Select
		0000	ADCINT1 is the input for interrupt task 1. (default)
		0010	ePWM1 is the input for interrupt task 1. (EPWM1_INT)
		xxx1	No interrupt source

(1) All values not shown are reserved.
(2) This register is protected by EALLOW and the code security module.

10.7.3.4 Interrupt Flag Register (MIFR)

Each bit in the interrupt flag register corresponds to a CLA task. The corresponding bit is automatically set when the task request is received from the peripheral interrupt. The bit can also be set by the main CPU writing to the MIFRC register or using the IACK instruction to start the task. To use the IACK instruction to begin a task first enable this feature in the MCTL register. If the bit is already set when a new peripheral interrupt is received, then the corresponding overflow bit will be set in the MIOVF register.

The corresponding MIFR bit is automatically cleared when the task begins execution. This will occur if the interrupt is enabled in the MIER register and no other higher priority task is pending. The bits can also be cleared manually by writing to the MICLR register. Writes to the MIFR register are ignored.

Figure 10-6. Interrupt Flag Register (MIFR)

15							8
Reserved							
R -0							

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 10-25. Interrupt Flag Register (MIFR) Field Descriptions

Bits	Name	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Flag
		0	A task 8 interrupt is currently not flagged. (default)
		1	A task 8 interrupt has been received and is pending execution.
6	INT7		Task 7 Interrupt Flag
		0	A task 7 interrupt is currently not flagged. (default)
		1	A task 7 interrupt has been received and is pending execution.
5	INT6		Task 6 Interrupt Flag
		0	A task 6 interrupt is currently not flagged. (default)
		1	A task 6 interrupt has been received and is pending execution.
4	INT5		Task 5 Interrupt Flag
		0	A task 5 interrupt is currently not flagged. (default)
		1	A task 5 interrupt has been received and is pending execution.
3	INT4		Task 4 Interrupt Flag
		0	A task 4 interrupt is currently not flagged. (default)
		1	A task 4 interrupt has been received and is pending execution.
2	INT3		Task 3 Interrupt Flag
		0	A task 3 interrupt is currently not flagged. (default)
		1	A task 3 interrupt has been received and is pending execution.
1	INT2		Task 2 Interrupt Flag
		0	A task 2 interrupt is currently not flagged. (default)
		1	A task 2 interrupt has been received and is pending execution.

Table 10-25. Interrupt Flag Register (MIFR) Field Descriptions (continued)

Bits	Name	Value	Description (1)
0	INT1		Task 1 Interrupt Flag
		0	A task 1 interrupt is currently not flagged. (default)
		1	A task 1 interrupt has been received and is pending execution.

(1) This register is protected by the code security module.

10.7.3.5 Interrupt Overflow Flag Register (MIOVF)

Each bit in the overflow flag register corresponds to a CLA task. The bit is set when an interrupt overflow event has occurred for the specific task. An overflow event occurs when the MIFR register bit is already set when a new interrupt is received from a peripheral source. The MIOVF bits are only affected by peripheral interrupt events. They do not respond to a task request by the main CPU IACK instruction or by directly setting MIFR bits. The overflow flag will remain latched and can only be cleared by writing to the overflow flag clear (MICLROVF) register. Writes to the MIOVF register are ignored.

Figure 10-7. Interrupt Overflow Flag Register (MIOVF)

15							8
Reserved							
R -0							
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 10-26. Interrupt Overflow Flag Register (MIOVF) Field Descriptions

Bits	Name	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Overflow Flag
		0	A task 8 interrupt overflow has not occurred. (default)
		1	A task 8 interrupt overflow has occurred.
6	INT7		Task 7 Interrupt Overflow Flag
		0	A task 7 interrupt overflow has not occurred. (default)
		1	A task 7 interrupt overflow has occurred.
5	INT6		Task 6 Interrupt Overflow Flag
		0	A task 6 interrupt overflow has not occurred. (default)
		1	A task 6 interrupt overflow has occurred.
4	INT5		Task 5 Interrupt Overflow Flag
		0	A task 5 interrupt overflow has not occurred. (default)
		1	A task 5 interrupt overflow has occurred.
3	INT4		Task 4 Interrupt Overflow Flag
		0	A task 4 interrupt overflow has not occurred. (default)
		1	A task 4 interrupt overflow has occurred.
2	INT3		Task 3 Interrupt Overflow Flag
		0	A task 3 interrupt overflow has not occurred. (default)
		1	A task 3 interrupt overflow has occurred.

Table 10-26. Interrupt Overflow Flag Register (MIOVF) Field Descriptions (continued)

Bits	Name	Value	Description (1)
1	INT2		Task 2 Interrupt Overflow Flag
		0	A task 2 interrupt overflow has not occurred. (default)
		1	A task 2 interrupt overflow has occurred.
0	INT1		Task 1 Interrupt Overflow Flag
		0	A task 1 interrupt overflow has not occurred. (default)
		1	A task 1 interrupt overflow has occurred.

(1) This register is protected by the code security module.

10.7.3.6 Interrupt Force Register (MIFRC)

The interrupt force register can be used by the main CPU to start tasks through software. Writing a 1 to a MIFRC bit will set the corresponding bit in the MIFR register. Writes of 0 are ignored and reads always return 0. The IACK #16bit operation can also be used to start tasks and has the same effect as the MIFRC register. To enable IACK to set MIFR bits you must first set the MCTL[IACKE] bit. Using IACK has the advantage of not having to first set the EALLOW bit. This allows the main CPU to efficiently trigger CLA tasks through software.

Figure 10-8. Interrupt Force Register (MIFRC)

15							8
Reserved							
R -0							
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 10-27. Interrupt Force Register (MIFRC) Field Descriptions

Bits	Name	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 8 interrupt.
6	INT7		Task 7 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 7 interrupt.
5	INT6		Task 6 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 6 interrupt.
4	INT5		Task 5 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 5 interrupt.
3	INT4		Task 4 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 4 interrupt.
2	INT3		Task 3 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 3 interrupt.
1	INT2		Task 2 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 2 interrupt.
0	INT1		Task 1 Interrupt Force
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to force the task 1 interrupt.

(1) This register is protected by EALLOW and the code security module.

10.7.3.7 Interrupt Flag Clear Register (MICLR)

Normally bits in the MIFR register are automatically cleared when a task begins. The interrupt flag clear register can be used to instead manually clear bits in the interrupt flag (MIFR) register. Writing a 1 to a MICLR bit will clear the corresponding bit in the MIFR register. Writes of 0 are ignored and reads always return 0.

Figure 10-9. Interrupt Flag Clear Register (MICLR)

15							8
Reserved							
R -0							

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 10-28. Interrupt Flag Clear Register (MICLR) Field Descriptions

Bits	Name	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 8 interrupt flag.
6	INT7		Task 7 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 7 interrupt flag.
5	INT6		Task 6 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 6 interrupt flag.
4	INT5		Task 5 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 5 interrupt flag.
3	INT4		Task 4 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 4 interrupt flag.
2	INT3		Task 3 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 3 interrupt flag.
1	INT2		Task 2 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 2 interrupt flag.
0	INT1		Task 1 Interrupt Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 1 interrupt flag.

(1) This register is protected by EALLOW and the code security module.

10.7.3.8 Interrupt Overflow Flag Clear Register (MICLROVF)

Overflow flag bits in the MIOVF register are latched until manually cleared using the MICLROVF register. Writing a 1 to a MICLROVF bit will clear the corresponding bit in the MIOVF register. Writes of 0 are ignored and reads always return 0.

Figure 10-10. Interrupt Overflow Flag Clear Register (MICLROVF)

15							8
Reserved							
R -0							

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 10-29. Interrupt Overflow Flag Clear Register (MICLROVF) Field Descriptions

Bits	Name	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 8 interrupt overflow flag.
6	INT7		Task 7 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 7 interrupt overflow flag.
5	INT6		Task 6 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 6 interrupt overflow flag.
4	INT5		Task 5 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 5 interrupt overflow flag.
3	INT4		Task 4 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 4 interrupt overflow flag.
2	INT3		Task 3 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 3 interrupt overflow flag.
1	INT2		Task 2 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 2 interrupt overflow flag.
0	INT1		Task 1 Interrupt Overflow Flag Clear
		0	This bit always reads back 0 and writes of 0 have no effect.
		1	Write a 1 to clear the task 1 interrupt overflow flag.

(1) This register is protected by EALLOW and the code security module.

10.7.3.9 Interrupt Enable Register (MIER)

Setting the bits in the interrupt enable register (MIER) allow an incoming interrupt or main CPU software to start the corresponding CLA task. Writing a 0 will block the task, but the interrupt request will still be latched in the flag register (MIFLG). Setting the MIER register bit to 0 while the corresponding task is executing will have no effect on the task. The task will continue to run until it hits the MSTOP instruction.

When a soft reset is issued, the MIER bits are cleared. There should always be at least a 1 SYSCLKOUT delay between issuing the soft reset and reconfiguring the MIER bits.

Figure 10-11. Interrupt Enable Register (MIER)

15	14	13	12	11	10	9	8
Reserved							
R -0							

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 10-30. Interrupt Enable Register (MIER) Field Descriptions

Bits	Name	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Interrupt Enable
		0	Task 8 interrupt is disabled. (default)
		1	Task 8 interrupt is enabled.
6	INT7		Task 7 Interrupt Enable
		0	Task 7 interrupt is disabled. (default)
		1	Task 7 interrupt is enabled.
5	INT6		Task 6 Interrupt Enable
		0	Task 6 interrupt is disabled. (default)
		1	Task 6 interrupt is enabled.
4	INT5		Task 5 Interrupt Enable
		0	Task 5 interrupt is disabled. (default)
		1	Task 5 interrupt is enabled.
3	INT4		Task 4 Interrupt Enable
		0	Task 4 interrupt is disabled. (default)
		1	Task 4 interrupt is enabled.
2	INT3		Task 3 Interrupt Enable
		0	Task 3 interrupt is disabled. (default)
		1	Task 3 interrupt is enabled.
1	INT2		Task 2 Interrupt Enable
		0	Task 2 interrupt is disabled. (default)
		1	Task 2 interrupt is enabled.
0	INT1		Task 1 Interrupt Enable
		0	Task 1 interrupt is disabled. (default)
		1	Task 1 interrupt is enabled.

(1) This register is protected by EALLOW and the code security module.

10.7.3.10 Interrupt Run Status Register (MIRUN)

The interrupt run status register (MIRUN) indicates which task is currently executing. Only one MIRUN bit will ever be set to a 1 at any given time. The bit is automatically cleared when the task competes and the respective interrupt is fed to the peripheral interrupt expansion (PIE) block of the device. This lets the main CPU know when a task has completed. The main CPU can stop a currently running task by writing to the MCTL[SOFTRESET] bit. This will clear the MIRUN flag and stop the task. In this case no interrupt will be generated to the PIE.

Figure 10-12. Interrupt Run Status Register (MIRUN)

15							8
Reserved							
R -0							

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

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

Table 10-31. Interrupt Run Status Register (MIRUN) Field Descriptions

Bits	Name	Value	Description (1)
15-8	Reserved		Any writes to these bit(s) must always have a value of 0.
7	INT8		Task 8 Run Status
		0	Task 8 is not executing. (default)
		1	Task 8 is executing.
6	INT7		Task 7 Run Status
		0	Task 7 is not executing. (default)
		1	Task 7 is executing.
5	INT6		Task 6 Run Status
		0	Task 6 is not executing. (default)
		1	Task 6 is executing.
4	INT5		Task 5 Run Status
		0	Task 5 is not executing. (default)
		1	Task 5 is executing.
3	INT4		Task 4 Run Status
		0	Task 4 is not executing. (default)
		1	Task 4 is executing.
2	INT3		Task 3 Run Status
		0	Task 3 is not executing. (default)
		1	Task 3 is executing.
1	INT2		Task 2 Run Status
		0	Task 2 is not executing. (default)
		1	Task 2 is executing.
0	INT1		Task 1 Run Status
		0	Task 1 is not executing. (default)
		1	Task 1 is executing.

(1) This register is protected by the code security module.

10.7.4 Execution Registers

The CLA program counter is initialized by the appropriate MVECTx register when an interrupt is received and a task begins execution. The MPC points to the instruction in the decode 2 (D2) stage of the CLA pipeline. After a MSTOP operation, if no other tasks are pending, the MPC will remain pointing to the MSTOP instruction. The MPC register can be read by the main C28x CPU for debug purposes. The main CPU cannot write to MPC.

10.7.4.1 CLA Program Counter Register (MPC)

The MPC register is described in Figure 10-13 and described in Table 10-32.

Figure 10-13. Program Counter (MPC)

15 12	11 0
Reserved	MPC
R-0	R-0

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

Table 10-32. Program Counter (MPC) Field Descriptions

Bits	Name	Value	Description [1]
15-12	Reserved		Any writes to these bit(s) must always have a value of 0.
11-0	MPC	0000 - 0FFF	Points to the instruction currently in the decode 2 phase of the CLA pipeline. The value is the offset from the first address in the CLA program space.

(1) This register is protected by the code security module. The main CPU can read this register for debug purposes but it can not write to it.

10.7.4.2 CLA Floating-Point Status Register (MSTF)

The CLA status register (MSTF) reflects the results of different operations. These are the basic rules for the flags:

- Zero and negative flags are cleared or set based on:
  - floating-point moves to registers
  - the result of compare, minimum, maximum, negative, and absolute value operations
  - the integer result of operations such as MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32
- Overflow and underflow flags are set by floating-point math instructions such as multiply, add, subtract and 1/x. These flags may also be connected to the peripheral interrupt expansion (PIE) block on your device. This can be useful for debugging underflow and overflow conditions within an application.

The MSTF register is shown in Figure 10-14 and described in Table 10-33.

Figure 10-14. CLA Status Register (MSTF)

31 24	23 16
Reserved	RPC
R/W-0	R/W-0

15 12	11	10	9	8 7	6	5 4	3	2	1	0
RPC	MEALLOW	Rsvd	RND32	Reserved	TF	Reserved	ZF	NF	LUF	LVF
R/W-0	R/W-0	R-0	R/W-0	R-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 10-33. CLA Status (MSTF) Register Field Descriptions

Bits	Field	Value	Description [1]
31-24	Reserved	0	Reserved for future use.
23-12	RPC		Return program counter. The RPC is used to save and restore the MPC address by the MCCNDD and MRCNDD operations.
11	MEALLOW		This bit enables and disables CLA write access to EALLOW protected registers. This is independent of the state of the EALLOW bit in the main CPU status register. This status bit can be saved and restored by the MMOV32 STF, mem32 instruction.
		0	The CLA cannot write to EALLOW protected registers. This bit is cleared by the CLA instruction, MEDIS.
		1	The CLA is allowed to write to EALLOW protected registers. This bit is set by the CLA instruction, MEALLOW.
10	Reserved	0	Any writes to these bit(s) must always have a value of 0.
9	RND32		Round 32-bit Floating-Point Mode Use the MSETFLG and MMOV32 MSTF, mem32 instructions to change the rounding mode.
		0	If this bit is zero, the MMPYF32, MADDF32 and MSUBF32 instructions will round to zero (truncate).
		1	If this bit is one, the MMPYF32, MADDF32 and MSUBF32 instructions will round to the nearest even value.
8-7	Reserved	0	Reserved for future use.
6	TF		Test Flag The MTESTTF instruction can modify this flag based on the condition tested. The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	The condition tested with the MTESTTF instruction is false.
		1	The condition tested with the MTESTTF instruction is true.
5-4	Reserved		These two bits may change based on integer results. These flags are not, however, used by the CLA and therefore marked as reserved.
3	ZF		Zero Flag [2] [3] • Instructions that modify this flag based on the floating-point value stored in the destination register: MMOV32, MMOVD32, MABSF32, MNEGF32 • Instructions that modify this flag based on the floating-point result of the operation: MCMPF32, MMAXF32, and MMINF32 • Instructions that modify this flag based on the integer result of the operation: MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32 and MLSL32 The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag
		0	The value is not zero.
		1	The value is zero.
2	NF		Negative Flag [2] [3] • Instructions that modify this flag based on the floating-point value stored in the destination register: MMOV32, MMOVD32, MABSF32, MNEGF32 • Instructions that modify this flag based on the floating-point result of the operation: MCMPF32, MMAXF32, and MMINF32 • Instructions that modify this flag based on the integer result of the operation: MMOV16, MAND32, MOR32, MXOR32, MCMP32, MASR32, MLSR32 and MLSL32 The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	The value is not negative.
		1	The value is negative.

Table 10-33. CLA Status (MSTF) Register Field Descriptions (continued)

Bits	Field	Value	Description (1)
1	LUF		Latched Underflow Flag
			The following instructions will set this flag to 1 if an underflow occurs: MMPYF32, MADDF32, MSUBF32, MMACF32, MEINVF32, MEISQRTF32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	An underflow condition has not been latched.
		1	An underflow condition has been latched.
0	LVF		Latched Overflow Flag
			The following instructions will set this flag to 1 if an overflow occurs: MMPYF32, MADDF32, MSUBF32, MMACF32, MEINVF32, MEISQRTF32
			The MSETFLG and MMOV32 MSTF, mem32 instructions can also be used to modify this flag.
		0	An overflow condition has not been latched.
		1	An overflow condition has been latched.

(1) This register is protected by the code security module. The main CPU can read this register for debug purposes but it can not write to it.
(2) A negative zero floating-point value is treated as a positive zero value when configuring the ZF and NF flags.
(3) A DeNorm floating-point value is treated as a positive zero value when configuring the ZF and NF flags.

This page intentionally left blank.

Chapter 11

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 using devices such as shift registers, display drivers, and analog-to-digital converters (ADCs). Multi-device communications are supported by the master or slave operation of the SPI. The port supports a 4-level, receive and transmit FIFO for reducing CPU servicing overhead.

11.1 Introduction

11.1.1 Features

The SPI module features include:

- SPISOMI: SPI slave-output/master-input pin
- SPISIMO: SPI slave-input/master-output pin
- $\overline{\text{SPISTE}}$: SPI slave transmit-enable pin
- SPICLK: SPI serial-clock pin

Note

All four pins can be used as GPIO if the SPI module is not used.

- Two operational modes: Master and Slave
- Baud rate: 125 different programmable rates. The maximum baud rate that can be employed is limited by the maximum speed of the I/O buffers used on the SPI pins. See the device-specific data sheet for more details.
- Data word length: 1 to 16 data bits
- Four clocking schemes (controlled by clock polarity and clock phase bits) include:
  - Falling edge without phase delay: SPICLK active-high. SPI transmits data on the falling edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.
  - Falling edge with phase delay: SPICLK active-high. SPI transmits data one half-cycle ahead of the falling edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
  - Rising edge without phase delay: SPICLK inactive-low. SPI transmits data on the rising edge of the SPICLK signal and receives data on the falling edge of the SPICLK signal.
  - Rising edge with phase delay: SPICLK inactive-low. SPI transmits data one half-cycle ahead of the rising edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.
- Simultaneous receive and transmit operation (transmit function can be disabled in software)
- Transmitter and receiver operations are accomplished through either interrupt- driven or polled algorithm
- 4-level transmit/receive FIFO
- Delayed transmit control
- 3-wire SPI mode
- $\overline{\text{SPISTE}}$ inversion for digital audio interface receive mode on devices with two SPI modules

11.1.2 Block Diagram

Figure 11-1 shows the SPI CPU interfaces.

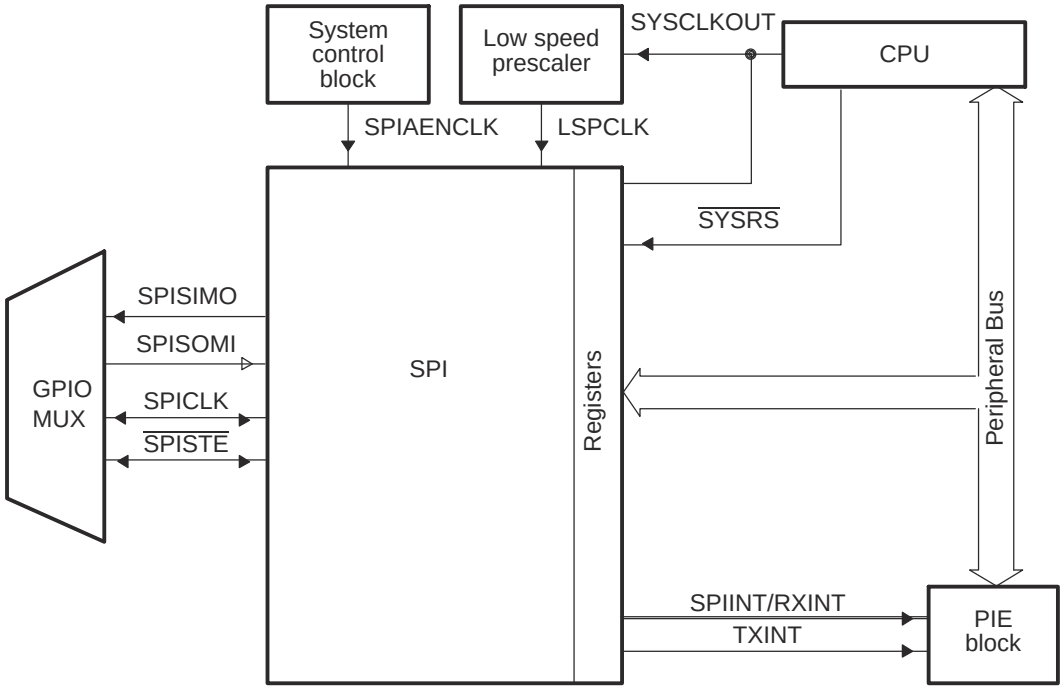


Figure 11-1. SPI CPU Interface

11.2 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.

11.2.1 SPI Module Signals

Table 11-1 classifies and provides a summary of the SPI module signals.

Table 11-1. 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. A HIGH logic signal on $\overline{\text{SPISTE}}$ will not allow the slave to receive data. This prevents the SPI slave from losing synchronization with the master. It is this reason that TI does not recommend that the $\overline{\text{SPISTE}}$ always be tied to the active state.

If the SPI slave does ever lose synchronization with the master, toggling SPISWRESET resets the internal bit counter as well as the various status flags in the module. By resetting the bit counter, the SPI interprets the next clock transition as the first bit of a new transmission. The register bit fields that are reset by SPISWRESET are found in Section 11.5.

Configuring a GPIO to Emulate $\overline{\text{SPISTE}}$

In many systems, a SPI master may be connected to multiple SPI slaves using multiple instances of $\overline{\text{SPISTE}}$. Though this SPI module does not natively support multiple $\overline{\text{SPISTE}}$ signals, it is possible to emulate this behavior in software using GPIOs. In this configuration, the SPI must be configured as the master. Rather than using the GPIO Mux to select $\overline{\text{SPISTE}}$, the application would configure pins to be GPIO outputs, one GPIO per SPI slave. Before transmitting any data, the application would drive the desired GPIO to the active state. Immediately after the transmission has been completed, the GPIO chip select would be driven to the inactive state. This process can be repeated for many slaves that share the SPICLK, SPISIMO, and SPISOMI lines.

11.2.2 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.

11.2.3 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 11-2 and Table 11-2 show how these control bits influence the SPI interrupt generation.

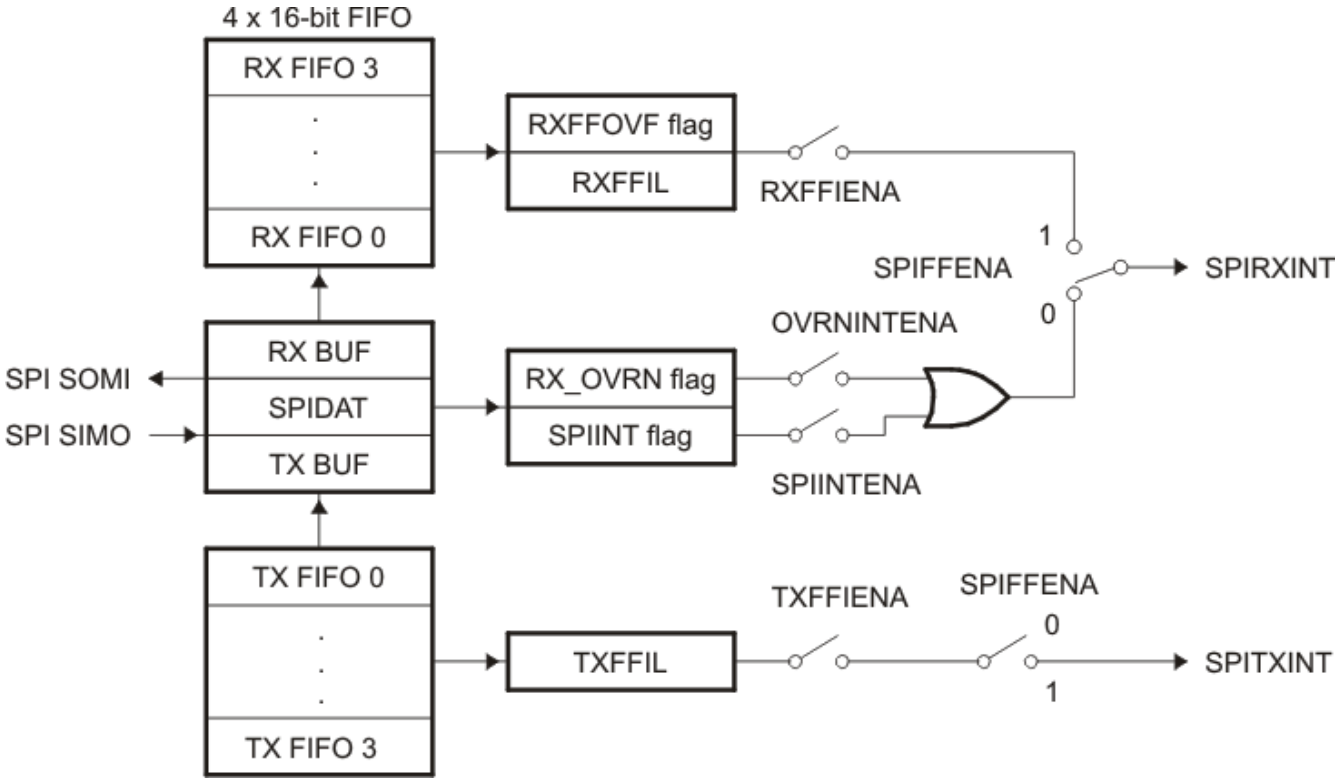


Figure 11-2. SPI Interrupt Flags and Enable Logic Generation

Table 11-2. 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 C28x devices.

11.3 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.

11.3.1 Introduction to Operation

Figure 11-3 shows typical connections of the SPI for communications between two controllers: a master and a slave.

The master transfers data by sending the SPICLK signal. For both the slave and the master, data is shifted out of the shift registers on one edge of the SPICLK and latched into the shift register on the opposite SPICLK clock edge. If the CLK_PHASE bit is high, data is transmitted and received a half-cycle before the SPICLK transition. As a result, both controllers send and receive data simultaneously. The application software determines whether the data is meaningful or dummy data. There are three possible methods for data transmission:

- Master sends data; slave sends dummy data.
- Master sends data; slave sends data.
- Master sends dummy data; slave sends data.

The master can initiate data transfer at any time because it controls the SPICLK signal. The software, however, determines how the master detects when the slave is ready to broadcast data.

The SPI can operate in master or slave mode. The MASTER_SLAVE bit selects the operating mode and the source of the SPICLK signal.

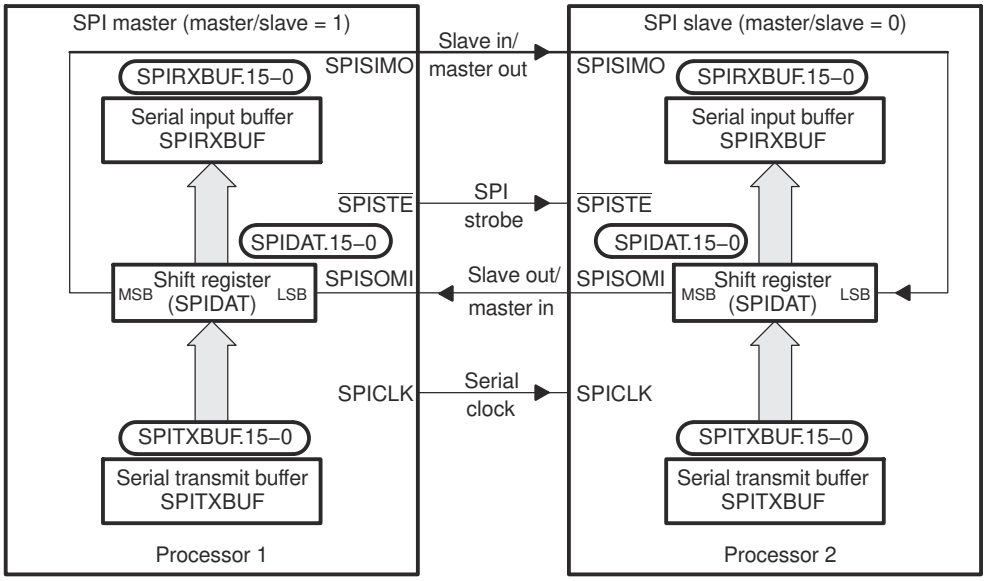


Figure 11-3. SPI Master/Slave Connection

Figure 11-4 is a block diagram of the SPI module showing all of the basic control blocks available on the SPI module.

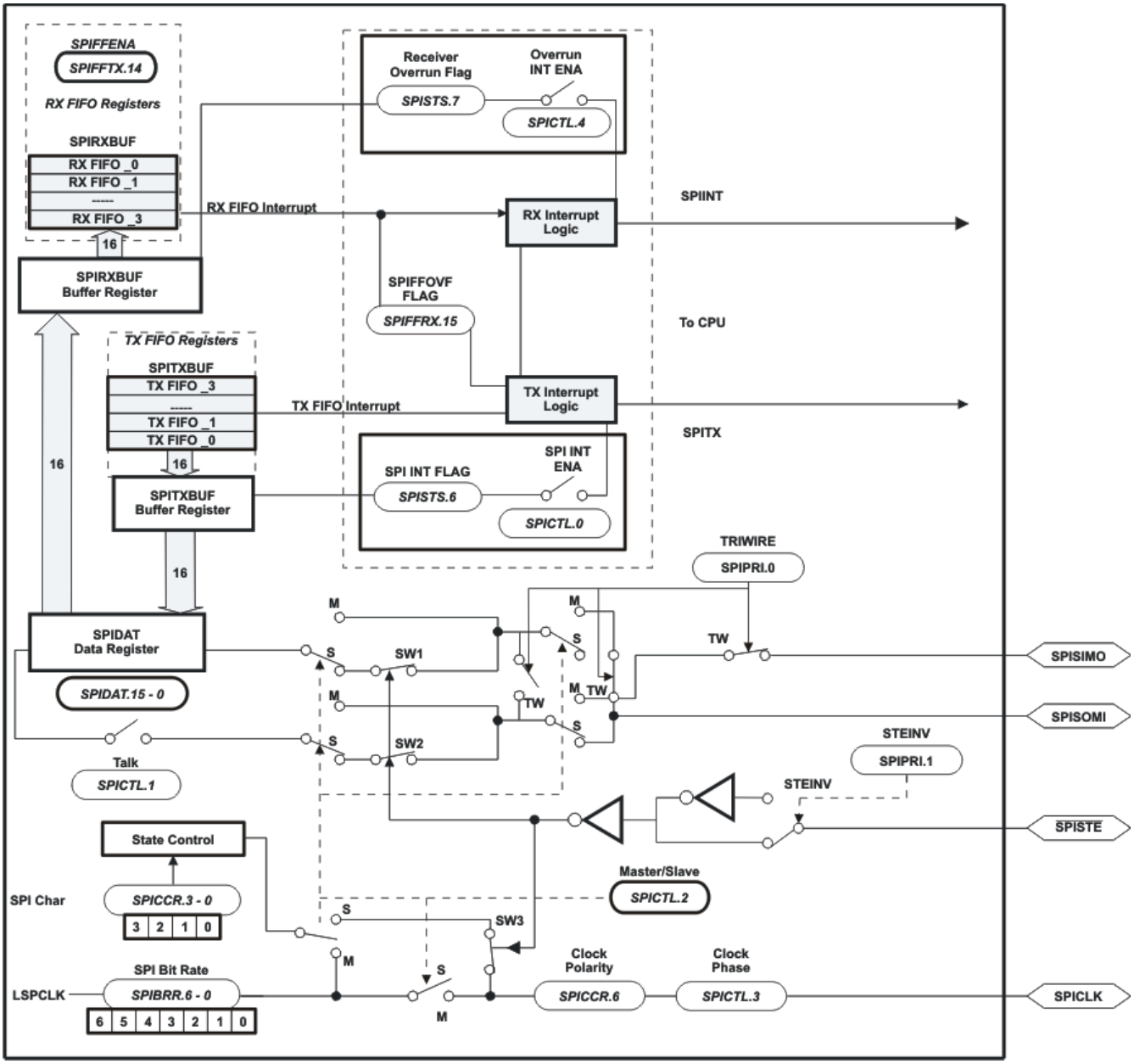


Figure 11-4. Serial Peripheral Interface Block Diagram

11.3.2 Master Mode

In master mode (MASTER_SLAVE = 1), the SPI provides the serial clock on the SPICLK pin for the entire serial communications network. Data is output on the SPISIMO pin and latched from the SPISOMI pin.

The SPIBRR register determines both the transmit and receive bit transfer rate for the network. SPIBRR can select 125 different data transfer rates.

Data written to SPIDAT or SPITXBUF initiates data transmission on the SPISIMO pin, MSB (most-significant bit) first. Simultaneously, received data is shifted through the SPISOMI pin into the LSB (least- significant bit) of SPIDAT. When the selected number of bits has been transmitted, the received data is transferred to the SPIRXBUF (buffered receiver) for the CPU to read. Data is stored right-justified in SPIRXBUF.

When the specified number of data bits has been shifted through SPIDAT, the following events occur:

- SPIDAT contents are transferred to SPIRXBUF.
- INT_FLAG bit is set to 1.
- If there is valid data in the transmit buffer SPITXBUF, as indicated by the transmit buffer full flag (BUFFULL_FLAG), this data is transferred to SPIDAT and is transmitted; otherwise, SPICLK stops after all bits have been shifted out of SPIDAT.
- If the SPIINTENA bit is set to 1, an interrupt is asserted.

In a typical application, the $\overline{\text{SPISTE}}$ pin serves as a chip-enable pin for a SPI slave device. This pin is driven low by the master before transmitting data to the slave and is taken high after the transmission is complete.

11.3.3 Slave Mode

In slave mode (MASTER_SLAVE = 0), data shifts out on the SPISOMI pin and in on the SPISIMO pin. The SPICLK pin is used as the input for the serial shift clock, which is supplied from the external network master. The transfer rate is defined by this clock. The SPICLK input frequency should be no greater than the LSPCLK frequency divided by 4.

Data written to SPIDAT or SPITXBUF is transmitted to the network when appropriate edges of the SPICLK signal are received from the network master. A character written to the SPITXBUF register will be copied to the SPIDAT register when all bits of the current character in SPIDAT have been shifted out. If no character was previously copied to SPIDAT, then any character written to SPITXBUF will be immediately copied to SPIDAT. If a character was previously copied to SPIDAT, any data written to SPITXBUF will not be copied to SPIDAT until the current character in SPIDAT has been shifted out. To receive data, the SPI waits for the network master to send the SPICLK signal and then shifts the data on the SPISIMO pin into SPIDAT. If data is to be transmitted by the slave simultaneously, and SPIDAT has not been previously loaded, the character must be written to SPITXBUF before the beginning of the SPICLK signal.

When the TALK bit is cleared, data transmission is disabled, and the output line (SPISOMI) is put into the high-impedance state. If this occurs while a transmission is active, the current character is completely transmitted even though SPISOMI is forced into the high-impedance state. This ensures that the SPI is still able to receive incoming data correctly. This TALK bit allows many slave devices to be tied together on the network, but only one slave at a time is allowed to drive the SPISOMI line.

The $\overline{\text{SPISTE}}$ pin operates as the slave-select pin. An active-low signal on the $\overline{\text{SPISTE}}$ pin allows the slave SPI to transfer data to the serial data line; an inactive- high signal causes the slave SPI serial shift register to stop and its serial output pin to be put into the high-impedance state. This allows many slave devices to be tied together on the network, although only one slave device is selected at a time.

11.3.4 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 11-1).

Example 11-1. Transmission of Bit from SPIRXBUF

Conditions:

1. Transmission character length = 1 bit (specified in SPICHAR bits)
2. The current value of SPIDAT = 737Bh

SPIDAT (before transmission)																
	0	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1
SPIDAT (after transmission)																
(TXed) 0 ←	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	x(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.

11.3.5 Baud Rate Selection

The SPI module supports 125 different baud rates and four different clock schemes. Depending on whether the SPI clock is in slave or master mode, the SPICLK pin can receive an external SPI clock signal or provide the SPI clock signal, respectively.

- In the slave mode, the SPI clock is received on the SPICLK pin from the external source and can be no greater than the LSPCLK frequency divided by 4.
- In the master mode, the SPI clock is generated by the SPI and is output on the SPICLK pin and can be no greater than the LSPCLK frequency divided by 4.

Note

The baud rate should be configured to not exceed the maximum rated GPIO toggle frequency. Refer to the device data sheet for the maximum GPIO toggle frequency

Example 11-2 shows how to determine the SPI baud rates.

Example 11-2. Baud Rate Determination

For SPIBRR = 3 to 127:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{(\text{SPIBRR} + 1)} \quad (5)$$

For SPIBRR = 0, 1, or 2:

$$\text{SPI Baud Rate} = \frac{\text{LSPCLK}}{4} \quad (6)$$

where:

LSPCLK = Low-speed peripheral clock frequency of the device

SPIBRR = Contents of the SPIBRR in the master SPI device

To determine what value to load into SPIBRR, you must know the device system clock (LSPCLK) frequency (that is device-specific) and the baud rate at which you will be operating.

Example 11-3 shows how to calculate the baud rate of the SPI module .

Example 11-3. Baud Rate Calculation

$$\text{Maximum SPI Baud Rate} = \frac{\text{LSPCLK}}{4} = \frac{40 \times 10^6}{4} = 10 \times 10^6 \text{ bps} \quad (7)$$

11.3.6 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:

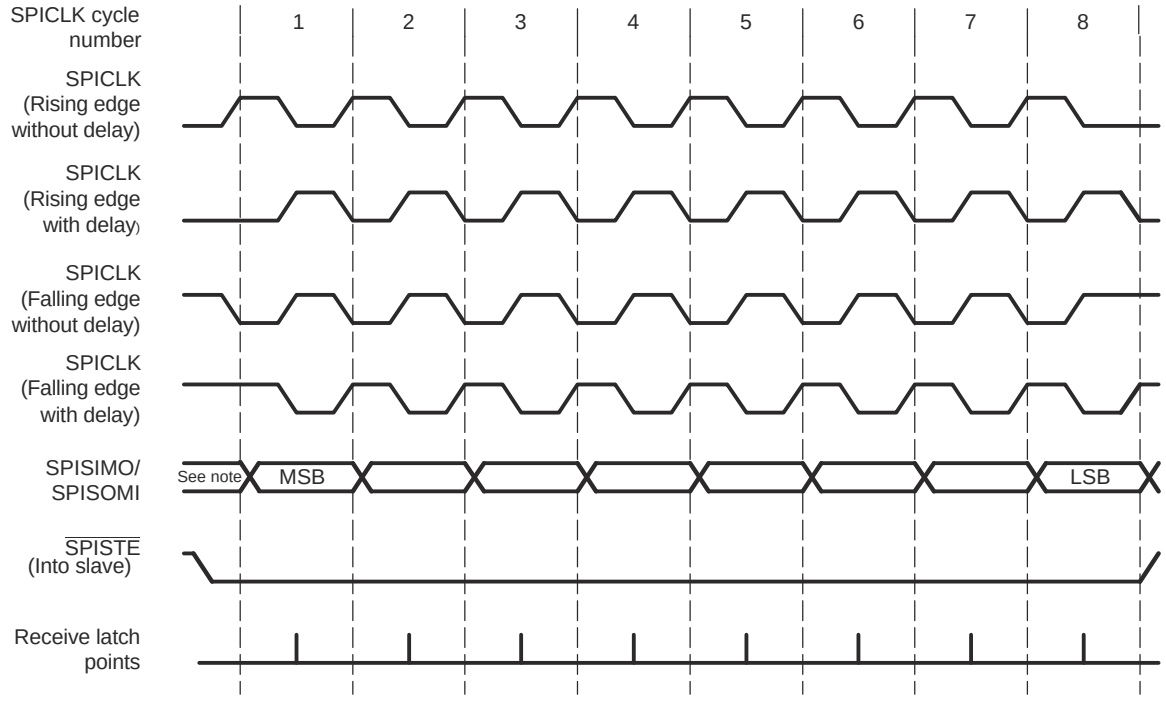
- Falling Edge Without Delay. The SPI transmits data on the falling edge of the SPICLK and receives data on the rising edge of the SPICLK.
- Falling Edge With Delay. The SPI transmits data one half-cycle ahead of the falling edge of the SPICLK signal and receives data on the falling edge of the SPICLK 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 11-3. Examples of these four clocking schemes relative to transmitted and received data are shown in Figure 11-5.

Table 11-3. 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.



Note: Previous data bit

Figure 11-5. SPICLK Signal Options

SPICLK symmetry is retained only when the result of (SPIBRR+1) is an even value. When (SPIBRR + 1) is an odd value and SPIBRR is greater than 3, SPICLK becomes asymmetrical. The low pulse of SPICLK is one LSPCLK cycle longer than the high pulse when CLKPOLARITY bit is clear (0). When CLKPOLARITY bit is set to 1, the high pulse of the SPICLK is one LSPCLK cycle longer than the low pulse, as shown in Figure 11-6.

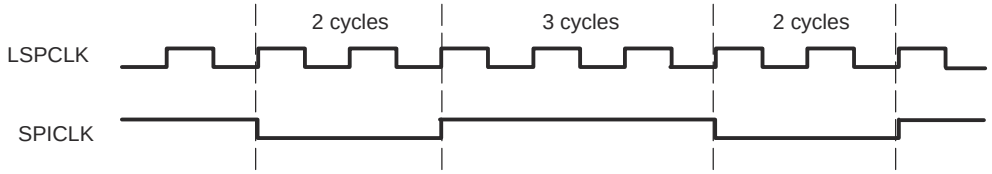


Figure 11-6. SPI: SPICLK-LSPCLK Characteristic When (BRR + 1) is Odd, BRR > 3, and CLKPOLARITY = 1

11.3.7 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 11.2.3.
6. **Buffers.** Transmit and receive buffers are each supplemented with a 4 word FIFO. The one-word transmit buffer (SPITXBUF) of the standard SPI functions as a transition buffer between the transmit FIFO and shift register. The one-word transmit buffer will be loaded from transmit FIFO only after the last bit of the shift register is shifted out.
7. **Delayed transfer.** The rate at which transmit words in the FIFO are transferred to transmit shift register is programmable. The SPIFFCT register bits (7−0) 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.

11.3.8 SPI 3-Wire Mode Description

SPI 3-wire mode allows for SPI communication over three pins instead of the normal four pins.

In master mode, if the TRIWIRE bit is set, enabling 3-wire SPI mode, SPISIMOx becomes the bi-directional SPIMOMIx (SPI master out, master in) pin, and SPISOMIx is no longer used by the SPI. In slave mode, if the TRIWIRE bit is set, SPISOMIx becomes the bi-directional SPISISOx (SPI slave in, slave out) pin, and SPISIMOx is no longer used by the SPI.

Table 11-4 indicates the pin function differences between 3-wire and 4-wire SPI mode for a master and slave SPI.

Table 11-4. 4-wire vs. 3-wire SPI Pin Functions

4-wire SPI	3-wire SPI (Master)	3-wire SPI (Slave)
SPICLKx	SPICLKx	SPICLKx
SPISTEx	SPISTEx	SPISTEx
SPISIMOx	SPIMOMIx	Free
SPISOMIx	Free	SPISISOx

Because in 3-wire mode, the receive and transmit paths within the SPI are connected, any data transmitted by the SPI module is also received by itself. The application software must take care to perform a dummy read to clear the SPI data register of the additional received data.

The TALK bit plays an important role in 3-wire SPI mode. The bit must be set to transmit data and cleared prior to reading data. In master mode, in order to initiate a read, the application software must write dummy data to the SPI data register (SPIDAT or SPIRXBUF) while the TALK bit is cleared (no data is transmitted out the SPIMOMI pin) before reading from the data register.

Figure 11-7 and Figure 11-8 illustrate 3-wire master and slave mode.

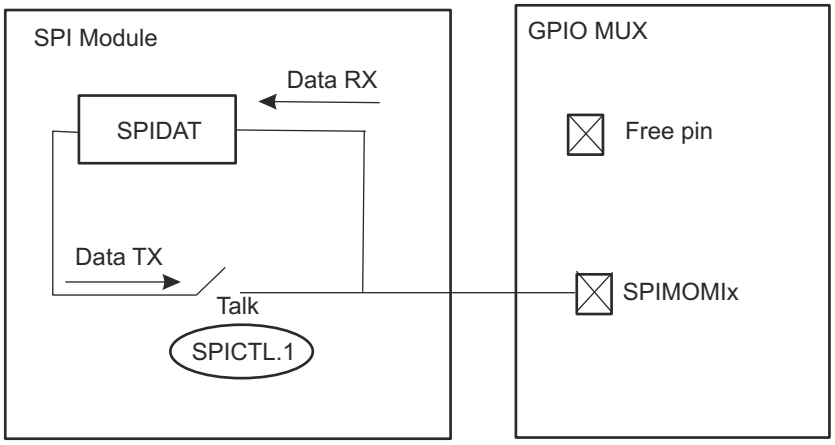


Figure 11-7. SPI 3-wire Master Mode

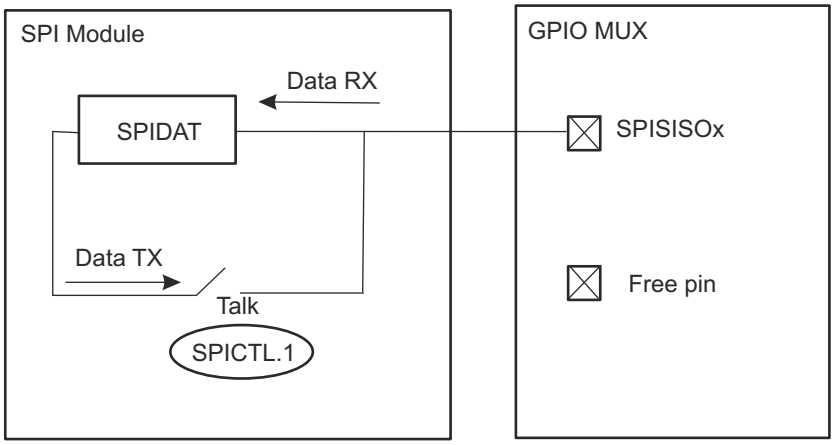


Figure 11-8. SPI 3-wire Slave Mode

Table 11-5 indicates how data is received or transmitted in the various SPI modes while the TALK bit is set or cleared.

Table 11-5. 3-Wire SPI Pin Configuration

Pin Mode	SPIPRI[TRIWIRE]	SPICTL[TALK]	SPISIMO	SPISOMI
		Master Mode		
4-wire	0	X	TX	RX
3-pin mode	1	0	RX	Disconnect from SPI
		1	TX/RX	
		Slave Mode		
4-wire	0	X	RX	TX
3-pin mode	1	0	Disconnect from SPI	RX
		1		TX/RX

11.4 Programming Procedure

This section describes the procedure for configuring the SPI for the various modes of operation.

11.4.1 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

11.4.2 Configuring the SPI

This section describes the procedure in which to configure the SPI module for operation. To prevent unwanted and unforeseen events from occurring during or as a result of initialization changes, clear the SPISWRESET bit before making initialization changes, and then set this bit after initialization is complete. While the SPI is held in reset (SPISWRESET = 0), configuration may be changed in any order. The following list shows the SPI configuration procedure in a logical order. However, the SPI registers can be written with single 16-bit writes, so the order is not required with the exception of SPISWRESET.

To change the SPI configuration:

1. Clear the SPI Software Reset bit (SPISWRESET) to 0 to force the SPI to the reset state.
2. Configure the SPI as desired:
   - Select either master or slave mode (MASTER_SLAVE).
   - Choose SPICLK polarity and phase (CLKPOLARITY and CLK_PHASE).
   - Set the desired baud rate (SPIBRR).
   - Set the SPI character length (SPICHAR).
   - Clear the SPI Flags (OVERRUN_FLAG, INT_FLAG).
   - Enable $\overline{\text{SPISTE}}$ inversion (STEINV), if needed.
   - Enable 3-wire mode (TRIWIRE), if needed.
   - If using FIFO enhancements:
     - Enable the FIFO enhancements (SPIFFENA).
     - Clear the FIFO Flags (TXFFINTCLR, RXFFOVFCLR, and RXFFINTCLR).
     - Release transmit and receive FIFO resets (TXFIFO and RXFIFORESET).
     - Release SPI FIFO channels from reset (SPIRST).
3. If interrupts are used:
   - In non-FIFO mode, enable the receiver overrun and/or SPI interrupts (OVERRUNINTENA and SPIINTENA).
   - In FIFO mode, set the transmit and receive interrupt levels (TXFFIL and RXFFIL) then enable the interrupts (TXFFIENA and RXFFIENA).
4. Set SPISWRESET to 1 to release the SPI from the reset state.

Note

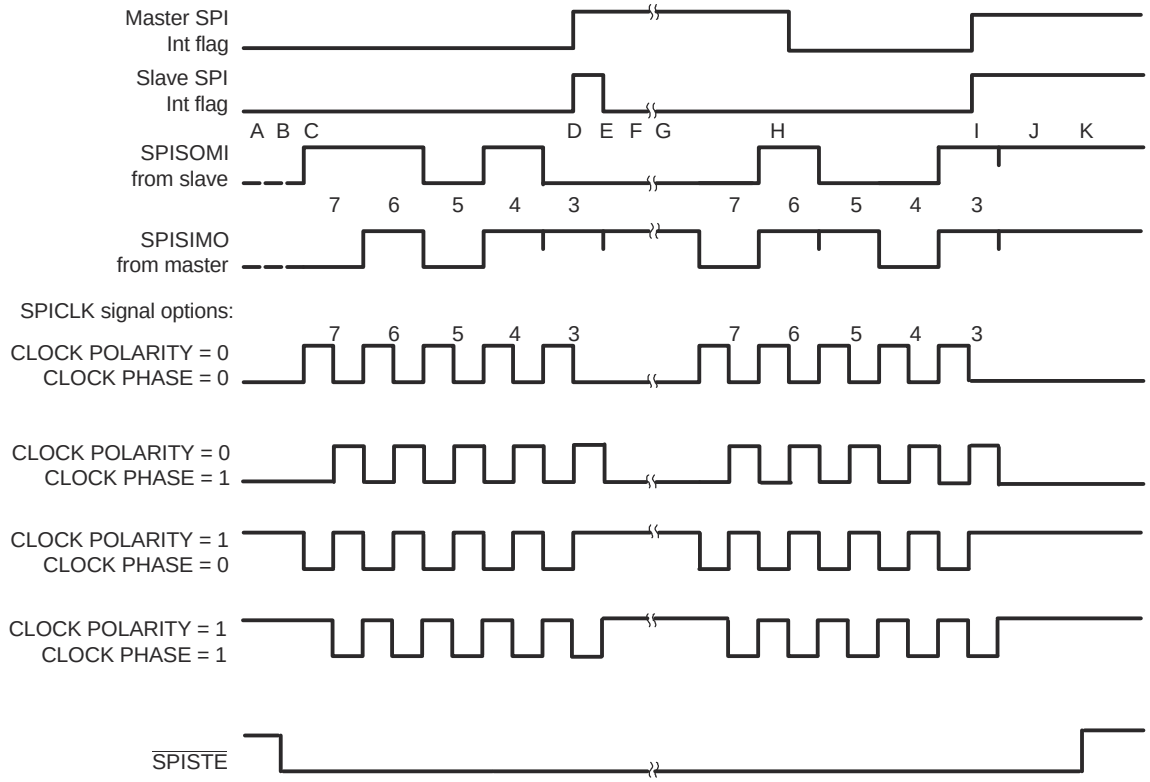
Do not change the SPI configuration when communication is in progress.

11.4.3 Data Transfer Example

The timing diagram shown in Figure 11-9 shows an SPI data transfer between two devices using a character length of five bits with the SPICLK being symmetrical.

The timing diagram with SPICLK asymmetrical (Figure 11-6) shares similar characterizations with Figure 11-9 except that the data transfer is one LSPCLK cycle longer per bit during the low pulse (CLKPOLARITY = 0) or during the high pulse (CLKPOLARITY = 1) of the SPICLK.

Figure 11-9 is applicable for 8-bit SPI only and is not for C28x devices that are capable of working with 16-bit data. The figure is shown for illustrative purposes only.



A. Slave writes 0D0h to SPIDAT and waits for the master to shift out the data.
B. Master sets the slave $\overline{\text{SPISTE}}$ signal low (active).
C. Master writes 058h to SPIDAT, which starts the transmission procedure.
D. First byte is finished and sets the interrupt flags.
E. Slave reads 0Bh from its SPIRXBUF (right-justified).
F. Slave writes 04Ch to SPIDAT and waits for the master to shift out the data.
G. Master writes 06Ch to SPIDAT, which starts the transmission procedure.
H. Master reads 01Ah from the SPIRXBUF (right-justified).
I. Second byte is finished and sets the interrupt flags.
J. Master reads 89h and the slave reads 8Dh from their respective SPIRXBUF. After the user software masks off the unused bits, the master receives 09h and the slave receives 0Dh.
K. Master clears the slave $\overline{\text{SPISTE}}$ signal high (inactive).

Figure 11-9. Five Bits per Character

11.4.4 SPI 3-Wire Mode Code Examples

In addition to the normal SPI initialization, to configure the SPI module for 3-wire mode, the TRIWIRE bit (SPIPRI.0) must be set to 1. After initialization, there are several considerations to take into account when transmitting and receiving data in 3-wire master and slave mode. The following examples demonstrate these considerations.

In 3-wire master mode, SPICLKx, SPISTEx, and SPISIMOx pins must be configured as SPI pins (SPISOMIx pin can be configured as non-SPI pin). When the master transmits, it receives the data it transmits (because SPISIMOx and SPISOMIx are connected internally in 3-wire mode). Therefore, the junk data received must be cleared from the receive buffer every time data is transmitted.

Example 11-4. 3-Wire Master Mode Transmit

```
Uint16 data;
Uint16 dummy;
     SpiaRegs.SPICTL.bit.TALK = 1;              // Enable Transmit path
     SpiaRegs.SPITXBUF = data; // Master transmits data
     while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Waits until data rx'd
     dummy = SpiaRegs.SPIRXBUF;                 // Clears junk data from itself
                                                // bc it rx'd same data tx'd
```

To receive data in 3-wire master mode, the master must clear the TALK (SPICTL.1) bit to 0 to close the transmit path and then transmit dummy data in order to initiate the transfer from the slave. Because the TALK bit is 0, unlike in transmit mode, the master dummy data does not appear on the SPISIMOx pin, and the master does not receive its own dummy data. Instead, the data from the slave is received by the master.

Example 11-5. 3-Wire Master Mode Receive

```
Uint16 rdata;
Uint16 dummy;
     SpiaRegs.SPICTL.bit.TALK = 0;              // Disable Transmit path
     SpiaRegs.SPITXBUF = dummy;                 // Send dummy to start tx
     // NOTE: because TALK = 0, data does not tx onto SPISIMOA pin
     while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Wait until data received
     rdata = SpiaRegs.SPIRXBUF;                 // Master reads data
```

In 3-wire slave mode, SPICLKx, SPISTEx, and SPISOMIx pins must be configured as SPI pins (SPISIMOx pin can be configured as non-SPI pin). Like in master mode, when transmitting, the slave receives the data it transmits and must clear this junk data from its receive buffer.

Example 11-6. 3-Wire Slave Mode Transmit

```
Uint16 data;
Uint16 dummy;
     SpiaRegs.SPICTL.bit.TALK = 1;              // Enable Transmit path
     SpiaRegs.SPITXBUF = data;                  // Slave transmits data
     while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Wait until data rx'd
     dummy = SpiaRegs.SPIRXBUF;                 // Clears junk data from itself
```

As in 3-wire master mode, the TALK bit must be cleared to 0. Otherwise, the slave receives data normally.

Example 11-7. 3-Wire Slave Mode Receive

```
Uint16 rdata;
    SpiaRegs.SPICTL.bit.TALK = 0;              // Disable Transmit path
    while(SpiaRegs.SPISTS.bit.INT_FLAG !=1) {} // Waits until data rx'd
    rdata = SpiaRegs.SPIRXBUF;                 // Slave reads data
```

11.4.5 SPI STEINV Bit in Digital Audio Transfers

On those devices with two SPI modules, enabling the STEINV bit on one of the SPI modules allows the pair of SPIs to receive both left and right-channel digital audio data in slave mode. The SPI module that receives a normal active-low $\overline{\text{SPISTE}}$ signal stores right-channel data, and the SPI module that receives an inverted active-high $\overline{\text{SPISTE}}$ signal stores left-channel data from the master. To receive digital audio data from a digital audio interface receiver, the SPI modules can be connected as shown in Figure 11-10.

Note

This configuration is only applicable to slave mode (MASTER_SLAVE = 0). When the SPI is configured as master (MASTER_SLAVE = 1), the STEINV bit will have no effect on the $\overline{\text{SPISTE}}$ pin.

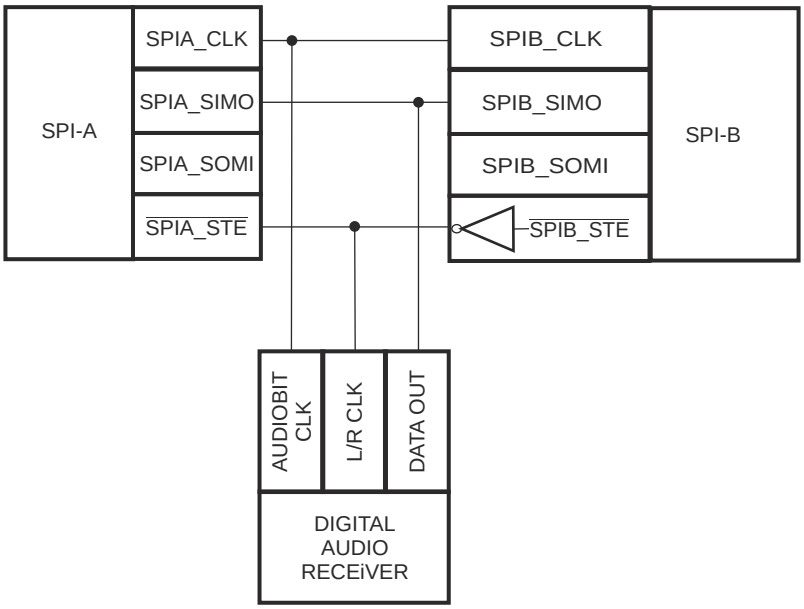


Figure 11-10. SPI Digital Audio Receiver Configuration Using Two SPIs

Standard C28x SPI timing requirements limit the number of digital audio interface formats supported using the 2-SPI configuration with the STEINV bit. See your device-specific data sheet electrical specifications for SPI timing requirements. With the SPI clock phase configured such that the CLKPOLARITY bit is 0 and the CLK_PHASE bit is 1 (data latched on rising edge of clock), standard right-justified digital audio interface data format is supported as shown in Figure 11-11.

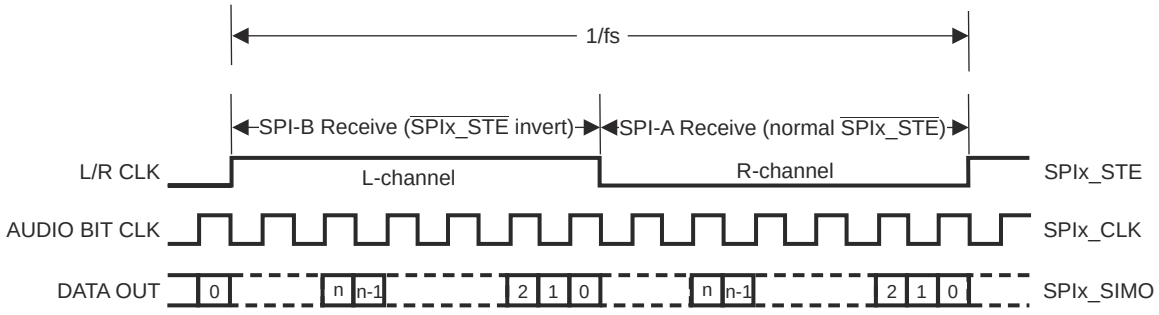


Figure 11-11. Standard Right-Justified Digital Audio Data Format

11.5 SPI Registers

This section describes the Serial Peripheral Interface registers. It is important to note that the SPI registers only allow 16-bit accesses.

11.5.1 SPI Base Addresses

Table 11-6. SPI Base Address Table (C28)

Bit Field Name		Base Address
Instance	Structure	
SpiaRegs	SPI_REGS	0x0000_7040
SpibRegs	SPI_REGS	0x0000_7740

11.5.2 SPI_REGS Registers

Table 11-7 lists the SPI_REGS registers. All register offset addresses not listed in Table 11-7 should be considered as reserved locations and the register contents should not be modified.

Table 11-7. SPI_REGS Registers

Offset	Acronym	Register Name	Section
0h	SPICCR	SPI Configuration Control Register	Section 11.5.2.1
1h	SPICTL	SPI Operation Control Register	Section 11.5.2.2
2h	SPISTS	SPI Status Register	Section 11.5.2.3
4h	SPIBRR	SPI Baud Rate Register	Section 11.5.2.4
6h	SPIRXEMU	SPI Emulation Buffer Register	Section 11.5.2.5
7h	SPIRXBUF	SPI Serial Input Buffer Register	Section 11.5.2.6
8h	SPITXBUF	SPI Serial Output Buffer Register	Section 11.5.2.7
9h	SPIDAT	SPI Serial Data Register	Section 11.5.2.8
Ah	SPIFFTX	SPI FIFO Transmit Register	Section 11.5.2.9
Bh	SPIFFRX	SPI FIFO Receive Register	Section 11.5.2.10
Ch	SPIFFCT	SPI FIFO Control Register	Section 11.5.2.11
Fh	SPIPRI	SPI Priority Control Register	Section 11.5.2.12

Complex bit access types are encoded to fit into small table cells. Table 11-8 shows the codes that are used for access types in this section.

Table 11-8. SPI_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
RC	R C	Read to Clear
Write Type		
W	W	Write
W1C	W 1C	Write 1 to clear
Reset or Default Value		
-n		Value after reset or the default value
Register Array Variables		
i,j,k,l,m,n		When these variables are used in a register name, an offset, or an address, they refer to the value of a register array where the register is part of a group of repeating registers. The register groups form a hierarchical structure and the array is represented with a formula.
y		When this variable is used in a register name, an offset, or an address it refers to the value of a register array.

11.5.2.1 SPI Configuration Control Register (SPICCR) (Offset = 0h) [reset = 0h]

SPICCR controls the setup of the SPI for operation.

Figure 11-12. SPI Configuration Control Register (SPICCR)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
SPISWRESET	CLKPOLARITY	RESERVED	SPILBK	SPICHAR			
R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h			

Table 11-9. SPI Configuration Control Register (SPICCR) 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.

Table 11-9. SPI Configuration Control Register (SPICCR) Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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
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.

Table 11-9. SPI Configuration Control Register (SPICCR) Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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

11.5.2.2 SPI Operation Control (SPICTL) Register (Offset = 1h) [reset = 0h]

SPICTL controls data transmission, the SPI's ability to generate interrupts, the SPICLK phase, and the operational mode (slave or master).

Figure 11-13. SPI Operation Control (SPICTL) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			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 11-10. SPI Operation Control (SPICTL) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-5	RESERVED	R	0h	Reserved
4	OVERRUNINTENA	R/W	0h	Overrun Interrupt Enable Overrun Interrupt Enable. Setting this bit causes an interrupt to be generated when the RECEIVER OVERRUN Flag bit (SPISTS.7) is set by hardware. Interrupts generated by the RECEIVER OVERRUN Flag bit and the SPI INT FLAG bit (SPISTS.6) share the same interrupt vector. Reset type: SYSRSn 0h (R/W) = Disable RECEIVER OVERRUN interrupts. 1h (R/W) = Enable RECEIVER_OVERRUN interrupts.
3	CLK_PHASE	R/W	0h	SPI Clock Phase Select This bit controls the phase of the SPICLK signal. CLOCK PHASE and CLOCK POLARITY (SPICCR.6) make four different clocking schemes possible (see clocking figures in SPI chapter). When operating with CLOCK PHASE high, the SPI (master or slave) makes the first bit of data available after SPIDAT is written and before the first edge of the SPICLK signal, regardless of which SPI mode is being used. Reset type: SYSRSn 0h (R/W) = Normal SPI clocking scheme, depending on the CLOCK POLARITY bit (SPICCR.6). 1h (R/W) = SPICLK signal delayed by one half-cycle. Polarity determined by the CLOCK POLARITY bit.
2	MASTER_SLAVE	R/W	0h	SPI Network Mode Control This bit determines whether the SPI is a network master or slave. SLAVE During reset initialization, the SPI is automatically configured as a network slave. Reset type: SYSRSn 0h (R/W) = SPI is configured as a slave. 1h (R/W) = SPI is configured as a master.

Table 11-10. SPI Operation Control (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.

11.5.2.3 SPI Status (SPISTS) Register (Offset = 2h) [reset = 0h]

SPISTS contains interrupt and status bits.

Figure 11-14. SPI Status (SPISTS) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
OVERRUN_FLAG	INT_FLAG	BUFFULL_FLAG	RESERVED				
W1C-0h	RC-0h	R-0h	R-0h				

Table 11-11. SPI Status (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 11-11. SPI Status (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

11.5.2.4 SPI Baud Rate Register (SPIBRR) (Offset = 4h) [reset = 0h]

SPIBRR contains the bits used for baud-rate selection.

Figure 11-15. SPI Baud Rate Register (SPIBRR)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	SPI_BIT_RATE						
R-0h	R/W-0h						

Table 11-12. SPI Baud Rate Register (SPIBRR) Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6-0	SPI_BIT_RATE	R/W	0h	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

11.5.2.5 SPI Emulation Buffer (SPIRXEMU) Register (Offset = 6h) [reset = 0h]

SPIRXEMU contains the received data. Reading SPIRXEMU does not clear the SPI INT FLAG bit of SPISTS. This is not a real register but a dummy address from which the contents of SPIRXBUF can be read by the debug probe connection without clearing the SPI INT FLAG.

Figure 11-16. SPI Emulation Buffer (SPIRXEMU) Register

15	14	13	12	11	10	9	8
ERXBn							
R-0h							
7	6	5	4	3	2	1	0
ERXBn							
R-0h							

Table 11-13. SPI Emulation Buffer (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 with a debug probe connection, the control registers are read to continually update the contents of these registers on the display screen. SPIRXEMU was created so that the debug probe connection can read this register and properly update the contents on the display screen. Reading SPIRXEMU does not clear the SPI INT FLAG bit, but reading SPIRXBUF clears this flag. In other words, SPIRXEMU enables the debug probe connection to emulate the true operation of the SPI more accurately. It is recommended that you view SPIRXEMU when the debug probe is connected. Reset type: SYSRSn

11.5.2.6 SPI Serial Input Buffer (SPIRXBUF) Register (Offset = 7h) [reset = 0h]

SPIRXBUF contains the received data. Reading SPIRXBUF clears the SPI INT FLAG bit in SPISTS. If FIFO mode is enabled, reading this register will also decrement the RXFFST counter in SPIFFRX.

Figure 11-17. SPI Serial Input Buffer (SPIRXBUF) Register

15	14	13	12	11	10	9	8
RXBn							
R-0h							
7	6	5	4	3	2	1	0
RXBn							
R-0h							

Table 11-14. SPI Serial Input Buffer (SPIRXBUF) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	RXBn	R	0h	Received Data Once SPIDAT has received the complete character, the character is transferred to SPIRXBUF, where it can be read. At the same time, the SPI INT FLAG bit (SPISTS.6) is set. Since data is shifted into the SPI's most significant bit first, it is stored right-justified in this register. Reset type: SYSRSn

11.5.2.7 SPI Serial Output Buffer (SPITXBUF) Register (Offset = 8h) [reset = 0h]

SPITXBUF stores the next character to be transmitted. Writing to this register sets the TX BUF FULL Flag bit in SPISTS. When the transmission of the current character is complete, the contents of this register are automatically loaded in SPIDAT and the TX BUF FULL Flag is cleared. If no transmission is currently active, data written to this register falls through into the SPIDAT register and the TX BUF FULL Flag is not set. In master mode, if no transmission is currently active, writing to this register initiates a transmission in the same manner that writing to SPIDAT does.

Figure 11-18. SPI Serial Output Buffer (SPITXBUF) Register

15	14	13	12	11	10	9	8
TXBn							
R/W-0h							
7	6	5	4	3	2	1	0
TXBn							
R/W-0h							

Table 11-15. SPI Serial Output Buffer (SPITXBUF) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	TXBn	R/W	0h	Transmit Data Buffer This is where the next character to be transmitted is stored. When the transmission of the current character has completed, if the TX BUF FULL Flag bit is set, the contents of this register is automatically transferred to SPIDAT, and the TX BUF FULL Flag is cleared. Writes to SPITXBUF must be left-justified. Reset type: SYSRSn

11.5.2.8 SPI Serial Data (SPIDAT) Register (Offset = 9h) [reset = 0h]

SPIDAT is the transmit and receive shift register. Data written to SPIDAT is shifted out (MSB) on subsequent SPICLK cycles. For every bit (MSB) shifted out of the SPI, a bit is shifted into the LSB end of the shift register.

Figure 11-19. SPI Serial Data (SPIDAT) Register

15	14	13	12	11	10	9	8
SDATn							
R/W-0h							
7	6	5	4	3	2	1	0
SDATn							
R/W-0h							

Table 11-16. SPI Serial Data (SPIDAT) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	SDATn	R/W	0h	Serial Data Shift Register - It provides data to be output on the serial output pin if the TALK bit (SPICTL.1) is set. - When the SPI is operating as a master, a data transfer is initiated. When initiating a transfer, check the CLOCK POLARITY bit (SPICCR.6) described in Section 10.2.1.1 and the CLOCK PHASE bit (SPICTL.3) described in Section 10.2.1.2, for the requirements. In master mode, writing dummy data to SPIDAT initiates a receiver sequence. Since the data is not hardware-justified for characters shorter than sixteen bits, transmit data must be written in left-justified form, and received data read in right-justified form. Reset type: SYSRSn

11.5.2.9 SPI FIFO Transmit (SPIFFTX) Register (Offset = Ah) [reset = A000h]

SPIFFTX contains both control and status bits related to the output FIFO buffer. This includes FIFO reset control, FIFO interrupt level control, FIFO level status, as well as FIFO interrupt enable and clear bits.

Figure 11-20. SPI FIFO Transmit (SPIFFTX) Register

15	14	13	12	11	10	9	8
SPIRST	SPIFFENA	TXFIFO	TXFFST				
R/W-1h	R/W-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL				
R-0h	W-0h	R/W-0h	R/W-0h				

Table 11-17. SPI FIFO Transmit (SPIFFTX) Register Field Descriptions

Bit	Field	Type	Reset	Description
15	SPIRST	R/W	1h	SPI Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the SPI transmit and receive channels. The SPI FIFO register configuration bits will be left as is. 1h (R/W) = SPI FIFO can resume transmit or receive. No effect to the SPI registers bits.
14	SPIFFENA	R/W	0h	SPI FIFO Enhancements Enable Reset type: SYSRSn 0h (R/W) = SPI FIFO enhancements are disabled. 1h (R/W) = SPI FIFO enhancements are enabled.
13	TXFIFO	R/W	1h	TX FIFO Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Release transmit FIFO from reset.
12-8	TXFFST	R	0h	Transmit FIFO Status Reset type: SYSRSn 0h (R/W) = Transmit FIFO is empty. 1h (R/W) = Transmit FIFO has 1 word. 2h (R/W) = Transmit FIFO has 2 words. 3h (R/W) = Transmit FIFO has 3 words. 4h (R/W) = Transmit FIFO has 4 words, which is the maximum. 1Fh (R/W) = Reserved.
7	TXFFINT	R	0h	TX FIFO Interrupt Flag Reset type: SYSRSn 0h (R/W) = TXFIFO interrupt has not occurred, This is a read-only bit. 1h (R/W) = TXFIFO interrupt has occurred, This is a read-only bit.
6	TXFFINTCLR	W	0h	TXFIFO Interrupt Clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFTX[TXFFINT] flag.

Table 11-17. SPI FIFO Transmit (SPIFFTX) Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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.
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. 3h (R/W) = A TX FIFO interrupt request is generated when there are 3 words or fewer remaining in the TX buffer. 4h (R/W) = A TX FIFO interrupt request is generated when there are 4 words or fewer remaining in the TX buffer. 1Fh (R/W) = Reserved.

11.5.2.10 SPI FIFO Receive (SPIFFRX) Register (Offset = Bh) [reset = 201Fh]

SPIFFRX contains both control and status bits related to the input FIFO buffer. This includes FIFO reset control, FIFO interrupt level control, FIFO level status, as well as FIFO interrupt enable and clear bits.

Figure 11-21. SPI FIFO Receive (SPIFFRX) Register

15	14	13	12	11	10	9	8
RXFFOVF	RXFFOVFCLR	RXFIFORESET	RXFFST				
R-0h	W-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL				
R-0h	W-0h	R/W-0h	R/W-1Fh				

Table 11-18. SPI FIFO Receive (SPIFFRX) Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RXFFOVF	R	0h	Receive FIFO Overflow Flag Reset type: SYSRSn 0h (R/W) = Receive FIFO has not overflowed. This is a read-only bit. 1h (R/W) = Receive FIFO has overflowed, read-only bit. More than 4 words have been received in to the FIFO, and the first received word is lost.
14	RXFFOVFCLR	W	0h	Receive FIFO Overflow Clear Reset type: SYSRSn 0h (R/W) = Write 0 does not affect RXFFOVF flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFRX[RXFFOVF].
13	RXFIFORESET	R/W	1h	Receive FIFO Reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Re-enable receive FIFO operation.
12-8	RXFFST	R	0h	Receive FIFO Status Reset type: SYSRSn 0h (R/W) = Receive FIFO is empty. 1h (R/W) = Receive FIFO has 1 word. 2h (R/W) = Receive FIFO has 2 words. 3h (R/W) = Receive FIFO has 3 words. 4h (R/W) = Receive FIFO has 4 words, which is the maximum. 1Fh (R/W) = Reserved.
7	RXFFINT	R	0h	Receive FIFO Interrupt Flag Reset type: SYSRSn 0h (R/W) = RXFIFO interrupt has not occurred. This is a read-only bit. 1h (R/W) = RXFIFO interrupt has occurred. This is a read-only bit.
6	RXFFINTCLR	W	0h	Receive FIFO Interrupt Clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFIFINT flag bit, Bit reads back a zero. 1h (R/W) = Write 1 to clear SPIFFRX[RXFFINT] flag

Table 11-18. SPI FIFO Receive (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. 3h (R/W) = A RX FIFO interrupt request is generated when there are 3 or more words in the RX buffer. 4h (R/W) = A RX FIFO interrupt request is generated when there are 4 words in the RX buffer. 1Fh (R/W) = Reserved.

11.5.2.11 SPI FIFO Control (SPIFFCT) Register (Offset = Ch) [reset = 0h]

SPIFFCT controls the FIFO transmit delay bits.

Figure 11-22. SPI FIFO Control (SPIFFCT) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXDLY							
R/W-0h							

Table 11-19. SPI FIFO Control (SPIFFCT) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	TXDLY	R/W	0h	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.

11.5.2.12 SPI Priority Control (SPIPRI) Register (Offset = Fh) [reset = 0h]

SPIPRI controls auxiliary functions for the SPI including emulation control, and 3-wire control.

Figure 11-23. SPI Priority Control (SPIPRI) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	RESERVED	SOFT	FREE	RESERVED		STEINV	TRIWIRE
R-0h	R/W-0h	R/W-0h	R/W-0h	R-0h		R/W-0h	R/W-0h

Table 11-20. SPI Priority Control (SPIPRI) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	RESERVED	R/W	0h	Reserved
5	SOFT	R/W	0h	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.
3-2	RESERVED	R	0h	Reserved

Table 11-20. SPI Priority Control (SPIPRI) Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
1	STEINV	R/W	0h	SPISTEn Inversion Bit On devices with 2 SPI modules, inverting the SPISTE signal on one of the modules allows the device to receive left and right- channel digital audio data. This bit is only applicable to slave mode. Writing to this bit while configured as master (MASTER_SLAVE = 1) has no effect Reset type: SYSRSn 0h (R/W) = SPISTEn is active low (normal) 1h (R/W) = SPISTE is active high (inverted)
0	TRIWIRE	R/W	0h	SPI 3-wire Mode Enable Reset type: SYSRSn 0h (R/W) = Normal 4-wire SPI mode. 1h (R/W) = 3-wire SPI mode enabled. The unused pin becomes a GPIO pin. In master mode, the SPISIMO pin becomes the SPIMOMI (master receive and transmit) pin and SPISOMI is free for non-SPI use. In slave mode, the SPISOMI pin becomes the SPISISO (slave receive and transmit) pin and SPISIMO is free for non-SPI use.

This page intentionally left blank.

Chapter 12

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 4-level deep FIFO for reducing servicing overhead, and each has its own separate enable and interrupt bits. Both can be operated independently for half-duplex communication, or simultaneously for full-duplex communication.

To specify data integrity, the SCI checks received data for break detection, parity, overrun, and framing errors. The bit rate is programmable to different speeds through a 16-bit baud-select register.

12.1 Introduction

The SCI interfaces are shown in Figure 12-1.

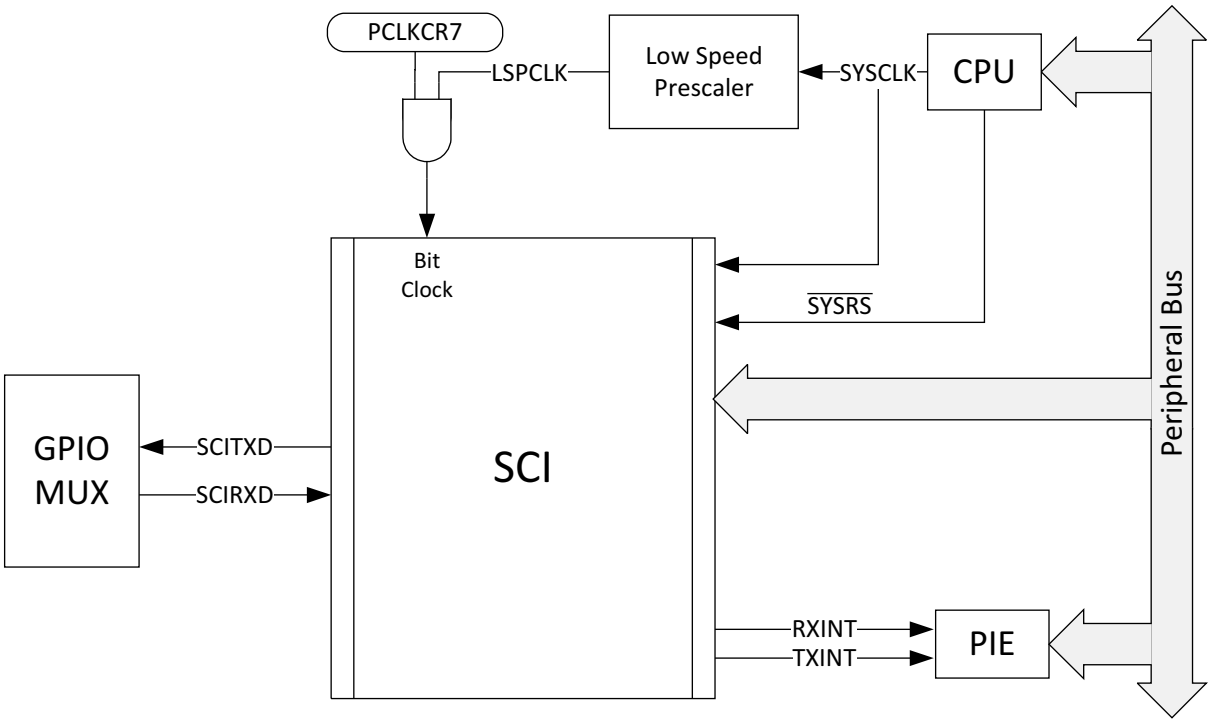


Figure 12-1. SCI CPU Interface

12.1.1 Features

Features of the SCI module include:

- Two external pins:
  - SCITXD: SCI transmit-output pin
  - SCIRXD: SCI receive-input pin

  Both pins can be used as GPIO if not used for SCI.
- Baud rate programmable to 64K different rates
- Data-word format
  - One start bit
  - Data-word length programmable from one to eight bits
  - Optional even/odd/no parity bit
  - One or two stop bits
  - An extra bit to distinguish addresses from data (address bit mode only)
- Four error-detection flags: parity, overrun, framing, and break detection
- Two wake-up multiprocessor modes: idle-line and address bit
- Half- or full-duplex operation
- Double-buffered receive and transmit functions
- Transmitter and receiver operations can be accomplished through interrupt-driven or polled algorithms with status flags
- Separate enable bits for transmitter and receiver interrupts (except BRKDT)
- NRZ (non-return-to-zero) format

Enhanced features include:

- Auto-baud-detect hardware logic
- 4-level transmit/receive FIFO

12.1.2 SCI Related Collateral

Foundational Materials

- One Minute RS-485 Introduction (Video)
- RS-232, RS-422, RS-485: What Are the Differences? (Video)

Getting Started Materials

- [FAQ] My C2000 SCI is not Transmitting and/or Receiving data correctly, how do I fix this?

12.1.3 Block Diagram

Figure 12-2 shows the SCI module block diagram. The SCI port operation is configured and controlled by the registers listed in Section 12.14.

12.2 Architecture

The major elements used in full-duplex operation are shown in Figure 12-2 and include:

- A transmitter (TX) and its major registers (upper half of Figure 12-2)
  - 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 12-2)
  - 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.

12.3 SCI Module Signal Summary

A summarized description of each SCI signal name is shown in Table 12-1.

Table 12-1. 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

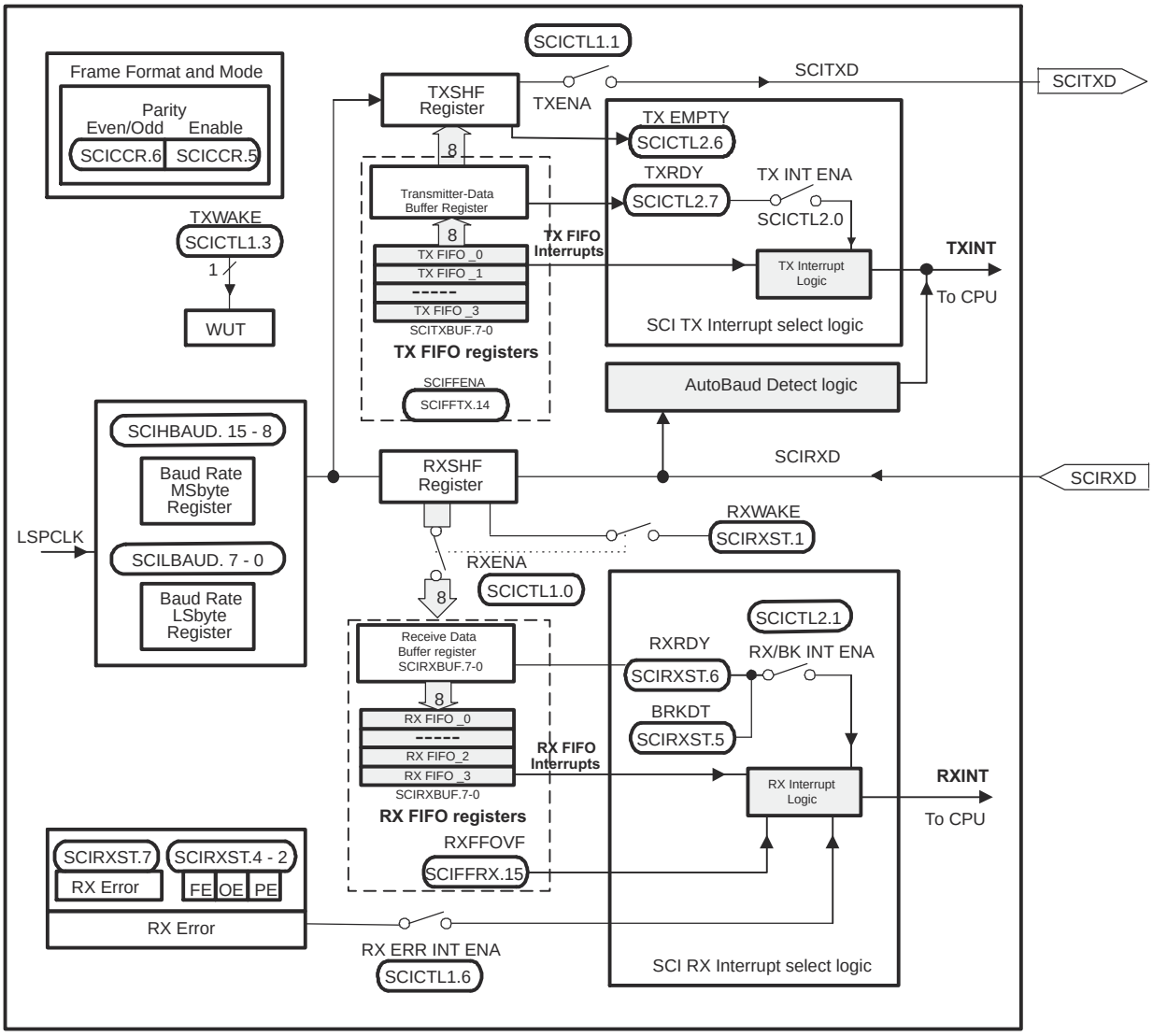


Figure 12-2. Serial Communications Interface (SCI) Module Block Diagram

12.4 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins. To avoid glitches on the pins, the GPyMUX bits must be configured first (while keeping the corresponding GPyMUX bits at the default of zero), followed by writing the GPyMUX register to the desired value.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification should be set to asynchronous mode by setting the appropriate GPxQSELn register bits to 11b. The internal pullups can be configured in the GPyPUD register.

See the *General-Purpose Input/Output (GPIO)* chapter for more details on GPIO mux and settings.

12.5 Multiprocessor and Asynchronous Communication Modes

The SCI has two multiprocessor protocols, the idle-line multiprocessor mode (see Section 12.8) and the address-bit multiprocessor mode (see Section 12.9). 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 12.10) 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

12.6 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 12-3, 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 12-3.

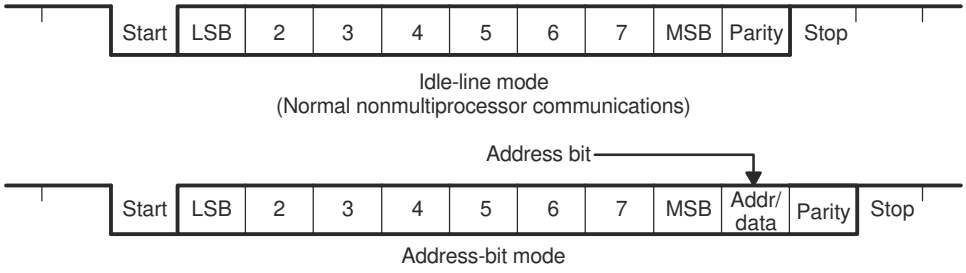


Figure 12-3. Typical SCI Data Frame Formats

To program the data format, use the SCICCR register. The bits used to program the data format are shown in Table 12-2.

Table 12-2. Programming the Data Format Using SCICCR

Bit(s)	Bit Name	Designation	Functions
2-0	SCICHAR	SCICCR.2:0	Select the character (data) length (one to eight bits).
5	PARITYENA (ENABLE)	SCICCR.5	Enables the parity function if set to 1, or disables the parity function if cleared to 0.
6	PARITY (EVEN/ODD)	SCICCR.6	If parity is enabled, selects odd parity if cleared to 0; even parity if set to 1.
7	STOPBITS	SCICCR.7	Determines the number of stop bits transmitted—one stop bit if cleared to 0 or two stop bits if set to 1.

12.7 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.

12.7.1 Recognizing the Address Byte

A processor recognizes an address byte differently, depending on the multiprocessor mode used. For example:

- The idle-line mode (Section 12.8) 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 12.9) 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.

12.7.2 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.

12.7.3 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.

12.8 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 12-4 (ADDR/IDLE MODE bit is bit 3 of SCICCR).

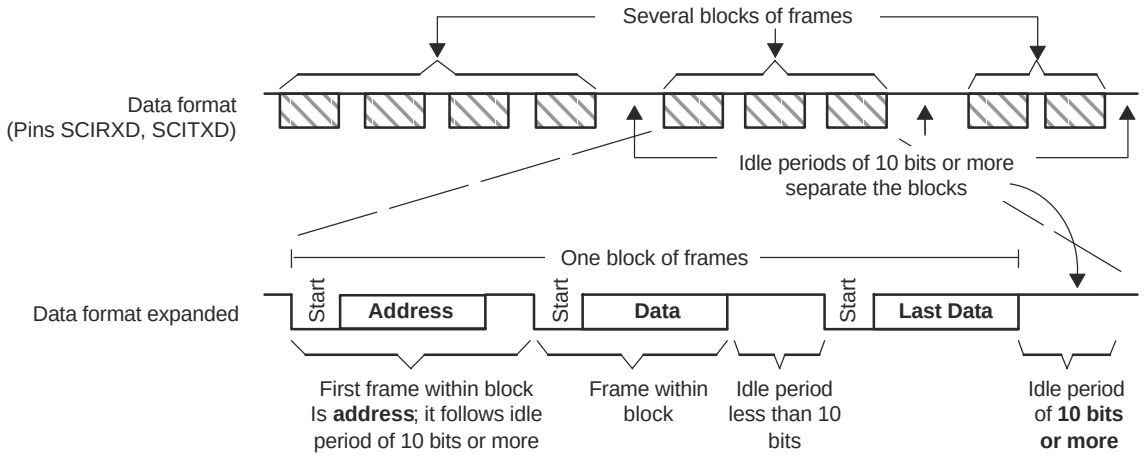


Figure 12-4. Idle-Line Multiprocessor Communication Format

12.8.1 Idle-Line Mode Steps

The steps followed by the idle-line mode:

1. SCI wakes up after receipt of the block-start signal.
2. The processor recognizes the next SCI interrupt.
3. The interrupt service routine compares the received address (sent by a remote transmitter) to its own.
4. If the CPU is being addressed, the service routine clears the SLEEP bit and receives the rest of the data block.
5. If the CPU is not being addressed, the SLEEP bit remains set. This lets the CPU continue to execute its main program without being interrupted by the SCI port until the next detection of a block start.

12.8.2 Block Start Signal

There are two ways to send a block-start signal:

- **Method 1:** Deliberately leave an idle time of ten bits or more by delaying the time between the transmission of the last frame of data in the previous block and the transmission of the address frame of the new block.
- **Method 2:** The SCI port first sets the TXWAKE bit (SCICTL1, bit 3) to 1 before writing to the SCITXBUF register. This sends an idle time of exactly 11 bits. In this method, the serial communications line is not idle any longer than necessary. (A don't care byte has to be written to SCITXBUF after setting TXWAKE, and before sending the address, so as to transmit the idle time.)

12.8.3 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 12-5.

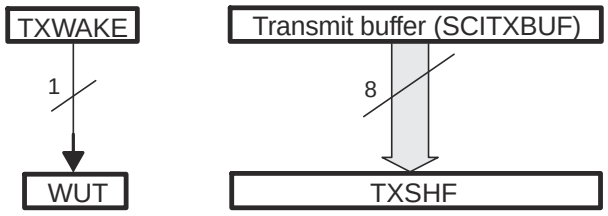


Figure 12-5. Double-Buffered WUT and TXSHF

12.8.3.1 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.

12.8.4 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.

12.9 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 12-6).

12.9.1 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.

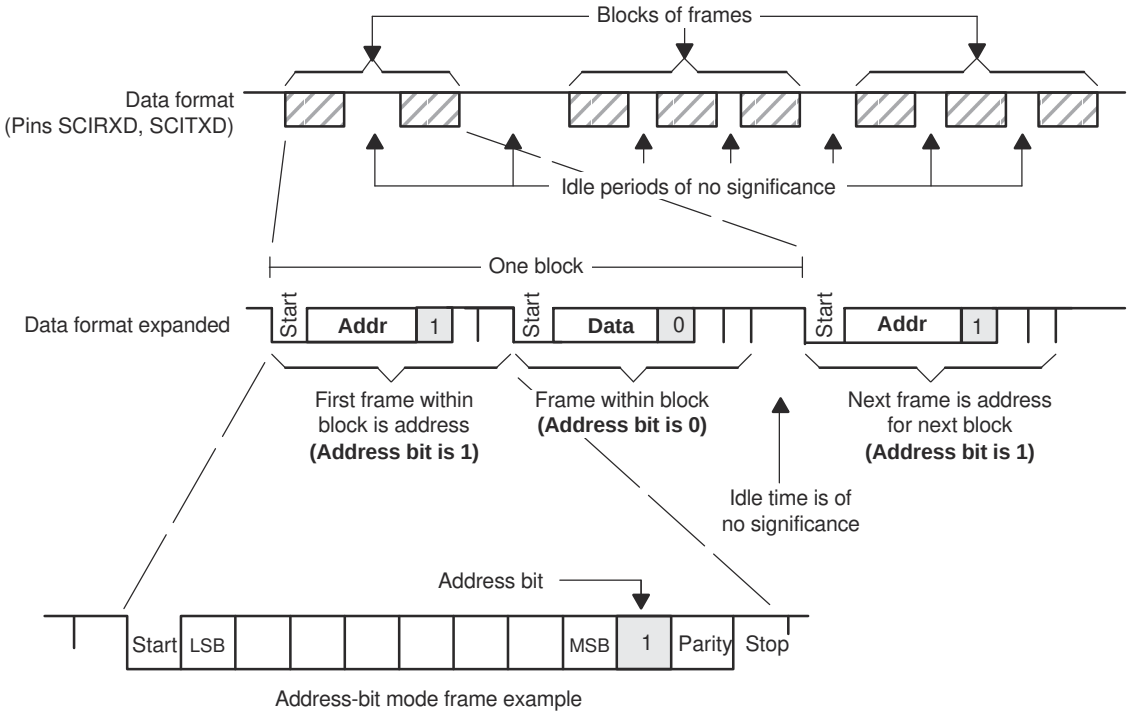


Figure 12-6. Address-Bit Multiprocessor Communication Format

12.10 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 12-7). 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 12-7. 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 12-7 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.

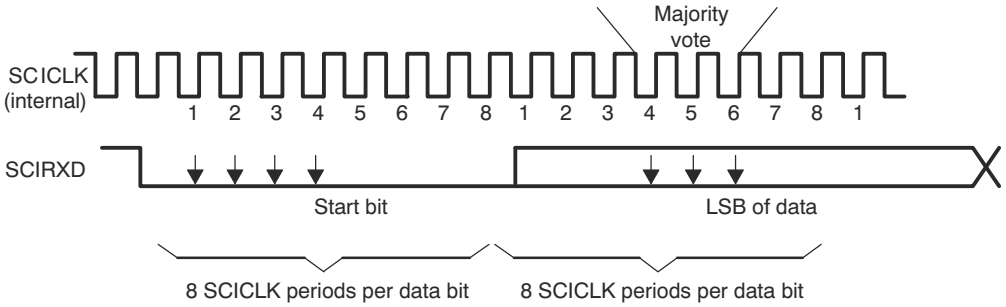


Figure 12-7. SCI Asynchronous Communications Format

12.10.1 Receiver Signals in Communication Modes

Figure 12-8 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

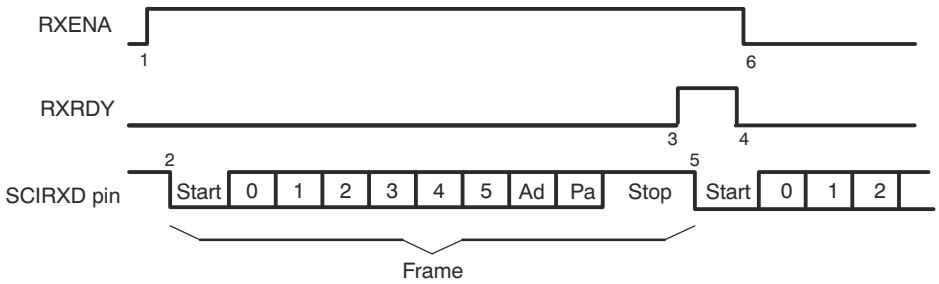


Figure 12-8. SCI RX Signals in Communication Modes

Notes:

1. Flag bit RXENA (SCICTL1, bit 0) goes high to enable the receiver.
2. Data arrives on the SCIRXD pin, start bit detected.
3. Data is shifted from RXSHF to the receiver buffer register (SCIRXBUF); an interrupt is requested. Flag bit RXRDY (SCIRXST, bit 6) goes high to signal that a new character has been received.
4. The program reads SCIRXBUF; flag RXRDY is automatically cleared.
5. The next byte of data arrives on the SCIRXD pin; the start bit is detected, then cleared.
6. Bit RXENA is brought low to disable the receiver. Data continues to be assembled in RXSHF but is not transferred to the receiver buffer register.

12.10.2 Transmitter Signals in Communication Modes

Figure 12-9 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

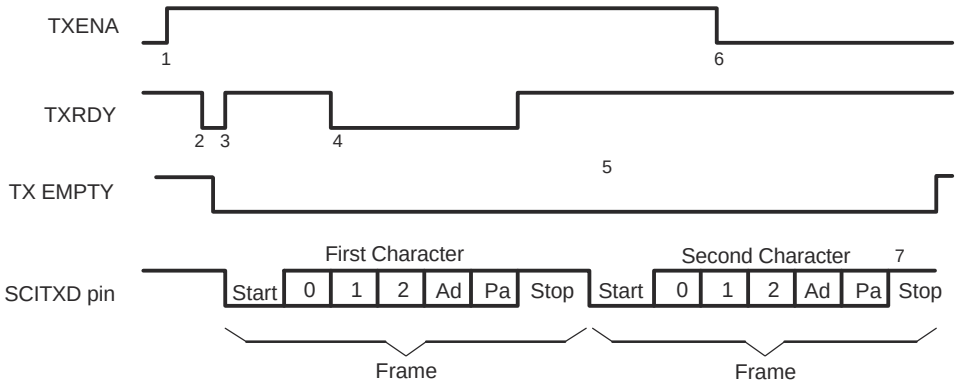


Figure 12-9. SCI TX Signals in Communications Mode

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.

12.11 SCI Port Interrupts

The SCI receiver and transmitter can be interrupt controlled. The SCICTL2 register has one flag bit (TXRDY) that indicates active interrupt conditions, and the SCIRXST register has two interrupt flag bits (RXRDY and BRKDT), plus the RX ERROR interrupt flag that is a logical-OR of the FE, OE, BRKDT, and PE conditions. The transmitter and receiver have separate interrupt-enable bits. When not enabled, the interrupts are not asserted; however, the condition flags remain active, reflecting transmission and receipt status.

The SCI has independent peripheral interrupt vectors for the receiver and transmitter. Peripheral interrupt requests can be either high priority or low priority. This is indicated by the priority bits that are output from the peripheral to the PIE controller. When both RX and TX interrupt requests are made at the same priority level, the receiver always has higher priority than the transmitter, reducing the possibility of receiver overrun.

The operation of peripheral interrupts is described in the Peripheral Interrupts section of the *System Control and Interrupts* chapter.

- If the RX/BK INT ENA bit (SCICTL2, bit 1) is set, the receiver peripheral interrupt request is asserted when one of the following events occurs:
  - The SCI receives a complete frame and transfers the data in the RXSHF register to the SCIRXBUF register. This action sets the RXRDY flag (SCIRXST, bit 6) and initiates an interrupt.
  - A break detect condition occurs (the SCIRXD is low for 9.625 bit periods following a missing stop bit). This action sets the BRKDT flag bit (SCIRXST, bit 5) and initiates an interrupt.
- If the TX INT ENA bit (SCICTL2.0) is set, the transmitter peripheral interrupt request is asserted whenever the data in the SCITXBUF register is transferred to the TXSHF register, indicating that the CPU can write to SCITXBUF; this action sets the TXRDY flag bit (SCICTL2, bit 7) and initiates an interrupt.

Note

Interrupt generation due to the RXRDY and BRKDT bits is controlled by the RX/BK INT ENA bit (SCICTL2, bit 1). Interrupt generation due to the RX ERROR bit is controlled by the RX ERR INT ENA bit (SCICTL1, bit 6).

12.12 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 12-3 shows the baud-select values for common SCI bit rates. LSPCLK/16 is the maximum baud rate. For example, if LSPCLK is 100 MHz, then the maximum baud rate is 6.25 Mbps.

Table 12-3. 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

12.13 SCI Enhanced Features

The C28x SCI features autobaud detection and transmit/receive FIFO. The following section explains the FIFO operation.

12.13.1 SCI FIFO Description

The following steps explain the FIFO features and help with programming the SCI with FIFOs.

1. **Reset.** At reset the SCI powers up in standard SCI mode and the FIFO function is disabled. The FIFO registers SCIFFTX, SCIFFRX, and SCIFFCT remain inactive.
2. **Standard SCI.** The standard SCI modes will work normally with TXINT/RXINT interrupts as the interrupt source for the module.
3. **FIFO enable.** FIFO mode is enabled by setting the SCIFFEN bit in the SCIFFTX register. SCIRST can reset the FIFO mode at any stage of its operation.
4. **Active registers.** All the SCI registers and SCI FIFO registers (SCIFFTX, SCIFFRX, and SCIFFCT) are active.
5. **Interrupts.** FIFO mode has two interrupts; one for transmit FIFO, TXINT and one for receive FIFO, RXINT. RXINT is the common interrupt for SCI FIFO receive, receive error, and receive FIFO overflow conditions. The TXINT of the standard SCI will be disabled and this interrupt will service as SCI transmit FIFO interrupt.
6. **Buffers.** Transmit and receive buffers are supplemented with two 4-level FIFOs. The transmit FIFO registers are 8 bits wide and receive FIFO registers are 10 bits wide. The one-word transmit buffer (SCITXBUF) of the standard SCI functions as a transition buffer before the transmit FIFO and shift register. SCITXBUF is loaded into either the FIFO (when FIFO is enabled) or TXSHF (when FIFO is disabled). When FIFO is enabled, SCITXBUF loads into the FIFO only after the last bit of the shift register is shifted out, so SCITXBUF should not be treated as an additional level of buffer. With the FIFO enabled, TXSHF is directly loaded from the FIFO (not TXBUF) after an optional delay value (SCIFFCT). When FIFO mode is enabled for SCI, characters written to SCITXBUF are queued in to SCI-TXFIFO and the characters received in SCI-RXFIFO can be read using SCIRXBUF.
7. **Delayed transfer.** The rate at which words in the FIFO are transferred to the transmit shift register is programmable. The SCIFFCT register bits (7−0) FFTXDLY7−FFTXDLY0 define the delay between the word transfer. The delay is defined in the number SCI baud clock cycles. The 8 bit register can define a minimum delay of 0 baud clock cycles and a maximum of 256-baud clock cycles. With zero delay, the SCI module can transmit data in continuous mode with the FIFO words shifting out back to back. With the 256 clock delay the SCI module can transmit data in a maximum delayed mode with the FIFO words shifting out with a delay of 256 baud clocks between each words. The programmable delay facilitates communication with slow SCI/UARTs with little CPU intervention.
8. **FIFO status bits.** Both the transmit and receive FIFOs have status bits TXFFST or RXFFST (bits 12−8) that define the number of words available in the FIFOs at any time. The transmit FIFO reset bit TXFIFO and receive reset bit RXFIFO reset the FIFO pointers to zero when these bits are cleared to 0. The FIFOs resumes operation from start once these bits are set to 1.
9. **Programmable interrupt levels.** Both transmit and receive FIFO can generate CPU interrupts. The interrupt trigger is generated whenever the transmit FIFO status bits TXFFST (bits 12−8) match (less than or equal to) the interrupt trigger level bits TXFFIL (bits 4−0). This provides a programmable interrupt trigger for transmit and receive sections of the SCI. Default value for these trigger level bits will be 0x11111 for receive FIFO and 0x00000 for transmit FIFO, respectively.

Figure 12-10 and Table 12-4 explain the operation/configuration of SCI interrupts in nonFIFO/FFO mode.

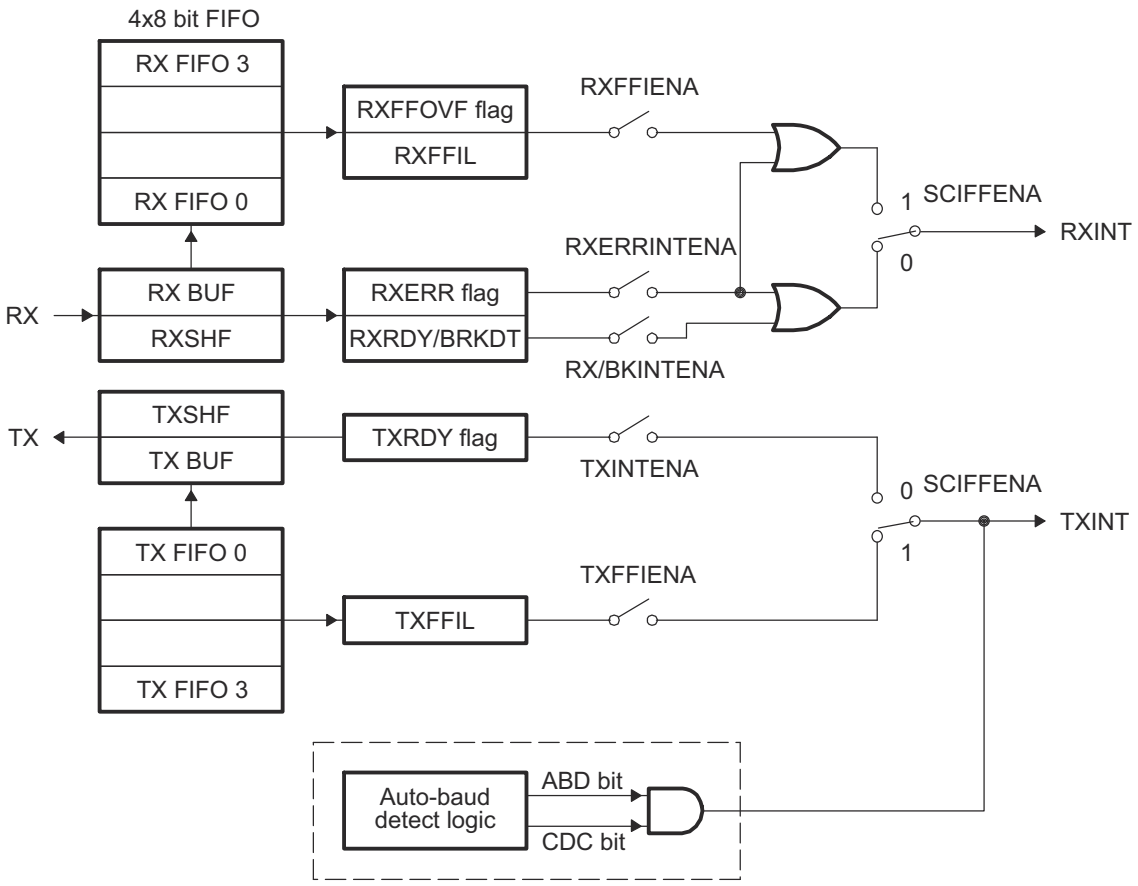


Figure 12-10. SCI FIFO Interrupt Flags and Enable Logic

Table 12-4. 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.

12.13.2 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.

12.13.3 Autobaud-Detect Sequence

Bits ABD and CDC in SCIFFCT control the autobaud logic. The SCIRST bit should be enabled to make autobaud logic work.

If ABD is set while CDC is 1, which indicates auto-baud alignment, SCI transmit FIFO interrupt will occur (TXINT). After the interrupt service CDC bit has to be cleared by software. If CDC remains set even after interrupt service, there should be no repeat interrupts.

1. Enable autobaud-detect mode for the SCI by setting the CDC bit (bit 13) in SCIFFCT and clearing the ABD bit (Bit 15) by writing a 1 to ABDCLR bit (bit 14).
2. Initialize the baud register to be 1 or less than a baud rate limit of 500 Kbps.
3. Allow SCI to receive either character "A" or "a" from a host at the desired baud rate. If the first character is either "A" or "a", the autobaud-detect hardware will detect the incoming baud rate and set the ABD bit.
4. The auto-detect hardware will update the baud rate register with the equivalent baud value hex. The logic will also generate an interrupt to the CPU.
5. Respond to the interrupt clear ADB bit by writing a 1 to ABD CLR (bit 14) of SCIFFCT register and disable further autobaud locking by clearing CDC bit by writing a 0.
6. Read the receive buffer for character "A" or "a" to empty the buffer and buffer status.
7. If ABD is set while CDC is 1, which indicates autobaud alignment, the SCI transmit FIFO interrupt will occur (TXINT). After the interrupt service CDC bit must be cleared by software.

Note

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

To avoid this, the following is recommended:

- Achieve a baud-lock between the host and C28x SCI boot loader using a lower baud rate.
- The host may then handshake with the loaded C28x application to set the SCI baud rate register to the desired higher baud rate.

12.14 SCI Registers

The section describes the Serial Communication Interface module registers.

12.14.1 SCI Base Addresses

Table 12-5. SCI Base Address Table (C28)

Bit Field Name		Base Address
Instance	Structure	
SciaRegs	SCI_REGS	0x0000_7050

12.14.2 SCI_REGS Registers

Table 12-6 lists the SCI_REGS registers. All register offset addresses not listed in Table 12-6 should be considered as reserved locations and the register contents should not be modified.

Table 12-6. SCI_REGS Registers

Offset	Acronym	Register Name	Section
0h	SCICCR	SCI Communications Control Register	Section 12.14.2.1
1h	SCICTL1	SCI Control Register 1	Section 12.14.2.2
2h	SCIHBAUD	SCI Baud Rate (high) Register	Section 12.14.2.3
3h	SCILBAUD	SCI Baud Rate (low) Register	Section 12.14.2.4
4h	SCICTL2	SCI Control Register 2	Section 12.14.2.5
5h	SCIRXST	SCI Receive Status Register	Section 12.14.2.6
6h	SCIRXEMU	SCI Receive Emulation Buffer Register	Section 12.14.2.7
7h	SCIRXBUF	SCI Receive Data Buffer Register	Section 12.14.2.8
9h	SCITXBUF	SCI Transmit Data Buffer Register	Section 12.14.2.9
Ah	SCIFFTX	SCI FIFO Transmit Register	Section 12.14.2.10
Bh	SCIFFRX	SCI FIFO Receive Register	Section 12.14.2.11
Ch	SCIFFCT	SCI FIFO Control Register	Section 12.14.2.12
Fh	SCIPRI	SCI Priority Control Register	Section 12.14.2.13

Complex bit access types are encoded to fit into small table cells. Table 12-7 shows the codes that are used for access types in this section.

Table 12-7. SCI_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
R-0	R -0	Read Returns 0s
Write Type		
W	W	Write
W1S	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.

12.14.2.1 SCI Communications Control Register (SCICCR) (Offset = 0h) [reset = 0h]

SCICCR defines the character format, protocol, and communications mode used by the SCI.

Figure 12-11. SCI Communications Control Register (SCICCR)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
STOPBITS	PARITY	PARITYENA	LOOPBKENA	ADDRIDLE_MODE	SCICHAR		
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h		

Table 12-8. SCI Communications Control Register (SCICCR) Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7	STOPBITS	R/W	0h	SCI number of stop bits. This bit specifies the number of stop bits transmitted. The receiver checks for only one stop bit. Reset type: SYSRSn 0h (R/W) = One stop bit 1h (R/W) = Two stop bits
6	PARITY	R/W	0h	SCI parity odd/even selection. If the PARITY ENABLE bit (SCICCR, bit 5) is set, PARITY (bit 6) designates odd or even parity (odd or even number of bits with the value of 1 in both transmitted and received characters). Reset type: SYSRSn 0h (R/W) = Odd parity 1h (R/W) = Even parity
5	PARITYENA	R/W	0h	SCI parity enable. This bit enables or disables the parity function. If the SCI is in the address bit multiprocessor mode (set using bit 3 of this register), the address bit is included in the parity calculation (if parity is enabled). For characters of less than eight bits, the remaining unused bits should be masked out of the parity calculation. Reset type: SYSRSn 0h (R/W) = Parity disabled no parity bit is generated during transmission or is expected during reception 1h (R/W) = Parity is enabled
4	LOOPBKENA	R/W	0h	Loop Back test mode enable. This bit enables the Loop Back test mode where the Tx pin is internally connected to the Rx pin. Reset type: SYSRSn 0h (R/W) = Loop Back test mode disabled 1h (R/W) = Loop Back test mode enabled
3	ADDRIDLE_MODE	R/W	0h	SCI multiprocessor mode control bit. This bit selects one of the multiprocessor protocols.Multiprocessor communication is different from the other communication modes because it uses SLEEP and TXWAKE functions (bits SCICTL1, bit 2 and SCICTL1, bit 3, respectively). The idle-line mode is usually used for normal communications because the address-bit mode adds an extra bit to the frame. The idle-line mode does not add this extra bit and is compatible with RS-232 type communications. Reset type: SYSRSn 0h (R/W) = Idle-line mode protocol selected 1h (R/W) = Address-bit mode protocol selected

Table 12-8. SCI Communications Control Register (SCICCR) Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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

12.14.2.2 SCI Control Register 1 (SCICTL1) (Offset = 1h) [reset = 0h]

SCICTL1 controls the receiver/transmitter enable, TXWAKE and SLEEP functions, and the SCI software reset.

Figure 12-12. SCI Control Register 1 (SCICTL1)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	RXERRINTENA	SWRESET	RESERVED	TXWAKE	SLEEP	TXENA	RXENA
R-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 12-9. SCI Control Register 1 (SCICTL1) Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	RXERRINTENA	R/W	0h	SCI receive error interrupt enable. Setting this bit enables an interrupt if the RX ERROR bit (SCIRXST, bit 7) becomes set because of errors occurring. Reset type: SYSRSn 0h (R/W) = Receive error interrupt disabled 1h (R/W) = Receive error interrupt enabled
5	SWRESET	R/W	0h	SCI software reset (active low). Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition. The SW RESET bit does not affect any of the configuration bits. All affected logic is held in the specified reset state until a 1 is written to SW RESET (the bit values following a reset are shown beneath each register diagram in this section). Thus, after a system reset, re-enable the SCI by writing a 1 to this bit. Clear this bit after a receiver break detect (BRKDT flag, bit SCIRXST, bit 5). SW RESET affects the operating flags of the SCI, but it neither affects the configuration bits nor restores the reset values. Once SW RESET is asserted, the flags are frozen until the bit is deasserted. The affected flags are as follows: Value After SW SCI Flag Register Bit RESET 1 TXRDY SCICTL2, bit 7 1 TX EMPTY SCICTL2, bit 6 0 RXWAKE SCIRXST, bit 1 0 PE SCIRXST, bit 2 0 OE SCIRXST, bit 3 0 FE SCIRXST, bit 4 0 BRKDT SCIRXST, bit 5 0 RXRDY SCIRXST, bit 6 0 RX ERROR SCIRXST, bit 7 Reset type: SYSRSn 0h (R/W) = Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition. 1h (R/W) = After a system reset, re-enable the SCI by writing a 1 to this bit.
4	RESERVED	R	0h	Reserved

Table 12-9. SCI Control Register 1 (SCICTL1) Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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

12.14.2.3 SCI Baud Rate (high) (SCIHBAUD) Register (Offset = 2h) [reset = 0h]

The values in SCIHBAUD and SCILBAUD specify the baud rate for the SCI.

Figure 12-13. SCI Baud Rate (high) (SCIHBAUD) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
BAUD							
R/W-0h							

Table 12-10. SCI Baud Rate (high) (SCIHBAUD) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	BAUD	R/W	0h	SCI 16-bit baud selection Registers SCIHBAUD (MSbyte). The internally-generated serial clock is determined by the low speed peripheral clock (LSPCLK) signal and the two baud-select registers. The SCI uses the 16-bit value of these registers to select one of 64K serial clock rates for the communication modes. BRR = (SCIHBAUD << 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

12.14.2.4 SCI Baud Rate (low) (SCILBAUD) Register (Offset = 3h) [reset = 0h]

The values in SCIHBAUD and SCILBAUD specify the baud rate for the SCI.

Figure 12-14. SCI Baud Rate (low) (SCILBAUD) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
BAUD							
R/W-0h							

Table 12-11. SCI Baud Rate (low) (SCILBAUD) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	BAUD	R/W	0h	See SCIHBAUD Detailed Description Reset type: SYSRSn

12.14.2.5 SCI Control Register 2 (SCICTL2) (Offset = 4h) [reset = C0h]

SCICTL2 enables the receive-ready, break-detect, and transmit-ready interrupts as well as transmitter-ready and -empty flags.

Figure 12-15. SCI Control Register 2 (SCICTL2)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXRDY	TXEMPTY	RESERVED				RXBKINTENA	TXINTENA
R-1h	R-1h	R-0h				R/W-0h	R/W-0h

Table 12-12. SCI Control Register 2 (SCICTL2) 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
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

12.14.2.6 SCI Receive Status (SCIRXST) Register (Offset = 5h) [reset = 0h]

SCIRXST contains seven bits that are receiver status flags (two of which can generate interrupt requests). Each time a complete character is transferred to the receiver buffers (SCIRXEMU and SCIRXBUF), the status flags are updated.

Figure 12-16. SCI Receive Status (SCIRXST) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RXERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	RESERVED
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

Table 12-13. SCI Receive Status (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 condition occurs when the SCI receiver data line (SCIRXD) remains continuously low for at least ten bits, beginning after a missing first stop bit. The occurrence of a break causes a receiver interrupt to be generated if the RX/BK INT ENA bit is a 1, but it does not cause the receiver buffer to be loaded. A BRKDT interrupt can occur even if the receiver SLEEP bit is set to 1. BRKDT is cleared by an active SW RESET or by a system reset. It is not cleared by receipt of a character after the break is detected. In order to receive more characters, the SCI must be reset by toggling the SW RESET bit or by a system reset. Reset type: SYSRSn 0h (R/W) = No break condition 1h (R/W) = Break condition occurred

Table 12-13. SCI Receive Status (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

12.14.2.7 SCI Receive Emulation Buffer (SCIRXEMU) Register (Offset = 6h) [reset = 0h]

Normal SCI data-receive operations read the data received from the SCIRXBUF register. The SCIRXEMU register is used principally during a debug connection because it can continuously read the data received for screen updates without clearing the RXRDY flag. SCIRXEMU is cleared by a system reset. This is the register that should be used in the CCS watch window to view the contents of the SCIRXBUF register. SCIRXEMU is not physically implemented
it is just a different address location to access the SCIRXBUF register without clearing the RXRDY flag.

Figure 12-17. SCI Receive Emulation Buffer (SCIRXEMU) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
ERXDT							
R-0h							

Table 12-14. SCI Receive Emulation Buffer (SCIRXEMU) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	ERXDT	R	0h	Receive emulation buffer data Reset type: SYSRSn

12.14.2.8 SCI Receive Data Buffer (SCIRXBUF) Register (Offset = 7h) [reset = 0h]

When the current data received is shifted from RXSHF to the receiver buffer, flag bit RXRDY is set and the data is ready to be read. If the RXBKINTENA bit (SCICTL2.1) is set, this shift also causes an interrupt. When SCIRXBUF is read, the RXRDY flag is reset. SCIRXBUF is cleared by a system reset.

Figure 12-18. SCI Receive Data Buffer (SCIRXBUF) Register

15	14	13	12	11	10	9	8
SCIFFFE	SCIFFPE	RESERVED					
R-0h	R-0h	R-0h					
7	6	5	4	3	2	1	0
SAR							
R-0h							

Table 12-15. SCI Receive Data Buffer (SCIRXBUF) Register Field Descriptions

Bit	Field	Type	Reset	Description
15	SCIFFFE	R	0h	SCIFFFE. SCI FIFO Framing error flag bit (applicable only if the FIFO is enabled) Reset type: SYSRSn 0h (R/W) = No frame error occurred while receiving the character, in bits 7-0. This bit is associated with the character on the top of the FIFO. 1h (R/W) = A frame error occurred while receiving the character in bits 7-0. This bit is associated with the character on the top of the FIFO.
14	SCIFFPE	R	0h	SCIFFPE. SCI FIFO parity error flag bit (applicable only if the FIFO is enabled) Reset type: SYSRSn 0h (R/W) = No parity error occurred while receiving the character, in bits 7-0. This bit is associated with the character on the top of the FIFO. 1h (R/W) = A parity error occurred while receiving the character in bits 7-0. This bit is associated with the character on the top of the FIFO.
13-8	RESERVED	R	0h	Reserved
7-0	SAR	R	0h	Receive Character bits Reset type: SYSRSn

12.14.2.9 SCI Transmit Data Buffer (SCITXBUF) Register (Offset = 9h) [reset = 0h]

Data bits to be transmitted are written to SCITXBUF. These bits must be right-justified because the leftmost bits are ignored for characters less than eight bits long. The transfer of data from this register to the TXSHF transmitter shift register sets the TXRDY flag (SCICTL2.7), indicating that SCITXBUF is ready to receive another set of data. If bit TXINTENA (SCICTL2.0) is set, this data transfer also causes an interrupt.

Figure 12-19. SCI Transmit Data Buffer (SCITXBUF) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
TXDT							
R/W-0h							

Table 12-16. SCI Transmit Data Buffer (SCITXBUF) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	TXDT	R/W	0h	Transmit data buffer Reset type: SYSRSn

12.14.2.10 SCI FIFO Transmit (SCIFFTX) Register (Offset = Ah) [reset = A000h]

SCIFFTX controls the transmit FIFO interrupt, FIFO enhancements, and reset for the SCI transmit and receive channels.

Figure 12-20. SCI FIFO Transmit (SCIFFTX) Register

15	14	13	12	11	10	9	8
SCIRST	SCIFFENA	TXFIFORESET	TXFFST				
R/W-1h	R/W-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL				
R-0h	R-0/W1S-0h	R/W-0h	R/W-0h				

Table 12-17. SCI FIFO Transmit (SCIFFTX) Register Field Descriptions

Bit	Field	Type	Reset	Description
15	SCIRST	R/W	1h	SCI Reset 0 Write 0 to reset the SCI transmit and receive channels. SCI FIFO register configuration bits will be left as is. 1 SCI FIFO can resume transmit or receive. SCIRST should be 1 even for Autobaud logic to work. Reset type: SYSRSn
14	SCIFFENA	R/W	0h	SCI FIFO enable Reset type: SYSRSn 0h (R/W) = SCI FIFO enhancements are disabled 1h (R/W) = SCI FIFO enhancements are enabled
13	TXFIFORESET	R/W	1h	Transmit FIFO reset Reset type: SYSRSn 0h (R/W) = Reset the FIFO pointer to zero and hold in reset 1h (R/W) = Re-enable transmit FIFO operation
12-8	TXFFST	R	0h	FIFO status Reset type: SYSRSn 0h (R/W) = Transmit FIFO is empty 1h (R/W) = Transmit FIFO has 1 words 2h (R/W) = Transmit FIFO has 2 words 3h (R/W) = Transmit FIFO has 3 words 4h (R/W) = Transmit FIFO has 4 words
7	TXFFINT	R	0h	Transmit FIFO interrupt Reset type: SYSRSn 0h (R/W) = TXFIFO interrupt has not occurred, read-only bit 1h (R/W) = TXFIFO interrupt has occurred, read-only bit
6	TXFFINTCLR	R-0/W1S	0h	Transmit FIFO clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on TXFIFINT flag bit, Bit reads back a zero 1h (R/W) = Write 1 to clear TXFFINT flag in bit 7
5	TXFFIENA	R/W	0h	Transmit FIFO interrupt enable Reset type: SYSRSn 0h (R/W) = TX FIFO interrupt is disabled 1h (R/W) = TX FIFO interrupt is enabled. This interrupt is triggered whenever the transmit FIFO status (TXFFST) bits match (equal to or less than) the interrupt trigger level bits TXFFIL (bits 4-0).
4-0	TXFFIL	R/W	0h	TXFFIL4-0 Transmit FIFO interrupt level bits. The transmit FIFO generates an interrupt whenever the FIFO status bits (TXFFST4-0) are less than or equal to the FIFO level bits (TXFFIL4-0). The maximum value that can be assigned to these bits to generate an interrupt cannot be more than the depth of the TX FIFO. The default value of these bits after reset is 00000b. Users should set TXFFIL to best fit their application needs by weighing between the CPU overhead to service the ISR and the best possible usage of SCI bus bandwidth. Reset type: SYSRSn

12.14.2.11 SCI FIFO Receive (SCIFFRX) Register (Offset = Bh) [reset = 201Fh]

SCIFFTX controls the receive FIFO interrupt, receive FIFO reset, and status of the receive FIFO overflow.

Figure 12-21. SCI FIFO Receive (SCIFFRX) Register

15	14	13	12	11	10	9	8
RXFFOVF	RXFFOVRCLR	RXFIFORESET	RXFFST				
R-0h	R-0/W1S-0h	R/W-1h	R-0h				
7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL				
R-0h	W-0h	R/W-0h	R/W-1Fh				

Table 12-18. SCI FIFO Receive (SCIFFRX) Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RXFFOVF	R	0h	Receive FIFO overflow. This will function as flag, but cannot generate interrupt by itself. This condition will occur while receive interrupt is active. Receive interrupts should service this flag condition. Reset type: SYSRSn 0h (R/W) = Receive FIFO has not overflowed, read-only bit 1h (R/W) = Receive FIFO has overflowed, read-only bit. More than 16 words have been received in to the FIFO, and the first received word is lost
14	RXFFOVRCLR	R-0/W1S	0h	RXFFOVF clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFFOVF flag bit, Bit reads back a zero 1h (R/W) = Write 1 to clear RXFFOVF flag in bit 15
13	RXFIFORESET	R/W	1h	Receive FIFO reset Reset type: SYSRSn 0h (R/W) = Write 0 to reset the FIFO pointer to zero, and hold in reset. 1h (R/W) = Re-enable receive FIFO operation
12-8	RXFFST	R	0h	FIFO status Reset type: SYSRSn 0h (R/W) = Receive FIFO is empty 1h (R/W) = Receive FIFO has 1 words 2h (R/W) = Receive FIFO has 2 words 3h (R/W) = Receive FIFO has 3 words 4h (R/W) = Receive FIFO has 4 words
7	RXFFINT	R	0h	Receive FIFO interrupt Reset type: SYSRSn 0h (R/W) = RXFIFO interrupt has not occurred, read-only bit 1h (R/W) = RXFIFO interrupt has occurred, read-only bit
6	RXFFINTCLR	W	0h	Receive FIFO interrupt clear Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on RXFIFINT flag bit. Bit reads back a zero. 1h (R/W) = Write 1 to clear RXFFINT flag in bit 7

Table 12-18. SCI FIFO Receive (SCIFFRX) Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
5	RXFFIENA	R/W	0h	Receive FIFO interrupt enable Reset type: SYSRSn 0h (R/W) = RX FIFO interrupt is disabled 1h (R/W) = RX FIFO interrupt is enabled. This interrupt is triggered whenever the receive FIFO status (RXFFST) bits match (equal to or greater than) the interrupt trigger level bits RXFFIL (bits 4-0).
4-0	RXFFIL	R/W	1Fh	Receive FIFO interrupt level bits The receive FIFO generates an interrupt whenever the FIFO status bits (RXFFST4-0) are greater than or equal to the FIFO level bits (RXFFIL4-0). The maximum value that can be assigned to these bits to generate an interrupt cannot be more than the depth of the RX FIFO. The default value of these bits after reset is 11111b. Users should set RXFFIL to best fit their application needs by weighing between the CPU overhead to service the ISR and the best possible usage of received SCI data. Reset type: SYSRSn

12.14.2.12 SCI FIFO Control (SCIFFCT) Register (Offset = Ch) [reset = 0h]

SCIFFCT contains the status of auto-baud detect, clears the auto-baud flag, and calibrate for A-detect bit.

Figure 12-22. SCI FIFO Control (SCIFFCT) Register

15	14	13	12	11	10	9	8
ABD	ABDCLR	CDC	RESERVED				
R-0h	W-0h	R/W-0h	R-0h				
7	6	5	4	3	2	1	0
FFTXDLY							
R/W-0h							

Table 12-19. SCI FIFO Control (SCIFFCT) Register Field Descriptions

Bit	Field	Type	Reset	Description
15	ABD	R	0h	Auto-baud detect (ABD) bit Reset type: SYSRSn 0h (R/W) = Auto-baud detection is not complete. "A","a" character has not been received successfully. 1h (R/W) = Auto-baud hardware has detected "A" or "a" character on the SCI receive register. Auto-detect is complete.
14	ABDCLR	W	0h	ABD-clear bit Reset type: SYSRSn 0h (R/W) = Write 0 has no effect on ABD flag bit. Bit reads back a zero. 1h (R/W) = Write 1 to clear ABD flag in bit 15.
13	CDC	R/W	0h	CDC calibrate A-detect bit Reset type: SYSRSn 0h (R/W) = Disables auto-baud alignment 1h (R/W) = Enables auto-baud alignment
12-8	RESERVED	R	0h	Reserved
7-0	FFTXDLY	R/W	0h	FIFO transfer delay. These bits define the delay between every transfer from FIFO transmit buffer to transmit shift register. The delay is defined in the number of SCI serial baud clock cycles. The 8 bit register could define a minimum delay of 0 baud clock cycles and a maximum of 256 baud clock cycles. In FIFO mode, the buffer (TXBUF) between the shift register and the FIFO should be filled only after the shift register has completed shifting of the last bit. This is required to pass on the delay between transfers to the data stream. In FIFO mode, TXBUF should not be treated as one additional level of buffer. The delayed transmit feature will help to create an auto-flow scheme without RTS/CTS controls as in standard UARTS. When SCI is configured for one stop-bit, delay introduced by FFTXDLY between one frame and the next frame is equal to number of baud clock cycles that FFTXDLY is set to. When SCI is configured for two stop-bits, delay introduced by FFTXDLY between one frame and the next frame is equal to number of baud clock cycles that FFTXDLY is set to minus 1. Reset type: SYSRSn

12.14.2.13 SCI Priority Control (SCIPRI) Register (Offset = Fh) [reset = 0h]

SCIPRI determines what happens when an emulation suspend event occurs.

Figure 12-23. SCI Priority Control (SCIPRI) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED			FREESOFT		RESERVED		
R-0h			R/W-0h		R-0h		

Table 12-20. SCI Priority Control (SCIPRI) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-5	RESERVED	R	0h	Reserved
4-3	FREESOFT	R/W	0h	These bits determine what occurs when an emulation suspend event occurs (for example, when the debugger hits a break point). The peripheral can continue whatever it is doing (free-run mode), or if in stop mode, it can either stop immediately or stop when the current operation (the current receive/transmit sequence) is complete. Reset type: SYSRSn 0h (R/W) = Immediate stop on suspend 1h (R/W) = Complete current receive/transmit sequence before stopping 2h (R/W) = Free run 3h (R/W) = Free run
2-0	RESERVED	R	0h	Reserved

Chapter 13

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 chapter assumes the reader is familiar with the I2C bus specification.

Note

A unit of data transmitted or received by the I2C module can have fewer than 8 bits; however, for convenience, a unit of data is called a data byte throughout this document. The number of bits in a data byte is selectable by way of the BC bits of the mode register, I2CMDR.

13.1 Introduction

The I2C module supports any slave or master I2C-compatible device. Figure 13-1 shows an example of multiple I2C modules connected for a two-way transfer from one device to other devices.

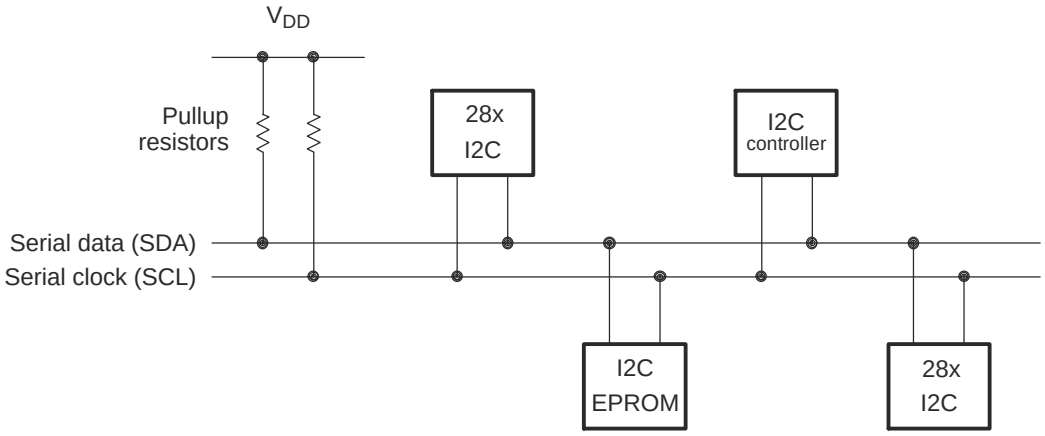


Figure 13-1. Multiple I2C Modules Connected

13.1.1 I2C Related Collateral

Foundational Materials

- I2C Hardware Overview (Video)
- I2C Protocol Overview (Video)
- *Understanding the I2C Bus Application Report*

Getting Started Materials

- *Configuring the TMS320F280x DSP as an I2C Processor Application Report*
- I2C Buffers Overview (Video)
- *I2C Dynamic Addressing Application Report*
- I2C translators overview (Video)
- *Why, When, and How to use I2C Buffers Application Report*

Expert Materials

- *I2C Bus Pull-Up Resistor Calculation Application Report*
- *Maximum Clock Frequency of I2C Bus Using Repeaters Application Report*

13.1.2 Features

The I2C module has the following features:

- Compliance with the NXP Semiconductors I2C bus specification (version 2.1):
  - Support for 8-bit format transfers
  - 7-bit and 10-bit addressing modes
  - General call
  - START byte mode
  - Support for multiple master-transmitters and slave-receivers
  - Support for multiple slave-transmitters and master-receivers
  - Combined master transmit/receive and receive/transmit mode
  - Data transfer rate from 10 kbps up to 400 kbps (Fast-mode)
- Receive FIFO and Transmitter FIFO (4-deep x 8-bit FIFO)
- Supports two ePIE interrupts:
  - I2Cx Interrupt – Any of the following events can be configured to generate an I2Cx interrupt:
    - Transmit-data ready
    - Receive-data ready
    - Register-access ready
    - No-acknowledgment received
    - Arbitration lost
    - Stop condition detected
    - Addressed as slave
  - I2Cx_FIFO interrupts:
    - Transmit FIFO interrupt
    - Receive FIFO interrupt
- Module enable and disable capability
- Free data format mode

13.1.3 Features Not Supported

The I2C module does not support:

- High-speed mode (Hs-mode)
- CBUS-compatibility mode

13.1.4 Functional Overview

Each device connected to an I2C bus is recognized by a unique address. Each device can operate as either a transmitter or a receiver, depending on the function of the device. A device connected to the I2C bus can also be considered as the master or the slave when performing data transfers. A master device is the device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. During this transfer, any device addressed by this master is considered a slave. The I2C module supports the multi-master mode, in which one or more devices capable of controlling an I2C bus can be connected to the same I2C bus.

For data communication, the I2C module has a serial data pin (SDA) and a serial clock pin (SCL), as shown in Figure 13-2. These two pins carry information between the C28x device and other devices connected to the I2C bus. The SDA and SCL pins both are bidirectional. They each must be connected to a positive supply voltage using a pull-up resistor. When the bus is free, both pins are high. The driver of these two pins has an open-drain configuration to perform the required wired-AND function.

There are two major transfer techniques:

- Standard Mode: Send exactly n data values, where n is a value you program in an I2C module register. See the I2CCNT register in Section 13.6 for more information.
- Repeat Mode: Keep sending data values until you use software to initiate a STOP condition or a new START condition. See the I2CMDR register in Section 13.6 for RM bit information.

The I2C module consists of the following primary blocks:

- A serial interface: one data pin (SDA) and one clock pin (SCL)
- Data registers and FIFOs to temporarily hold receive data and transmit data traveling between the SDA pin and the CPU
- Control and status registers
- A peripheral bus interface to enable the CPU to access the I2C module registers and FIFOs.
- A clock synchronizer to synchronize the I2C input clock (from the device clock generator) and the clock on the SCL pin, and to synchronize data transfers with masters of different clock speeds
- A prescaler to divide down the input clock that is driven to the I2C module
- A noise filter on each of the two pins, SDA and SCL
- An arbitrator to handle arbitration between the I2C module (when it is a master) and another master
- Interrupt generation logic, so that an interrupt can be sent to the CPU
- FIFO interrupt generation logic, so that FIFO access can be synchronized to data reception and data transmission in the I2C module

Figure 13-2 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.

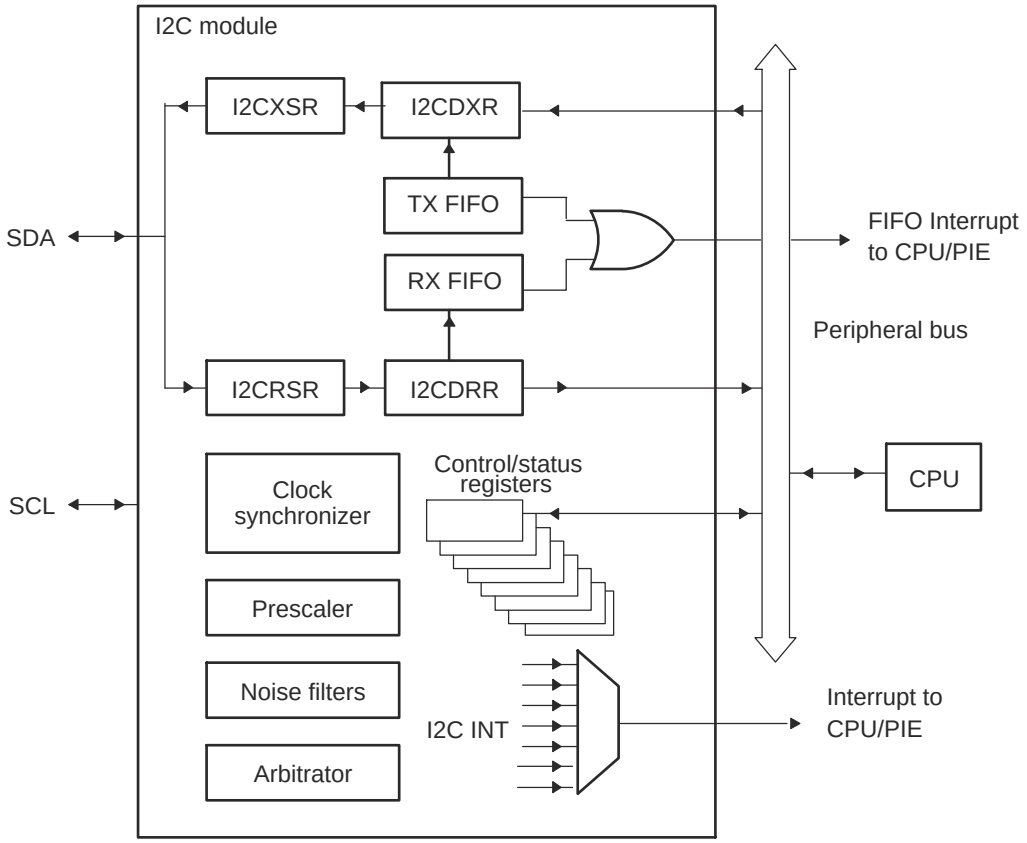


Figure 13-2. I2C Module Conceptual Block Diagram

13.1.5 Clock Generation

The I2C module clock determines the frequency at which the I2C module operates. A programmable prescaler in the I2C module divides down the SYSCLK to produce the I2C module clock and this I2C module clock is divided further to produce the I2C master clock on the SCL pin. Figure 13-3 shows the clock generation diagram for I2C module.

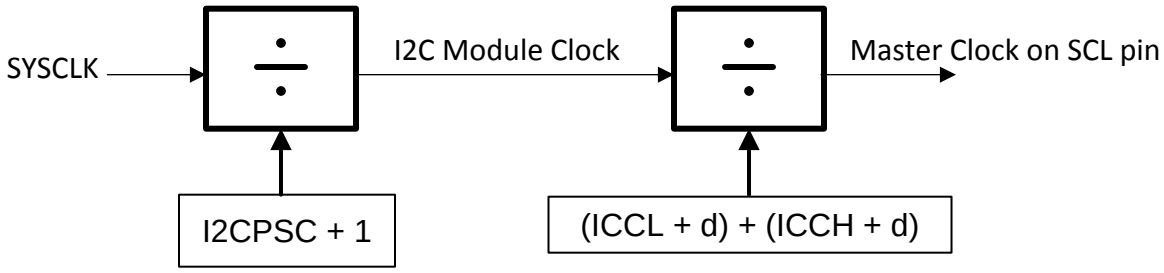


Figure 13-3. Clocking Diagram for the I2C Module

Note

To meet all of the I2C protocol timing specifications, the I2C module clock must be between 7-12 MHz.

To specify the divide-down value, initialize the IPSC field of the prescaler register, I2CPSC. The resulting frequency is:

$$\text{I2C Module Clock (Fmod)} = \frac{\text{SYSCLK}}{(\text{I2CPSC} + 1)} \quad (8)$$

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 13-3, 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 13.1.6 for the master clock frequency equation.

13.1.6 I2C Clock Divider Registers (I2CCLKL and I2CCLKH)

As explained in Section 13.1.5, when the I2C module is a master, the I2C module clock is divided down further to use as the master clock on the SCL pin. As shown in Figure 13-4, 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.

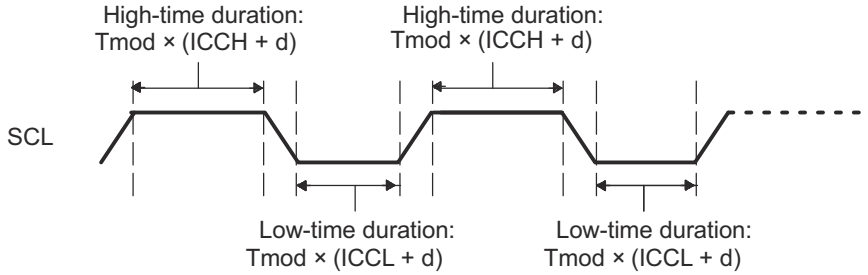


Figure 13-4. Roles of the Clock Divide-Down Values (ICCL and ICCH)

13.1.6.1 Formula for the Master Clock Period

The master clock period (Tmst) is a multiple of the period of the I2C Module Clock (Tmod):

$$\text{Master Clock period (Tmst)} = \frac{[\,(\text{ICCH} + \text{d}) + (\text{ICCL} + \text{d})\,]}{\text{I2C Module Clock (Fmod)}} \quad (9)$$

where d depends on the divide-down value IPSC, as shown in Table 13-1. IPSC is described in the I2CPSC register.

Table 13-1. Dependency of Delay d on the Divide-Down Value IPSC

IPSC	d
0	7
1	6
Greater than 1	5

13.2 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.

13.3 I2C Module Operational Details

This section provides an overview of the I2C bus protocol and how it is implemented.

13.3.1 Input and Output Voltage Levels

One clock pulse is generated by the master device for each data bit transferred. Due to a variety of different technology devices that can be connected to the I2C bus, the levels of logic 0 (low) and logic 1 (high) are not fixed and depend on the associated level of V_{DD}. For details, see your device-specific data sheet.

13.3.2 Data Validity

The data on SDA must be stable during the high period of the clock (see Figure 13-5). The high or low state of the data line, SDA, should change only when the clock signal on SCL is low.

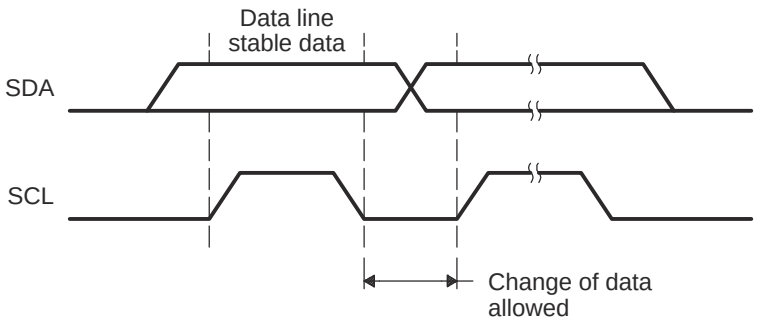


Figure 13-5. Bit Transfer on the I2C bus

13.3.3 Operating Modes

The I2C module has four basic operating modes to support data transfers as a master and as a slave. See Table 13-2 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 13-2. Operating Modes of the I2C Module

Operating Mode	Description
Slave-receiver mode	The I2C module is a slave and receives data from a master. All slaves begin in this mode. In this mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master. As a slave, the I2C module does not generate the clock signal, but it can hold SCL low while the intervention of the device is required (RSFULL = 1 in I2CSTR) after a byte has been received. See Section 13.3.7 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 13.3.7 for more details.
Master-receiver mode	The I2C module is a master and receives data from a slave. This mode can be entered only from the master-transmitter mode; the I2C module must first transmit a command to the slave. When you are using any of the 7-bit/10-bit addressing formats, the I2C module enters its master-receiver mode after transmitting the slave address byte and R/ $\overline{W}$ = 1. Serial data bits on SDA are shifted into the I2C module with the clock pulses generated by the I2C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the device is required (RSFULL = 1 in I2CSTR) after a byte has been received.
Master-transmitter mode	The I2C module is a master and transmits control information and data to a slave. All masters begin in this mode. In this mode, data assembled in any of the 7-bit/10-bit addressing formats is shifted out on SDA. The bit shifting is synchronized with the clock pulses generated by the I2C module on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the device is required (XSMT = 0 in I2CSTR) after a byte has been transmitted.

To summarize, SCL will be held low in the following conditions:

- When an overrun condition is detected (RSFULL = 1), in Slave-receiver mode.
- When an underflow condition is detected (XSMT = 0), in Slave-transmitter mode.

I2C slave nodes have to accept and provide data when the I2C master node requests it.

- To release SCL in slave-receiver mode, read data from I2CDRR.
- To release SCL in slave-transmitter mode, write data to I2CDXR.
- To force a release without handling the data, reset the module using the I2CMDR.IRS bit.

Table 13-3. 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;

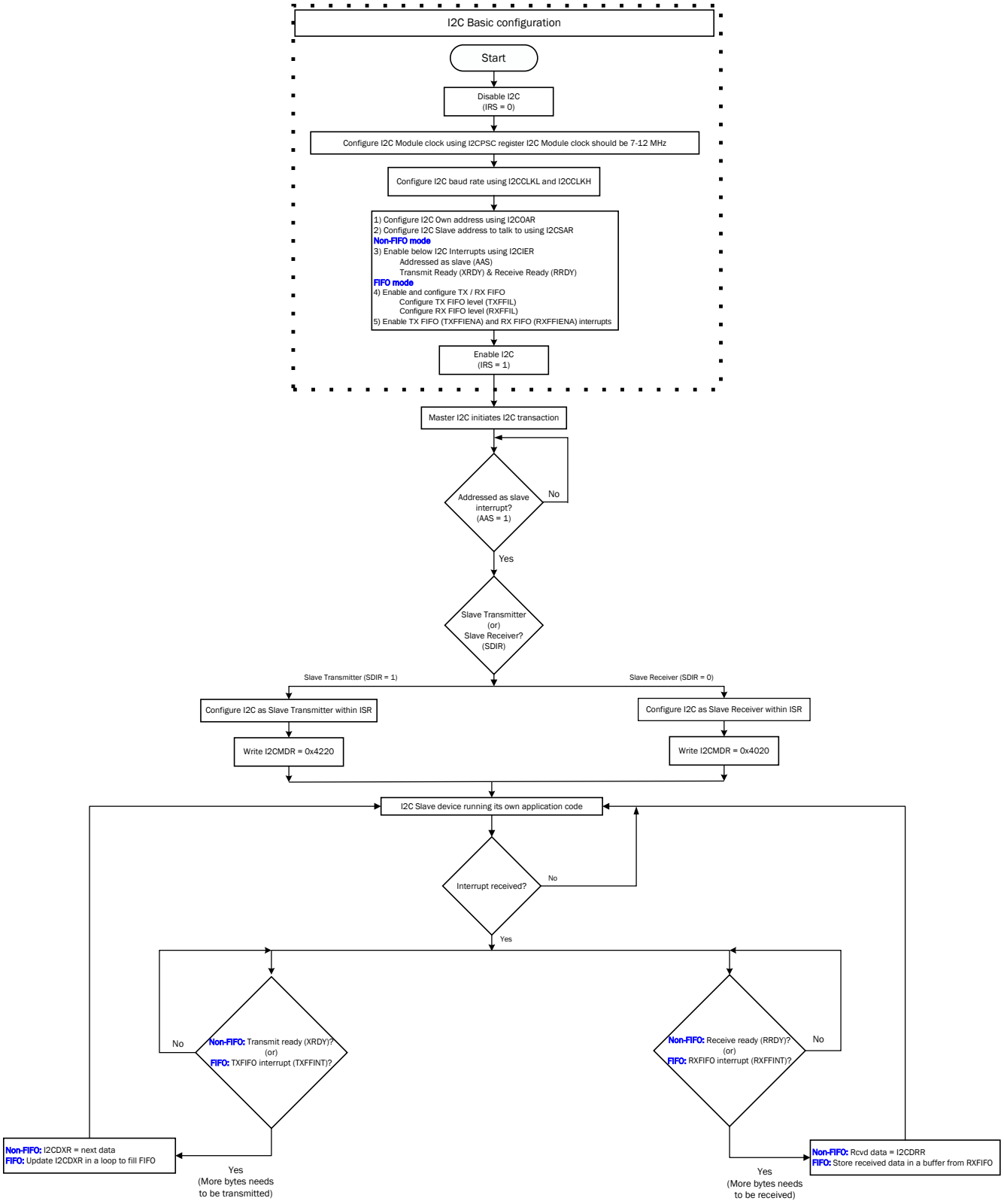


Figure 13-6. I2C Slave TX / RX Flowchart

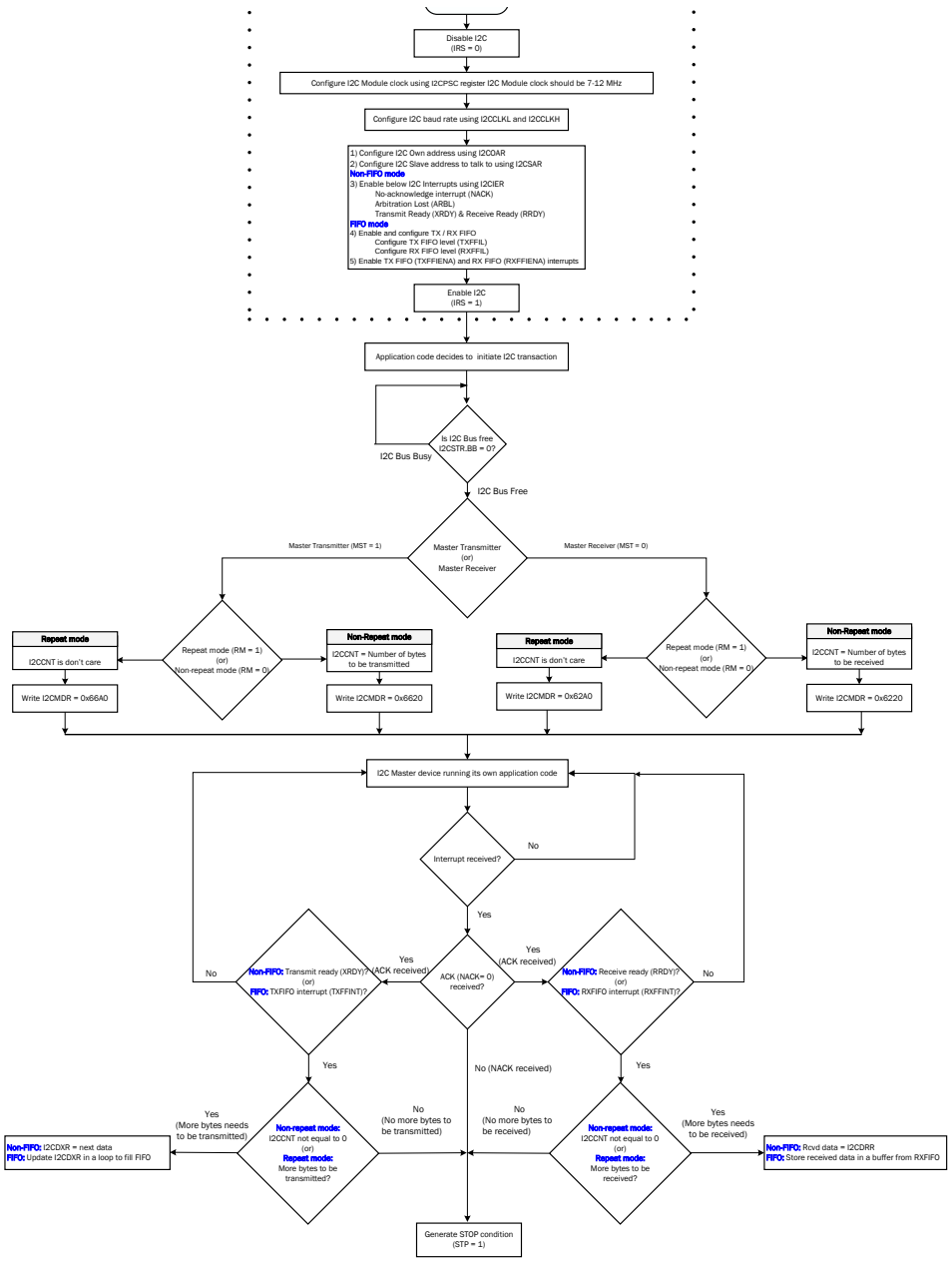


Figure 13-7. I2C Master TX / RX Flowchart

13.3.4 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 13-8:

- The START condition is defined as a high-to-low transition on the SDA line while SCL is high. A master drives this condition to indicate the start of a data transfer.
- The STOP condition is defined as a low-to-high transition on the SDA line while SCL is high. A master drives this condition to indicate the end of a data transfer.

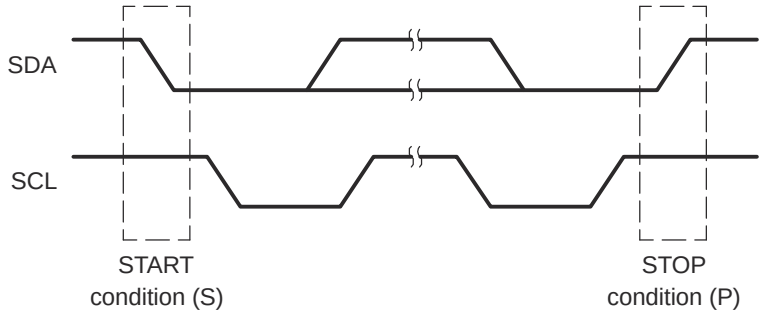


Figure 13-8. I2C Module START and STOP Conditions

After a START condition and before a subsequent STOP condition, the I2C bus is considered busy, and the bus busy (BB) bit of I2CSTR is 1. Between a STOP condition and the next START condition, the bus is considered free, and BB is 0.

For the I2C module to start a data transfer with a START condition, the master mode bit (MST) and the START condition bit (STT) in I2CMDR must both be 1. For the I2C module to end a data transfer with a STOP condition, the STOP condition bit (STP) must be set to 1. When the BB bit is set to 1 and the STT bit is set to 1, a repeated START condition is generated. For a description of I2CMDR and its bits (including MST, STT, and STP), see Section 13.6.

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.

13.3.5 Non-repeat Mode versus Repeat Mode

Non-repeat mode:

- When I2CMDR.RM = 0, I2C module is configured in non-repeat mode.
- I2CCNT register determines the number of bytes to be transmitted (or) received.
- If STP = 0 in I2CMDR, the ARDY bit is set when the internal data counter counts down to 0.
- If STP = 1, ARDY bit doesn't get set and I2C module generates a STOP condition when the internal data counter counts down to 0.

Note

In non-repeat mode (RM = 0), if I2CCNT is set to 0, I2C state machine expects to transmit (or) receive 65536 bytes and not 0 bytes.

Repeat mode:

- When I2CMDR.RM = 1, I2C module is configured in repeat mode.
- I2CCNT register contents don't determine the number of bytes to be transmitted (or) received.
- Number of bytes to be transmitted (or) received can be controlled by software.
- ARDY bit gets set at end of transmission and reception of each byte.

Note

Once you start I2C transaction in non-repeat mode (or) repeat mode, you cannot switch into another mode until the I2C transaction is completed with a STOP condition.

13.3.6 Serial Data Formats

Figure 13-9 shows an example of a data transfer on the I2C bus. The I2C module supports 1 to 8-bit data values. In Figure 13-9, 8-bit data is transferred. Each bit put on the SDA line equates to 1 pulse on the SCL line, and the values are always transferred with the most significant bit (MSB) first. The number of data values that can be transmitted or received is unrestricted. The serial data format used in Figure 13-9 is the 7-bit addressing format. The I2C module supports the formats shown in Figure 13-10 through Figure 13-12 and described in the paragraphs that follow the figures.

Note

In Figure 13-9 through Figure 13-12, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

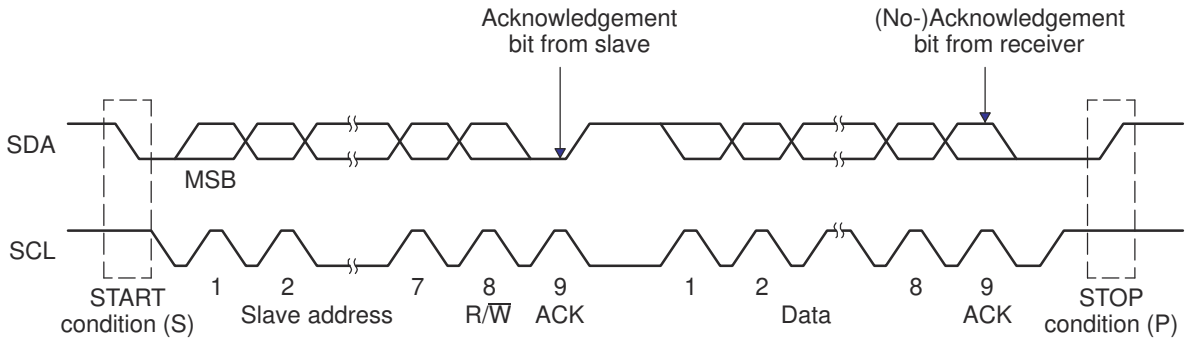


Figure 13-9. I2C Module Data Transfer (7-Bit Addressing with 8-bit Data Configuration Shown)

13.3.6.1 7-Bit Addressing Format

The 7-bit addressing format is the default format after reset. Disabling expanded address (I2CMDR.XA = 0) and free data format (I2CMDR.FDF = 0) enables 7-bit addressing format.

In this format (see Figure 13-10), 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)

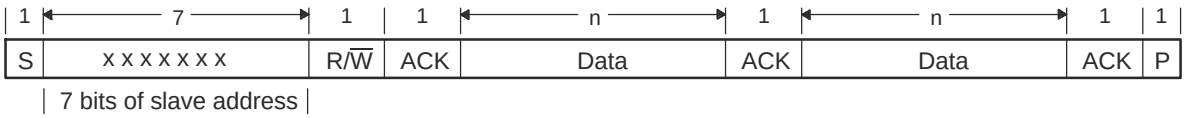


Figure 13-10. I2C Module 7-Bit Addressing Format (FDF = 0, XA = 0 in I2CMDR)

An extra clock cycle dedicated for acknowledgment (ACK) is inserted after each byte. If the ACK bit is inserted by the slave after the first byte from the master, it is followed by n bits of data from the transmitter (master or slave, depending on the R/ $\overline{W}$ 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.

13.3.6.2 10-Bit Addressing Format

The 10-bit addressing format can be enabled by setting expanded address (I2CMDR.XA = 1) and disabling free data format (I2CMDR.FDF = 0).

The 10-bit addressing format (see Figure 13-11) is similar to the 7-bit addressing format, but the master sends the slave address in two separate byte transfers. The first byte consists of 11110b, the two MSBs of the 10-bit slave address, and R/ $\overline{W}$. 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.

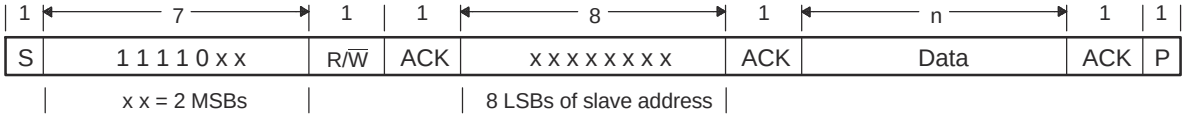


Figure 13-11. I2C Module 10-Bit Addressing Format (FDF = 0, XA = 1 in I2CMDR)

13.3.6.3 Free Data Format

The free data format can be enabled by setting I2CMDR. FDF = 1.

In this format (see Figure 13-12), the first byte after a START condition (S) is a data byte. An ACK bit is inserted after each data byte, which can be from 1 to 8 bits, depending on the BC field of I2CMDR. No address or data-direction bit is sent. Therefore, the transmitter and the receiver must both support the free data format, and the direction of the data must be constant throughout the transfer.

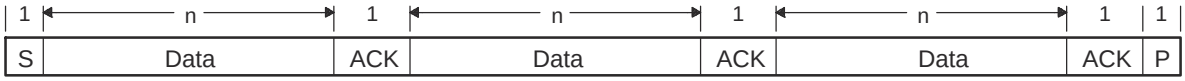


Figure 13-12. I2C Module Free Data Format (FDF = 1 in I2CMDR)

Note

The free data format is not supported in the digital loopback mode (I2CMDR.DLB = 1).

Table 13-4. 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.

13.3.6.4 Using a Repeated START Condition

I2C master can communicate with multiple slave addresses without having to give up control of the I2C bus by driving a STOP condition. This can be achieved by driving another START condition at the end of each data type. The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, and free data formats. Figure 13-13 shows a repeated START condition in the 7-bit addressing format.

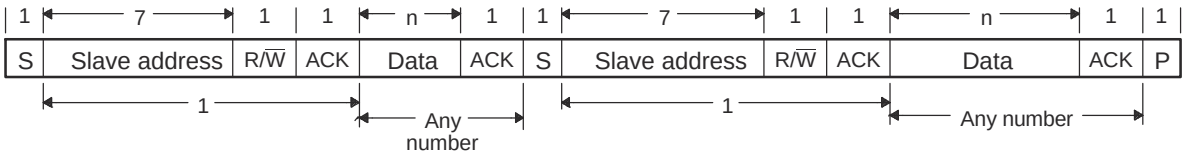


Figure 13-13. Repeated START Condition (in This Case, 7-Bit Addressing Format)

Note

In Figure 13-13, n = the number of data bits (from 1 to 8) specified by the bit count (BC) field of I2CMDR.

13.3.7 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 13-14 illustrates the clock synchronization. The wired-AND property of SCL means that a device that first generates a low period on SCL overrules the other devices. At this high-to-low transition, the clock generators of the other devices are forced to start their own low period. The SCL is held low by the device with the longest low period. The other devices that finish their low periods must wait for SCL to be released, before starting their high periods. A synchronized signal on SCL is obtained, where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

If a device pulls down the clock line for a longer time, the result is that all clock generators must enter the wait state. In this way, a slave slows down a fast master and the slow device creates enough time to store a received byte or to prepare a byte to be transmitted.

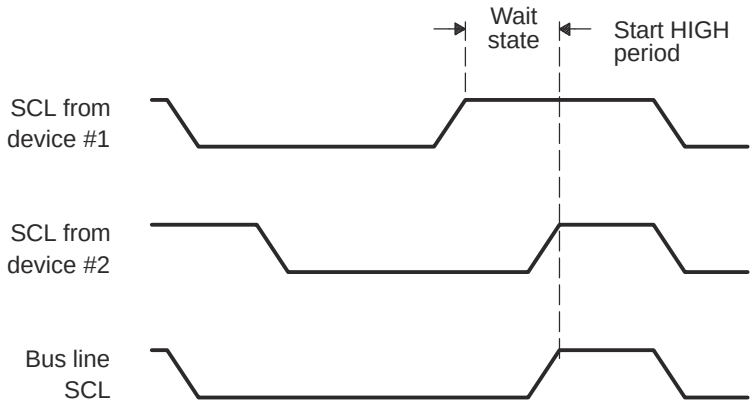


Figure 13-14. Synchronization of Two I2C Clock Generators During Arbitration

13.3.8 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 13-15 illustrates the arbitration procedure between two devices. The first master-transmitter that releases the SDA line high is overruled by another master-transmitter that drives the SDA low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If the I2C module is the losing master, it switches to the slave-receiver mode, sets the arbitration lost (ARBL) flag, and generates the arbitration-lost interrupt request.

If during a serial transfer the arbitration procedure is still in progress when a repeated START condition or a STOP condition is transmitted to SDA, the master-transmitters involved must send the repeated START condition or the STOP condition at the same position in the format frame. Arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

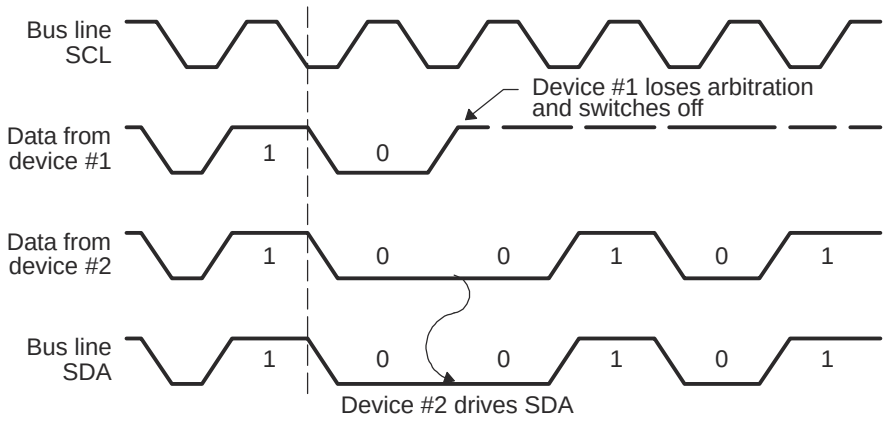


Figure 13-15. Arbitration Procedure Between Two Master-Transmitters

13.3.9 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 13-16 shows the signal routing in digital loopback mode.

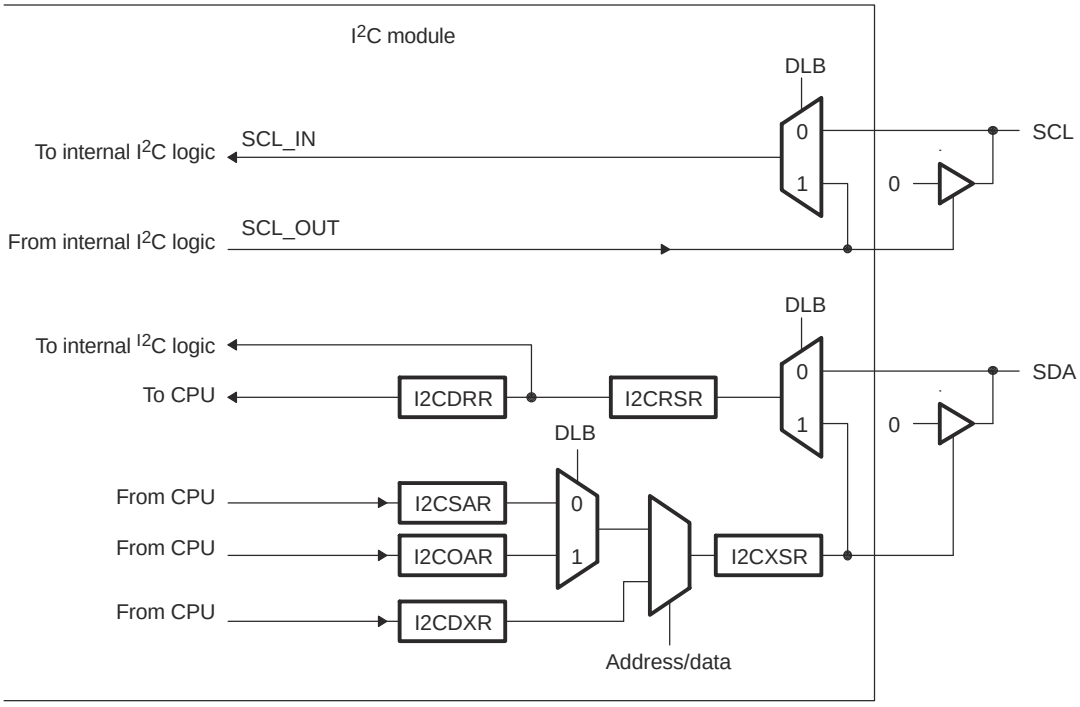


Figure 13-16. Pin Diagram Showing the Effects of the Digital Loopback Mode (DLB) Bit

Note

The free data format (I2CMDR.FDF = 1) is not supported in digital loopback mode.

13.3.10 NACK Bit Generation

When the I2C module is a receiver (master or slave), it can acknowledge or ignore bits sent by the transmitter. To ignore any new bits, the I2C module must send a no-acknowledge (NACK) bit during the acknowledge cycle on the bus. Table 13-5 summarizes the various ways you can allow the I2C module to send a NACK bit.

Table 13-5. 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

13.4 Interrupt Requests Generated by the I2C Module

Each I2C module can generate two CPU interrupts.

1. Basic I2C interrupt: Possible basic I2C interrupt sources that can trigger this interrupt are described in Section 13.4.1.
2. I2C FIFO interrupt: Possible I2C FIFO interrupt sources that can trigger this interrupt are described in Section 13.4.2

13.4.1 Basic I2C Interrupt Requests

The I2C module generates the interrupt requests described in Table 13-6. As shown in Figure 13-17, all requests are multiplexed through an arbiter to a single I2C interrupt request to the CPU. Each interrupt request has a flag bit in the status register (I2CSTR) and an enable bit in the interrupt enable register (I2CIER). When one of the specified events occurs, its flag bit is set. If the corresponding enable bit is 0, the interrupt request is blocked. If the enable bit is 1, the request is forwarded to the CPU as an I2C interrupt.

The I2C interrupt is one of the maskable interrupts of the CPU. As with any maskable interrupt request, if it is properly enabled in the CPU, the CPU executes the corresponding interrupt service routine (I2CINT1A_ISR). The I2CINT1A_ISR for the I2C interrupt can determine the interrupt source by reading the interrupt source register, I2CISRC. Then the I2CINT1A_ISR can branch to the appropriate subroutine.

After the CPU reads I2CISRC, the following events occur:

1. The flag for the source interrupt is cleared in I2CSTR. Exception: The ARDY, RRDY, and XRDY bits in I2CSTR are not cleared when I2CISRC is read. To clear one of these bits, write a 1 to it.
2. The arbiter determines which of the remaining interrupt requests has the highest priority, writes the code for that interrupt to I2CISRC, and forwards the interrupt request to the CPU.

Table 13-6. 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.

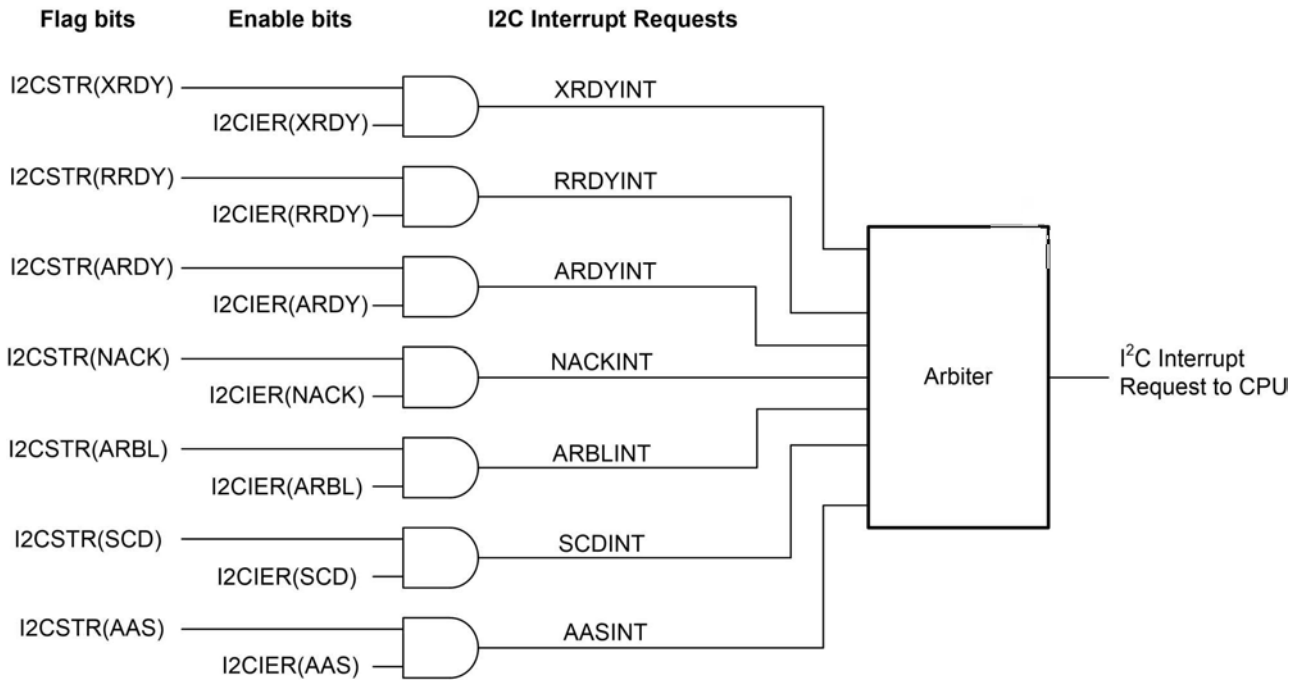


Figure 13-17. Enable Paths of the I2C Interrupt Requests

The priorities of the basic I2C interrupt requests are listed in order of highest priority to lowest priority:

1. ARBLINT
2. NACKINT
3. ARDYINT
4. RRDYINT
5. XRDYINT
6. SCDINT
7. AASINT

The I2C module has a backwards compatibility bit (BC) in the I2CEMDR register. The timing diagram in demonstrates the effect the backwards compatibility bit has on I2C module registers and interrupts when configured as a slave-transmitter.

13.4.2 I2C FIFO Interrupts

In addition to the seven basic I2C interrupts, the transmit and receive FIFOs each contain the ability to generate an interrupt (I2CINT2A). The transmit FIFO can be configured to generate an interrupt after transmitting a defined number of bytes, up to 4. The receive FIFO can be configured to generate an interrupt after receiving a defined number of bytes, up to 4. These two interrupt sources are ORed together into a single maskable CPU interrupt. Figure 13-18 shows the structure of I2C FIFO interrupt. The interrupt service routine can then read the FIFO interrupt status flags to determine from which source the interrupt came. See the I2C transmit FIFO register (I2CFFTX) and the I2C receive FIFO register (I2CFFRX) descriptions.

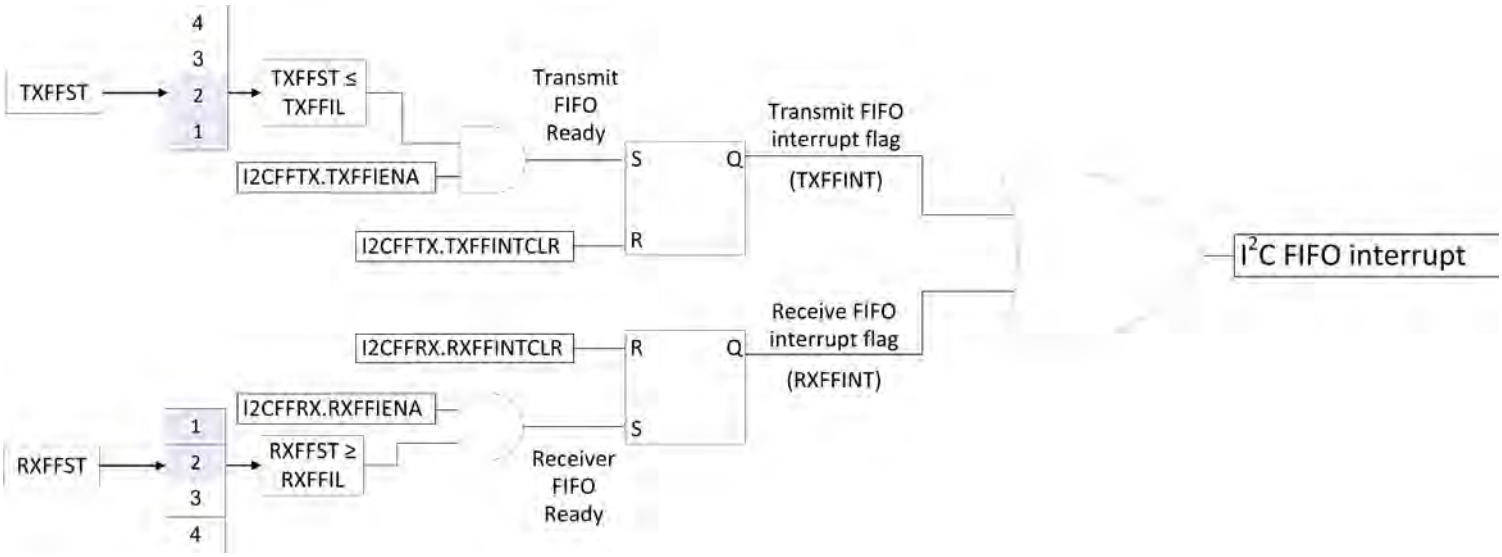


Figure 13-18. I2C FIFO Interrupt

13.5 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.

13.6 I2C Registers

This section describes the C28x I2C Module Registers.

13.6.1 I2C Base Address Table (C28)

Table 13-7. I2C Base Address Table (C28)

Bit Field Name		Base Address
Instance	Structure	
I2caRegs	I2C_REGS	0x0000_7900

13.6.2 I2C_REGS Registers

Table 13-8 lists the I2C_REGS registers. All register offset addresses not listed in Table 13-8 should be considered as reserved locations and the register contents should not be modified.

Table 13-8. I2C_REGS Registers

Offset	Acronym	Register Name	Write Protection	Section
0h	I2COAR	I2C Own Address		Section 13.6.2.1
1h	I2CIER	I2C Interrupt Enable		Section 13.6.2.2
2h	I2CSTR	I2C Status		Section 13.6.2.3
3h	I2CCLKL	I2C Clock Low-time Divider		Section 13.6.2.4
4h	I2CCLKH	I2C Clock High-time Divider		Section 13.6.2.5
5h	I2CCNT	I2C Data Count		Section 13.6.2.6
6h	I2CDRR	I2C Data Receive		Section 13.6.2.7
7h	I2CSAR	I2C Slave Address		Section 13.6.2.8
8h	I2CDXR	I2C Data Transmit		Section 13.6.2.9
9h	I2CMDR	I2C Mode		Section 13.6.2.10
Ah	I2CISRC	I2C Interrupt Source		Section 13.6.2.11
Bh	I2CEMDR	I2C Extended Mode		Section 13.6.2.12
Ch	I2CPSC	I2C Prescaler		Section 13.6.2.13
20h	I2CFFTX	I2C FIFO Transmit		Section 13.6.2.14
21h	I2CFFRX	I2C FIFO Receive		Section 13.6.2.15

Complex bit access types are encoded to fit into small table cells. Table 13-9 shows the codes that are used for access types in this section.

Table 13-9. 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.

13.6.2.1 I2C Own Address Register (I2COAR) (Offset = 0h) [reset = 0h]

The I2C own address register (I2COAR) is a 16-bit register. The I2C module uses this register to specify its own slave address, which distinguishes it from other slaves connected to the I2C-bus. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6-0 are used
write 0s to bits 9-7.

Figure 13-19. I2C Own Address Register (I2COAR)

15	14	13	12	11	10	9	8
RESERVED						OAR	
R-0h						R/W-0h	
7	6	5	4	3	2	1	0
OAR							
R/W-0h							

Table 13-10. I2C Own Address Register (I2COAR) Field Descriptions

Bit	Field	Type	Reset	Description
15-10	RESERVED	R	0h	Reserved
9-0	OAR	R/W	0h	In 7-bit addressing mode (XA = 0 in I2CMDR): 00h-7Fh Bits 6-0 provide the 7-bit slave address of the I2C module. Write 0s to bits 9-7. In 10-bit addressing mode (XA = 1 in I2CMDR): 000h-3FFh Bits 9-0 provide the 10-bit slave address of the I2C module. Reset type: SYSRSn

13.6.2.2 I2C Interrupt Enable Register (I2CIER) (Offset = 1h) [reset = 0h]

I2CIER is used by the CPU to individually enable or disable I2C interrupt requests.

Figure 13-20. I2C Interrupt Enable Register (I2CIER)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	AAS	SCD	XRDY	RRDY	ARDY	NACK	ARBL
R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 13-11. I2C Interrupt Enable Register (I2CIER) Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved
6	AAS	R/W	0h	Addressed as slave interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
5	SCD	R/W	0h	Stop condition detected interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
4	XRDY	R/W	0h	Transmit-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
3	RRDY	R/W	0h	Receive-data-ready interrupt enable bit. This bit should not be set when using FIFO mode. Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
2	ARDY	R/W	0h	Register-access-ready interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
1	NACK	R/W	0h	No-acknowledgment interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled
0	ARBL	R/W	0h	Arbitration-lost interrupt enable Reset type: SYSRSn 0h (R/W) = Interrupt request disabled 1h (R/W) = Interrupt request enabled

13.6.2.3 I2C Status Register (I2CSTR) (Offset = 2h) [reset = 410h]

The I2C status register (I2CSTR) is a 16-bit register used to determine which interrupt has occurred and to read status information.

Figure 13-21. I2C Status Register (I2CSTR)

15	14	13	12	11	10	9	8
RESERVED	SDIR	NACKSNT	BB	RSFULL	XSMT	AAS	AD0
R-0h	R/W1C-0h	R/W1C-0h	R-0h	R-0h	R-1h	R-0h	R-0h
7	6	5	4	3	2	1	0
RESERVED		SCD	XRDY	RRDY	ARDY	NACK	ARBL
R/W-0h		R/W1C-0h	R-1h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h

Table 13-12. I2C Status Register (I2CSTR) Field Descriptions

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reserved
14	SDIR	R/W1C	0h	Slave direction bit Reset type: SYSRSn 0h (R/W) = I2C is not addressed as a slave transmitter. SDIR is cleared by one of the following events: - It is manually cleared. To clear this bit, write a 1 to it. - Digital loopback mode is enabled. - A START or STOP condition occurs on the I2C bus. 1h (R/W) = I2C is addressed as a slave transmitter.
13	NACKSNT	R/W1C	0h	NACK sent bit. This bit is used when the I2C module is in the receiver mode. One instance in which NACKSNT is affected is when the NACK mode is used (see the description for NACKMOD in Reset type: SYSRSn 0h (R/W) = NACK not sent. NACKSNT bit is cleared by any one of the following events: - It is manually cleared. To clear this bit, write a 1 to it. - The I2C module is reset (either when 0 is written to the IRS bit of I2CMDR or when the whole device is reset). 1h (R/W) = NACK sent: A no-acknowledge bit was sent during the acknowledge cycle on the I2C-bus.
12	BB	R	0h	Bus busy bit. BB indicates whether the I2C-bus is busy or is free for another data transfer. See the paragraph following the table for more information Reset type: SYSRSn 0h (R/W) = Bus free. BB is cleared by any one of the following events: - The I2C module receives or transmits a STOP bit (bus free). - The I2C module is reset. 1h (R/W) = Bus busy: The I2C module has received or transmitted a START bit on the bus.

Table 13-12. I2C Status Register (I2CSTR) 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. Debug probe 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). Debug probe 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 13-12. I2C Status Register (I2CSTR) 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. Debug probe 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 13-12. I2C Status Register (I2CSTR) 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 master transmitter. NACK indicates whether the I2C module has detected an acknowledge bit (ACK) or a noacknowledge bit (NACK) from the slave receiver. The CPU can poll NACK or use the NACK interrupt request. 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 slave 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. Debug probe 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 master-transmitter. 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. Debug probe 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.

13.6.2.4 I2C Clock Low-time Divider (I2CCLKL) Register (Offset = 3h) [reset = 0h]

I2C Clock low-time divider

Figure 13-22. I2C Clock Low-time Divider (I2CCLKL) Register

15	14	13	12	11	10	9	8
I2CCLKL							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCLKL							
R/W-0h							

Table 13-13. I2C Clock Low-time Divider (I2CCLKL) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	I2CCLKL	R/W	0h	Clock low-time divide-down value. To produce the low time duration of the master clock, the period of the module clock is multiplied by (ICCL + d). d is an adjustment factor based on the prescaler. See the Clock Divider Registers section of the Introduction for details. Note: These bits must be set to a non-zero value for proper I2C clock generation. Reset type: SYSRSn

13.6.2.5 I2C Clock High-time Divider (I2CCLKH) Register (Offset = 4h) [reset = 0h]

I2C Clock high-time divider

Figure 13-23. I2C Clock High-time Divider (I2CCLKH) Register

15	14	13	12	11	10	9	8
I2CCLKH							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCLKH							
R/W-0h							

Table 13-14. I2C Clock High-time Divider (I2CCLKH) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-0	I2CCLKH	R/W	0h	Clock high-time divide-down value. To produce the high time duration of the master clock, the period of the module clock is multiplied by (ICCL + d). d is an adjustment factor based on the prescaler. See the Clock Divider Registers section of the Introduction for details. Note: These bits must be set to a non-zero value for proper I2C clock generation. Reset type: SYSRSn

13.6.2.6 I2C Data Count (I2CCNT) Register (Offset = 5h) [reset = 0h]

I2CCNT is a 16-bit register used to indicate how many data bytes to transfer when the I2C module is configured as a transmitter, or to receive when configured as a master receiver. In the repeat mode (RM = 1), I2CCNT is not used.The value written to I2CCNT is copied to an internal data counter. The internal data counter is decremented by 1 for each byte transferred (I2CCNT remains unchanged). If a STOP condition is requested in the master mode (STP = 1 in I2CMDR), the I2C module terminates the transfer with a STOP condition when the countdown is complete (that is, when the last byte has been transferred).

Figure 13-24. I2C Data Count (I2CCNT) Register

15	14	13	12	11	10	9	8
I2CCNT							
R/W-0h							
7	6	5	4	3	2	1	0
I2CCNT							
R/W-0h							

Table 13-15. I2C Data Count (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. If a STOP condition is specified (STP=1) then I2CCNT will decrease after each byte is sent until it reaches zero, which in turn will generate a STOP condition. 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 0h (R/W) = data count value is 65536 1h (R/W) = data count value is 1 2h (R/W) = data count value is 2 FFFFh (R/W) = data count value is 65535

13.6.2.7 I2C Data Receive Register (I2CDRR) (Offset = 6h) [reset = 0h]

I2CDRR is a 16-bit register used by the CPU to read received data. The I2C module can receive a data byte with 1 to 8 bits. The number of bits is selected with the bit count (BC) bits in 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 13-25. I2C Data Receive Register (I2CDRR)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
DATA							
R-0h							

Table 13-16. I2C Data Receive Register (I2CDRR) Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	DATA	R	0h	Receive data Reset type: SYSRSn

13.6.2.8 I2C Slave Address Register (I2CSAR) (Offset = 7h) [reset = 3FFh]

The I2C slave address register (I2CSAR) is a 16-bit register for storing the next slave address that will be transmitted by the I2C module when it is a master. The SAR field of I2CSAR contains a 7-bit or 10-bit slave address. When the I2C module is not using the free data format (FDF = 0 in I2CMDR), it uses this address to initiate data transfers with a slave, or slaves. When the address is nonzero, the address is for a particular slave. When the address is 0, the address is a general call to all slaves. If the 7-bit addressing mode is selected (XA = 0 in I2CMDR), only bits 6-0 of I2CSAR are used
write 0s to bits 9-7.

Figure 13-26. I2C Slave Address Register (I2CSAR)

15	14	13	12	11	10	9	8
RESERVED						SAR	
R-0h						R/W-3FFh	
7	6	5	4	3	2	1	0
SAR							
R/W-3FFh							

Table 13-17. I2C Slave Address Register (I2CSAR) Field Descriptions

Bit	Field	Type	Reset	Description
15-10	RESERVED	R	0h	Reserved
9-0	SAR	R/W	3FFh	In 7-bit addressing mode (XA = 0 in I2CMDR): 00h-7Fh Bits 6-0 provide the 7-bit slave address that the I2C module transmits when it is in the master-transmitter mode. Write 0s to bits 9-7. In 10-bit addressing mode (XA = 1 in I2CMDR): 000h-3FFh Bits 9-0 provide the 10-bit slave address that the I2C module transmits when it is in the master transmitter mode. Reset type: SYSRSn

13.6.2.9 I2C Data Transmit Register (I2CDXR) (Offset = 8h) [reset = 0h]

The CPU writes transmit data to I2CDXR. This 16-bit register accepts a data byte with 1 to 8 bits. Before writing to I2CDXR, specify how many bits are in a data byte by loading the appropriate value into the bit count (BC) bits of I2CMDR. When writing a data byte with fewer than 8 bits, make sure the value is right-aligned in I2CDXR.After a data byte is written to I2CDXR, the I2C module copies the data byte to the transmit shift register (I2CXSR). The CPU cannot access I2CXSR directly. From I2CXSR, the I2C module shifts the data byte out on the SDA pin, one bit at a time.When in the transmit FIFO mode, the I2CDXR register acts as the transmit FIFO buffer.

Figure 13-27. I2C Data Transmit Register (I2CDXR)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
DATA							
R/W-0h							

Table 13-18. I2C Data Transmit Register (I2CDXR) Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	DATA	R/W	0h	Transmit data Reset type: SYSRSn

13.6.2.10 I2C Mode Register (I2CMDR) (Offset = 9h) [reset = 0h]

The I2C mode register (I2CMDR) is a 16-bit register that contains the control bits of the I2C module.

Figure 13-28. I2C Mode Register (I2CMDR)

15	14	13	12	11	10	9	8
NACKMOD	FREE	STT	RESERVED	STP	MST	TRX	XA
R/W-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
RM	DLB	IRS	STB	FDF	BC		
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h		

Table 13-19. I2C Mode Register (I2CMDR) 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

Table 13-19. I2C Mode Register (I2CMDR) Field Descriptions (continued)

Bit	Field	Type	Reset	Description
12	RESERVED	R	0h	Reserved
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).

Table 13-19. I2C Mode Register (I2CMDR) Field Descriptions (continued)

Bit	Field	Type	Reset	Description
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.
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.

Table 13-19. I2C Mode Register (I2CMDR) Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3	FDF	R/W	0h	Free data format mode bit. Reset type: SYSRSn 0h (R/W) = Free data format mode is disabled. Transfers use the 7-/10-bit addressing format selected by the XA bit. 1h (R/W) = Free data format mode is enabled. Transfers have the free data (no address) format described in Section 9.2.5. The free data format is not supported in the digital loopback mode (DLB=1).
2-0	BC	R/W	0h	Bit count bits. BC defines the number of bits (1 to 8) in the next data byte that is to be received or transmitted by the I2C module. The number of bits selected with BC must match the data size of the other device. Notice that when BC = 000b, a data byte has 8 bits. BC does not affect address bytes, which always have 8 bits. Note: If the bit count is less than 8, receive data is right-justified in I2CDRR(7-0), and the other bits of I2CDRR(7-0) are undefined. Also, transmit data written to I2CDXR must be right-justified Reset type: SYSRSn 0h (R/W) = 8 bits per data byte 1h (R/W) = 1 bit per data byte 2h (R/W) = 2 bits per data byte 3h (R/W) = 3 bits per data byte 4h (R/W) = 4 bits per data byte 5h (R/W) = 5 bits per data byte 6h (R/W) = 6 bits per data byte 7h (R/W) = 7 bits per data byte

13.6.2.11 I2C Interrupt Source (I2CISRC) Register (Offset = Ah) [reset = 0h]

The I2C interrupt source register (I2CISRC) is a 16-bit register used by the CPU to determine which event generated the I2C interrupt.

Figure 13-29. I2C Interrupt Source (I2CISRC) Register

15	14	13	12	11	10	9	8
RESERVED				WRITE_ZEROS			
R-0h				R/W-0h			
7	6	5	4	3	2	1	0
RESERVED						INTCODE	
R-0h						R-0h	

Table 13-20. I2C Interrupt Source (I2CISRC) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-12	RESERVED	R	0h	Reserved
11-8	WRITE_ZEROS	R/W	0h	TI internal testing bits These reserved bit locations should always be written as zeros. Reset type: SYSRSn
7-3	RESERVED	R	0h	Reserved
2-0	INTCODE	R	0h	Interrupt code bits. The binary code in INTCODE indicates the event that generated an I2C interrupt. A CPU read will clear this field. If another lower priority interrupt is pending and enabled, the value corresponding to that interrupt will then be loaded. Otherwise, the value will stay cleared. In the case of an arbitration lost, a no-acknowledgment condition detected, or a stop condition detected, a CPU read will also clear the associated interrupt flag bit in the I2CSTR register. Inversely, in the case of a Register-access-ready (ARDY), a Receive-data-ready (RRDY), a Transmit-data-ready (XRDY), or an Adressed-as-slave (AAS) condition, a CPU read will NOT clear the associated interrupt flag bit in the I2CSTR register Debug probe reads will not affect the state of this field or of the status bits in the I2CSTR register. Reset type: SYSRSn 0h (R/W) = None 1h (R/W) = Arbitration lost 2h (R/W) = No-acknowledgment condition detected 3h (R/W) = Registers ready to be accessed 4h (R/W) = Receive data ready 5h (R/W) = Transmit data ready 6h (R/W) = Stop condition detected 7h (R/W) = Addressed as slave

13.6.2.12 I2C Extended Mode Register (I2CEMDR) (Offset = Bh) [reset = 1h]

I2C Extended Mode

Figure 13-30. I2C Extended Mode Register (I2CEMDR)

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							BC
R-0h							R/W-1h

Table 13-21. I2C Extended Mode Register (I2CEMDR) Field Descriptions

Bit	Field	Type	Reset	Description
15-1	RESERVED	R	0h	Reserved
0	BC	R/W	1h	Backwards compatibility mode. This bit affects the timing of the transmit status bits (XRDY and XSMT) in the I2CSTR register when in slave transmitter mode. Check Backwards Compatibility Mode Bit, Slave Transmitter diagram for more details. Reset type: SYSRSn 0h (R/W) = See the "Backwards Compatibility Mode Bit, Slave Transmitter" Figure for details. 1h (R/W) = See the "Backwards Compatibility Mode Bit, Slave Transmitter" Figure for details.

13.6.2.13 I2C Prescaler (I2CPSC) Register (Offset = Ch) [reset = 0h]

The I2C prescaler register (I2CPSC) is a 16-bit register used for dividing down the I2C input clock to obtain the desired module clock for the operation of the I2C module. See the device-specific data manual for the supported range of values for the module clock frequency. IPSC must be initialized while the I2C module is in reset (IRS = 0 in I2CMDR). The prescaled frequency takes effect only when IRS is changed to 1. Changing the IPSC value while IRS = 1 has no effect.

Figure 13-31. I2C Prescaler (I2CPSC) Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
IPSC							
R/W-0h							

Table 13-22. I2C Prescaler (I2CPSC) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	RESERVED	R	0h	Reserved
7-0	IPSC	R/W	0h	I2C prescaler divide-down value. IPSC determines how much the CPU clock is divided to create the module clock of the I2C module: module clock frequency = I2C input clock frequency/(IPSC + 1) Note: IPSC must be initialized while the I2C module is in reset (IRS = 0 in I2CMDR). Reset type: SYSRSn

13.6.2.14 I2C FIFO Transmit (I2CFFTX) Register (Offset = 20h) [reset = 0h]

The I2C transmit FIFO register (I2CFFTX) is a 16-bit register that contains the I2C FIFO mode enable bit as well as the control and status bits for the transmit FIFO mode of operation on the I2C peripheral.

Figure 13-32. I2C FIFO Transmit (I2CFFTX) Register

15	14	13	12	11	10	9	8
RESERVED	I2CFFEN	TXFFRST	TXFFST				
R-0h	R/W-0h	R/W-0h	R-0h				
7	6	5	4	3	2	1	0
TXFFINT	TXFFINTCLR	TXFFIENA	TXFFIL				
R-0h	R-0/W1S-0h	R/W-0h	R/W-0h				

Table 13-23. I2C FIFO Transmit (I2CFFTX) Register Field Descriptions

Bit	Field	Type	Reset	Description
15	RESERVED	R	0h	Reserved
14	I2CFFEN	R/W	0h	I2C FIFO mode enable bit. This bit must be enabled for either the transmit or the receive FIFO to operate correctly. Reset type: SYSRSn 0h (R/W) = Disable the I2C FIFO mode. 1h (R/W) = Enable the I2C FIFO mode.
13	TXFFRST	R/W	0h	Transmit FIFO Reset Reset type: SYSRSn 0h (R/W) = Reset the transmit FIFO pointer to 0000 and hold the transmit FIFO in the reset state. 1h (R/W) = Enable the transmit FIFO operation.
12-8	TXFFST	R	0h	Contains the status of the transmit FIFO: xxxxx Transmit FIFO contains xxxxx bytes. 00000 Transmit FIFO is empty. Note: Since these bits are reset to zero, the transmit FIFO interrupt flag will be set when the transmit FIFO operation is enabled and the I2C is taken out of reset. This will generate a transmit FIFO interrupt if enabled. To avoid any detrimental effects from this, write a one to the TXFFINTCLR once the transmit FIFO operation is enabled and the I2C is taken out of reset. Reset type: SYSRSn
7	TXFFINT	R	0h	Transmit FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the TXFFINTCLR bit. If the TXFFIENA bit is set, this bit will generate an interrupt when it is set. Reset type: SYSRSn 0h (R/W) = Transmit FIFO interrupt condition has not occurred. 1h (R/W) = Transmit FIFO interrupt condition has occurred.
6	TXFFINTCLR	R-0/W1S	0h	Transmit FIFO Interrupt Flag Clear Reset type: SYSRSn 0h (R/W) = Writes of zeros have no effect. Reads return a 0. 1h (R/W) = Writing a 1 to this bit clears the TXFFINT flag.
5	TXFFIENA	R/W	0h	Transmit FIFO Interrupt Enable Reset type: SYSRSn 0h (R/W) = Disabled. TXFFINT flag does not generate an interrupt when set. 1h (R/W) = Enabled. TXFFINT flag does generate an interrupt when set.

Table 13-23. I2C FIFO Transmit (I2CFFTX) Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4-0	TXFFIL	R/W	0h	Transmit FIFO interrupt level. These bits set the status level that will set the transmit interrupt flag. When the TXFFST4-0 bits reach a value equal to or less than these bits, the TXFFINT flag will be set. This will generate an interrupt if the TXFFIENA bit is set. Because the I2C on this device has a 4-level transmit FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels. Reset type: SYSRSn

13.6.2.15 I2C FIFO Receive (I2CFFRX) Register (Offset = 21h) [reset = 0h]

The I2C receive FIFO register (I2CFFRX) is a 16-bit register that contains the control and status bits for the receive FIFO mode of operation on the I2C peripheral.

Figure 13-33. I2C FIFO Receive (I2CFFRX) Register

15	14	13	12	11	10	9	8
RESERVED		RXFFRST	RXFFST				
R-0h		R/W-0h	R-0h				
7	6	5	4	3	2	1	0
RXFFINT	RXFFINTCLR	RXFFIENA	RXFFIL				
R-0h	R-0/W1S-0h	R/W-0h	R/W-0h				

Table 13-24. I2C FIFO Receive (I2CFFRX) Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	RESERVED	R	0h	Reserved
13	RXFFRST	R/W	0h	I2C receive FIFO reset bit Reset type: SYSRSn 0h (R/W) = Reset the receive FIFO pointer to 0000 and hold the receive FIFO in the reset state. 1h (R/W) = Enable the receive FIFO operation.
12-8	RXFFST	R	0h	Contains the status of the receive FIFO: xxxxx Receive FIFO contains xxxxx bytes 00000 Receive FIFO is empty. Reset type: SYSRSn
7	RXFFINT	R	0h	Receive FIFO interrupt flag. This bit cleared by a CPU write of a 1 to the RXFFINTCLR bit. If the RXFFIENA bit is set, this bit will generate an interrupt when it is set Reset type: SYSRSn 0h (R/W) = Receive FIFO interrupt condition has not occurred. 1h (R/W) = Receive FIFO interrupt condition has occurred.
6	RXFFINTCLR	R-0/W1S	0h	Receive FIFO interrupt flag clear bit. Reset type: SYSRSn 0h (R/W) = Writes of zeros have no effect. Reads return a zero. 1h (R/W) = Writing a 1 to this bit clears the RXFFINT flag.
5	RXFFIENA	R/W	0h	Receive FIFO interrupt enable bit. Reset type: SYSRSn 0h (R/W) = Disabled. RXFFINT flag does not generate an interrupt when set. 1h (R/W) = Enabled. RXFFINT flag does generate an interrupt when set.
4-0	RXFFIL	R/W	0h	Receive FIFO interrupt level. These bits set the status level that will set the receive interrupt flag. When the RXFFST4-0 bits reach a value equal to or greater than these bits, the RXFFINT flag is set. This will generate an interrupt if the RXFFIENA bit is set. Note: Since these bits are reset to zero, the receive FIFO interrupt flag will be set if the receive FIFO operation is enabled and the I2C is taken out of reset. This will generate a receive FIFO interrupt if enabled. To avoid this, modify these bits on the same instruction as or prior to setting the RXFFRST bit. Because the I2C on this device has a 4-level receive FIFO, these bits cannot be configured for an interrupt of more than 4 FIFO levels. Reset type: SYSRSn

Chapter 14

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.

14.1 Introduction

14.1.1 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.

14.1.2 ECAN Related Collateral

Foundational Materials

- Automotive CAN Overview and Training (Video)
- CAN Physical layer (Video)
- CAN and CAN FD Overview (Video)

Getting Started Materials

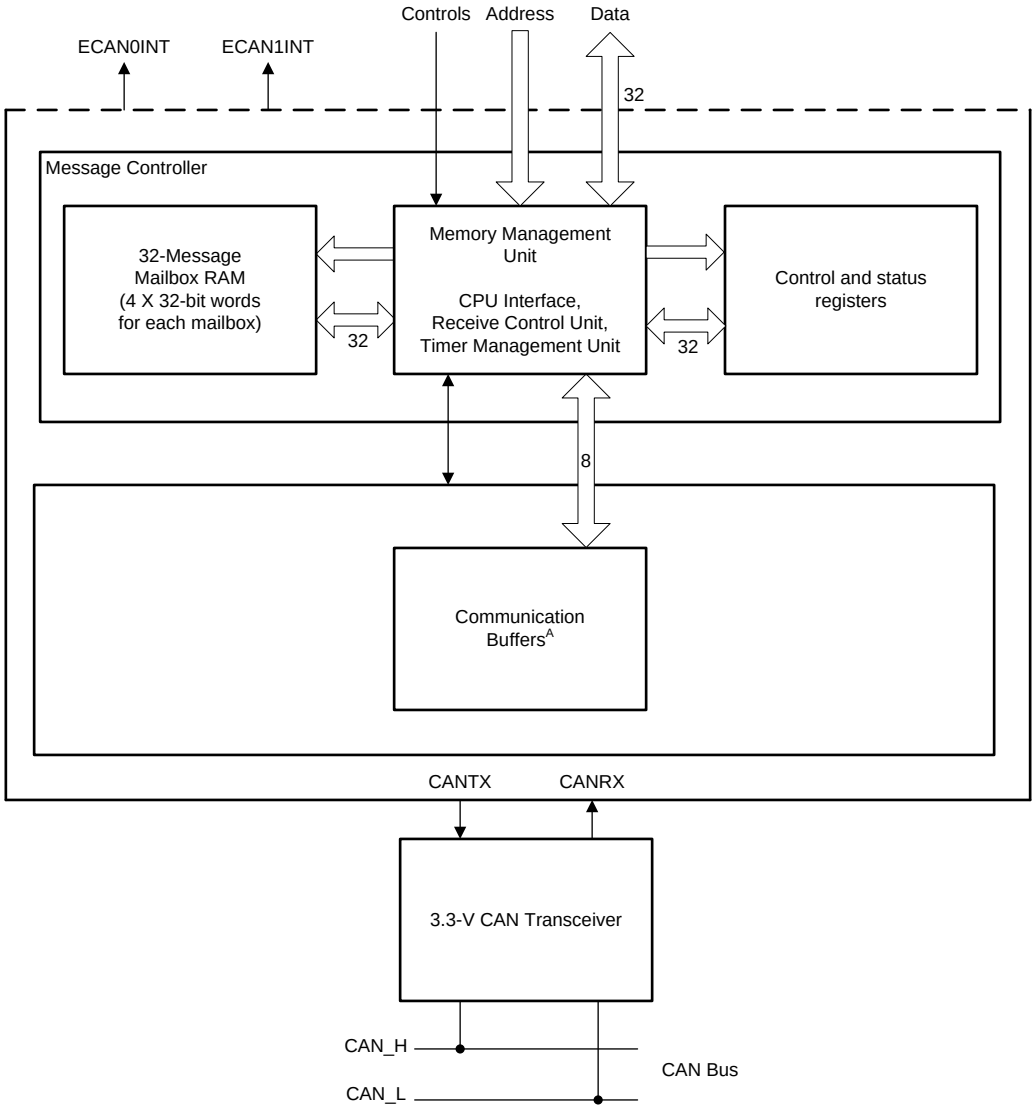
- *MCU CAN Module Operation Using the On-Chip Zero-Pin Oscillator Application Report*
- *Programming Examples for the TMS320x28xx eCAN Application Report*
- *Using the CAN Bootloader at High Temperature Application Report*

Expert Materials

- *Calculator for CAN Bit Timing Parameters Application Report*

14.1.3 Block Diagram

Figure 14-1 shows the block diagram of eCAN.



A. The communication buffers are transparent to the user and are not accessible by user code.

Figure 14-1. eCAN Block Diagram and Interface Circuit

14.2 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.

14.3 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.

14.3.1 CAN Protocol Overview

The CAN protocol supports four different frame types for communication:

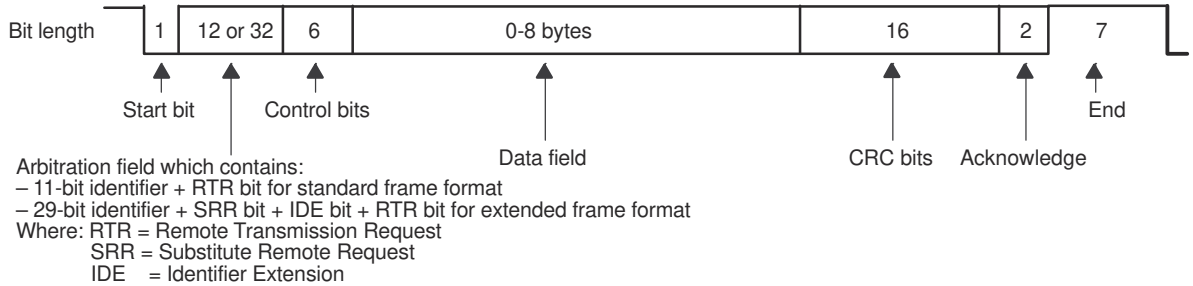
- Data frames that carry data from a transmitter node to the receiver node(s).
- Remote frames that are transmitted by a node to request the transmission of a data frame with the same identifier.
- Error frames that are transmitted by any node upon detecting an error condition.
- Overload frames that 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 14-2 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

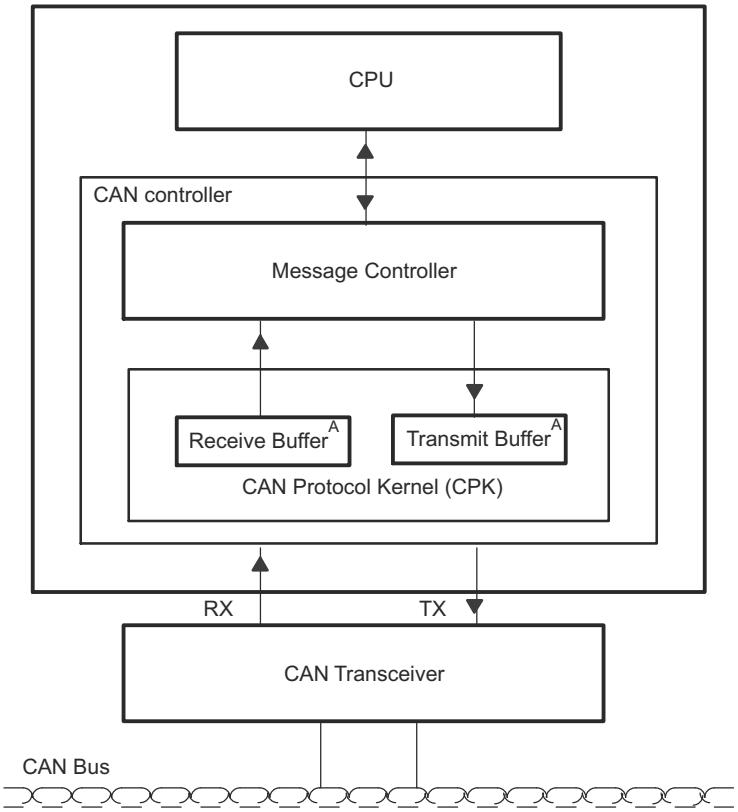


Note: Unless otherwise noted, numbers are amount of bits in field.

Figure 14-2. CAN Data Frame

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 14-3, is composed of a CAN protocol kernel (CPK) and a message controller.



A. The receive and transmit buffers are transparent to the user and are not accessible by user code.

Figure 14-3. Architecture of the eCAN Module

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.

14.4 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.

14.4.1 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).

14.4.2 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 14-4, 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.

14.4.2.1 32-bit Access to Control and Status Registers

As indicated in Section 14.4.2, 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 14-1. 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 14-2. 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 Example 14-2, 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);
```

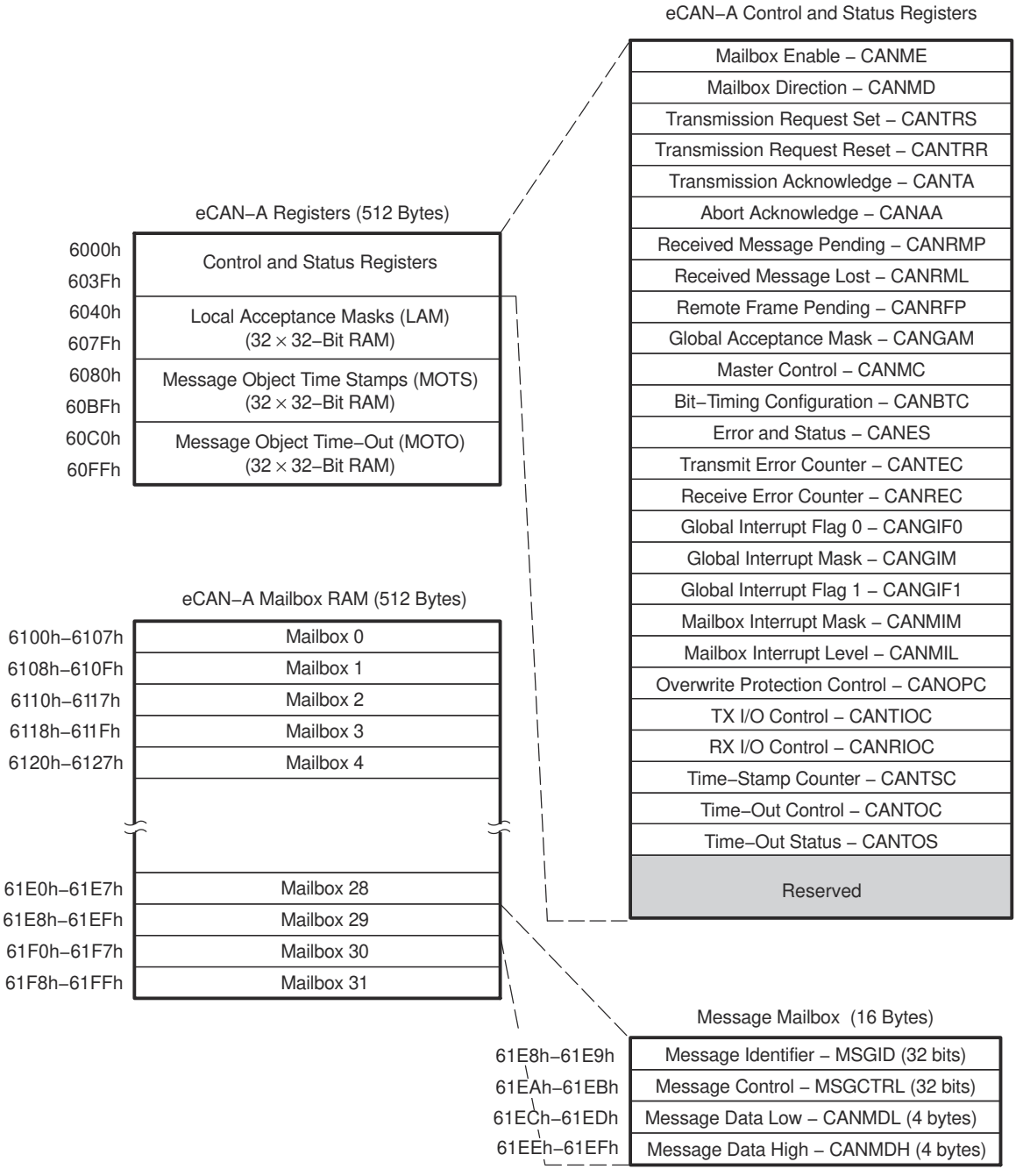


Figure 14-4. eCAN-A Memory Map

14.5 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.

14.6 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 14-1). 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 14-2 lists the mailbox RAM layout.

Table 14-1. 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 14-2. 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 14-3. 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

14.6.1 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.

14.6.2 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. In addition to the TRS bit, the RTR bit is also cleared. The mailbox needs to be disabled in order to set this bit again.

14.6.3 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).

14.7 eCAN Configuration

This section explains the process of initialization and describes the procedures to configure the eCAN module.

14.7.1 CAN Module Initialization

The CAN module must be initialized before the utilization. Initialization is only possible if the module is in initialization mode. Figure 14-5 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.

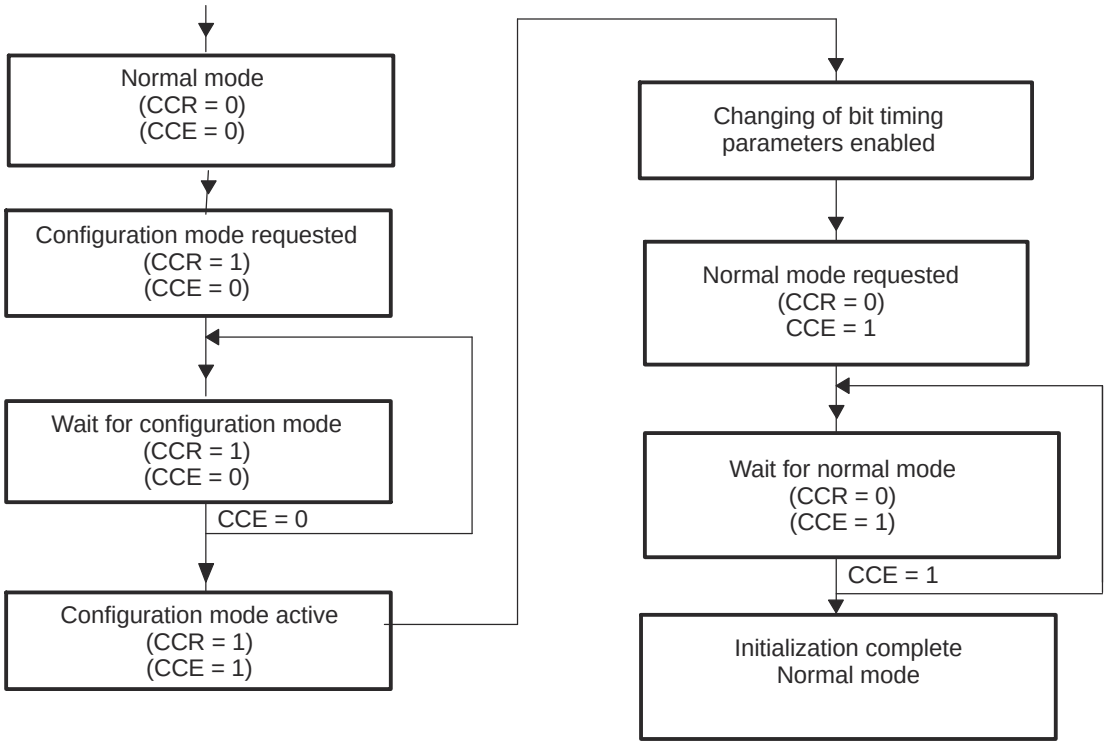


Figure 14-5. Initialization Sequence

Note

The transition between initialization mode and normal mode and conversely 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.

14.7.1.1 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.

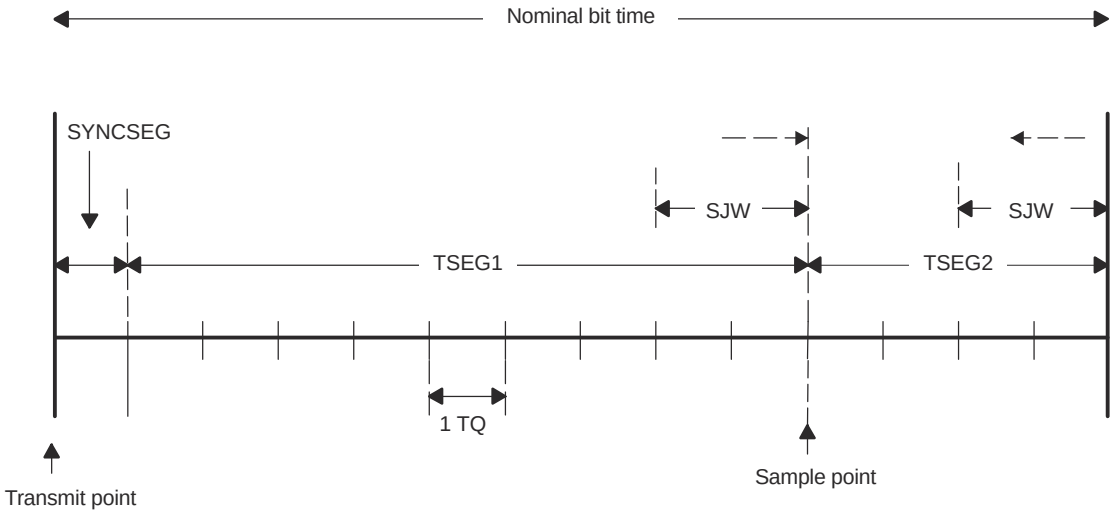
In the eCAN module, the length of a bit on the CAN bus is determined by the parameters TSEG1 (BTC.6-3), TSEG2 (BTC.2-0), and BRP (BTC.23.16).

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



A. TSEG1 can be lengthened or TSEG2 shortened by the SJW

Figure 14-6. CAN Bit Timing

14.7.1.2 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 $TSEG1_{reg}$, $TSEG2_{reg}$, 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 14-6.

Bit-time = TSEG1 + TSEG2 + 1

14.7.1.3 Bit Configuration Parameters for 30-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 14-4 shows how the BRP_{reg} field may be changed to achieve different bit rates with a BT of 15 for a 80% SP.

Table 14-4. BRP Field for Bit Rates (BT = 15, $TSEG1_{reg}$ = 10, $TSEG2_{reg}$ = 2, Sampling Point = 80%)

CAN Bus Speed	BRP	CAN Module Clock
1 Mbps	BRP_{reg} +1 = 2	15 MHz
500 kbps	BRP_{reg} +1 = 4	7.5 MHz
250 kbps	BRP_{reg} +1 = 8	3.75 MHz
125 kbps	BRP_{reg} +1 = 16	1.875 MHz
100 kbps	BRP_{reg} +1 = 20	1.5 MHz
50 kbps	BRP_{reg} +1 = 40	0.75 MHz

Table 14-5 shows how to achieve different sampling points with a BT of 15.

Table 14-5. 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 60 MHz, the lowest bit-rate that can be achieved is 4.687 kbps.

14.7.1.4 Bit Configuration Parameters for 30-MHz CAN Clock

Table 14-6 shows how BRP_{reg} field may be changed to achieve different bit rates with a BT of 10 for an 80% sampling point.

Table 14-6. BRP Field for Bit Rates (BT = 10, $TSEG1_{reg}$ = 6, $TSEG2_{reg}$ = 1, Sampling Point = 80%)

CAN Bus Speed	BRP	CAN Module Clock
1 Mbps	BRP_{reg} +1 = 3	10 MHz
500 kbps	BRP_{reg} +1 = 6	5 MHz
250 kbps	BRP_{reg} +1 = 12	2.5 MHz
125 kbps	BRP_{reg} +1 = 24	1.25 MHz
100 kbps	BRP_{reg} +1 = 30	1 MHz
50 kbps	BRP_{reg} +1 = 60	0.5 MHz

Table 14-7 shows how to achieve different sampling points with a BT of 20.

Table 14-7. Achieving Different Sampling Points With a BT of 20

$TSEG1_{reg}$	$TSEG2_{reg}$	SP
15	2	85%
14	3	80%
13	4	75%
12	5	70%
11	6	65%
10	7	60%

14.7.1.5 EALLOW Protection

To protect against inadvertent modification, some critical registers/bits of the eCAN module are EALLOW protected. These registers/bits can be changed only if the EALLOW protection has been disabled. Following are the registers/ bits that are EALLOW protected in the eCAN module:

- CANMC[15:9, 7:6]
- CANBTC
- CANGIM
- CANMIM[31..0]
- CANTSC[31..0]
- CANTIOC[3]
- CANRIOC[3]

14.7.2 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:

1. Enable clock to the CAN module.
2. Set the CANTX and the CANRX pins to CAN functions:
   a. Write CANTIOC.3:0 = 0x08
   b. Write CANRIOC.3:0 = 0x08
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).
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.
5. For the SCC, program the acceptance masks now. For example:
   Write LAM(3) = 0x3C0000
6. Program the master control register (CANMC) as follows:
   a. Clear CCR (CANMC.12) = 0
   b. Clear PDR (CANMC.11) = 0
   c. Clear DBO (CANMC.10) = 0
   d. Clear WUBA (CANMC.9)= 0
   e. Clear CDR (CANMC.8) = 0
   f. Clear ABO (CANMC.7) = 0
   g. Clear STM (CANMC.6) = 0
   h. Clear SRES (CANMC.5) = 0
   i. Clear MBNR (CANMC.4-0) = 0
7. Initialize all bits of MSGCTRLn registers to zero.
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.

14.7.2.1 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:
   a. Write MSGID(1) = 0x15AC0000
   b. Write the data length into the DLC field of the message control field register (MSGCTRL.3:0). The RTR flag is usually cleared (MSGCTRL.4 = 0).
   c. Set the mailbox direction by clearing the corresponding bit in the CANMD register.
   d. Clear CANMD.1 = 0
4. Set the mailbox enable by setting the corresponding bit in the CANME register

   Set CANME.1 = 1

This configures mailbox 1 for transmit mode.

14.7.2.2 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.

14.7.2.3 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.

14.7.2.4 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.

14.7.2.5 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.

14.7.3 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.

14.7.3.1 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.

14.7.3.2 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.

14.7.3.3 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

14.7.4 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 14-7.

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

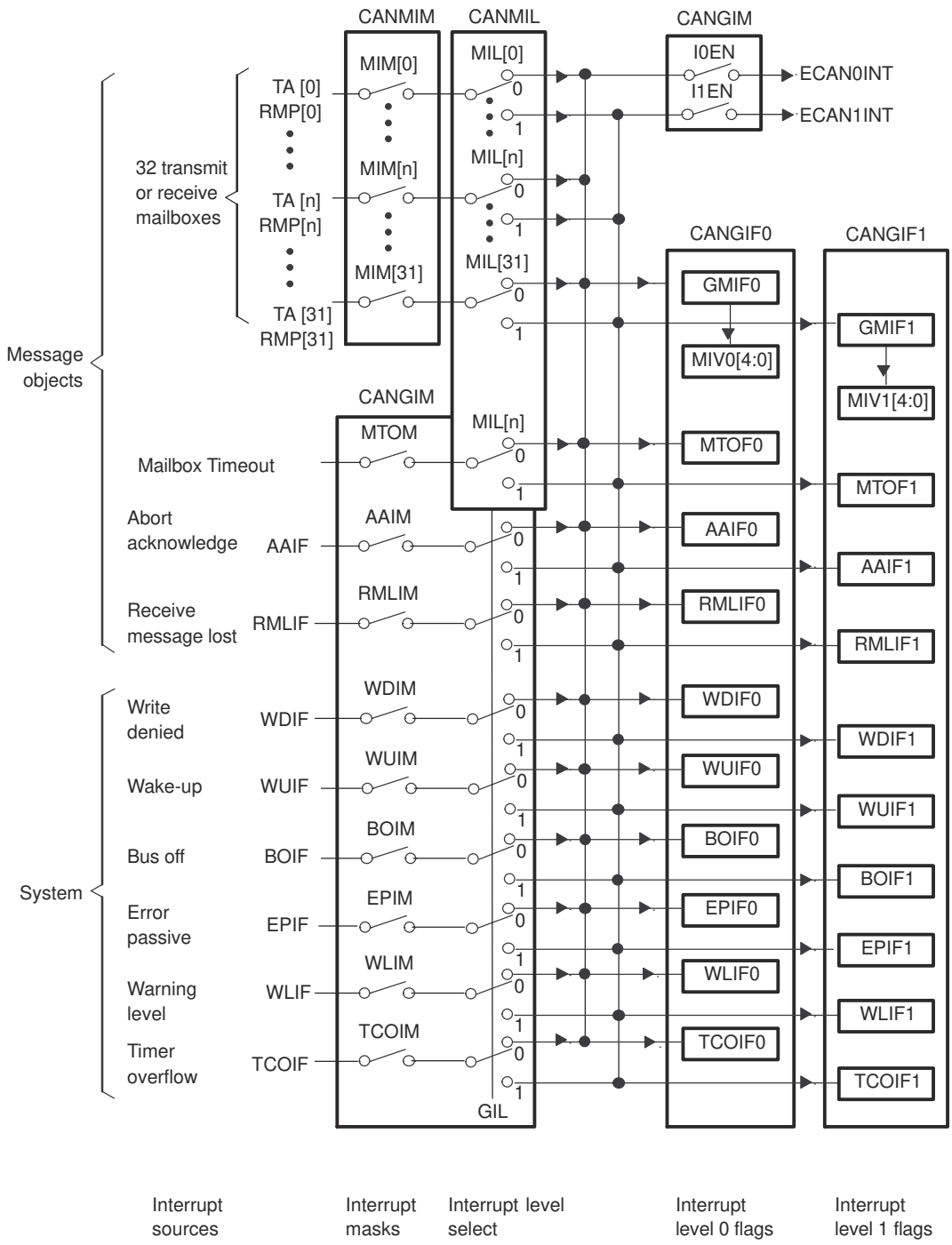


Figure 14-7. Interrupts Scheme

14.7.4.1 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.

14.7.4.2 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.

14.7.4.3 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 14-8. This also releases the interrupt line if no other interrupt is pending.

Table 14-8. eCAN Interrupt Assertion/Clearing

Interrupt Flag (1)	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 appropriate bit in CANTA or CANRMP registers
TCOFn	The MSB of 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 dependent 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.

14.7.4.3.1 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.

14.7.4.3.2 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.

14.7.4.3.3 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 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.

14.7.5 CAN Power-Down Mode

A local power-down mode has been implemented where the CAN module internal clock is deactivated by the CAN module itself.

14.7.5.1 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.

14.7.5.2 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.

14.7.5.3 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 . 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.

14.7.5.4 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

14.8 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 Section 14.8.1.

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 14.7.1).

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.

14.8.1 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.

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.

Figure 14-8. 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 14-9. 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.

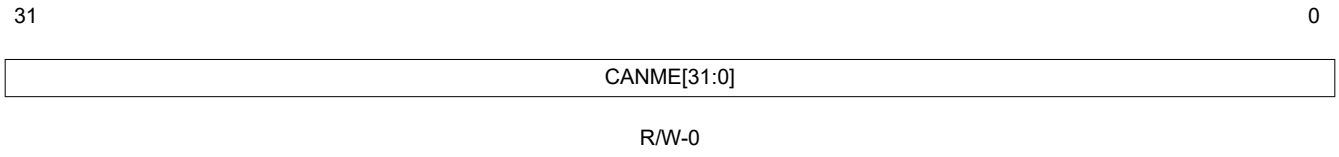
14.9 eCAN Registers

This chapter describes the Enhanced Controller Area Network (eCAN) registers.

14.9.1 Mailbox Enable Register (CANME)

This register is used to enable/disable individual mailboxes.

Figure 14-9. Mailbox-Enable Register (CANME)



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

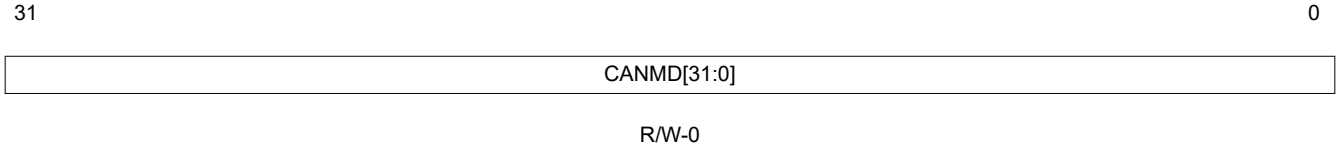
Table 14-10. Mailbox-Enable Register (CANME) Field Descriptions

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.

14.9.2 Mailbox-Direction Register (CANMD)

This register is used to configure a mailbox for transmit or receive operation.

Figure 14-10. Mailbox-Direction Register (CANMD)



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

Table 14-11. 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.

14.9.3 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 14-11. Transmission-Request Set Register (CANTRS)

31	0
TRS[31:0]	
RS-0	

LEGEND: RS = Read/Set; -*n* = value after reset

Table 14-12. 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

14.9.4 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 14-12. Transmission-Request-Reset Register (CANTRR)

31	0
TRR[31:0]	
RS-0	

LEGEND: RS = Read/Set; -*n* = value after reset

Table 14-13. 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

14.9.5 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 14-13. Transmission-Acknowledge Register (CANTA)

31	0
TA[31:0]	
RC-0	

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 14-14. 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.

14.9.6 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 14-14. Abort-Acknowledge Register (CANAA)

31	0
AA[31:0]	
RC-0	

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 14-15. 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.

14.9.7 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 14-15. Received-Message-Pending Register (CANRMP)

31	0
RMP[31:0]	
RC-0	

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 14-16. 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.

14.9.8 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 14-16. Received-Message-Lost Register (CANRML)

31	0
RML[31:0]	
R-0	

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

Table 14-17. 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.

14.9.9 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 14-17. Remote-Frame-Pending Register (CANRFP)

31	0
RFP.31:0	
RC-0	

LEGEND: RC = Read/Clear; -*n* = value after reset

Table 14-18. 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.

14.9.9.1 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 14.9.7.

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

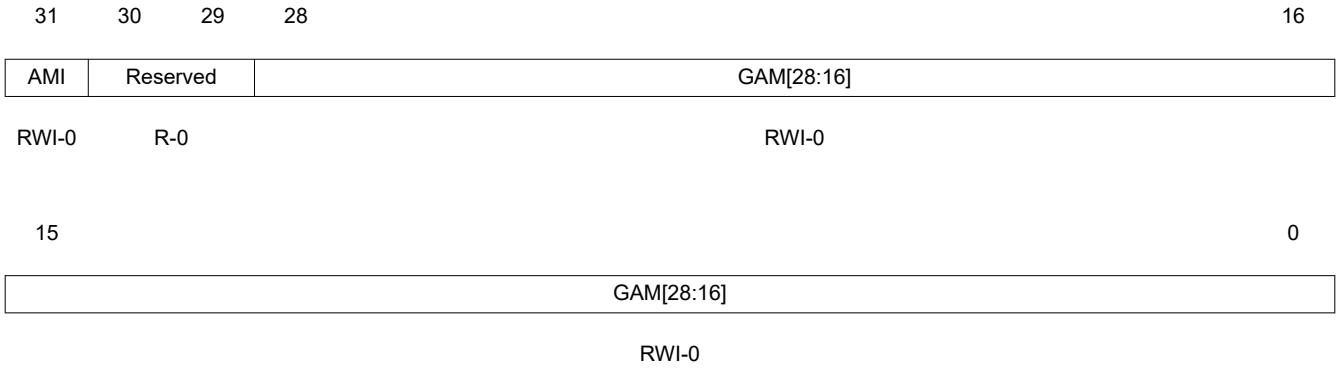
When a remote transmission request is successfully transmitted with a message object configured in request mode, the CANTA register is not set and no interrupt is generated. When the remote reply message is received, the behavior of the message object is the same as a message object configured in receive mode.

14.9.10 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 14-18. Global Acceptance Mask Register (CANGAM)



LEGEND: RWI = Read at any time, write during initialization mode only; -*n* = value after reset

Table 14-19. 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".

14.9.11 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 14-19. 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 14-20. 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.

Table 14-20. Master Control Register (CANMC) Field Descriptions (continued)

Bit	Field	Value	Description
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.
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.

Table 14-20. Master Control Register (CANMC) Field Descriptions (continued)

Bit	Field	Value	Description
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.

14.9.11.1 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.

14.9.12 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 14.7.1).

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 14.7.1.1.

Figure 14-20. 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 14-21. Bit-Timing Configuration Register (CANBTC) Field Descriptions

Bit	Field	Value	Description
31:24	Reserved		Reads are undefined and writes have no effect.
23:16	BRP_{reg}.7:0		Baud rate prescaler. This register sets the prescaler for the baud rate settings. The length of one TQ is defined by: $$TQ = \frac{1}{SYSCLKOUT/2} \times (BRP_{reg} + 1)$$ where SYSCLKOUT /2 is the frequency of the CAN module clock. BRP_{reg} denotes the "register value" of the prescaler; 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]

Table 14-21. Bit-Timing Configuration Register (CANBTC) Field Descriptions (continued)

Bit	Field	Value	Description
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.
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 14.7.1.1.
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. $TSEG2 = TSEG2_{reg} + 1$

14.9.13 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 14-21. 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 14-22. 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.

Table 14-22. Error and Status Register (CANES) Field Descriptions (continued)

Bit	Field	Value	Description
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.
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.

14.9.14 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.

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.

Figure 14-22. Transmit-Error-Counter Register (CANTEC)

31	8	7	0
Reserved		TEC	
R-x		R-0	

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

Figure 14-23. Receive-Error-Counter Register (CANREC)

31	8	7	0
Reserved		REC	
R-x		R-0	

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

14.9.15 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.

14.9.15.1 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.

Note

The 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.

Figure 14-24. Global Interrupt Flag 0 Register (CANGIF0)

31						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 14-25. Global Interrupt Flag 1 Register (CANGIF1)

31						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

Table 14-23. 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.
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.

14.9.15.2 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 14-26. 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 14-24. 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

Table 14-24. Global Interrupt Mask Register (CANGIM) Field Descriptions (continued)

Bit	Field	Value	Description
10	BOIM		Bus-off interrupt mask
		1	Enabled
		0	Disabled
9	EPIM		Error-passive interrupt mask
		1	Enabled
		0	Disabled
8	WLIM		Warning level interrupt mask
		1	Enabled
		0	Disabled
7:3	Reserved		Reads are undefined and writes have no effect.
2	GIL		Global interrupt level for the interrupts TCOF, WDIF, WUIF, BOIF, EPIF, RMLIF, AAIF and WLIF.
		1	All global interrupts are mapped to the ECAN1INT interrupt line.
		0	All global interrupts are mapped to the ECAN0INT interrupt line.
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.

14.9.15.3 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 14-27. Mailbox Interrupt Mask Register (CANMIM)

31	0
MIM.31:0	
R/W-0	

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

Table 14-25. 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.

14.9.15.4 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 14-28. Mailbox Interrupt Level Register (CANMIL)

31	0
MIL.31:0	
R/W-0	

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

Table 14-26. 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.

14.9.16 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 14-29. Overwrite Protection Control Register (CANOPC)

31	0
OPC.31:0	
R/W-0	

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

Table 14-27. 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.

14.9.17 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 14-30. 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 14-28. 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 14-31. 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 14-29. 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

14.9.18 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.

14.9.18.1 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.

14.9.18.1.1 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 14-32. 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 14-30. 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.

14.9.18.2 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.

14.9.18.2.1 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 14-33. 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 14-31. 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.

14.9.18.2.2 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 14-34. 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 14-32. 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.

14.9.18.2.3 Time-Out Control Register (CANTOC)

This register controls whether or not time-out functionality is enabled for a given mailbox.

Figure 14-35. Time-Out Control Register (CANTOC)

31	0
TOC31:0	
R/W-0	

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

Table 14-33. 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.

14.9.18.2.4 Time-Out Status Register (CANTOS)

This register holds the status information of mailboxes that have timed out.

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.

Figure 14-36. Time-Out Status Register (CANTOS)

31	0
TOS31:0	
R/C-0	

LEGEND: R/C = Read/Clear; -*n* = value after reset

Table 14-34. 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.

14.9.18.3 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.

14.9.19 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

14.9.19.1 Message Identifier Register (MSGID)

This register contains the message ID and other control bits for a given mailbox.

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.

Figure 14-37. 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

Table 14-35. 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.

14.9.19.2 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.

14.9.19.3 Message-Control Register (MSGCTRL)

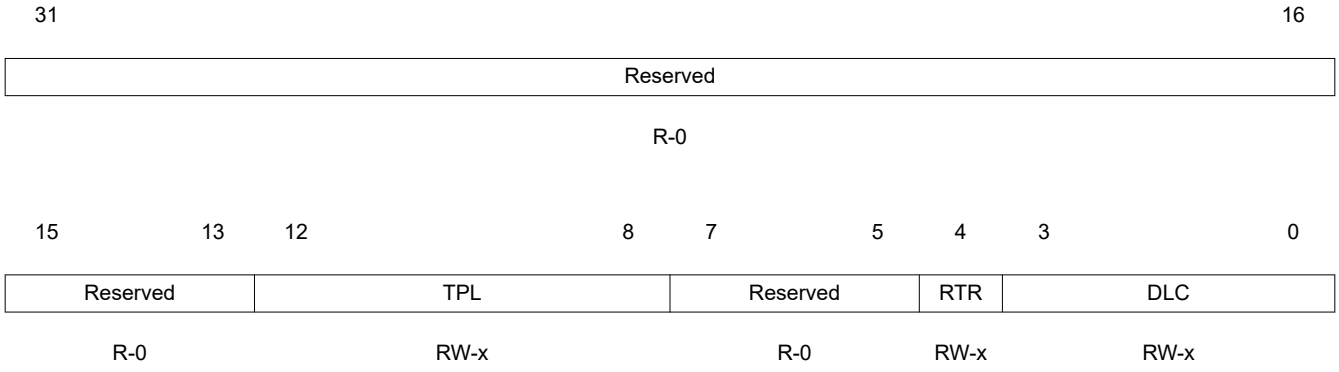
For a transmit mailbox, this register specifies the number of bytes to be transmitted and the transmission priority. It also specifies the remote-frame operation.

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.

The register MSGCTRL(*n*) can only be written if the mailbox *n* is configured for transmission (CANMD[*n*] (CANMD.31-0)=0) or if the mailbox is disabled (CANME[*n*] (CANME.31-0) =0).

Figure 14-38. Message-Control Register (MSGCTRL)



LEGEND: RW = Read any time, write when mailbox is disabled or configured for transmission; -*n* = value after reset; x = indeterminate

Table 14-36. 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.

14.10 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 the 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 14-39. Message-Data-Low Register With DBO = 0 (CANMDL)

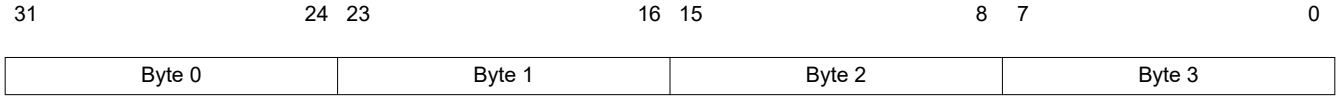


Figure 14-40. Message-Data-High Register With DBO = 0 (CANMDH)

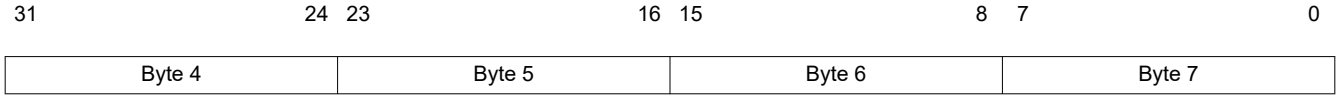


Figure 14-41. Message-Data-Low Register With DBO = 1 (CANMDL)

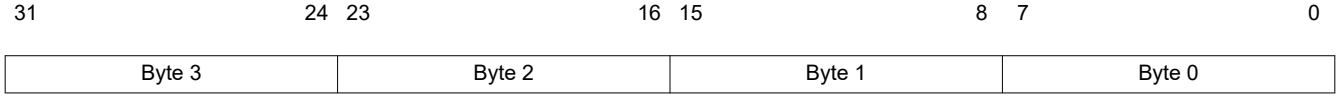
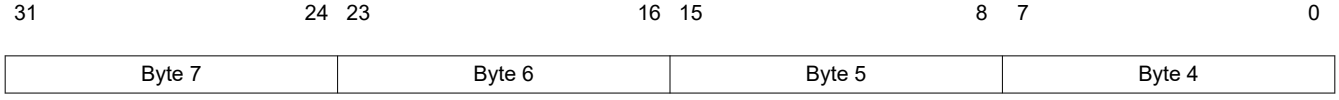


Figure 14-42. Message-Data-High Register With DBO = 1 (CANMDH)



Note

The data field beyond the valid received data is modified by any message reception and is indeterminate.

Chapter 15

Local Interconnect Network (LIN)

This chapter describes the local interconnect network (LIN) module. Since this module can also operate like a conventional serial communications interface (SCI) port, it is referred to as the SCI/LIN module in this document. In SCI compatibility mode, it is functionally compatible to the standalone SCI module. However, since the SCI/LIN module uses a different register/bit structure, code written for this module cannot be directly ported to the standalone SCI module and conversely.

This module can be configured to operate in either SCI (UART) or LIN mode.

15.1 Introduction

The SCI/LIN is compliant to the LIN 2.0 protocol specified in the *LIN Specification Package*.The SCI/LIN module can be programmed to work either as an SCI or as a LIN. The SCI's hardware features are augmented to achieve LIN compatibility.

The SCI module is a universal asynchronous receiver-transmitter (UART) that implements the standard non-return to zero format.

The LIN standard is based on the SCI (UART) serial data link format. The communication concept is single-master/multiple-slave with a message identification for multi-cast transmission between any network nodes.

Throughout the chapter, Compatibility Mode refers to SCI Mode functionality of the SCI/LIN Module. Section 15.2 explains about the SCI functionality and Section 15.3 explains about the LIN functionality. Though the registers are common for LIN and SCI, the register descriptions has notes to identify the register/bit usage in different modes.

15.1.1 SCI Features

The following are the features of the SCI module:

- Standard universal asynchronous receiver-transmitter (UART) communication
- Supports full- or half-duplex operation
- Standard non-return to zero (NRZ) format
- Double-buffered receive and transmit functions in compatibility mode
- Supports two individually enabled interrupt lines: level 0 and level 1
- Configurable frame format of 3 to 13 bits per character based on the following:
  - Data word length programmable from one to eight bits
  - Additional address bit in address-bit mode
  - Parity programmable for zero or one parity bit, odd or even parity
  - Stop programmable for one or two stop bits
- Asynchronous communication mode
- Two multiprocessor communication formats allow communication between more than two devices
- Sleep mode is available to free CPU resources during multiprocessor communication and then wake up to receive an incoming message
- The 24-bit programmable baud rate supports 2^{24} different baud rates provide high accuracy baud rate selection
- At 100-MHz peripheral clock, 3.125 Mbits/s is the Max Baud Rate achievable
- Five error flags and Seven status flags provide detailed information regarding SCI events
- Two external pins: LINRX and LINTX
- Multi-buffered receive and transmit units

Note

The SCI/LIN module is functionally compatible with the C2000™ SCI modules, but not directly software compatible due to different register control structures.

The SCI/LIN module does not support UART hardware flow control. This feature can be implemented in software using a general-purpose I/O pin.

The SCI/LIN module does not support isosynchronous mode as there is no SCICLK pin.

15.1.2 LIN Features

The following are the features of the LIN module:

- Compatibility with LIN 1.3 and 2.0 protocols
- Configurable Baud Rate up to 20 kpbs
- Two external pins: LINRX and LINTX.
- Multi-buffered receive and transmit units
- Identification masks for message filtering
- Automatic master header generation
  - Programmable synchronization break field
  - Synchronization field
  - Identifier field
- Slave automatic synchronization
  - Synchronization break detection
  - Optional baud rate update
  - Synchronization validation
- 2^{28} programmable transmission rates
- Wake up on LINRX dominant level from transceiver
- Automatic wake up support
  - Wakeup signal generation
  - Expiration times on wakeup signals
- Automatic bus idle detection
- Error detection
  - Bit error
  - Bus error
  - No-response error
  - Checksum error
  - Synchronization field error
  - Parity error
- 2 Interrupt lines with priority encoding for:
  - Receive
  - Transmit
  - ID, error, and status
- Support for LIN 2.0 checksum
- Enhanced synchronizer finite state machine (FSM) support for frame processing
- Enhanced handling of extended frames
- Enhanced baud rate generator
- Update wakeup/go to sleep

15.1.3 Block Diagram

The SCI/LIN module contains the core SCI block with added sub-blocks to support LIN protocol.

The three major components of the SCI Module are:

- **Transmitter** (TX) contains two major registers to perform the double-buffering:
  - The transmitter data buffer register (SCITD) contains data loaded by the CPU to be transferred to the shift register for transmission.
  - The transmitter shift register (SCITXSHF) loads data from the data buffer (SCITD) and shifts data onto the LINTX pin, one bit at a time.
- **Baud Clock Generator**
  - A programmable baud generator produces a baud clock scaled from the input clock VCLK
- **Receiver** (RX) contains two major registers to perform the double-buffering:
  - The receiver shift register (SCIRXSHF) shifts data in from the LINRX pin one bit at a time and transfers completed data into the receive data buffer.
  - The receiver data buffer register (SCIRD) contains received data transferred from the receiver shift register

The SCI receiver and transmitter are double-buffered, and each has its own separate enable and interrupt bits. The receiver and transmitter may each be operated independently or simultaneously in full duplex mode.

To ensure data integrity, the SCI checks the data it receives for breaks, parity, overrun, and framing errors. The bit rate (baud) is programmable to over 16 million different rates through a 24-bit baud-select register. Figure 15-1 shows the detailed SCI block diagram.

The SCI/LIN module is based on the standalone SCI with the addition of an error detector (parity calculator, checksum calculator, and bit monitor), a mask filter, a synchronizer, and a multi-buffered receiver and transmitter. The SCI interface and the baud generator are modified as part of the hardware enhancements for LIN compatibility. Figure 15-2 shows the SCI/LIN block diagram.

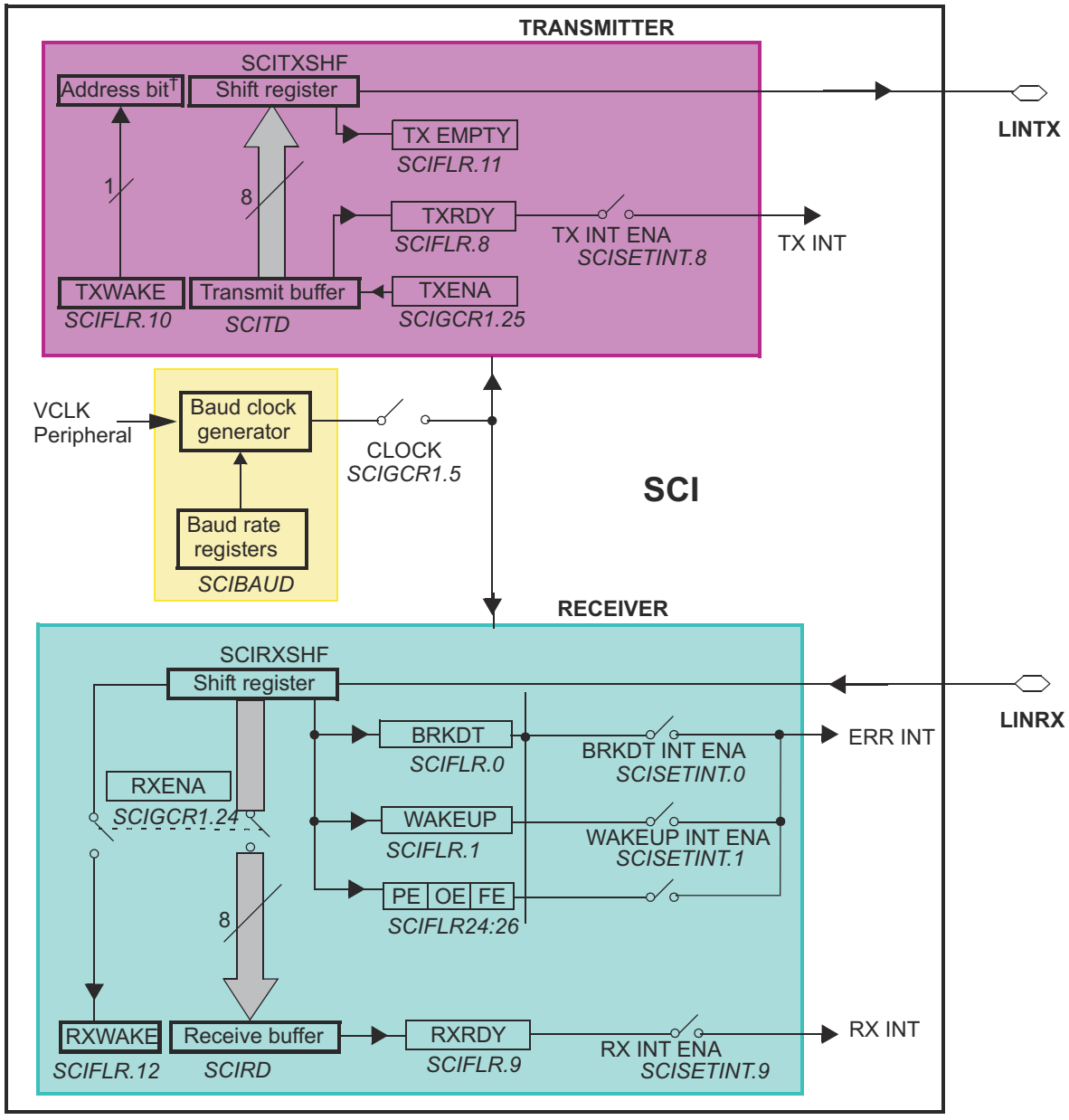


Figure 15-1. SCI Block Diagram

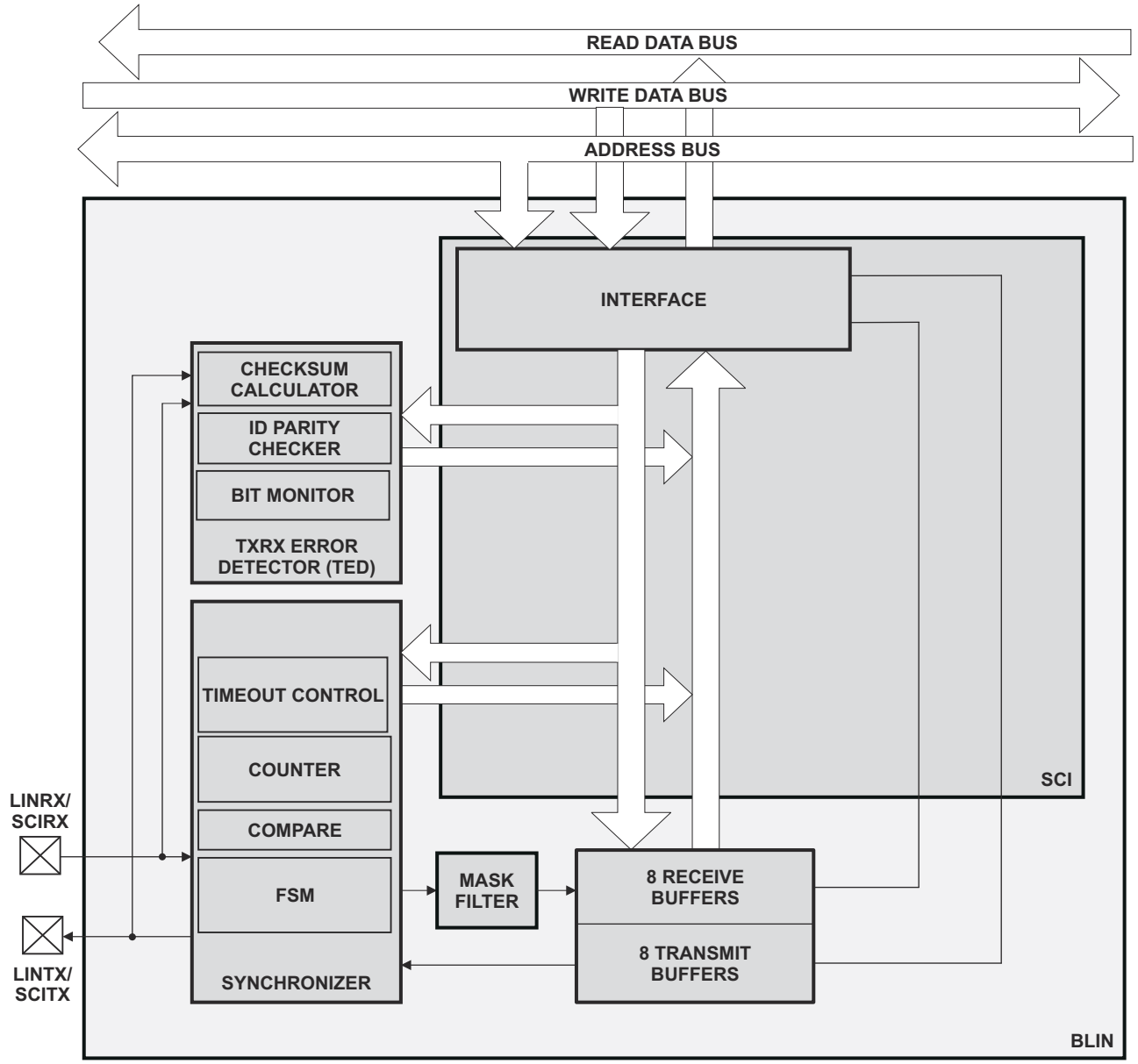


Figure 15-2. SCI/LIN Block Diagram

15.2 Serial Communications Interface Module

15.2.1 SCI Communication Formats

The SCI module can be configured to meet the requirements of many applications. Because communication formats vary depending on the specific application, many attributes of the SCI/LIN are user configurable. The configuration options are:

- SCI Frame format
- SCI Timing modes
- SCI Baud rate
- SCI Multiprocessor modes

15.2.1.1 SCI Frame Formats

The SCI uses a programmable frame format. All frames consist of the following:

- One start bit
- One to eight data bits
- Zero or one address bit
- Zero or one parity bit
- One or two stop bits

The frame format for both the transmitter and receiver is programmable through the bits in the SCIGCR1 register. Both receive and transmit data is in nonreturn to zero (NRZ) format, which means that the transmit and receive lines are at logic high when idle. Each frame transmission begins with a start bit, in which the transmitter pulls the SCI line low (logic low). Following the start bit, the frame data is sent and received least significant bit first (LSB).

An address bit is present in each frame if the SCI is configured to be in address-bit mode but is not present in any frame if the SCI is configured for idle-line mode. The format of frames with and without the address bit is illustrated in Figure 15-3.

A parity bit is present in every frame when the PARITY ENA bit is set. The value of the parity bit depends on the number of one bits in the frame and whether odd or even parity has been selected via the PARITY ENA bit. Both examples in Figure 15-3 have parity enabled.

All frames include one stop bit, which is always a high level. This high level at the end of each frame is used to indicate the end of a frame to ensure synchronization between communicating devices. Two stop bits are transmitted if the STOP bit in SCIGCR1 register is set. The examples shown in Figure 15-3 use one stop bit per frame.

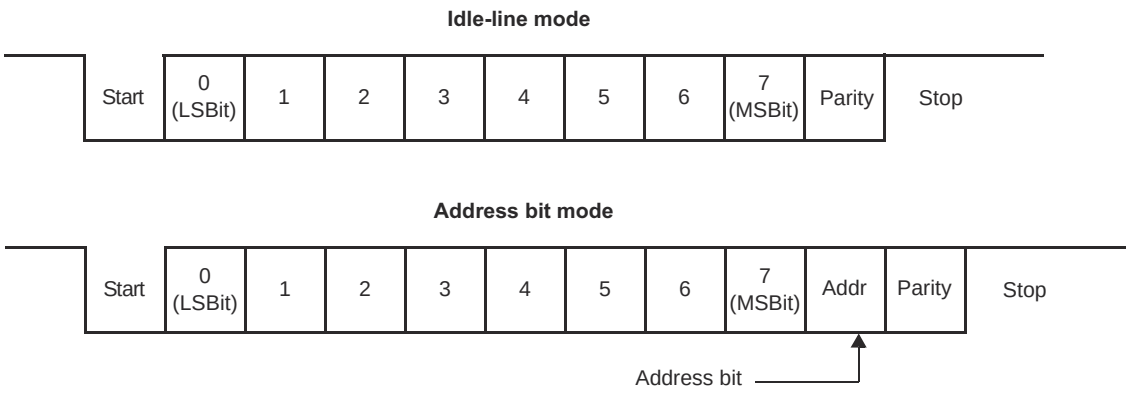


Figure 15-3. Typical SCI Data Frame Formats

15.2.1.2 SCI Asynchronous Timing Mode

The SCI can be configured to use the asynchronous timing mode using TIMING MODE bit in SCIGCR1 register.

The asynchronous timing mode uses only the receive and transmit data lines to interface with devices using the standard universal asynchronous receiver- transmitter (UART) protocol.

In the asynchronous timing mode, each bit in a frame has a duration of 16 SCI baud clock periods. Each bit therefore consists of 16 samples (one for each clock period). When the SCI is using asynchronous mode, the baud rates of all communicating devices must match as closely as possible. Receive errors result from devices communicating at different baud rates.

With the receiver in the asynchronous timing mode, the SCI detects a valid start bit if the first four samples after a falling edge on the LINRX pin are of logic level 0. As soon as a falling edge is detected on LINRX, the SCI assumes that a frame is being received and synchronizes itself to the bus.

To prevent interpreting noise as Start bit SCI expects LINRX line to be low for at least four contiguous SCI baud clock periods to detect a valid start bit. The bus is considered idle if this condition is not met. When a valid start bit is detected, the SCI determines the value of each bit by sampling the LINRX line value during the seventh, eighth, and ninth SCI baud clock periods. A majority vote of these three samples is used to determine the value stored in the SCI receiver shift register. By sampling in the middle of the bit, the SCI reduces errors caused by propagation delays and rise and fall times and data line noises. Figure 15-4 illustrates how the receiver samples a start bit and a data bit in asynchronous timing mode.

The transmitter transmits each bit for a duration of 16 SCI baud clock periods. During the first clock period for a bit, the transmitter shifts the value of that bit onto the LINTX pin. The transmitter then holds the current bit value on LINTX for 16 SCI baud clock periods.

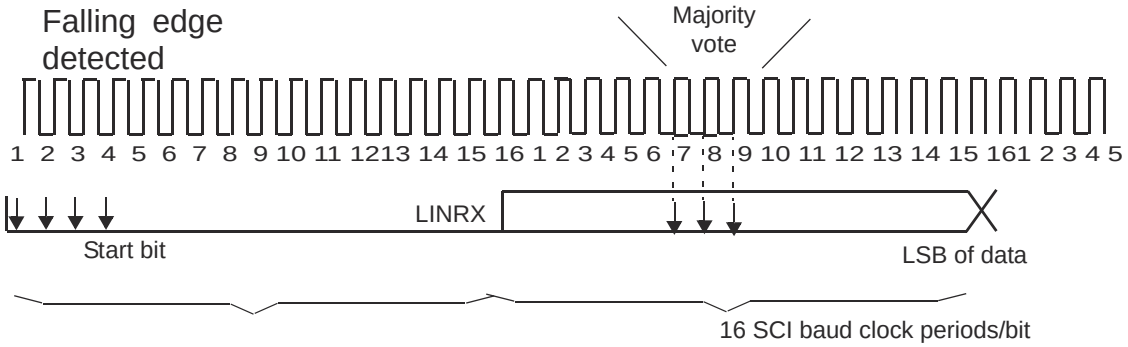


Figure 15-4. Asynchronous Communication Bit Timing

15.2.1.3 SCI Baud Rate

The SCI/LIN has an internally generated serial clock determined by the peripheral VCLK and the prescalers P and M in this register. The SCI uses the 24-bit integer prescaler P value in the BRS register to select the required baud rates. The additional 4-bit fractional divider M refines the baud rate selection.

In asynchronous timing mode, the SCI generates a baud clock according to the following formula:

$$SCICLK\ Frequency = \frac{VCLK\ Frequency}{P + 1 + \frac{M}{16}}$$

$$Asynchronous\ baud\ value = \frac{SCICLK\ Frequency}{16}$$

For P = 0,

$$Asynchronous\ baud\ value = \frac{VCLK\ Frequency}{32} \quad (10)$$

Table 15-1. P and M Values for Standard Bit Rates

60 MHz Device, LIN Module Input Clock = 30 MHz				
Desired Bit Rate	**Actual Bit Rate**	**Percentage Error**	**P Value**	**M Value**
115200	115384.62	0.16	15	4
57600	57581.57	0.03	31	9
38400	38412.29	0.03	47	13
19200	19193.86	0.03	96	11
9600	9600	0	194	5
4800	4800	0	389	10
2400	2400	0	780	4
1200	1200	0	1561	8

15.2.1.4 SCI Multiprocessor Communication Modes

In some applications, the SCI may be connected to more than one serial communication device. In such a multiprocessor configuration, several frames of data may be sent to all connected devices or to an individual device. In the case of data sent to an individual device, the receiving devices must determine when they are being addressed. When a message is not intended for them, the devices can ignore the following data. When only two devices make up the SCI network, addressing is not needed, so multiprocessor communication schemes are not required.

SCI supports two multiprocessor communication modes which can be selected using COMM MODE bit:

- Idle-Line Mode
- Address Bit Mode

When the SCI is not used in a multiprocessor environment, software can consider all frames as data frames. In this case, the only distinction between the idle-line and address-bit modes is the presence of an extra bit (the address bit) in each frame sent with the address-bit protocol.

The SCI allows full-duplex communication where data can be sent and received via the transmit and receive pins simultaneously. However, the protocol used by the SCI assumes that only one device transmits data on the same bus line at any one time. No arbitration is done by the SCI.

15.2.1.4.1 Idle-Line Multiprocessor Modes

In idle-line multiprocessor mode, a frame that is preceded by an idle period (10 or more idle bits) is an address frame. A frame that is preceded by fewer than 10 idle bits is a data frame. Figure 15-5 illustrates the format of several blocks and frames with idle-line mode.

There are two ways to transmit an address frame using idle-line mode:

Method 1: In software, deliberately leave an idle period between the transmission of the last data frame of the previous block and the address frame of the new block.

Method 2: Configure the SCI to automatically send an idle period between the last data frame of the previous block and the address frame of the new block.

Although Method 1 is only accomplished by a delay loop in software, Method 2 can be implemented by using the transmit buffer and the TXWAKE bit in the following manner:

Step 1: Write a 1 to the TXWAKE bit.

Step 2: Write a dummy data value to the SCITD register. This triggers the SCI to begin the idle period as soon as the transmitter shift register is empty.

Step 3: Wait for the SCI to clear the TXWAKE flag.

Step 4: Write the address value to SCITD.

As indicated by Step 3, software should wait for the SCI to clear the TXWAKE bit. However, the SCI clears the TXWAKE bit at the same time it sets TXRDY (that is, transfers data from SCITD into SCITXSHF). Therefore, if the TX INT ENA bit is set, the transfer of data from SCITD to SCITXSHF causes an interrupt to be generated at the same time that the SCI clears the TXWAKE bit. If this interrupt method is used, software is not required to poll the TXWAKE bit waiting for the SCI to clear it.

When idle-line multiprocessor communications are used, software must ensure that the idle time exceeds 10 bit periods before addresses (using one of the methods mentioned above), and software must also ensure that data frames are written to the transmitter quickly enough to be sent without a delay of 10 bit periods between frames. Failure to comply with these conditions will result in data interpretation errors by other devices receiving the transmission.

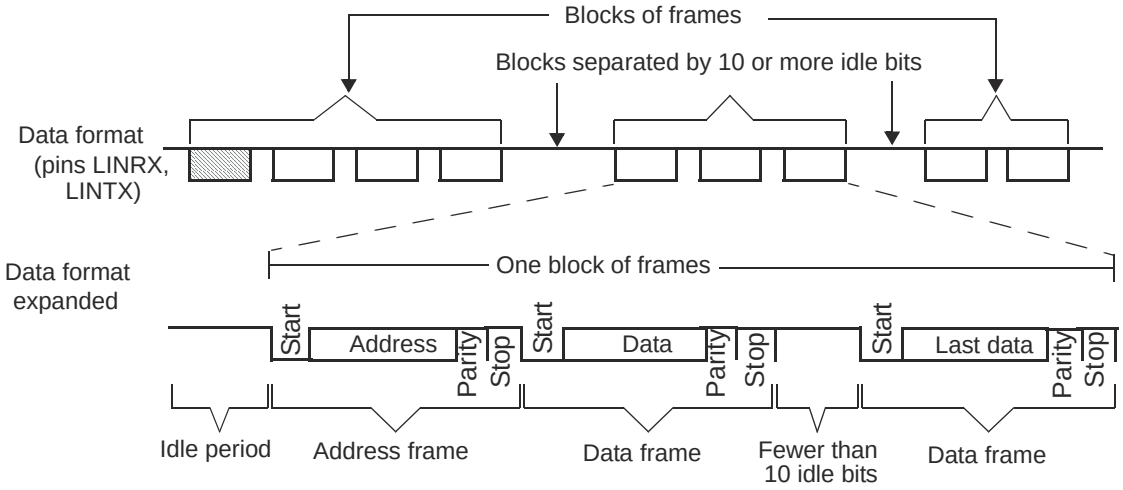


Figure 15-5. Idle-Line Multiprocessor Communication Format

15.2.1.4.2 Address-Bit Multiprocessor Mode

In the address-bit protocol, each frame has an extra bit immediately following the data field called an address bit. A frame with the address bit set to 1 is an address frame; a frame with the address bit set to 0 is a data frame. The idle period timing is irrelevant in this mode. Figure 15-6 illustrates the format of several blocks and frames with the address-bit mode.

When address-bit mode is used, the value of the TXWAKE bit is the value sent as the address bit. To send an address frame, software must set the TXWAKE bit. This bit is cleared as the contents of the SCITD are shifted from the TXWAKE register so that all frames sent are data except when the TXWAKE bit is written as a 1.

No dummy write to SCITD is required before an address frame is sent in address-bit mode. The first byte written to SCITD after the TXWAKE bit is written to 1 is transmitted with the address bit set when address-bit mode is used.

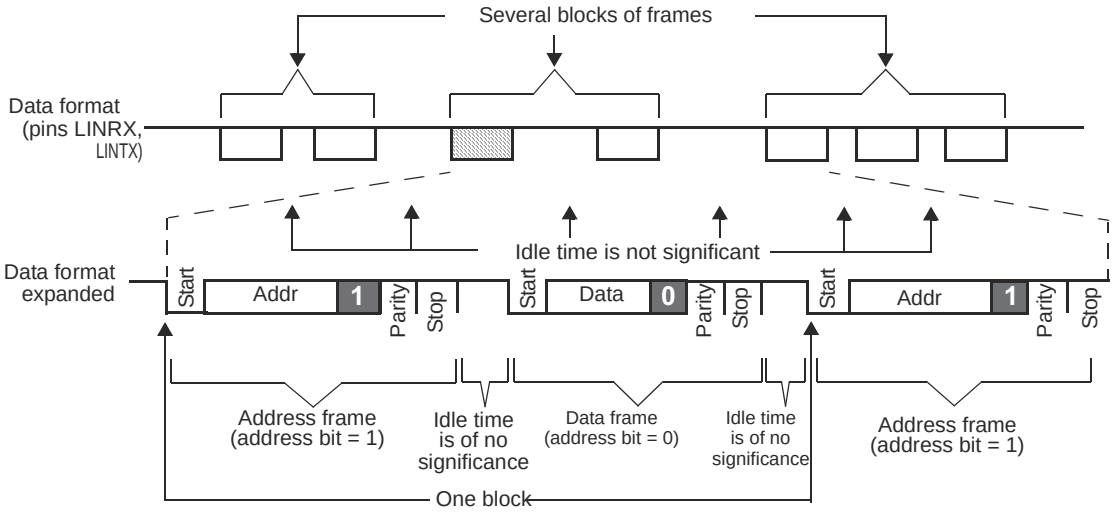


Figure 15-6. Address-Bit Multiprocessor Communication Format

15.2.1.5 SCI Multi Buffered Mode

To reduce CPU load when Receiving or Transmitting data , the SCI/LIN module has eight separate receive and transmit buffers. Multi buffered mode is enabled by setting the MBUF MODE bit.

The multi-buffer 3-bit counter counts the data bytes transferred from the SCIRXSHF register to the RDy receive buffers and TDy transmit buffers register to SCITXSHF register. The 3-bit compare register contains the number of data bytes expected to be received or transmitted. the LENGTH value in SCIFORMAT register indicates the expected length and is used to load the 3-bit compare register.

A receive interrupt (RX interrupt; see the SCIINTVECT0 and SCIINTVECT1registers), and a receive ready RXRDY flag set in SCIFLR register could occur after receiving a response if there are no response receive errors for the frame (such as, there is, frame error, and overrun error).

A transmit interrupt (TX interrupt), and a transmit ready flag (TXRDY flag in SCIFLR register) could occur after transmitting a response.

Figure 15-7 and Figure 15-8 show the receive and transmit multi-buffer functional block diagram, respectively.

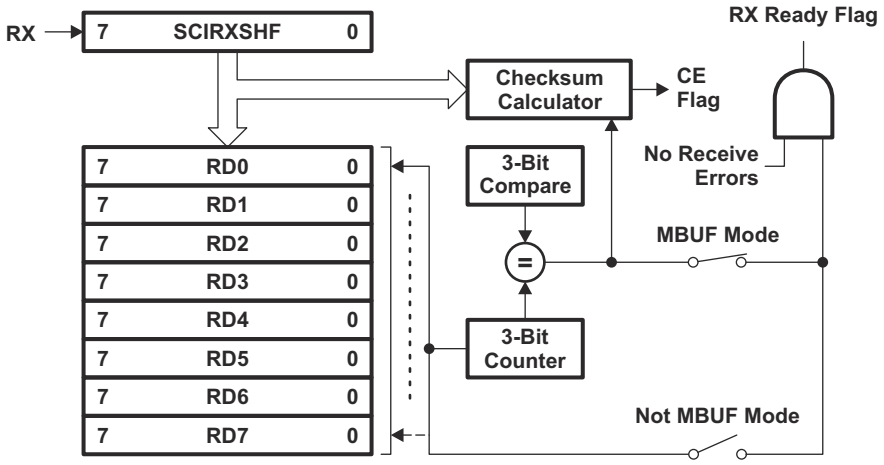


Figure 15-7. Receive Buffers

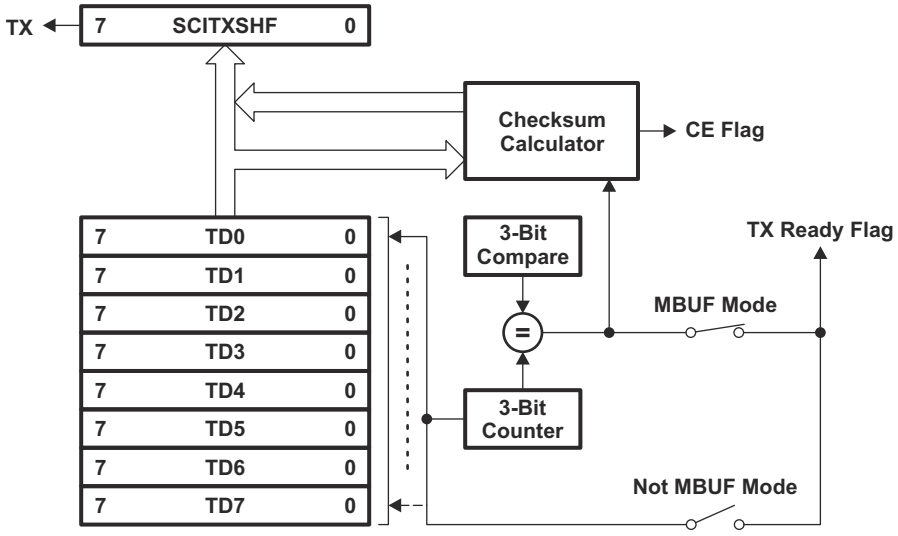


Figure 15-8. Transmit Buffers

15.2.2 SCI Interrupts

The SCI/LIN module has two interrupt lines, level 0 and level 1, to the vectored interrupt manager (VIM) module (see Figure 15-9). Two offset registers SCIINTVECT0 and SCIINTVECT1 determine which flag triggered the interrupt according to the respective priority encoders. Each interrupt condition has a bit to enable/disable the interrupt in the SCISETINT and SCICLRINT registers, respectively.

Each interrupt also has a bit that can be set as interrupt level 0(INT0) or as interrupt level 1(INT1). By default, interrupts are in interrupt level 0. SCISETINTLVL sets a given interrupt to level1. SCICLEARINTLVL resets a given interrupt level to the default level 0.

The interrupt vector registers SCIINTVECT0 and SCIINTVECT1 return the vector of the pending interrupt line INT0 or INT1. If more than one interrupt is pending, the interrupt vector register holds the highest priority interrupt.

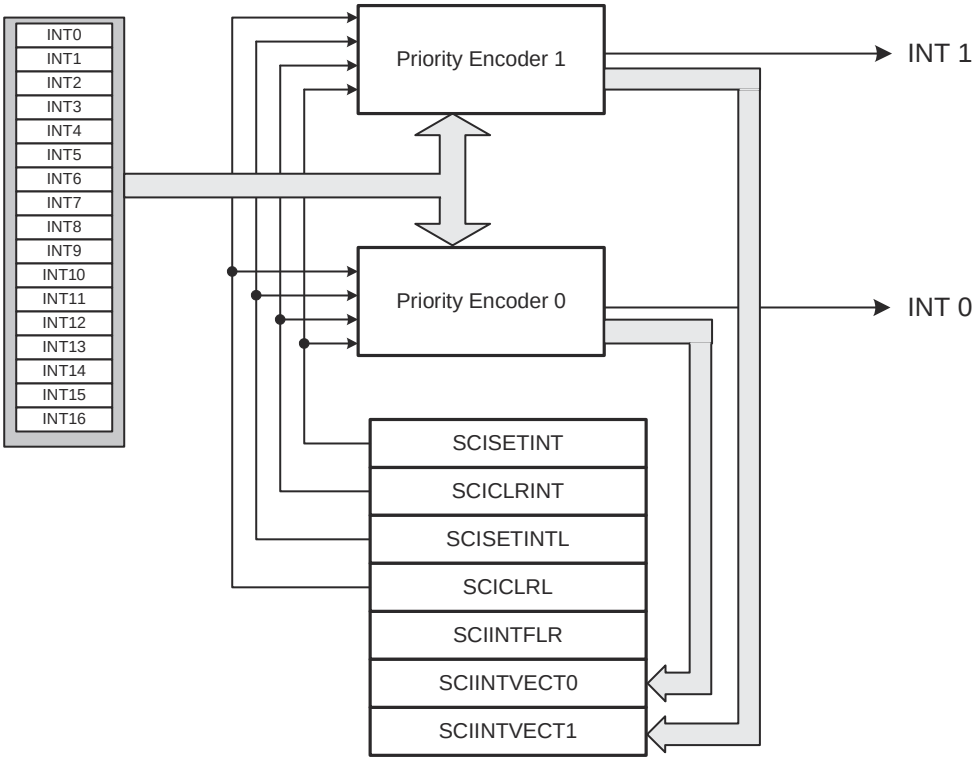


Figure 15-9. General Interrupt Scheme

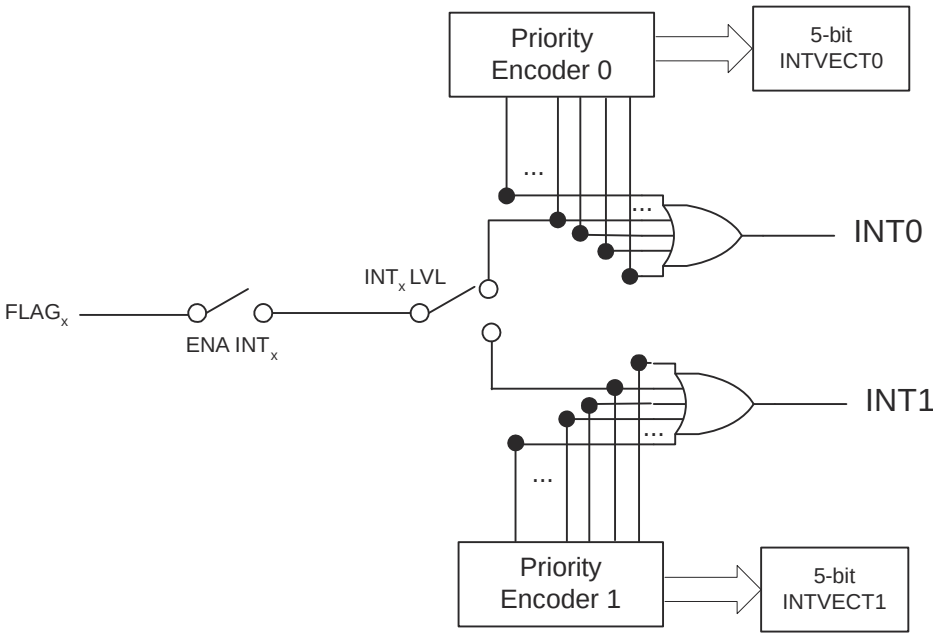


Figure 15-10. Interrupt Generation for Given Flags

15.2.2.1 Transmit Interrupt

To use transmit interrupt functionality, SET TX INT bit must be enabled . The transmit ready (TXRDY) flag is set when the SCI transfers the contents of SCITD to the shift register, SCITXSHF. The TXRDY flag indicates that SCITD is ready to be loaded with more data. In addition, the SCI sets the TX EMPTY bit if both the SCITD and SCITXSHF registers are empty. If the SET TX INT bit is set, then a transmit interrupt is generated when the TXRDY flag goes high. Transmit Interrupt is not generated immediately after setting the SET TX INT . Transmit Interrupt is generated only after the first transfer from SCITD to SCITXSHF, that is first data has to be written to SCITD before any interrupt gets generated. To transmit further data, data can be written to SCITD in the transmit Interrupt service routine.

Writing data to the SCITD register clears the TXRDY bit. When this data has been moved to the SCITXSHF register, the TXRDY bit is set again. The interrupt request can be suspended by setting the CLR TX INT bit; however, when the SET TX INT bit is again set to 1, the TXRDY interrupt is asserted again. The transmit interrupt request can be eliminated until the next series of values is written to SCITD, by disabling the transmitter via the TXENA bit, by a software reset SWnRST, or by a device hardware reset.

15.2.2.2 Receive Interrupt

The receive ready (RXRDY) flag is set when the SCI transfers newly received data from SCIRXSHF to SCIRD. The RXRDY flag therefore indicates that the SCI has new data to be read. Receive interrupts are enabled by the SET RX INT bit. If the SET RX INT is set when the SCI sets the RXRDY flag, then a receive interrupt is generated. The received data can be read in the Interrupt Service routine.

15.2.2.3 WakeUp Interrupt

SCI sets the WAKEUP flag if bus activity on the RX line either prevents power-down mode from being entered, or RX line activity causes an exit from power-down mode. If enabled (SET WAKEUP INT), wakeup interrupt is triggered once WAKEUP flag is set.

15.2.2.4 Error Interrupts

The following error detections are supported with an interrupt by the SCI module:

- Parity errors (PE)
- Frame errors (FE)
- Break Detect errors (BRKDT)
- Overrun errors (OE)
- Bit errors (BE)

If all of these errors (PE, FE, BRKDT, OE, BE) are flagged, an interrupt for the flagged errors will be generated if enabled. A message is valid for both the transmitter and the receiver, if there is no error detected until the end of the frame. Each of these flags is located in the receiver status (SCIFLR) register.

There are 16 interrupt sources in the SCI/LIN module. In SCI mode, 8 interrupts are supported, as listed in Table 15-2.

Table 15-2. SCI/LIN Interrupts

Offset(1)	Interrupt	Applicable to SCI	Applicable to LIN
0	No interrupt	-	-
1	Wakeup	Yes	Yes
2	Inconsistent-sync-field error	No	Yes
3	Parity error	Yes	Yes
4	ID	No	Yes
5	Physical bus error	No	Yes
6	Frame error	Yes	Yes
7	Break detect	Yes	No
8	Checksum error	No	Yes
9	Overrun error	Yes	Yes
10	Bit error (BE)	Yes	Yes
11	Receive	Yes	Yes
12	Transmit	Yes	Yes
13	No-response error	No	Yes
14	Timeout after wakeup signal (150 ms)	No	Yes
15	Timeout after three wakeup signals (1.5 s)	No	Yes
16	Timeout (Bus Idle, 4s)	No	Yes

(1) Offset 1 is the highest priority. Offset 16 is the lowest priority.

Table 15-3. SCI Receiver Status Flags

SCI Flag	Register	Bit	Value After Reset(1)
CE	SCIFLR	29	0
ISFE	SCIFLR	28	0
NRE	SCIFLR	27	0
FE	SCIFLR	26	0
OE	SCIFLR	25	0
PE	SCIFLR	24	0
RXWAKE	SCIFLR	12	0
RXRDY	SCIFLR	9	0
BUSY	SCIFLR	3	0
IDLE	SCIFLR	2	1
WAKEUP	SCIFLR	1	0
BRKDT	SCIFLR	0	0

(1) The flags are frozen with their reset value while SWnRST = 0.

Table 15-4. SCI Transmitter Status Flags

SCI Flag	Register	Bit	Value After Reset(1)
BE	SCIFLR	31	0
PBE	SCIFLR	30	0
TXWAKE	SCIFLR	10	0
TXEMPTY	SCIFLR	11	1
TXRDY	SCIFLR	8	1

(1) The flags are frozen with their reset value while SWnRST = 0.

15.2.3 SCI Configurations

Before the SCI sends or receives data, its registers should be properly configured. Upon power-up or a system-level reset, each bit in the SCI registers is set to a default state. The registers are writable only after the RESET bit in the SCIGCR0 register is set to 1. Of particular importance is the SWnRST bit in the SCIGCR1 register. The SWnRST is an active-low bit initialized to 0 and keeps the SCI in a reset state until it is programmed to 1. Therefore, all SCI configuration should be completed before a 1 is written to the SWnRST bit.

The following list details the configuration steps that software should perform prior to the transmission or reception of data. As long as the SWnRST bit is cleared to 0 the entire time that the SCI is being configured, the order in which the registers are programmed is not important.

- Enable SCI by setting the RESET bit to 1.
- Clear the SWnRST bit to 0 before SCI is configured.
- Select the desired frame format by programming the SCIGCR1 register.
- Select the baud rate to be used for communication by programming the BRS register.
- Set the CLOCK bit in SCIGCR1 to 1 to select the internal clock.
- Set the CONT bit in SCIGCR1 to 1 to make SCI not halt for an emulation breakpoint until its current reception or transmission is complete (this bit is used only in an emulation environment).
- Set the LOOP BACK bit in SCIGCR1 to 1 to connect the transmitter to the receiver internally (this feature is used to perform a self-test).
- Set the RXENA bit in SCIGCR1 to 1, if data is to be received.
- Set the TXENA bit in SCIGCR1 to 1, if data is to be transmitted.
- Set the SWnRST bit to 1 after SCI is configured.
- Perform receiving or transmitting data (see Section 15.2.3.1 or Section 15.2.3.2).

15.2.3.1 Receiving Data

SCI module can receive data in one of the following modes:

- Single-Buffer (Normal) Mode
- Multi-Buffer Mode

After a valid idle period is detected, data is automatically received as it arrives on the LINRX pin.

15.2.3.1.1 Receiving Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit in SCIGCR1 is cleared to 0. In this mode, SCI sets the RXRDY bit when it transfers newly received data from SCIRXSHF to SCIRD. The SCI clears the RXRDY bit after the new data in SCIRD has been read. Also, as data is transferred from SCIRXSHF to SCIRD, the SCI sets the FE, OE, or PE flags if any of these error conditions were detected in the received data. These error conditions are supported with configurable interrupt capability. The wake-up and break-detect status bits are also set if one of these errors occurs, but they do not necessarily occur at the same time that new data is being loaded into SCIRD.

You can receive data by:

1. Polling Receive Ready Flag
2. Receive Interrupt

In polling method, software can poll for the RXRDY bit and read the data from the SCIRD register once the RXRDY bit is set high. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use interrupts. To use the interrupt method, the SET RX INT bit is set. An interrupt is generated the moment the RXRDY bit is set.

15.2.3.1.2 Receiving Data in Multi-Buffer Mode

Multi-buffer mode is selected when the MBUFMODE bit in SCIGCR1 is set to 1. In this mode, SCI sets the RXRDY bit after receiving the programmed number of data in the receive buffer, the complete frame. The error condition detection logic is similar to the single-buffer mode, except that it monitors for the complete frame. Like single-buffer mode, you can use the polling or interrupt method to read the data. The SCI clears the RXRDY bit after the new data in SCIRD has been read.

15.2.3.2 Transmitting Data

The SCI transmitter is enabled if both the TX FUNC bit and the TXENA bit are set to 1. If the TX FUNC bit is not set, the LINTX pin functions as a general-purpose I/O pin rather than as an SCI function pin. Any value written to the SCITD before TXENA is set to 1 is not transmitted. Both of these control bits allow for the SCI transmitter to be held inactive independently of the receiver.

SCI module can transmit data in one of the following modes:

- Single-Buffer (Normal) Mode
- Multi-Buffered or Buffered SCI Mode

15.2.3.2.1 Transmitting Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit in SCIGCR1 is cleared to 0. In this mode, SCI waits for data to be written to SCITD, transfers it to SCITXSHF, and transmits the data. The TXRDY and TX EMPTY bits indicate the status of the transmit buffers. That is, when the transmitter is ready for data to be written to SCITD, the TXRDY bit is set. Additionally, if both SCITD and SCITXSHF are empty, then the TX EMPTY bit is also set.

You can transmit data by:

1. Polling Transmit Ready Flag
2. Transmit Interrupt

In polling method, software can poll for the TXRDY bit to go high before writing the data to the SCITD register. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use the interrupt method. To use the interrupt method, the SET TX INT bit is set. An interrupt is generated the moment the TXRDY bit is set. When the SCI has completed transmission of all pending frames, the SCITXSHF register and SCITD are empty, the TXRDY bit is set, and an interrupt request is generated, if enabled. Because all data has been transmitted, the interrupt request should be halted. This can either be done by disabling the transmit interrupt (CLR TX INT) or by disabling the transmitter (clear TXENA bit).

Note

The TXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0 or SCIINTVECT1 register.

15.2.3.2.2 Transmitting Data in Multi-Buffer Mode

Multi-buffer mode is selected when the MBUF MODE bit in SCIGCR1 is set to 1. Like single-buffer mode, you can use the polling or interrupt method to write the data to be transmitted. The transmitted data has to be written to the SCITD registers. SCI waits for data to be written to the SCITD register and transfers the programmed number of bytes to SCITXSHF to transmit one by one automatically.

15.2.4 SCI Low-Power Mode

The SCI/LIN can be put in either local or global low-power mode. Global low-power mode is asserted by the system and is not controlled by the SCI/LIN. During global low-power mode, all clocks to the SCI/LIN are turned off so the module is completely inactive.

Local low-power mode is asserted by setting the POWERDOWN bit; setting this bit stops the clocks to the SCI/LIN internal logic and the module registers. Setting the POWERDOWN bit causes the SCI to enter local low-power mode and clearing the POWERDOWN bit causes SCI/LIN to exit from local low-power mode. All the registers are accessible during local power-down mode as any register access enables the clock to SCI for that particular access alone.

The wake-up interrupt is used to allow the SCI to exit low-power mode automatically when a low level is detected on the LINRX pin and also this clears the POWERDOWN bit. If wake-up interrupt is disabled, then the SCI/LIN immediately enters low-power mode whenever it is requested and also any activity on the LINRX pin does not cause the SCI to exit low-power mode.

Note

Enabling Local Low-Power Mode During Receive and Transmit

If the wake-up interrupt is enabled and low-power mode is requested while the receiver is receiving data, then the SCI immediately generates a wake-up interrupt to clear the powerdown bit and prevents the SCI from entering low-power mode and thus completes the current reception. Otherwise, if the wake-up interrupt is disabled, then the SCI completes the current reception and then enters the low-power mode.

15.2.4.1 Sleep Mode for Multiprocessor Communication

When the SCI receives data and transfers that data from SCIRXSHF to SCIRD, the RXRDY bit is set and if RX INT ENA is set, the SCI also generates an interrupt. The interrupt triggers the CPU to read the newly received frame before another one is received. In multiprocessor communication modes, this default behavior may be enhanced to provide selective indication of new data. When SCI receives an address frame that does not match its address, the device can ignore the data following this non-matching address until the next address frame by using sleep mode. Sleep mode can be used with both idle-line and address-bit multiprocessor modes.

If sleep mode is enabled by the SLEEP bit, then the SCI transfers data from SCIRXSHF to SCIRD only for address frames. Therefore, in sleep mode, all data frames are assembled in the SCIRXSHF register without being shifted into the SCIRD and without initiating a receive interrupt request. Upon reception of an address frame, the contents of the SCIRXSHF are moved into SCIRD, and the software must read SCIRD and determine if the SCI is being addressed by comparing the received address against the address previously set in the software and stored somewhere in memory (the SCI does not have hardware available for address comparison). If the SCI is being addressed, the software must clear the SLEEP bit so that the SCI will load SCIRD with the data of the data frames that follow the address frame.

When the SCI has been addressed and sleep mode has been disabled (in software) to allow the receipt of data, the SCI should check the RXWAKE bit (SCIFLR.12) to determine when the next address has been received. This bit is set to 1 if the current value in SCIRD is an address and set to 0 if SCIRD contains data. If the RXWAKE bit is set, then software should check the address in SCIRD against its own address. If it is still being addressed, then sleep mode should remain disabled. Otherwise, the SLEEP bit should be set again.

Following is a sequence of events typical of sleep mode operation:

- The SCI is configured and both sleep mode and receive actions are enabled.
- An address frame is received and a receive interrupt is generated.
- Software compares the received address frame against that set by software and determines that the SCI is not being addressed, so the value of the SLEEP bit is not changed.
- Several data frames are shifted into SCIRXSHF, but no data is moved to SCIRD and no receive interrupts are generated.
- A new address frame is received and a receive interrupt is generated.
- Software compares the received address frame against that set by software and determines that the SCI is being addressed and clears the SLEEP bit.
- Data shifted into SCIRXSHF is transferred to SCIRD, and a receive interrupt is generated after each data frame is received.
- In each interrupt routine, software checks RXWAKE to determine if the current frame is an address frame.
- Another address frame is received, RXWAKE is set, software determines that the SCI is not being addressed and sets the SLEEP bit back to 1. No receive interrupts are generated for the data frames following this address frame.

By ignoring data frames that are not intended for the device, fewer interrupts are generated. These interrupts would otherwise require CPU intervention to read data that is of no significance to this specific device. Using sleep mode can help free some CPU resources.

Except for the RXRDY flag, the SCI continues to update the receiver status flags (see Table 15-3) while sleep mode is active. In this way, if an error occurs on the receive line, an application can immediately respond to the error and take the appropriate corrective action.

Because the RXRDY bit is not updated for data frames when sleep mode is enabled, the SCI can enable sleep mode and use a polling algorithm if desired. In this case, when RXRDY is set, software knows that a new address has been received. If the SCI is not being addressed, then the software should not change the value of the SLEEP bit and should continue to poll RXRDY.

15.3 Local Interconnect Network Module

15.3.1 LIN Communication Formats

The SCI/LIN module can be used in LIN mode or SCI mode. The enhancements for baud generation and additional receive/transmit buffers necessary for LIN mode operation are also part of the enhanced buffered SCI module. LIN mode is selected by enabling LIN MODE bit in SCIGCR1 register.

Note

The SCI/LIN is built around the SCI platform and uses a similar sampling scheme: 16 samples for each bit with majority vote on samples 8, 9, and 10. For the START bit, the first three samples are used.

The SCI/LIN control registers are located at the SCI/LIN base address. For a detailed description of each register, see Section 15.7.

15.3.1.1 LIN Standards

For compatibility with LIN2.0 standard the following additional features are implemented over LIN1.3:

1. Support for LIN 2.0 checksum
2. Enhanced synchronizer FSM support for frame processing
3. Enhanced handling of extended frames
4. Enhanced baud rate generator
5. Update wakeup/go to sleep

The LIN module covers the CPU performance-consuming features, defined in the *LIN Specification Package* Revision 1.3 and 2.0 by hardware.

15.3.1.2 Message Frame

The LIN protocol defines a message frame format, illustrated in Figure 15-11. Each frame includes one master header, one response, one in-frame response space, and inter-byte spaces. In-frame-response and inter-byte spaces may be 0.

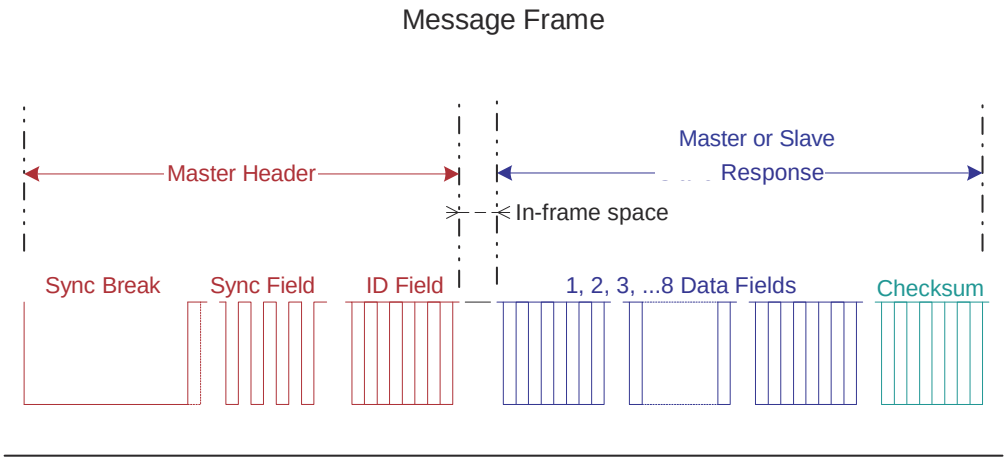


Figure 15-11. LIN Protocol Message Frame Format: Master Header and Slave Response

There is no arbitration in the definition of the LIN protocol; therefore, multiple slave nodes responding to a header might be detected as an error.

The LIN bus is a single channel wired-AND. The bus has a binary level: either dominant for a value of 0, or recessive for a value of 1.

15.3.1.2.1 Message Header

The header of a message is initiated by a master (see Figure 15-12) and consists of a three field-sequence:

- The sync break field signaling the beginning of a message
- The sync field conveying bit rate information of the LIN bus
- The ID field denoting the content of a message

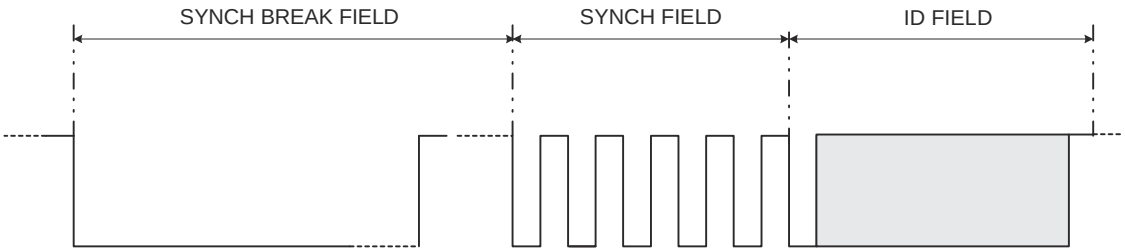


Figure 15-12. Header 3 Fields: Sync Break, Sync, and ID

15.3.1.2.2 Response

The format of the response is as illustrated in Figure 15-13. There are two types of fields in a response: data and checksum. The data field consists of exactly one data byte, one start bit, and one stop bit, for a total of 10 bits. The LSB is transmitted first. The checksum field consists of one checksum byte, one start bit and one stop bit. The checksum byte is the inverted modulo-256 sum over all data bytes in the data fields of the response.

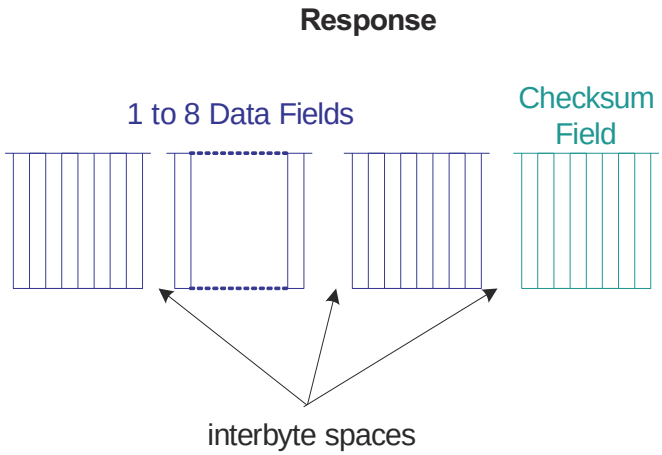


Figure 15-13. Response Format of LIN Message Frame

The format of the response is a stream of N data fields and one checksum field. Typically N is from 1 to 8, with the exception of the extended command frames (Section 15.3.1.6). The length N of the response is indicated either with the optional length control bits of the ID Field (this is used in standards earlier than LIN 1.x); see Table 15-5, or by LENGTH value in SCIFORMAT[18:16] register; see Table 15-6. The SCI/LIN module supports response lengths from 1 to 8 bytes in compliance with LIN 2.0.

Table 15-5. Response Length Info Using IDBYTE Field Bits [5:4] for LIN Standards Earlier than v1.3

ID5	ID4	Number of Data Bytes
0	0	2
0	1	2
1	0	4
1	1	8

Table 15-6. Response Length with SCIFORMAT[18:16] Programming

SCIFORMAT[18:16]	Number of Bytes
000	1
001	2
010	3
011	4
100	5
101	6
110	7
111	8

15.3.1.3 Synchronizer

The synchronizer has three major functions in the messaging between master and slave nodes. It generates the master header data stream, it synchronizes to the LIN bus for responding, and it locally detects timeouts. A bit rate is programmed using the prescalers in the BRSR register to match the indicated LIN_speed value in the LIN description file.

The LIN synchronizer will perform the following functions: master header signal generation, slave detection and synchronization to message header with optional baud rate adjustment, response transmission timing and timeout control.

The LIN synchronizer is capable of detecting an incoming break and initializing communication at all times.

15.3.1.4 Baud Rate

The LIN module is clocked at a frequency of one-half the CPU clock (SYSCLKOUT), that is, LIN Module input clock (LM_CLK or VCLK) = SYSCLKOUT/2. For a 60 MHz device, LM_CLK = 30 MHz.

The transmission baud rate of any node is configured by the CPU at the beginning; this defines the bit time T_{bit}. The bit time is derived from the fields P and M in the baud rate selection register (BRSR).

The ranges for the prescaler values in the BRSR register are:

P = 0, 1, 2, 3, . . . , 2^{24} - 1

M = 0, 1, 2, . . . , 15

The P and M values in the BRSR register are user programmable. The P and M dividers could be used for both SCI mode and LIN mode to select a baud rate. If the ADAPT bit is set and the LIN slave is in adaptive baud rate mode, then all these divider values are automatically obtained during header reception when the synchronization field is measured.

The LIN protocol defines baud rate boundaries as follows:

1kHz ≤ F_{LINCLK} ≤ 20kHz

All transmitted bits are shifted in and out at T_{bit} periods.

15.3.1.4.1 Fractional Divider

The M field of the BRSR register modifies the integer prescaler P for fine tuning of the baud rate. The M value adds in increments of 1/16 of the P value.

The bit time, T_{bit} is expressed in terms of the VCLK period T_{VCLK} as follows:

For all P other than 0, and all M,

$$T_{bit} = 16\left(P + 1 + \frac{M}{16}\right)T_{VCLK}$$

For P= 0 : $T_{bit} = 32T_{VCLK}$

Therefore, the LINCLK frequency is given by:

$$F_{LINCLK} = \frac{F_{VCLK}}{16(P + 1 + \frac{M}{16})} \quad \text{For all } P \text{ other than zero}$$

$$F_{LINCLK} = \frac{F_{VCLK}}{32} \quad \text{For } P = 0$$

15.3.1.5 Header Generation

Automatic generation of the LIN protocol header data stream is supported without CPU interaction. The CPU will trigger a message header generation and the LIN state machine will handle the generation itself. A master node initiates header generation on CPU writes to the IDBYTE in the LINID register. The header is always sent by the master to initiate a LIN communication and consists of three fields: break field, synchronization field, and identification field, as seen in Figure 15-14.

Note

The LIN protocol uses the parity bits in the identifier. The control length bits are optional to the LIN protocol.

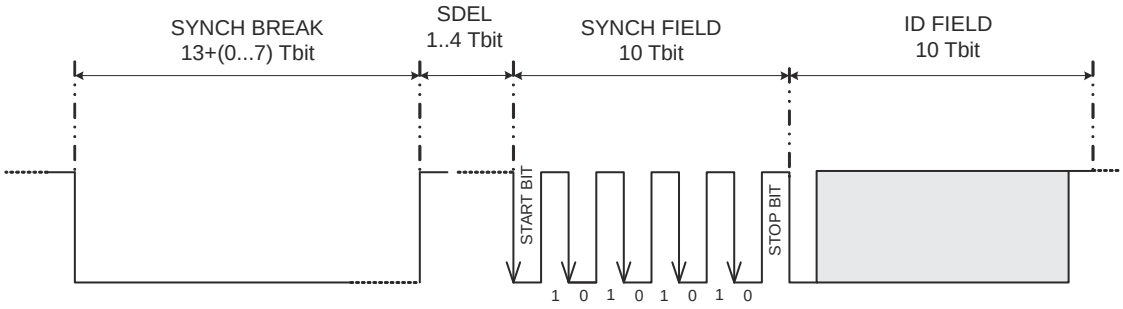


Figure 15-14. Message Header in Terms of T_{bit}

- The break field consists of two components:
  - The synchronization break (SYNC BREAK) consists of a minimum of 13 (dominant) low bits to a maximum of 20 dominant bits. The sync break length may be extended from the minimum with the 3-bit SBREAK value in the LINCOMP register.
  - The synchronization break delimiter (SDEL) consists of a minimum of 1 (recessive) high bit to a maximum of 4 recessive bits. The delimiter marks the end of the synchronization break field. The sync break delimiter length depends on the 2-bit SDEL value in the LINCOMP register.
- The synchronization field (SYNC FIELD) consists of one start bit, byte 0x55, and a stop bit. It is used to convey T_{bit} information and resynchronize LIN bus nodes.
- The identifier field's ID byte may use six bits as an identifier, with optional length control and two optional bits as parity of the identifier. The identifier parity is used and checked if the PARITY ENA bit is set. If length control bits are not used, then there can be a total of 64 identifiers plus parity. If neither length control or parity are used there can be up to 256 identifiers. See Figure 15-15 for an illustration of the ID field.

Note

Optional Control Length Bits

The control length bits only apply to LIN standards prior to LIN 1.3. IDBYTE field conveys response length information if compliant to standards earlier than LIN1.3. The SCIFORMAT register stores the length of the response for later versions of the LIN protocol.

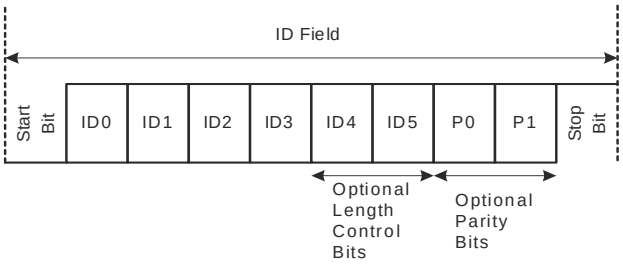


Figure 15-15. ID Field

Note

If the LIN module, configured as Slave in multi-buffer mode, is in the process of transmitting data while a new header comes in, the module might end up in responding with the data from the previous interrupted response (not the data corresponding to the new ID). To avoid this scenario the following procedure could be used:

1. Check for the Bit Error (BE) during the response transmission. If the BE flag is set, this indicates that a collision has happened on the LIN bus (here because of the new Sync Break).
2. In the Bit Error ISR, configure the TD0 and TD1 registers with the next set of data to be transmitted on a TX Match for the incoming ID. Before writing to TD0/TD1 make sure that there was not already an update because of a Bit Error; otherwise TD0/TD1 might be written twice for one ID.
3. Once the complete ID is received, based on the match, the newly configured data will be transmitted by the node.

15.3.1.5.1 Event Triggered Frame Handling

The LIN 2.0 protocol uses event-triggered frames that may occasionally cause collisions. Event-triggered frames have to be handled in software.

If no slave answers to an event triggered frame header, the master node will set the NRE flag, and a NRE interrupt will occur if enabled. If a collision occurs, a frame error and checksum error may arise before the NRE error. Those errors are flagged and the appropriate interrupts will occur, if enabled.

Frame errors and checksum errors depend on the behavior and synchronization of the responding slaves. If the slaves are totally synchronized and stop transmission once the collision occurred, it is possible that only the NRE error is flagged despite the occurrence of a collision. To detect if there has been a reception of one byte before the NRE error is flagged, the BUS BUSY flag can be used as an indicator.

The bus busy flag is set on the reception of the first bit of the header and remains set until the header reception is complete, and again is set on the reception of the first bit of the response. In the case of a collision the flag is cleared in the same cycle as the NRE flag is set.

Software could implement the following sequence:

- Once the reception of the header is done (poll for RXID flag), wait for the bus busy flag to get set or NRE flag to get set.
- If bus busy flag is not set before NRE flag, then it is a true no response case (no data has been transmitted onto the bus).
- If bus busy flag gets set, then wait for NRE flag to get set or for successful reception. If NRE flag is set, then in this case a collision has occurred on the bus.

Even in the case of a collision, the received (corrupted) data is accessible in the RX buffers; registers LINRD0 and LINRD1.

15.3.1.5.2 Header Reception and Adaptive Baud Rate

A slave node baud rate can optionally be adjusted to the detected bit rate as an option to the LIN module. The adaptive baud rate option is enabled by setting the ADAPT bit. During header reception, a slave measures the baud rate during detection of the sync field. If ADAPT bit is set, then the measured baud rate is compared to the slave node's programmed baud rate and adjusted to the LIN bus baud rate if necessary.

The LIN synchronizer determines two measurements: BRK_count and BAUD_count (Figure 15-16). These values are always calculated during the Header reception for sync field validation (Figure 15-17).

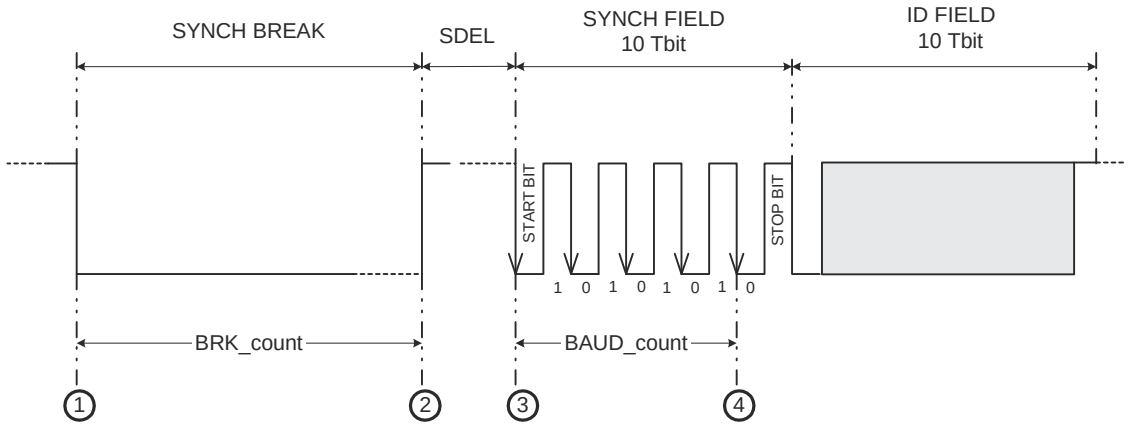


Figure 15-16. Measurements for Synchronization

By measuring the values BRK_count and BAUD_count, a valid sync break sequence can be detected as described in Figure 15-17. The four numbered events in Figure 15-16 signal the start/stop of the synchronizer counter. The synchronizer counter uses VCLK as the time base.

The synchronizer counter is used to measure the sync break relative to the detecting node T_{bit}. For a slave node receiving the sync break, a threshold of 11 T_{bit} is used as required by the LIN protocol. For detection of the dominant data stream of the sync break, the synchronizer counter is started on a falling edge and stopped on a rising edge of the LINRX. On detection of the sync break delimiter, the synchronizer counter value is saved and then reset.

On detection of five consecutive falling edges, the BAUD_count is measured. Bit timing calculation and consistency to required accuracy is implemented following the recommendations of LIN revision 2.0. A slave node can calculate a single T_{bit} time by division of BAUD_count by 8. In addition, for consistency between the detected edges the following is evaluated:

BAUD_count + BAUD_count » 2 + BAUD_count » 3 ≤ BRK_count

The BAUD_count value is shifted 3 times to the right and rounded using the first insignificant bit to obtain a T_{bit} unit. If the ADAPT bit is set, then the detected baud rate is compared to the programmed baud rate.

During the header reception processing as illustrated in Figure 15-17, if the measured BRK_count value is less than 11 T_{bit}, the sync break is not valid according to the protocol for a fixed rate. If the ADAPT bit is set, then the MBRS register is used for measuring BRK_count and BAUD_count values and automatically adjusts to any allowed LIN bus rate (refer to *LIN Specification Package 2.0*).

Note

In adaptive mode the MBRS divider should be set to allow a maximum baud rate that is not more than 10% above the expected operating baud rate in the LIN network. Otherwise, a 0x00 data byte could mistakenly be detected as a sync break.

The break-threshold relative to the slave node is 11 T_{bit}. The break is 13 T_{bit} as specified in LIN v1.3.

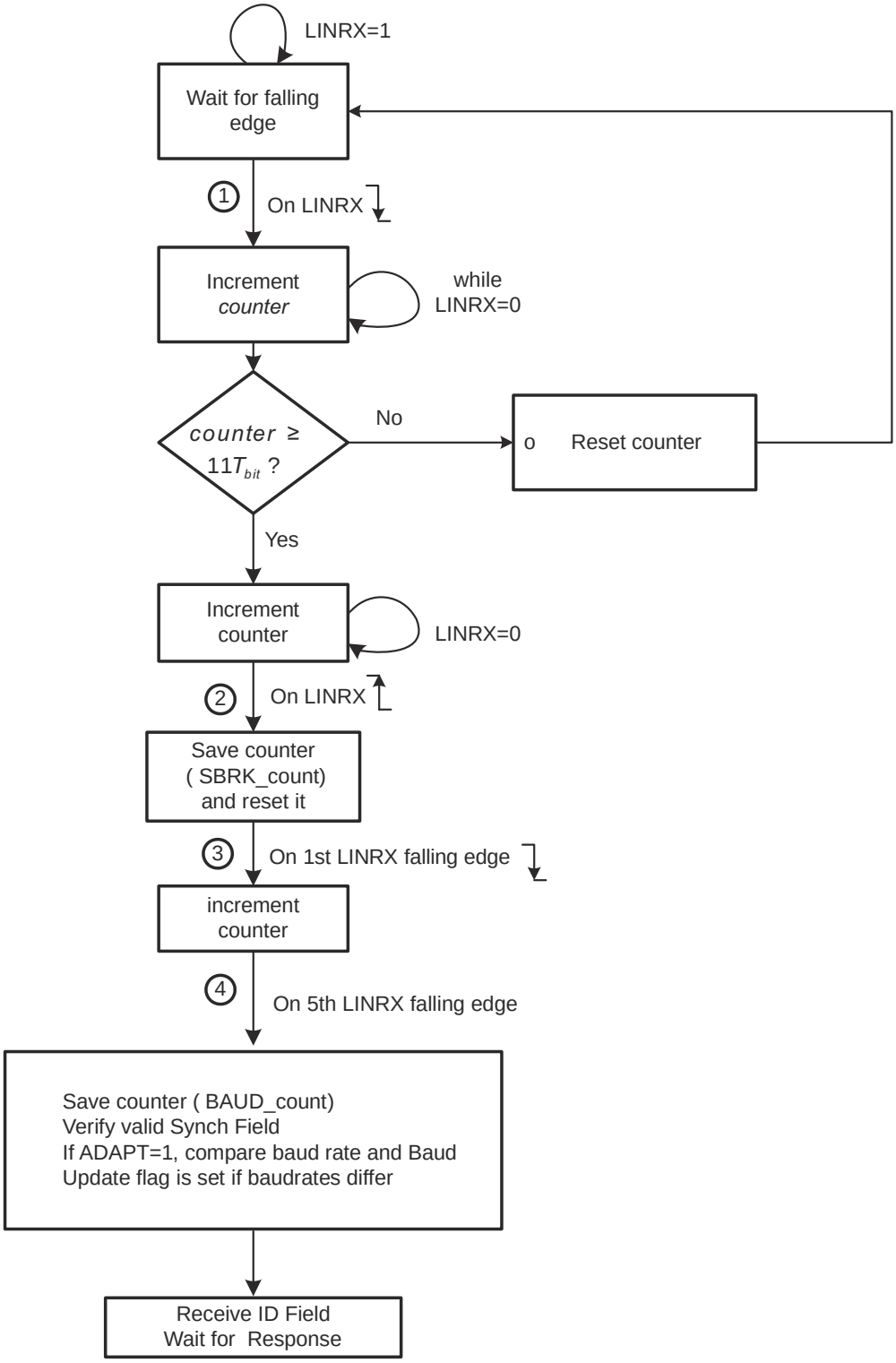


Figure 15-17. Synchronization Validation Process and Baud Rate Adjustment

If the sync field is not detected within the given tolerances, the inconsistent-sync-field-error (ISFE) flag will be set. An ISFE interrupt will be generated, if enabled by its respective bit in the SCISETINT register. The ID byte should be received after the sync field validation was successful. Any time a valid break (larger than 11 T_{bit}) is detected, the receiver's state machine should reset to reception of this new frame. This reset condition is only valid during response state, not if an additional sync break occurs during header reception.

Note

When an inconsistent sync field (ISFE) error occurs, suggested action for the application is to Reset the SWnRST bit and set the SWnRST bit to make sure that the internal state machines are back to their normal states.

15.3.1.6 Extended Frames Handling

The LIN protocol 2.0 and prior includes two extended frames with identifiers 62 (user-defined) and 63 (reserved extended). The response data length of the user-defined frame (ID 62, or 0x3E) is unlimited. The length for this identifier will be set at network configuration time to be shared with the LIN bus nodes.

Extended frame communication is triggered on reception of a header with identifier 0x3E; see Figure 15-18. Once the extended frame communication is triggered, unlike normal frames, this communication needs to be stopped before issuing another header. To stop the extended frame communication the STOP EXT FRAME bit must be set.

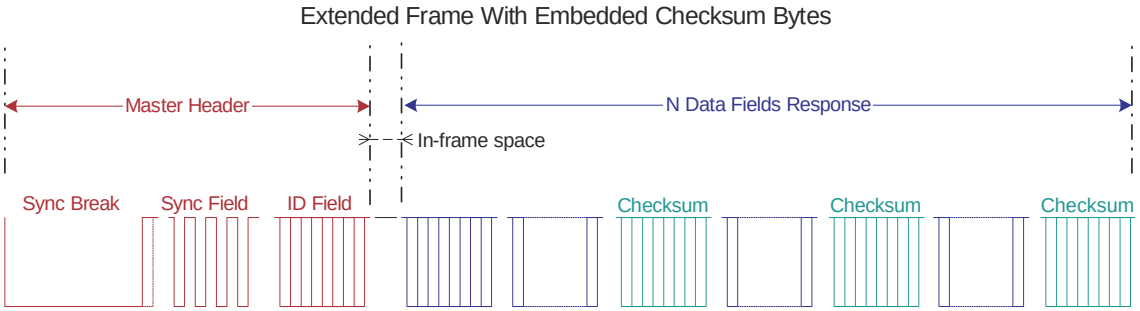


Figure 15-18. Optional Embedded Checksum in Response for Extended Frames

An ID interrupt will be generated (if enabled and there is a match) on reception of ID 62 (0x3E). This interrupt allows the CPU using a software counter to keep track of the bytes that are being sent out and decides when to calculate and insert a checksum byte (recommended at periodic rates). To handle this procedure, SC bit is used. A write to the send checksum bit SC will initiate an automatic send of the checksum byte. The last data field should always be a checksum in compliance with the LIN protocol.

The periodicity of the checksum insertion, defined at network configuration time, is used by the receiving node to evaluate the checksum of the ongoing message, and has the benefit of enhanced reliability.

For the sending node, the checksum is automatically embedded each time the send checksum bit SC is set. For the receiving node, the checksum is compared each time the compare checksum bit CC is set; see Figure 15-19.

Note

The LIN 2.0 enhanced checksum does not apply to the reserved identifiers. The reserved identifiers always use the classic checksum.

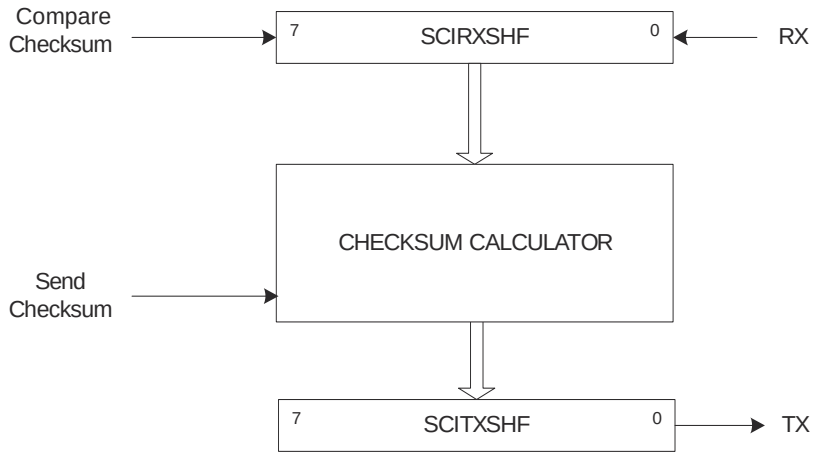


Figure 15-19. Checksum Compare and Send for Extended Frames

15.3.1.7 Timeout Control

Any LIN node listening to the bus and expecting a response initiated from a master node could flag a no-response error timeout event. The LIN protocol defines four types of timeout events, which are all handled by the hardware of the LIN module. The four LIN protocol events are:

- No-response timeout error
- Bus idle detection
- Timeout after wakeup signal
- Timeout after three wakeup signals

15.3.1.7.1 No-Response Error (NRE)

The no-response error will occur when any node expecting a response waits for T_{FRAME_MAX} time and the message frame is not fully completed within the maximum length allowed, T_{FRAME_MAX}. After this time a no-response error (NRE) is flagged in the NRE bit of the SCIFLR register. An interrupt is triggered if enabled.

As specified in the LIN 1.3 standard, the minimum time to transmit a frame is:

$T_{FRAME\_MIN} = T_{HEADER\_MIN} + T_{DATA\_FIELD} + T_{CHECKSUM\_FIELD} = 44 + 10N$

where N = number of data fields.

And the maximum time frame is given by:

$T_{FRAME\_MAX} = T_{FRAME\_MIN} * 1.4 = (44 + 10N) * 1.4$

The timeout value T_{FRAME_MAX} is derived from the *N* number of data fields value see Table 15-7. The *N* value is either embedded in the header's ID field for messages or is part of the description file. In the latter case, the 3-bit CHAR value in SCIFORMAT register, will indicate the value for *N*.

Note

The length coding of the ID field does not apply to two extended frame identifiers, ID fields of 0x3E (62) and 0x3F (63). In these cases, the ID field can be followed by an arbitrary number of data byte fields. Also, the LIN 2.0 protocol specification mentions that ID field 0x3F (63) cannot be used. For these two cases, the NRE will not be handled by the LIN controller hardware.

Table 15-7. Timeout Values in T_{bit} Units

N	T_{DATA_FIELD}	T_{FRAME_MIN}	T_{FRAME_MAX}
1	10	54	76
2	20	64	90
3	30	74	104
4	40	84	118
5	50	94	132
6	60	104	146
7	70	114	160
8	80	124	174

15.3.1.7.2 Bus Idle Detection

The second type of timeout can occur when a node detects an inactive LIN bus: no transitions between recessive and dominant values are detected on the bus. This happens after a minimum of 4 s (this is 80,000 F_{LINCLK} cycles with the fastest bus rate of 20 kbps). If a node detects no activity in the bus as the TIMEOUT bit is set, then it can be assumed that the LIN bus is in sleep mode. Application software can use the Timeout flag to determine when the LIN bus is inactive and put the LIN into sleep mode by writing the POWERDOWN bit.

Note

After the timeout was flagged, a SWnRESET should be asserted before entering Low-Power Mode. This is required to reset the receiver in case that an incomplete frame was on the bus before the idle period.

15.3.1.7.3 Timeout After Wakeup Signal and Timeout After Three Wakeup Signals

The third and fourth types of timeout are related to the wakeup signal. A node initiating a wakeup should expect a header from the master within a defined amount of time: timeout after wakeup signal. See Section 15.4.3 for more details.

15.3.1.8 TXRX Error Detector (TED)

The following sources of error are detected by the TXRX error detector logic (TED). The TED logic consists of a bit monitor, an ID parity checker, and a checksum error. The following errors are detected:

- Bit errors (BE)
- Physical bus errors (PBE)
- Identifier parity errors (PE)
- Checksum errors (CE)

All of these errors (BE, PBE, PE, CE) are flagged. An interrupt for the flagged errors will be generated if enabled. A message is valid for both the transmitter and the receiver if there is no error detected until the end of the frame.

15.3.1.8.1 Bit Errors

A bit error (BE) is detected at the bit time when the bit value that is monitored is different from the bit value that is sent. A bit error is indicated by the BE flag in SCIFLR. After signaling a BE, the transmission is aborted no later than the next byte. The bit monitor ensures that the transmitted bit in LINTX is the correct value on the LIN bus by reading back on the LINRX pin as shown in Figure 15-20.

Note

If a bit occurs due to receiving a header during a slave response, NRE/TIMEOUT flag will not be set for the new frame.

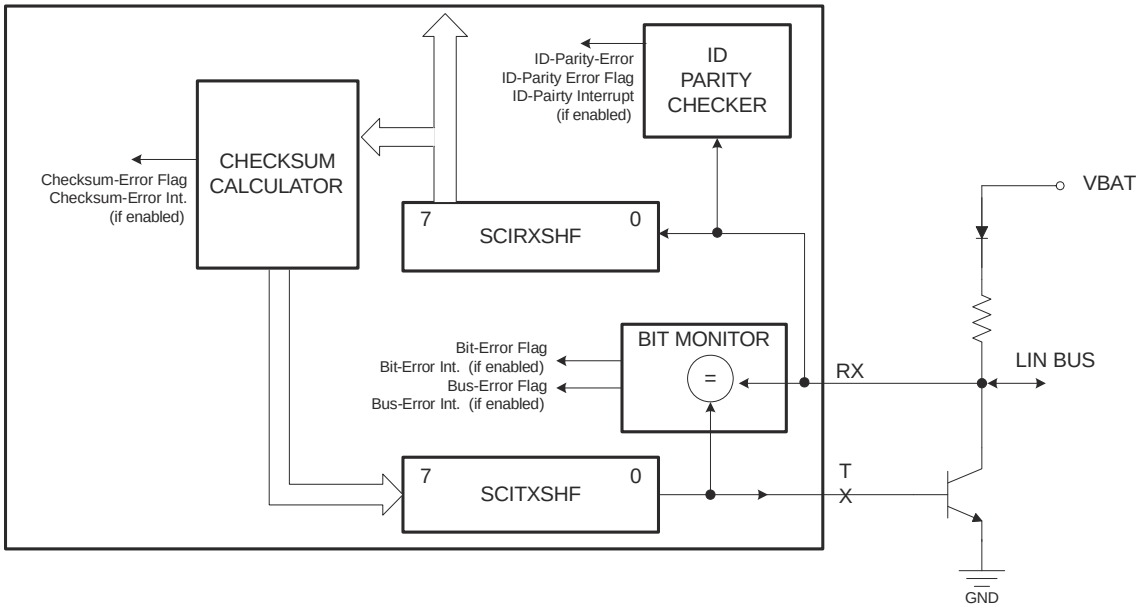


Figure 15-20. TXRX Error Detector

15.3.1.8.2 Physical Bus Errors

A Physical Bus Error (PBE) has to be detected by a master if no valid message can be generated on the bus (Bus shorted to GND or VBAT). The bit monitor detects a PBE during the header transmission, if no Sync Break can be generated (for example, because of a bus shortage to VBAT) or if no sync break delimiter can be generated (for example, because of a bus shortage to GND). Once the Sync Break Delimiter was validated, all other deviations between the monitored and the sent bit value are flagged as Bit Errors (BE) for this frame.

15.3.1.8.3 ID Parity Errors

If parity is enabled, an ID parity error (PE) is detected if any of the two parity bits of the sent ID byte are not equal to the calculated parity on the receiver node. The two parity bits are generated using the following mixed parity algorithm:

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4 \ (even\ Parity)$$
$$P1 = ID1 \oplus ID3 \oplus ID4 \oplus ID5 \ (odd\ Parity)$$

If an ID-parity error is detected, the ID-parity error is flagged, and the received ID is not valid. See Section 15.3.1.9 for details.

15.3.1.8.4 Checksum Errors

A checksum error (CE) is detected and flagged at the receiving end if the calculated modulo-256 sum over all received data bytes (including the ID byte if it is the enhanced checksum type) plus the checksum byte does not result in 0xFF. The modulo-256 sum is calculated over each byte by adding with carry, where the carry bit of each addition is added to the LSB of its resulting sum.

For the transmitting node, the checksum byte sent at the end of a message is the inverted sum of all the data bytes (see Figure 15-21) for classic checksum implementation. The checksum byte is the inverted sum of the identifier byte and all the data bytes (see Figure 15-22) for the LIN 2.0 compliant enhanced checksum implementation. The classic checksum implementation should always be used for reserved identifiers 60 to 63; therefore, the CTYPE bit will be overridden in this case. For signal-carrying-frame identifiers (0 to 59) the type of checksum used depends on the CTYPE bit.

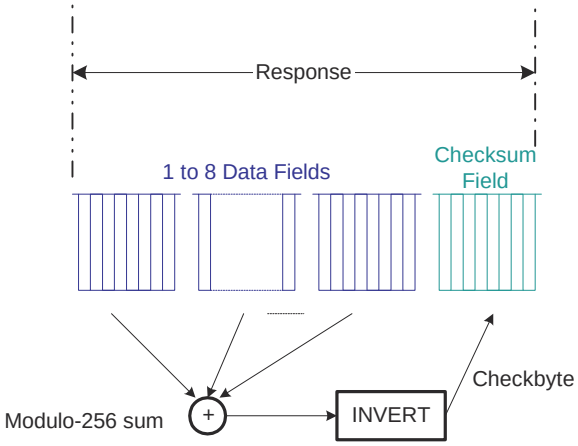


Figure 15-21. Classic Checksum Generation at Transmitting Node

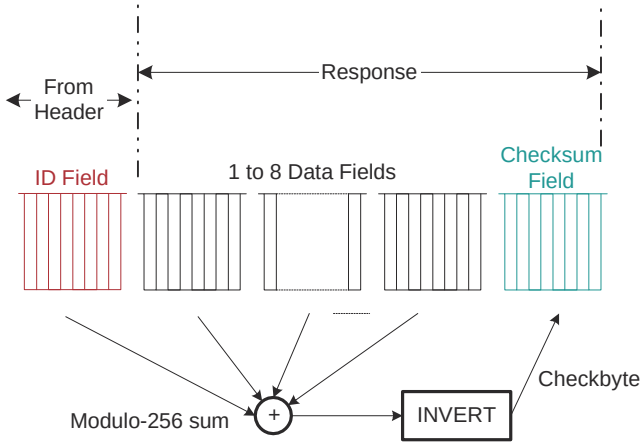


Figure 15-22. LIN 2.0-Compliant Checksum Generation at Transmitting Node

15.3.1.9 Message Filtering and Validation

Message filtering uses the entire identifier to determine which nodes will participate in a response, either receiving or transmitting a response. Therefore, two acceptance masks are used as shown in Figure 15-23. During header reception, all nodes filter the ID-Field (ID-Field is the part of the header explained in Figure 15-15) to determine whether they transmit a response or receive a response for the current message. There are two masks for message ID filtering: one to accept a response reception, the other to initiate a response transmission. See Figure 15-23. All nodes compare the received ID to the identifier stored in the ID-SlaveTask BYTE of the LINID register and use the RX ID MASK and the TX ID MASK fields in the LINMASK register to filter the bits of the identifier that should not be compared.

If there is an RX match with no parity error and the RXENA bit is set, there will be an ID RX flag and an interrupt will be triggered if enabled. If there is a TX match with no parity error and the TXENA bit is set, there will be an ID TX flag and an interrupt will be triggered if enabled in the SCISETINT register.

The masked bits become don't cares for the comparison. To build a mask for a set of identifiers, an XOR function could be used.

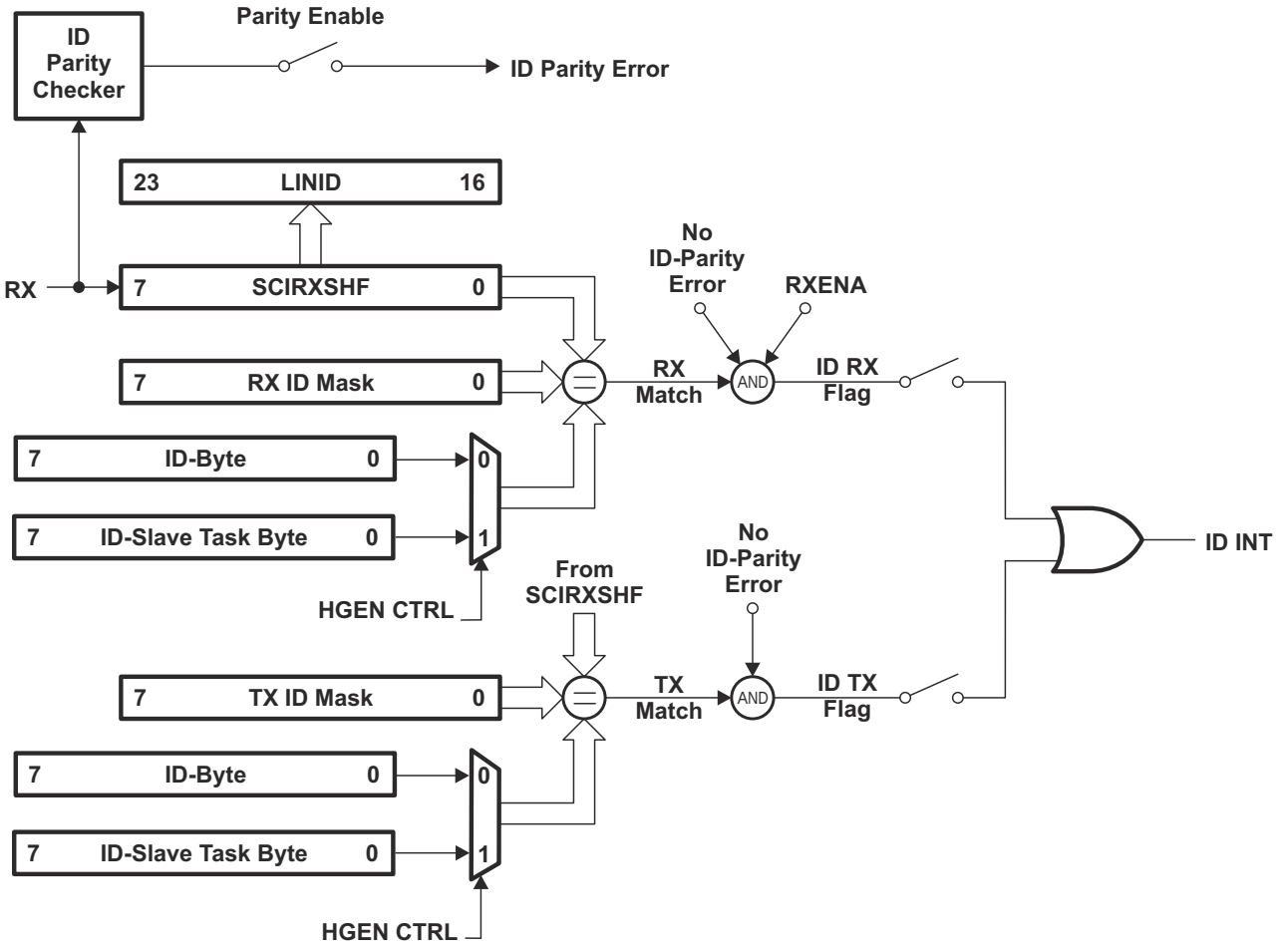


Figure 15-23. ID Reception, Filtering and Validation

For example, to build a mask to accept IDs 0x26 and 0x25 using LINID[7:0] = 0x20; that is, compare 5 most-significant bits (MSBs) and filter 3 least-significant bits (LSBs), the acceptance mask could be:

$$(0x26 + 0x25) \oplus 0x20 = 0x07$$

A mask of all zeros will compare all bits of the received identifier in the shift register with the ID-BYTE in LINID[7:0]. If HGEN CTRL is set to 1, a mask of 0xFF will always cause a match. A mask of all 1s will filter all bits of the received identifier, and thus there will be an ID match regardless of the content of the ID-SlaveTask BYTE field in the LINID register.

Note

When the HGEN CTRL bit = 0, the LIN nodes compare the received ID to the ID-BYTE field in the LINID register, and use the RX ID MASK and the TX ID MASK in the LINMASK register to filter the bits of the identifier that should not be compared.

If there is an RX match with no parity error and the RXENA bit is set, there will be an ID RX flag and an interrupt will be triggered if enabled. A mask of all 0s will compare all bits of the received identifier in the shift register with the ID-BYTE field in LINID[7:0]. A mask of all 1s will filter all bits of the received identifier and there will be no match.

If HGEN CTLR = 1:

- Received ID is compared with the ID-Slave-Task byte, using the RXID mask and the TXID mask.
- A mask of all ones will always result in a match.
- A mask of all zeroes means all the bits must be the same to result in a match.
- If a mask has some bits which are ones, then those bits will not be used for the filtering criterion.

If HGEN CTRL = 0:

- Received ID is compared with the ID byte, using the RXID mask and the TXID mask.
- A mask of all ones will result in no match.
- A mask of all zeroes means all the bits must be the same to result in a match.
- If a mask has some bits which are ones, then those bits will not be used for the filtering criterion.

During header reception, the received identifier is copied to the Received ID field LINID[23:16]. If there is no parity error and there is either a TX match or an RX match, then the corresponding TX or RX ID flag is set. If the ID interrupt is enabled, then an ID interrupt is generated.

After the ID interrupt is generated, the CPU may read the Received ID field LINID[23:16] and determine what response to load into the transmit buffers.

Note

When byte 0 is written to TD0 (LINTD0[31:24]), the response transmission is automatically generated.

In multi-buffer mode, the TXRDY flag will be set when all the response data bytes and checksum byte are copied to the shift register SCITXSHF. In non-multibuffer mode, the TXRDY flag is set each time a byte is copied to the SCITXSHF register, and also for the last byte of the frame after the checksum byte is copied to the SCITXSHF register.

In multi-buffer mode, the TXEMPTY flag is set when both the transmit buffer(s) TDy and the SCITXSHF shift register are emptied and the checksum has been sent. In non-multibuffer mode, TXEMPTY is set each time TD0 and SCITXSHF are emptied, except for the last byte of the frame where the checksum byte must also be transmitted.

If parity is enabled, all slave receiving nodes will validate the identifier using all eight bits of the received ID byte. The SCI/LIN will flag a corrupted identifier if an ID-parity error is detected.

15.3.1.10 Receive Buffers

To reduce CPU load when receiving a LIN N-byte (with N = 1–8) response in interrupt mode , the SCI/LIN module has eight receive buffers. These buffers can store an entire LIN response in the RDy receive buffers. Figure 15-7 illustrates the receive buffers.

The checksum byte following the data bytes is validated by the internal checksum calculator. The checksum error (CE) flag indicates a checksum error and a CE interrupt will be generated if enabled in the SCISETINT register.

The multi-buffer 3-bit counter counts the data bytes transferred from the SCIRXSHF register to the RDy receive buffers if multi-buffer mode is enabled, or to RD0 if multi-buffer mode is disabled. The 3-bit compare register contains the number of data bytes expected to be received. In cases where the ID BYTE field does not convey message length (see *Note: Optional Control Length Bits* in Section 15.3.1.5), the LENGTH value, indicates the expected length and is used to load the 3-bit compare register. Whether the length control field or the LENGTH value is used is selectable with the COMM MODE bit.

A receive interrupt, and a receive ready RXRDY flag could occur after receiving a response if there are no response receive errors for the frame (such as, there is no checksum error, frame error, and overrun error). The checksum byte will be compared before acknowledging a reception.

Note

In multi-buffer mode following are the scenarios associated with clearing the "RXRDY" flag bit:

1. The RXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0/1 register.
2. For LENGTH less than or equal to 4, Read to RD0 register will clear the “RXRDY” flag.
3. For LENGTH greater than 4, Read to RD1 register will clear the “RXRDY” flag.

15.3.1.11 Transmit Buffers

To reduce the CPU load when transmitting a LIN N-byte (with N = 1–8) response in interrupt mode , the SCI/LIN module has 8 transmit buffers, TD0–TD7 in LINTD0 and LINTD1. With these transmit buffers, an entire LIN response field can be preloaded in the TXy transmit buffers. Figure 15-8 illustrates the transmit buffers.

The multi-buffer 3-bit counter counts the data bytes transferred from the TDy transmit buffers register if multi-buffer mode is enabled, or from TD0 to SCITXSHF if multi-buffer mode is disabled. The 3-bit compare register contains the number of data bytes expected to be transmitted. If the ID field is not used to convey message length (see *Note: Optional Control Length Bits* in Section 15.3.1.5), the LENGTH value indicates the expected length and is used instead to load the 3-bit compare register. Whether the length control field or the LENGTH value is used is selectable with the COMM MODE bit.

A transmit interrupt (TX interrupt) and a transmit ready flag (TXRDY flag) could occur after transmitting a response.

The checksum byte will be automatically generated by the checksum calculator and sent after the data-fields transmission is finished. The multi-buffer 3-bit counter counts the data bytes transferred from the TDy buffers into the SCITXSHF register.

Note

The transmit interrupt request can be eliminated until the next series of data is written into the transmit buffers LINTD0 and LINTD1, by disabling the corresponding interrupt via the SCICLRINT register or by disabling the transmitter via the TXENA bit.

15.3.2 LIN Interrupts

LIN and SCI mode have a common Interrupt block as explained in Section 15.2.2. There are 16 interrupt sources in the SCI/LIN module, with 8 of them being LIN mode only, as seen in Table 15-2.

A LIN message frame indicating the timing and sequence of the LIN interrupts that could occur is shown in Figure 15-24.

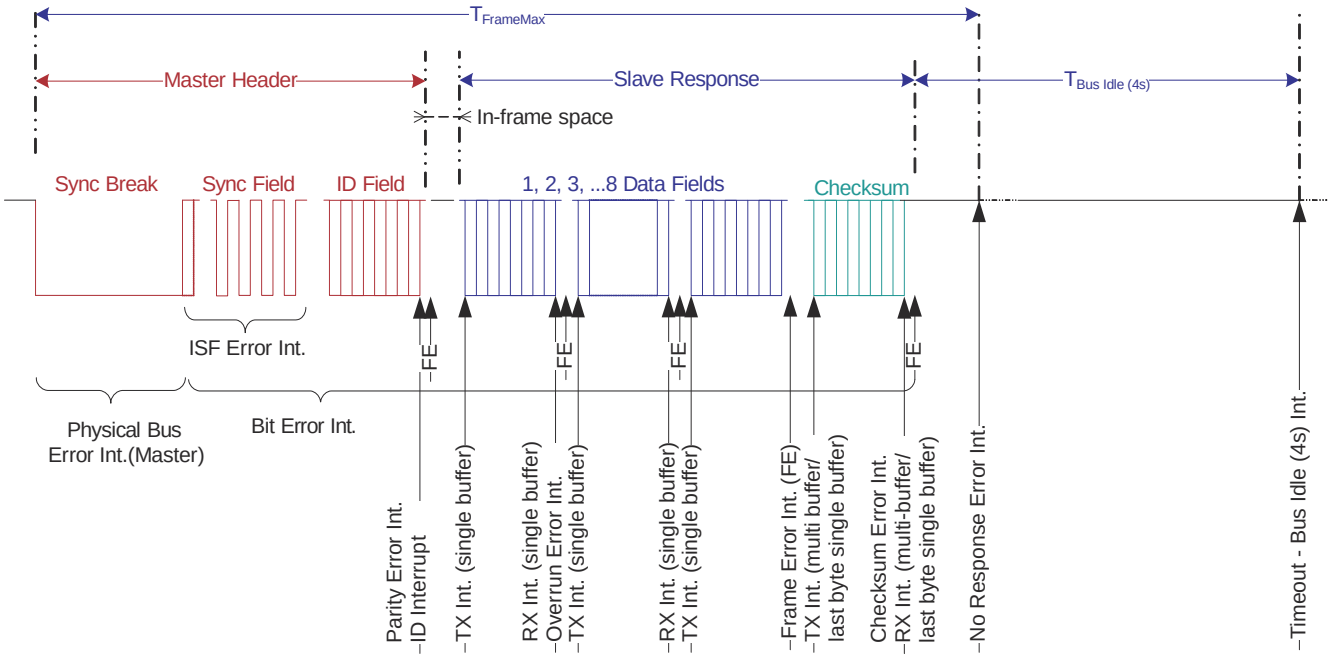


Figure 15-24. LIN Message Frame Showing LIN Interrupt Timing and Sequence

15.3.3 Servicing LIN Interrupts

When servicing an interrupt, clear the corresponding flag in the flag register (SCIFLR) before clearing the global interrupt flag (LIN_GLB_INT_CLR). The ISR should follow the guidelines below. This will prevent any spurious or duplicate interrupt from occurring.

- Clear the LIN interrupt flag in the SCIFLR register.
- Read the LIN interrupt status register to make sure the flag is cleared.
- Clear the global interrupt flag bit in LIN_GLB_INT_CLR.

Note

The transmit interrupt is generated before the LIN transmitter is ready to accept new data. Inside of the LIN transmit ISR, the software should wait until the buffer is completely empty before loading the next data. This can be done by polling for the Bus Busy Flag (SCIFLR.BUSY) to be 0.

15.3.4 LIN Configurations

The following list details the configuration steps that software should perform prior to the transmission or reception of data in LIN mode. As long as the SWnRST bit in the SCIGCR1 register is cleared to 0 the entire time that the LIN is being configured, the order in which the registers are programmed is not important.

- Enable LIN by setting RESET bit.
- Clear SWnRST to 0 before configuring the LIN.
- Select LIN mode by programming LIN MODE bit.
- Select Master or Slave mode by programming the CLOCK bit.
- Select the desired frame format (checksum, parity, length control) by programming SCIGCR1.

- Select multi-buffer mode by programming MBUF MODE bit.
- Select the baud rate to be used for communication by programming BRSR.
- Set the maximum baud rate to be used for communication by programming MBRSR.
- Set the CONT bit to make LIN not halt for an emulation breakpoint until its current reception or transmission is complete (this bit is used only in an emulation environment).
- Set LOOP BACK bit to connect the transmitter to the receiver internally if needed (this feature is used to perform a self-test).
- Select the receiver enable RXENA bit if data is to be received.
- Select the transmit enable TXENA bit if data is to be transmitted.
- Select the RX ID MASK and the TX ID MASK fields in the LINMASK register.
- Set SWnRST to 1 after the LIN is configured.
- Perform Receive or Transmit data (see Section 15.3.1.9, Section 15.3.4.1, and Section 15.3.4.2).

15.3.4.1 Receiving Data

The ID RX FLAG is set after a valid LIN ID is received with RX Match. An ID interrupt is generated, if enabled.

15.3.4.1.1 Receiving Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit is cleared to 0. In this mode, LIN sets the RXRDY bit when it transfers newly received data from SCIRXSHF to RD0. The SCI clears the RXRDY bit after the new data in RD0 has been read. Also, as data is transferred from SCIRXSHF to RD0, the LIN sets the FE, OE, or PE flags if any of these error conditions were detected in the received data. These error conditions are supported with configurable interrupt capability.

You can receive data by:

1. Polling Receive Ready Flag
2. Receive Interrupt

In polling method, software can poll for the RXRDY bit and read the data from RD0 byte of the LINRD0 register once the RXRDY bit is set high. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use the interrupt method. To use the interrupt method, the SET RX INT bit is set. An interrupt is generated the moment the RXRDY bit is set. If the checksum scheme is enabled by setting the Compare Checksum (CC) bit to 1, the checksum will be compared on the byte that is currently being received, which is expected to be the checksum byte. The CC bit will be cleared once the checksum is received. A CE will immediately be flagged if there is a checksum error.

15.3.4.1.2 Receiving Data in Multi-Buffer Mode

Multi-buffer mode is selected when the MBUF MODE bit is set to 1. In this mode, LIN sets the RXRDY bit after receiving the programmed number of data in the receive buffer and the checksum field, the complete frame. The error condition detection logic is similar to the single-buffer mode, except that it monitors for the complete frame. Like single-buffer mode, you can use the polling or interrupt method to read the data. The received data has to be read from the LINRD0 and LINRD1 registers, based on the number of bytes. For a LENGTH less than or equal to 4, a read from the LINRD0 register clears the RXRDY flag. For a LENGTH greater than 4, a read from the LINRD1 register clears the RXRDY flag. If the checksum scheme is enabled by setting the Compare Checksum (CC) bit to 1 during the reception of the data, then the byte that is received after the reception of the programmed number of data bytes indicated by the LENGTH field is treated as a checksum byte. The CC bit will be cleared once the checksum is received and compared.

15.3.4.2 Transmitting Data

The LIN transmitter is enabled if both the TX FUNC bit and the TXENA bit are set to 1. If the TX FUNC bit is not set, the LINTX pin functions as a general-purpose I/O pin rather than as a LIN function pin. Any value written to the TD0 before the TXENA bit is set to 1 is not transmitted. Both of these control bits allow for the LIN transmitter to be held inactive independently of the receiver.

The ID TX flag is set after a valid LIN ID is received with TX Match. An ID interrupt is generated, if enabled.

15.3.4.2.1 Transmitting Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit is cleared to 0. In this mode, LIN waits for data to be written to TD0, transfers it to SCITXSHF, and transmits the data. The TXRDY and TX EMPTY bits indicate the status of the transmit buffers. That is, when the transmitter is ready for data to be written to TD0, the TXRDY bit is set. Additionally, if both TD0 and SCITXSHF are empty, then the TX EMPTY bit is also set.

You can transmit data by:

1. Polling Transmit Ready Flag
2. Transmit Interrupt

In polling method, software can poll for the TXRDY bit to go high before writing the data to the TD0. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use the interrupt method. To use the interrupt method, the SET TX INT bit is set. An interrupt is generated the moment the TXRDY bit is set. When the LIN has completed transmission of all pending frames, the SCITXSHF register and the TD0 are empty, the TXRDY bit is set, and an interrupt request is generated, if enabled. Because all data has been transmitted, the interrupt request should be halted. This can either be done by disabling the transmit interrupt (CLR TX INT) or by disabling the transmitter (clear TXENA bit). If the checksum scheme is enabled by setting the Send Checksum (SC) bit to 1, the checksum byte will be sent after the current byte transmission. The SC bit will be cleared after the checksum byte has been transmitted.

Note

The TXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0 or SCIINTVECT1 register.

15.3.4.2.2 Transmitting Data in Multi-Buffer Mode

Multi-buffer mode is selected when the MBUF MODE bit is set to 1. Like single-buffer mode, you can use the polling or interrupt method to write the data to be transmitted. The transmitted data has to be written to the LINTD0 and LINTD1 registers, based on the number of bytes. LIN waits for data to be written to Byte 0 (TD0) of the LINTD0 register and transfers the programmed number of bytes to SCITXSHF to transmit one by one automatically. If the checksum scheme is enabled by setting the Send Checksum (SC) bit to 1, the checksum will be sent after transmission of the last byte of the programmed number of data bytes, indicated by the LENGTH field. The SC bit will be cleared after the checksum byte has been transmitted.

15.4 Low-Power Mode

The SCI/LIN module can be put in either local or global low-power mode. Global low-power mode is asserted by the system and is not controlled by the SCI/LIN module. During global low-power mode, all clocks to the SCI/LIN are turned off so the module is completely inactive. If global low-power mode is requested while the receiver is receiving data, then the SCI/LIN completes the current reception and then enters the low-power mode, that is, module enters low-power mode only when Busy bit (SCIFLR.3) is cleared.

The LIN module may enter low-power mode either when there was no activity on the LINRX pin for more than 4s (this can be either a constant recessive or dominant level) or when a Sleep Command frame was received. Once the Timeout flag (SCIFLR.4) was set or once a Sleep Command was received, the POWERDOWN bit (SCIGCR2.0) must be set by the application software to make the module enter local low-power mode. A wakeup signal will terminate the sleep mode of the LIN bus.

Note

Enabling Local Low-Power Mode During Receive and Transmit

If the wakeup interrupt is enabled and low-power mode is requested while the receiver is receiving data, then the SCI/LIN immediately generates a wake-up interrupt to clear the powerdown bit. Thus, the SCI/LIN is prevented from entering low-power mode and completes the current reception. Otherwise, if the wakeup interrupt is disabled, the SCI/LIN completes the current reception and then enters the low-power mode.

15.4.1 Entering Sleep Mode

In LIN protocol, a sleep command is used to broadcast the sleep mode to all nodes. The sleep command consists of a diagnostic master request frame with identifier 0x3C (60), with the first data field as 0x00. There should be no activity in the bus once all nodes receive the sleep command: the bus is in sleep mode.

Local low-power mode is asserted by setting the POWERDOWN bit; setting this bit stops the clocks to the SCI/LIN internal logic and registers. Clearing the POWERDOWN bit causes SCI/LIN to exit from local low-power mode. All the registers are accessible during local power-down mode. If a register is accessed in low-power mode, this access results in enabling the clock to the module for that particular access alone.

15.4.2 Wakeup

The wakeup interrupt is used to allow the SCI/LIN module to automatically exit a low-power mode. A SCI/LIN wakeup is triggered when a low level is detected on the receive RX pin, and this clears the POWERDOWN bit.

Note

If the wakeup interrupt is disabled, then the SCI/LIN enters low-power mode whenever it is requested to do so, but a low level on the receive RX pin does NOT cause the SCI/LIN to exit low-power mode.

In LIN mode, any node can terminate sleep mode by sending a wakeup signal; see Figure 15-25. A slave node that detects the bus in sleep mode, and with a wakeup request pending, will send a wakeup signal. The wakeup signal is a dominant value on the LIN bus for T_{WUSIG}; this is at least 5 T_{bits} for the LIN bus baud rates. The wakeup signal is generated by sending a 0xF0 byte containing 5 dominant T_{bits} and 5 recessive T_{bits}.

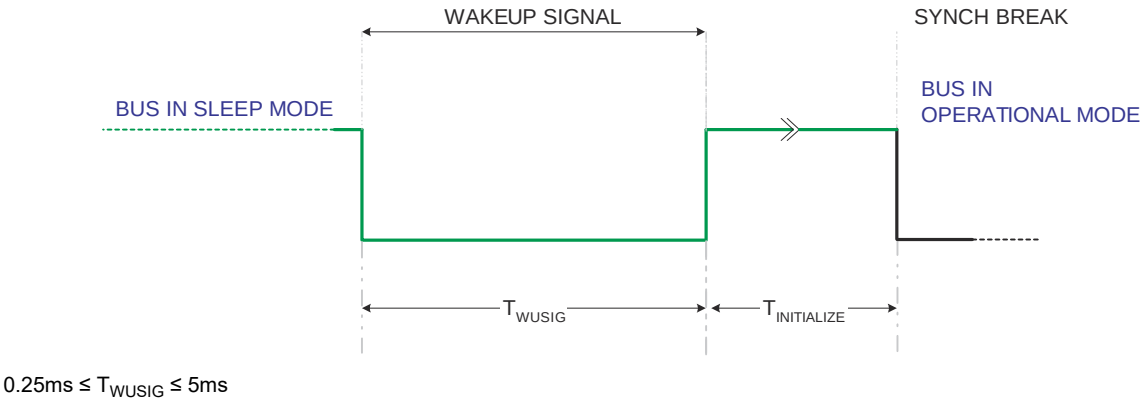


Figure 15-25. Wakeup Signal Generation

Assuming a perfect bus with no noise or loading effects, a write of 0xF0 to TD0 will load the transmitter to meet the wakeup signal timing requirement for T_{WUSIG}. Then, setting the GENWU bit will transmit the preloaded value in TD0 for a wakeup signal transmission.

Note

The GENWU bit can be set/reset only when SWnRST is set to'1' and the node is in power down mode. The bit will be cleared on a valid sync break detection. A master sending a wakeup request, will exit power down mode upon reception of the wakeup pulse. The bit will be cleared on a SWnRST. This can be used to stop a master from sending further wakeup requests.

The TI TPIC1021 LIN transceiver, upon receiving a wakeup signal, will translate it to the microcontroller for wakeup with a dominant level on the RX pin, or a signal to the voltage regulator. While the POWERDOWN bit is set, if the LIN module detects a recessive-to-dominant edge (falling edge) on the RX pin, it will generate a wakeup interrupt if enabled in the SCISETINT register.

According to LIN protocol 2.0, the TI TPIC1021 LIN transceiver detecting a dominant level on the bus longer than 150 ms will detect it as a wakeup request. The LIN controller's slave is ready to listen to the bus in less than 100 ms ($T_{INITIALIZE}$<100ms) after a dominant-to-recessive edge (end-of-wakeup signal).

15.4.3 Wakeup Timeouts

The LIN protocol defines the following timeouts for a wakeup sequence. After a wakeup signal has been sent to the bus, all nodes wait for the master to send a header. If no sync field is detected before 150 ms (3,000 cycles at 20 kHz) after a wakeup signal is transmitted, a new wakeup is sent by the same node that requested the first wakeup. This sequence is not repeated more than two times. After three attempts to wake up the LIN bus, wakeup signal generation is suspended for a 1.5 s (30,000 cycles at 20 kHz) period after three breaks.

Note

To achieve compatibility to LIN1.3 timeout conditions, the MBRS register must be set to assure that the LIN 2.0 (real-time-based) timings meet the LIN 1.3 bit time base. A node triggering the wakeup should set the MBRS register accordingly to meet the targeted time as 128 Tbits × programmed prescaler.

The LIN controller handles the wakeup expiration times defined by the LIN protocol with a hardware implementation.

15.5 Emulation Mode

In emulation mode, the CONT bit determines how the SCI/LIN operates when the program is suspended. The SCI/LIN counters are affected by this bit during debug mode. when set, the counters are not stopped and when cleared, the counters are stopped debug mode.

Any reads in emulation mode to a SCI/LIN register will not have any effect on the flags in the SCIFLR register.

Note

When emulation mode is entered during the Frame transmission or reception of the frame and CONT bit is not set, Communication is not expected to be successful. The suggested usage is to set CONT bit during emulation mode for successful communication.

15.6 LIN SCI versus Standard SCI

Table 15-8 compares the LIN/SCI with the standalone-SCI ("standard" SCI) available in all C2000 devices.

Table 15-8. SCI versus LIN-SCI Programming

Standard SCI	LIN SCI
CCS Register Structure	
SciaRegs.REGISTER	LinaRegs.REGISTER
Example Configurations	
Idle Line Mode:	**Idle Line Mode:**
SciaRegs.SCICCR.bit.ADDRIDLE_MODE = 0;	LinaRegs.SCIGCR1.bit.COMMMODE = 0;
Address Bit Mode:	**Address Bit Mode:**
SciaRegs.SCICCR.bit.ADDRIDLE_MODE = 1;	LinaRegs.SCIGCR1.bit.COMMMODE = 1;
FIFO Mode:	**Buffered Mode:**
SciaRegs.SCIFFTX.bit.SCIFFENA = 1;	LinaRegs.SCIGCR1.bit.MBUFMODE = 1;
No FIFO Mode:	**Unbuffered Mode:**
SciaRegs.SCIFFTX.bit.SCIFFENA = 0;	LinaRegs.SCIGCR1.bit.MBUFMODE = 0;

Table 15-8. SCI versus LIN-SCI Programming (continued)

Standard SCI	LIN SCI
Configuration Sequence	
//Into software reset SciaRegs.SCICTL1.bit.SWRESET = 0; *(Configuration instructions)* *//Relinquish SCI from reset* SciaRegs.SCICTL1.bit.SWRESET = 1;	*//Into reset* LinaRegs.SCIGCR0.bit.RESET = 0; *//Out of reset* LinaRegs.SCIGCR0.bit.RESET = 1; *//Into software reset* LinaRegs.SCIGCR1.bit.SWnRST = 0; *(Configuration instructions)* *//Bring out of software reset* LinaRegs.SCIGCR1.bit.SWnRST = 1;
Baud Rate Configuration	
Registers: SciaRegs.SCIHBAUD SciaRegs.SCILBAUD	**Registers**: LinaRegs.BRSR
Fields: BRR = (SciaRegs.SCIHBAUD << 8) + SciaRegs.SCILBAUD	**Fields**: Prescaler(P) = LinaRegs.BRSR.bit.P Fractional Divider(M) = LinaRegs.BRSR.bit.M
Base Clock Rate: $LSPCLK = \frac{SYSCLKOUT}{LowSpeedPrescaler}$	**Base Clock Rate:** $LM\_CLK = \frac{SYSCLKOUT}{2}$
Baud Rate Calculation: $Baud = \frac{LSPCLK}{(BRR+1)\times 8}$	**Baud Rate Calculation:** $Baud = \frac{LM\_CLK}{(P+1+\frac{M}{16})\times 16}$
Basic Transmission	
SciaRegs.SCITXBUF = data;	LinaRegs.SCITD = data;
FIFO/Data Buffer	

Structure:
FIFO
Depth:
8 bits x 16
Fill:
for(iter = 0; iter < depth; iter++)
{
SciaRegs.SCITXBUF = data[i];
}
 	
Empty:
for(iter = 0; iter < depth; iter++)
{
data[i] = SciaRegs.SCIRXBUF.all;
 	
}
 	
	Structure:
	Buffer
	Depth:
	8 bits x 8
	Fill:
	LinaRegs.LINTD0.all = data0123;
	LinaRegs.LINTD1.all = data4567;
	
	Empty:
	data4567 = LinaRegs.LINRD1.all;
	data0123 = LinaRegs.LINRD0.all;
	
	Note: The LINTD0/1 and LINRD0/1 should not be accessed with bytewise operations. Software must pack/unpack data into 32-bit words before writing/reading from the LIN data buffers.

Table 15-8. SCI versus LIN-SCI Programming (continued)

Standard SCI	LIN SCI
Flags	
Data Reception:	**Data Reception:**
SciaRegs.SCIRXST.bit.RXRDY	LinaRegs.SCIFLR.bit.RXRDY
Transmission Completion:	**Transmission Completion:**
SciaRegs.SCICTL2.bit.TXEMPTY	LinaRegs.SCIFLR.bit.TXEMPTY
Interrupts	
ISR Mapping:	**ISR Mapping:**
PieVectTable.SCIRXINTA = &sciaRxFifoIsr;	PieVectTable.LININT0A = &Lina_Level0_ISR;
PieVectTable.SCITXINTA = &sciaTxFifoIsr;	PieVectTable.LININT1A = &Lina_Level1_ISR;
ISR:	**ISR**:
interrupt void sciaTxFifoIsr(void)	interrupt void Lina_Level0_ISR(void)
{	{
//do TX stuff	*//read-clear interrupt vector*
	LinL0IntVect = LinaRegs.SCIINTVECT0.all;
//clear interrupt flag	
SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;	if(LinL0IntVect == TXVect)
//acknowledge PIE	{
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;	*//do TX stuff*
	{
}	if(LinL0IntVect == RXVect)
	{
interrupt void sciaRxFifoIsr(void)	*//do RX stuff*
{	}
//do RX stuff	
	//Acknowledge PIE
// clear overflow flag	PieCtrlRegs.PIEACK.all =
SciaRegs.SCIFFRX.bit.RXFFOVRCLR = 1;	PIEACK_GROUP9;
// clear interrupt flag	}
SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1;	
//acknowledge PIE	interrupt void Lina_Level1_ISR(void)
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;	{
	//read-clear interrupt vector
}	LinL1IntVect =
	LinaRegs.SCIINTVECT1.all;
	if(LinL1IntVect == TXVect)
	{
	//do TX stuff
	{
	if(LinL1IntVect == RXVect)
	{
	//do RX stuff
	}
	//Acknowledge PIE
	PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
	}

15.7 SCI/LIN Registers

The SCI/LIN module registers are based on the SCI registers, with added functionality registers enabled by the LIN MODE bit in the SCIGCR1 register.

These registers are accessible in 32-bit reads or writes. The SCI/LIN is controlled and accessed through the registers listed in the following sections. Among the features that can be programmed are the LIN protocol mode, communication and timing modes, baud rate value, frame format, and interrupt configuration.

15.7.1 LIN Base Addresses

Table 15-9. LIN Base Address Table (C28)

Bit Field Name		Base Address
Instance	Structure	
LinaRegs	LIN_REGS	0x0000_6C00

15.7.2 LIN_REGS Registers

Table 15-10 lists the LIN_REGS registers. All register offset addresses not listed in Table 15-10 should be considered as reserved locations and the register contents should not be modified.

Table 15-10. LIN_REGS Registers

Offset	Acronym	Register Name	Section
0h	SCIGCR0	Global Control Register 0	Go
4h	SCIGCR1	Global Control Register 1	Go
8h	SCIGCR2	Global Control Register 2	Go
Ch	SCISETINT	Interrupt Enable Register	Go
10h	SCICLEARINT	Interrupt Disable Register	Go
14h	SCISETINTLVL	Set Interrupt Level Register	Go
18h	SCICLEARINTLVL	Clear Interrupt Level Register	Go
1Ch	SCIFLR	Flag Register	Go
20h	SCIINTVECT0	Interrupt Vector Offset Register 0	Go
24h	SCIINTVECT1	Interrupt Vector Offset Register 1	Go
28h	SCIFORMAT	Length Control Register	Go
2Ch	BRSR	Baud Rate Selection Register	Go
30h	SCIED	Emulation buffer Register	Go
34h	SCIRD	Receiver data buffer Register	Go
38h	SCITD	Transmit data buffer Register	Go
44h	SCIPIO2	Pin control Register 2	Go
60h	LINCOMP	Compare register	Go
64h	LINRD0	Receive data register 0	Go
68h	LINRD1	Receive data register 1	Go
6Ch	LINMASK	Acceptance mask register	Go
70h	LINID	LIN ID Register	Go
74h	LINTD0	Transmit Data Register 0	Go
78h	LINTD1	Transmit Data Register 1	Go
7Ch	MBRSR	Maximum Baud Rate Selection Register	Go
90h	IODFTCTRL	IODFT for LIN	Go

Complex bit access types are encoded to fit into small table cells. Table 15-11 shows the codes that are used for access types in this section.

Table 15-11. LIN_REGS Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
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.

15.7.2.1 SCIGCR0 Register (Offset = 0h) [reset = 0h]

The SCIGCR0 register defines the module reset.

Figure 15-26. SCIGCR0 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							RESET
R-0h							R/W-0h

Table 15-12. SCIGCR0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-1	RESERVED	R	0h	Reserved
0	RESET	R/W	0h	This bit resets the SCI/LIN module. This bit is effective in LIN or SCI-compatible mode.. This bit affects the reset state of the SCI/LIN module. Reset type: SYSRSn 0h (R/W) = SCI/LIN module is in held in reset. 1h (R/W) = SCI/LIN module is out of reset.

15.7.2.2 SCIGCR1 Register (Offset = 4h) [reset = 0h]

The SCIGCR1 register defines the frame format, protocol, and communication mode used by the SCI.

Figure 15-27. SCIGCR1 Register

31	30	29	28	27	26	25	24
RESERVED						TXENA	RXENA
R-0h						R/W-0h	R/W-0h
23	22	21	20	19	18	17	16
RESERVED						CONT	LOOPBACK
R-0h						R/W-0h	R/W-0h
15	14	13	12	11	10	9	8
RESERVED		STOPEXT FRAME	HGENCTRL	CTYPE	MBUFMODE	ADAPT	SLEEP
R-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
SWnRST	LINMODE	CLK_MASTER	STOP	PARITY	PARITYENA	TIMINGMODE	COMMMODE
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 15-13. SCIGCR1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-26	RESERVED	R	0h	Reserved
25	TXENA	R/W	0h	Transmit enable. This bit is effective in LIN and SCI modes. Data is transferred from SCITD or the TDy (with y=0, 1,...7) buffers in LIN mode to the SCITXSHF shift out register only when the TXENA bit is set. Note: Data written to SCITD or the transmit multibuffer before TXENA is set is not transmitted. If TXENA is cleared while transmission is ongoing, the data previously written to SCITD is sent (including the checksum byte in LIN mode). Reset type: SYSRSn 0h (R/W) = Disable transfers from SCITD or TDy to SCITXSHF 1h (R/W) = Enable transfers of data from SCITD or TDy to SCITXSHF

Table 15-13. SCIGCR1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
24	RXENA	R/W	0h	Receive enable. This bit is effective in LIN or SCI-compatible mode. RXENA allows or prevents the transfer of data from SCIRXSHF to SCIRD or the receive multibuffers. Note: Clearing RXENA stops received characters from being transferred into the receive buffer or multibuffers, prevents the RX status flags (see Table 7) from being updated by receive data, and inhibits both receive and error interrupts. However, the shift register continues to assemble data regardless of the state of RXENA. Note: If RXENA is cleared before the time the reception of a frame is complete, the data from the frame is not transferred into the receive buffer. Note: If RXENA is set before the time the reception of a frame is complete, the data from the frame is transferred into the receive buffer. If RXENA is set while SCIRXSHF is in the process of assembling a frame, the status flags are not guaranteed to be accurate for that frame. To ensure that the status flags correctly reflect what was detected on the bus during a particular frame, RXENA should be set before the detection of that frame Reset type: SYSRSn 0h (R/W) = Prevents the receiver from transferring data from the shift buffer to the receive buffer or multibuffers 1h (R/W) = Allows the receiver to transfer data from the shift buffer to the receive buffer or multibuffers
23-18	RESERVED	R	0h	Reserved
17	CONT	R/W	0h	Continue on suspend. This bit has an effect only when a program is being debugged. The bit determines how the SCI/LIN operates when the program is suspended. This bit affects the LIN counters. When this bit is set, the counters are not stopped during debug. When this bit is cleared, the counters are stopped during debug. Reset type: SYSRSn 0h (R/W) = When debug mode is entered, the SCI/LIN state machine is frozen. Transmissions and LIN counters are halted and resume when debug mode is exited. 1h (R/W) = When debug mode is entered, the SCI/LIN continues to operate until the current transmit and receive functions are complete.
16	LOOPBACK	R/W	0h	Loopback bit. This bit is effective in LIN or SCI-compatible mode. The self-checking option for the SCI/LIN can be selected with this bit. If the LINTX and LINRX pins are configured with SCI/LIN functionality, then the LINTX pin is internally connected to the LINRX pin. Externally, during loop back operation, the LINTX pin outputs a high value and the LINRX pin is in a high-impedance state. If this bit value is changed while the SCI/LIN is transmitting or receiving data, errors may result. Reset type: SYSRSn 0h (R/W) = Loopback mode is disabled. 1h (R/W) = Loopback mode is enabled.
15-14	RESERVED	R	0h	Reserved
13	STOPEXTFRAME	R/W	0h	Stop extended frame communication. This bit is effective in LIN mode only. This bit can be written only during extended frame communication. When the extended frame communication is stopped, this bit is cleared automatically. Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Extended frame communication will be stopped, once current frame transmission/reception is completed.

Table 15-13. SCIGCR1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
12	HGENCTRL	R/W	0h	HGEN control bit. This bit is effective in LIN mode only. This bit controls the type of mask filtering comparison. Reset type: SYSRSn 0h (R/W) = ID filtering using ID-Byte. RECEIVEDID and IDBYTE fields in the LINID register are used for detecting a match (using TX/RXMASK values). Mask of 0xFF in LINMASK register will result in NO match. 1h (R/W) = ID filtering using ID-SlaveTask byte (Recommended). RECEIVEDID and IDSLAVETASKBYTE fields in the LINID register are used for detecting a match (using TX/RXMASK values). Mask of 0xFF in LINMASK register will result in ALWAYS match
11	CTYPE	R/W	0h	Checksum type. This bit is effective in LIN mode only. This bit controls the type of checksum to be used: classic or enhanced. Reset type: SYSRSn 0h (R/W) = Classic checksum is used. This checksum is compatible with LIN 1.3 slave nodes. The classic checksum contains the modulo-256 sum with carry over all data bytes. Frames sent with Identifier 60 (0x3C) to 63 (0x3F) must always use the classic checksum. 1h (R/W) = Enhanced checksum is used. The enhanced checksum is compatible with LIN 2.0 and newer slave nodes. The enhanced checksum contains the modulo-256 sum with carry over all data bytes AND the protected Identifier.
10	MBUFMODE	R/W	0h	Multibuffer mode. This bit is effective in LIN or SCI-compatible mode. This bit controls receive/transmit buffer usage, that is, whether the RX/TX multibuffers are used or a single register, RD0/TD0, is used. Reset type: SYSRSn 0h (R/W) = The multibuffer mode is disabled. 1h (R/W) = The multibuffer mode is enabled.
9	ADAPT	R/W	0h	Adapt mode enable. This mode is effective in LIN mode only. This bit has an effect during the detection of the Sync Field. There are two LIN protocol bit rate modes that could be enabled with this bit according to the Node capability file definition: automatic or select. Software and network configuration will decide which of the previous two modes. When this bit is cleared, the LIN 2.0 protocol fixed bit rate should be used. If the ADAPT bit is set, a LIN slave node detecting the baud rate will compare it to the prescalers in BRSR register and update it if they are different. The BRSR register will be updated with the new value. If this bit is not set there will be no adjustment to the BRSR register. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Automatic baud rate adjustment is disabled. 1h (R/W) = Automatic baud rate adjustment is enabled.

Table 15-13. SCIGCR1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
8	SLEEP	R/W	0h	SCI sleep. SCI compatibility mode only. In a multiprocessor configuration, this bit controls the receive sleep function. Clearing this bit brings the SCI out of sleep mode. The receiver still operates when the SLEEP bit is set however, RXRDY is updated and SCIRD is loaded with new data only when an address frame is detected. The remaining receiver status flags are updated and an error interrupt is requested if the corresponding interrupt enable bit is set, regardless of the value of the SLEEP bit. In this way, if an error is detected on the receive data line while the SCI is asleep, software can promptly deal with the error condition. The SLEEP bit is not automatically cleared when an address byte is detected. This field is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = Sleep mode is disabled. 1h (R/W) = Sleep mode is enabled.
7	SWnRST	R/W	0h	Software reset (active low). This bit is effective in LIN or SCI-compatible mode. The SCI/LIN should only be configured while SWnRST = 0. Only the following configuration bits can be changed in runtime (while SWnRESET = 1): - STOP EXT Frame (SCIGCR1[13]) - CC bit (SCIGCR2[17]) - SC bit (SCIGCR2[16]) Reset type: SYSRSn 0h (R/W) = The SCI/LIN is in its reset state no data will be transmitted or received. Writing a 0 to this bit initializes the SCI/LIN state machines and operating flags. All affected logic is held in the reset state until a 1 is written to this bit. 1h (R/W) = The SCI/LIN is in its ready state transmission and reception can occur. After this bit is set to 1, the configuration of the module should not change.
6	LINMODE	R/W	0h	LIN mode This bit controls the mode of operation of the module. Reset type: SYSRSn 0h (R/W) = LIN mode is disabled SCI compatibility mode is enabled. 1h (R/W) = LIN mode is enabled SCI compatibility mode is disabled.
5	CLK_MASTER	R/W	0h	SCI internal clock enable or LIN Master/Slave configuration. In the SCI mode, this bit enables the clock to the SCI module. In LIN mode, this bit determines whether a LIN node is a slave or master. Reset type: SYSRSn 0h (R/W) = SCI-compatible mode: Reserved. LIN mode: The module is in slave mode. 1h (R/W) = SCI-compatible mode: Enable clock to the SCI module. LIN mode: The node is in master mode.

Table 15-13. SCIGCR1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	STOP	R/W	0h	SCI number of stop bits. This bit is effective in SCI-compatible mode only. Note: The receiver checks for only one stop bit. However in idle-line mode, the receiver waits until the end of the second stop bit (if STOP = 1) to begin checking for an idle period. This field is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = One stop bit is used. 1h (R/W) = Two stop bits are used.
3	PARITY	R/W	0h	SCI parity odd/even selection. This bit is effective in SCI-compatible mode only. If the PARITY ENA bit (SCIGCR1.2) is set, PARITY designates odd or even parity. The parity bit is calculated based on the data bits in each frame and the address bit (in address-bit mode). The start and stop fields in the frame are not included in the parity calculation. This field is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = Odd parity is used. The SCI transmits and expects to receive a value in the parity bit that makes odd the total number of bits in the frame with the value of 1. 1h (R/W) = Even parity is used. The SCI transmits and expects to receive a value in the parity bit that makes even the total number of bits in the frame with the value of 1.
2	PARITYENA	R/W	0h	Parity enable. Enables or disables the parity function. Reset type: SYSRSn 0h (R/W) = SCI-compatible mode: Parity disabled no parity bit is generated during transmission or is expected during reception. LIN mode: ID-parity verification is disabled. 1h (R/W) = SCI compatible mode: Parity enabled. A parity bit is generated during transmission and is expected during reception. LIN mode: ID-parity verification is enabled.
1	TIMINGMODE	R/W	0h	SCI timing mode bit. This bit is effective in SCI-compatible mode only. It must be set to 1 when the SCI mode is used. This bit configures the SCI for asynchronous operation. Reset type: SYSRSn 0h (R/W) = Reserved. 1h (R/W) = Must be set to 1 when module is configured for SCI operation
0	COMMMODE	R/W	0h	SCI/LIN communication mode bit. In compatibility mode, it selects the SCI communication mode. In LIN mode it selects length control option for ID-field bits ID4 and ID5. Reset type: SYSRSn 0h (R/W) = SCI-compatible mode: Idle-line mode is used. LIN mode: ID4 and ID5 are not used for length control. 1h (R/W) = SCI-compatible mode: Address-bit mode is used. LIN mode: ID4 and ID5 are used for length control.

15.7.2.3 SCIGCR2 Register (Offset = 8h) [reset = 0h]

The SCIGCR2 register is used to send or compare a checksum byte during extended frames, to generate a wakeup and for low-power mode control of the LIN module.

Figure 15-28. SCIGCR2 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED						CC	SC
R-0h						R/W-0h	R/W-0h
15	14	13	12	11	10	9	8
RESERVED							GENWU
R-0h							R/W-0h
7	6	5	4	3	2	1	0
RESERVED							POWERDOWN
R-0h							R/W-0h

Table 15-14. SCIGCR2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-18	RESERVED	R	0h	Reserved
17	CC	R/W	0h	Compare Checksum. This mode is effective in LIN mode only. This bit is used by the receiver for extended frames to trigger a checksum compare. The user will initiate this transaction by writing a one to this bit. In non-multibuffer mode, once the CC bit is set, the checksum will be compared on the byte that is currently being received, expected to be the checkbyte. During Multibuffer mode, following are the scenarios associated with the CC bit : - If CC bit is set during the reception of the data, then the byte that is received after the reception of the programmed no. of data bytes indicated by SCIFORMAT[18:16], is treated as a checksum byte. - If CC bit is set during the IDLE period (that is, during inter-frame space), then the next immediate byte will be treated as a checksum byte. A CE will immediately be flagged if there is a checksum error. This bit is automatically cleared once the checksum is successfully compared. Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Compare checksum on expected checkbyte
16	SC	R/W	0h	Send Checksum This mode is effective in LIN mode only. This bit is used by the transmitter with extended frames to send a checkbyte. In non-multibuffer mode the checkbyte will be sent after the current byte transmission. In multibuffer mode the checkbyte will be sent after the last byte count, indicated by the SCIFORMAT[18:16]). This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No checkbyte will be sent. 1h (R/W) = A checkbyte will be sent. This bit will automatically get cleared after the checkbyte is transmitted. The checksum will not be sent if this bit is set before transmitting the very first byte, that is, during interframe space.
15-9	RESERVED	R	0h	Reserved

Table 15-14. SCIGCR2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
8	GENWU	R/W	0h	Generate wakeup signal. This bit controls the generation of a wakeup signal, by transmitting the TDO buffer value. This bit is cleared on reception of a valid sync break. Reset type: SYSRSn 0h (R/W) = No effect 1h (R/W) = Transmit TDO for wakeup. This bit will be cleared on a SWnRST (SCIGCR1.7)
7-1	RESERVED	R	0h	Reserved
0	POWERDOWN	R/W	0h	Power down. This bit is effective in LIN or SCI-compatible mode. When the powerdown bit is set, the SCI/LIN module attempts to enter local low-power mode. If the POWERDOWN bit is set while the receiver is actively receiving data and the wakeup interrupt is disabled, then the SCI/LIN will delay low-power mode from being entered until completion of reception. In LIN mode the user may set the POWERDOWN bit on Sleep Command reception or on idle bus detection (more than 4 seconds, that is, 80,000 cycles at 20 kHz) Reset type: SYSRSn 0h (R/W) = Normal operation 1h (R/W) = Request local low-power mode

15.7.2.4 SCISETINT Register (Offset = Ch) [reset = 0h]

The SCISETINT register is used to enable the various interrupts available in the LIN module.

Figure 15-29. SCISETINT Register

31	30	29	28	27	26	25	24
SETBEINT	SETPBEINT	SETCEINT	SETISFEINT	SETNREINT	SETFEINT	SETOEINT	SETPEINT
R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED		SETIDINT	RESERVED			SETRXINT	SETTXINT
R-0h		R/W1S-0h	R-0h			R/W1S-0h	R/W1S-0h
7	6	5	4	3	2	1	0
SETTOA3WU SINT	SETTOAWU SINT	RESERVED	SETTIMEOUT INT	RESERVED		SETWAKEUP INT	SETBRKDT INT
R/W1S-0h	R/W1S-0h	R-0h	R/W1S-0h	R-0h		R/W1S-0h	R/W1S-0h

Table 15-15. SCISETINT Register Field Descriptions

Bit	Field	Type	Reset	Description
31	SETBEINT	R/W1S	0h	Set bit error interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN module to generate an interrupt when there is a bit error. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
30	SETPBEINT	R/W1S	0h	Set physical bus error interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN module to generate an interrupt when a physical bus error occurs. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
29	SETCEINT	R/W1S	0h	Set checksum-error Interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN module to generate an interrupt when there is a checksum error. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
28	SETISFEINT	R/W1S	0h	Set inconsistent-sync-field-error interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN module to generate an interrupt when there is an inconsistent sync field error. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.

Table 15-15. SCISETINT Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
27	SETNREINT	R/W1S	0h	Set no-response-error interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN module to generate an interrupt when a no-response error occurs. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
26	SETFEINT	R/W1S	0h	Set framing-error interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit enables the SCI/LIN module to generate an interrupt when a framing error occurs. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
25	SETOEINT	R/W1S	0h	Set overrun-error interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit enables the SCI/LIN module to generate an interrupt when an overrun error occurs. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
24	SETPEINT	R/W1S	0h	Set parity interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit enables the SCI/LIN module to generate an interrupt when a parity error occurs. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
23-14	RESERVED	R	0h	Reserved
13	SETIDINT	R/W1S	0h	Set Identification interrupt. This bit is effective in LIN mode only. This bit is set to enable interrupt once a valid matching identifier is received. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
12-10	RESERVED	R	0h	Reserved
9	SETRXINT	R/W1S	0h	Set Receiver interrupt. Setting this bit enables the SCI/LIN to generate a receive interrupt after a frame has been completely received and the data is being transferred from SCIRXSHF to SCIRD. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
8	SETTXINT	R/W1S	0h	Set Transmitter interrupt. Setting this bit enables the SCI/LIN to generate a transmit interrupt as data is being transferred from SCITD to SCITXSHF and the TXRDY bit is being set. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.

Table 15-15. SCISETINT Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
7	SETTOA3WUSINT	R/W1S	0h	Set Timeout After 3 Wakeup Signals interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN to generate an interrupt when there is a timeout after 3 wakeup signals have been sent. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
6	SETTOAWUSINT	R/W1S	0h	Set Timeout After Wakeup Signal interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN to generate an interrupt when there is a timeout after one wakeup signal has been sent. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
5	RESERVED	R	0h	Reserved
4	SETTIMEOUTINT	R/W1S	0h	Set timeout interrupt. This bit is effective in LIN mode only. Setting this bit enables the SCI/LIN to generate an interrupt when no LIN bus activity (bus idle) occurs for at least 4 seconds. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
3-2	RESERVED	R	0h	Reserved
1	SETWAKEUPINT	R/W1S	0h	Set wake-up interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit enables the SCI/LIN to generate a wake-up interrupt and thereby exit low-power mode. The wake-up interrupt is asserted on falling edge of the wake-up pulse. If enabled, the wake-up interrupt is asserted when local low-power mode is requested while the receiver is busy or if a low level is detected on the SCIRX pin during low-power mode. Wake-up interrupt is not asserted upon a wakeup pulse if the module is not in power down mode. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.
0	SETBRKDTINT	R/W1S	0h	Set break-detect interrupt. This bit is effective in SCI-compatible mode only. Setting this bit enables the SCI/LIN to generate an interrupt if a break condition is detected on the LINRX pin. This field is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled.

15.7.2.5 SCICLEARINT Register (Offset = 10h) [reset = 0h]

The SCICLEARINT register is used to disable the enabled interrupts without accessing the SCISETINT register.

Figure 15-30. SCICLEARINT Register

31	30	29	28	27	26	25	24
CLRBEINT	CLRPBEINT	CLRCEINT	CLRISFEINT	CLRNREINT	CLRFEINT	CLROEINT	CLRPEINT
R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED		CLRIDINT	RESERVED			CLRRXINT	CLRTXINT
R-0h		R/W1C-0h	R-0h			R/W1C-0h	R/W1C-0h
7	6	5	4	3	2	1	0
CLRTOA3WUSINT	CLRTOAWUSINT	RESERVED	CLRTIMEOUTINT	RESERVED		CLRWAKEUPINT	CLRBRKDTINT
R/W1C-0h	R/W1C-0h	R-0h	R/W1C-0h	R-0h		R/W1C-0h	R/W1C-0h

Table 15-16. SCICLEARINT Register Field Descriptions

Bit	Field	Type	Reset	Description
31	CLRBEINT	R/W1C	0h	Clear Bit Error Interrupt. This bit is effective in LIN mode only. Setting this bit disables the bit error interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
30	CLRPBEINT	R/W1C	0h	Clear Physical Bus Error Interrupt. This bit is effective in LIN mode only. Setting this bit disables the physical-bus error interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
29	CLRCEINT	R/W1C	0h	Clear checksum-error Interrupt. This bit is effective in LIN mode only. Setting this bit disables the checksum-error interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
28	CLRISFEINT	R/W1C	0h	Clear Inconsistent-Sync-Field-Error Interrupt. This bit is effective in LIN mode only. Setting this bit disables the ISFE interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.

Table 15-16. SCICLEARINT Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
27	CLRNREINT	R/W1C	0h	Clear No-Reponse-Error Interrupt. This bit is effective in LIN mode only. Setting this bit disables the no-response error interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
26	CLRFEINT	R/W1C	0h	Clear Framing-Error Interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit disables framing-error interrupt. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
25	CLROEINT	R/W1C	0h	Clear Overrun-Error Interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit disables the overrun interrupt. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
24	CLRPEINT	R/W1C	0h	Clear Parity Interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit disables the parity error interrupt. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
23-14	RESERVED	R	0h	Reserved
13	CLRIDINT	R/W1C	0h	Clear Identifier interrupt. This bit is effective in LIN mode only. Setting this bit disables the ID interrupt. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
12-10	RESERVED	R	0h	Reserved
9	CLRRXINT	R/W1C	0h	Clear Receiver interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit disables the receiver interrupt. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
8	CLRTXINT	R/W1C	0h	Clear Transmitter interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit disables the transmitter interrupt. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.

Table 15-16. SCICLEARINT Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
7	CLRTOA3WUSINT	R/W1C	0h	Clear Timeout After 3 Wakeup Signals interrupt. This bit is effective in LIN mode only. Setting this bit disables the timeout after 3 wakeup signals interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
6	CLRTOAWUSINT	R/W1C	0h	Clear Timeout After Wakeup Signal interrupt. This bit is effective in LIN mode only. Setting this bit disables the timeout after one wakeup signal interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
5	RESERVED	R	0h	Reserved
4	CLRTIMEOUTINT	R/W1C	0h	Clear Timeout interrupt. This bit is effective in LIN mode only. Setting this bit disables the timeout (LIN bus idle) interrupt. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
3-2	RESERVED	R	0h	Reserved
1	CLRWAKEUPINT	R/W1C	0h	Clear Wake-up interrupt. This bit is effective in LIN or SCI-compatible mode. Setting this bit disables the wake-up interrupt. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.
0	CLRBRKDTINT	R/W1C	0h	Clear Break-detect interrupt. This bit is effective in SCI-compatible mode only. Setting this bit disables the Break-detect interrupt. This field is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = Interrupt is disabled. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt is enabled. Writing a 1 to this bit will disable the interrupt and clear this bit.

15.7.2.6 SCISETINTLVL Register (Offset = 14h) [reset = 0h]

The SCISETINTLVL register is used to map individual interrupt sources to the INT1 interrupt line.

Figure 15-31. SCISETINTLVL Register

31	30	29	28	27	26	25	24
SETBEINT LVL	SETPBEINT LVL	SETCEINT LVL	SETISFEINT LVL	SETNREINT LVL	SETFEINT LVL	SETOEINT LVL	SETPEINT LVL
R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h	R/W1S-0h
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED		SETIDINTLVL	RESERVED			SETRXINTOVO	SETTXINTLVL
R-0h		R/W1S-0h	R-0h			R/W1S-0h	R/W1S-0h
7	6	5	4	3	2	1	0
SETTOA3WU SINTLVL	SETTOAWU SINTLVL	RESERVED	SETTIMEOUT INTLVL	RESERVED		SETWAKEUP INTLVL	SETBRKDT INTLVL
R/W1S-0h	R/W1S-0h	R-0h	R/W1S-0h	R-0h		R/W1S-0h	R/W1S-0h

Table 15-17. SCISETINTLVL Register Field Descriptions

Bit	Field	Type	Reset	Description
31	SETBEINTLVL	R/W1S	0h	Set Bit Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Bit Error interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
30	SETPBEINTLVL	R/W1S	0h	Set Physical Bus Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Physical Bus Error interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
29	SETCEINTLVL	R/W1S	0h	Set Checksum-error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Checksum-error interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
28	SETISFEINTLVL	R/W1S	0h	Set Inconsistent-Sync-Field-Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Inconsistent-Sync-Field-Error interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.

Table 15-17. SCISETINTLVL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
27	SETNREINTLVL	R/W1S	0h	Set No-Reponse-Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the No-Response-Error interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
26	SETFEINTLVL	R/W1S	0h	Set Framing-Error interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Framing-Error interrupt level to the INT1 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
25	SETOEINTLVL	R/W1S	0h	Set Overrun-Error Interrupt Level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Overrun-Error interrupt level to the INT1 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
24	SETPEINTLVL	R/W1S	0h	Set Parity Error interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Parity error interrupt level to the INT1 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
23-14	RESERVED	R	0h	Reserved
13	SETIDINTLVL	R/W1S	0h	Set ID interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the ID interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
12-10	RESERVED	R	0h	Reserved
9	SETRXINTOVO	R/W1S	0h	Set Receiver interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the receiver interrupt level to the INT1 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
8	SETTXINTLVL	R/W1S	0h	Set Transmitter interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the transmitter interrupt level to the INT1 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
7	SETTOA3WUSINTLVL	R/W1S	0h	Set Timeout After 3 Wakeup Signals interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the timeout after 3 wakeup signals interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.

Table 15-17. SCISETINTLVL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
6	SETTOAWUSINTLVL	R/W1S	0h	Set Timeout After Wakeup Signal interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the timeout after wakeup interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
5	RESERVED	R	0h	Reserved
4	SETTIMEOUTINTLVL	R/W1S	0h	Set Timeout interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the timeout interrupt level to the INT1 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
3-2	RESERVED	R	0h	Reserved
1	SETWAKEUPINTLVL	R/W1S	0h	Set Wake-up interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Wake-up interrupt level to the INT1 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.
0	SETBRKDTINTLVL	R/W1S	0h	Set Break-detect interrupt level. This bit is effective in SCI-compatible mode only. Writing to this bit maps the Break-detect interrupt level to the INT1 line. This field is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line.

15.7.2.7 SCICLEARINTLVL Register (Offset = 18h) [reset = 0h]

The SCICLEARINTLVL register is used to map individual interrupt sources to the INT0 line.

Figure 15-32. SCICLEARINTLVL Register

31	30	29	28	27	26	25	24
CLRBEINTLVL	CLRPBEINTLVL	CLRCEINTLVL	CLRISFEINTLVL	CLRNREINTLVL	CLRFEINTLVL	CLROEINTLVL	CLRPEINTLVL
R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED		CLRIDINTLVL	RESERVED			CLRRXINTLVL	CLRTXINTLVL
R-0h		R/W1C-0h	R-0h			R/W1C-0h	R/W1C-0h
7	6	5	4	3	2	1	0
CLRTOA3WUSINTLVL	CLRTOAWUSINTLVL	RESERVED	CLRTIMEOUTINTLVL	RESERVED		CLRWAKEUPINTLVL	CLRBRKDTINTLVL
R/W1C-0h	R/W1C-0h	R-0h	R/W1C-0h	R-0h		R/W1C-0h	R/W1C-0h

Table 15-18. SCICLEARINTLVL Register Field Descriptions

Bit	Field	Type	Reset	Description
31	CLRBEINTLVL	R/W1C	0h	Clear Bit Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Bit Error interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
30	CLRPBEINTLVL	R/W1C	0h	Clear Physical Bus Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Physical Bus Error interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
29	CLRCEINTLVL	R/W1C	0h	Clear Checksum-error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Checksum-error interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
28	CLRISFEINTLVL	R/W1C	0h	Clear Inconsistent-Sync-Field-Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the Inconsistent-Sync-Field-Error interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.

Table 15-18. SCICLEARINTLVL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
27	CLRNREINTLVL	R/W1C	0h	Clear No-Reponse-Error interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the No-Response-Error interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
26	CLRFEINTLVL	R/W1C	0h	Clear Framing-Error interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Framing-Error interrupt level to the INT0 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
25	CLROEINTLVL	R/W1C	0h	Clear Overrun-Error Interrupt Level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Overrun-Error interrupt level to the INT0 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
24	CLRPEINTLVL	R/W1C	0h	Clear Parity Error interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Parity Error interrupt level to the INT0 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
23-14	RESERVED	R	0h	Reserved
13	CLRIDINTLVL	R/W1C	0h	Clear ID interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the ID interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
12-10	RESERVED	R	0h	Reserved
9	CLRRXINTLVL	R/W1C	0h	Clear Receiver interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the receiver interrupt level to the INT0 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
8	CLRTXINTLVL	R/W1C	0h	Clear Transmitter interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the transmitter interrupt level to the INT0 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.

Table 15-18. SCICLEARINTLVL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
7	CLRTOA3WUSINTLVL	R/W1C	0h	Clear Timeout After 3 Wakeup Signals interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the timeout after 3 wakeup signals interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
6	CLRTOAWUSINTLVL	R/W1C	0h	Clear Timeout After Wakeup Signal interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the timeout after wakeup interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
5	RESERVED	R	0h	Reserved
4	CLRTIMEOUTINTLVL	R/W1C	0h	Clear Timeout interrupt level. This bit is effective in LIN mode only. Writing to this bit maps the timeout interrupt level to the INT0 line. This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
3-2	RESERVED	R	0h	Reserved
1	CLRWAKEUPINTLVL	R/W1C	0h	Clear Wake-up interrupt level. This bit is effective in LIN or SCI-compatible mode. Writing to this bit maps the Wake-up interrupt level to the INT0 line. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. Writing a 0 to this bit has no effect. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.
0	CLRBRKDTINTLVL	R/W1C	0h	Clear Break-detect interrupt level. This bit is effective in SCI-compatible mode only. Writing to this bit maps the Break-detect interrupt level to the INT0 line. This field is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = Interrupt level mapped to INT0 line. 1h (R/W) = Interrupt level mapped to INT1 line. Writing a 1 to this bit will map the interrupt to INT0 and clear this bit.

15.7.2.8 SCIFLR Register (Offset = 1Ch) [reset = 900h]

The SCIFLR register indicates the current status of the various interrupt sources of the LIN module.

Figure 15-33. SCIFLR Register

31	30	29	28	27	26	25	24
BE	PBE	CE	ISFE	NRE	FE	OE	PE
R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h	R/W1C-0h
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED	IDRXFLAG	IDTXFLAG	RXWAKE	TXEMPTY	TXWAKE	RXRDY	TXRDY
R-0h	R/W1C-0h	R/W1C-0h	R-0h	R-1h	R/W-0h	R/W1C-0h	R-1h
7	6	5	4	3	2	1	0
TOA3WUS	TOAWUS	RESERVED	TIMEOUT	BUSY	IDLE	WAKEUP	BRKDT
R/W1C-0h	R/W1C-0h	R-0h	R/W1C-0h	R-0h	R-0h	R/W1C-0h	R/W1C-0h

Table 15-19. SCIFLR Register Field Descriptions

Bit	Field	Type	Reset	Description
31	BE	R/W1C	0h	Bit Error Flag. This bit is effective in LIN mode only. This bit is set when there has been a bit error. This is detected by the bit monitor in the internal bit monitor. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit - Reception of a new sync break This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No bit error detected. 1h (R/W) = Bit error detected.
30	PBE	R/W1C	0h	Physical Bus Error Flag. This bit is effective in LIN mode only. This bit is set when there has been a physical bus error. This is detected by the bit monitor in TED. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit - Reception of a new sync break Note: the PBE will only be flagged if no sync break can be generated. (because of a bus shortage to VBAT) or if no sync break delimiter can be generated (because of a bus shortage to GND). This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No physical bus error detected. 1h (R/W) = Physical bus error detected.

Table 15-19. SCIFLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
29	CE	R/W1C	0h	Checksum Error Flag. This bit is effective in LIN mode only. This bit is set when there is checksum error detected by a receiving node. The type of checksum to be used depends on the SCIGCR1.CTYPE bit. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit - Reception of a new sync break This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No Checksum error detected. 1h (R/W) = Checksum error detected.
28	ISFE	R/W1C	0h	Inconsistent Sync Field Error Flag. This bit is effective in LIN mode only. This bit is set when there has been an inconsistent Sync Field error detected by the synchronizer during header reception. See Section 15.3.1.5.2 for more information. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit - Reception of a new sync break This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No Inconsistent Sync Field error detected. 1h (R/W) = Inconsistent Sync Field error detected.
27	NRE	R/W1C	0h	No-Response Error Flag. This bit is effective in LIN mode only. This bit is set when there is no response to a master's header completed within TFRAME_MAX. This timeout period is applied for message frames of unknown length (identifiers 0 to 61). This error is detected by the synchronizer of the module. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit - Reception of a new sync break This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No No-Response error detected. 1h (R/W) = No-Response error detected.

Table 15-19. SCIFLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
26	FE	R/W1C	0h	Framing error flag. This bit is effective in LIN or SCI-compatible mode. This bit is set when an expected stop bit is not found. In SCI compatible mode, 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. Detection of a framing error causes the SCI to generate an error interrupt if the RXERR INT ENA bit is set. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit - Reception of a new character (SCI-compatible mode), or frame (LIN mode) In multibuffer mode the frame is defined in the SCIFORMAT register. Reset type: SYSRSn 0h (R/W) = No framing error detected. 1h (R/W) = Framing error detected.
25	OE	R/W1C	0h	Overrun error flag. This bit is effective in LIN or SCI-compatible mode. This bit is set when the transfer of data from SCIRXSHF to SCIRD overwrites unread data already in SCIRD or the RDy buffers. Detection of an overrun error causes the LIN to generate an error interrupt if the SET OE INT bit is one. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit Reset type: SYSRSn 0h (R/W) = No overrun error detected. 1h (R/W) = Overrun error detected.
24	PE	R/W1C	0h	Parity error flag. This bit is effective in LIN or SCI-compatible mode. This bit is set when a parity error is detected in the received data. In SCI address-bit mode, the parity is calculated on the data and address bit fields of the received frame. In idle-line mode, only the data is used to calculate parity. An error is generated when a character is received with a mismatch between the number of 1s and its parity bit. For more information on parity checking, see the "SCI Global Control Register (SCIGCR1)" description. If the parity function is disabled (that is, SCIGCR1.2 = 0), the PE flag is disabled and read as 0. Detection of a parity error causes the LIN to generate an error interrupt if the SET PE INT bit =1. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Reception of a new character (SCI-compatible mode) or frame (LIN mode) - Writing a 1 to this bit Reset type: SYSRSn 0h (R/W) = No parity error or parity disabled. 1h (R/W) = Parity error detected.
23-15	RESERVED	R	0h	Reserved

Table 15-19. SCIFLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
14	IDRXFLAG	R/W1C	0h	Identifier On Receive Flag. This bit is effective in LIN mode only. This flag is set once an identifier is received with an RX match and no ID-parity error. See Section 15.3.1.9 for more details. When this flag is set it indicates that a new valid identifier has been received on an RX match. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Reading the LINID register - Writing a 1 to this bit This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No valid ID received. 1h (R/W) = Valid ID RX received in LINID[23:16] on RX match.
13	IDTXFLAG	R/W1C	0h	Identifier On Transmit Flag. This bit is effective in LIN mode only. This flag is set once an identifier is received with a TX match and no ID-parity error. See Section 15.3.1.9 for more details. When this flag is set it indicates that a new valid identifier has been received on a TX match. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - RESET bit (SCIGCR0.0) - Setting SWnRESET - System reset - Reading the LINID register - Writing a 1 to this bit This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No valid ID received. 1h (R/W) = Valid ID received in LINID[23:16] on TX match.
12	RXWAKE	R	0h	Receiver wakeup detect flag. This bit is effective in SCI-compatible mode only. The SCI sets this bit to indicate that the data currently in SCIRD is an address. This bit is cleared by: - RESET bit - Setting the SWnRESET - System reset - Receipt of a data frame This bit is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = The data in SCIRD is not an address. 1h (R/W) = The data in SCIRD is an address. See Section 15.2.4.1 for more information on using the RXWAKE bit with sleep mode.

Table 15-19. SCIFLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
11	TXEMPTY	R	1h	Transmitter Empty flag. The value of this flag indicates the contents of the transmitter's buffer register(s) (SCITD/TDy) and shift register (SCITXSHF). In multibuffer mode, this flag indicates the value of the TDx registers and shift register (SCITXSHF). In non-multibuffer mode, this flag indicates the value of LINTDO (byte) and shift register (SCITXSHF). This bit is set by: - RESET bit (SCIGCR0.0) - Setting the SWnRESET (SCIGCR1.7 - System reset. Note: This bit does not cause an interrupt request. Reset type: SYSRSn 0h (R/W) = Compatible mode or LIN with no multibuffer: Transmitter buffer or shift register (or both) are loaded with data. In LIN mode using multibuffer mode: Multibuffer or shift register (or all) are loaded with data. 1h (R/W) = Compatible mode or LIN with no multibuffer: Transmitter buffer and shift registers are both empty. In LIN mode using multibuffer mode: Multibuffer and shift registers are all empty.
10	TXWAKE	R/W	0h	SCI transmitter wakeup method select. This bit is effective in SCI-compatible mode only. The TXWAKE bit controls whether the data in SCITD should be sent as an address or data frame using multiprocessor communication format. This bit is set to 1 or 0 by software before a byte is written to SCITD and is cleared by the SCI when data is transferred from SCITD to SCITXSHF or by a system reset. TXWAKE is not cleared by the SWnRESET bit (SCIGCR1.7). Reset type: SYSRSn 0h (R/W) = Address-bit mode: Frame to be transmitted will be data (address bit = 0). Idle-line mode: Frame to be transmitted will be data. 1h (R/W) = Address-bit mode: Frame to be transmitted will be an address (address bit=1). Idle-line mode: Following frame to be transmitted will be an address (writing a 1 to this bit followed by writing dummy data to the SCITD will result in a idle period of 11 bit periods before the next frame is transmitted).
9	RXRDY	R/W1C	0h	Receiver ready flag. In SCI compatibility mode, the receiver sets this bit to indicate that the SCIRD contains new data and is ready to be read by the CPU. In LIN mode, RXRDY is set once a valid frame is received in multibuffer mode, a valid frame being a message frame received with no errors. In non-multibuffer mode RXRDY is set for each received byte and will be set for the last byte of the frame if there are no errors. The SCI/LIN generates a receive interrupt when RXRDY flag bit is set if the interrupt-enable bit is set (SCISETINT.9). RXRDY is cleared by: - RESET bit (SCIGCR0.0) - Setting the SWnRESET - System reset - Writing a 1 to this bit - Reading SCIRD in while in SCI compatibility mode - Reading last data byte RDy of the response in LIN mode Note: The RXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0/1 register. Reset type: SYSRSn 0h (R/W) = No new data in SCIRD/RDy. 1h (R/W) = New data ready to be read from SCIRD.

Table 15-19. SCIFLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
8	TXRDY	R	1h	Transmitter buffer register ready flag. When set, this bit indicates that the transmit buffer(s) register (SCITD in compatibility mode and LINTDO, LINTD1 in MBUF mode) is/are ready to get another character from a CPU write. In SCI compatibility mode, writing data to SCITD automatically clears this bit. In LIN mode, this bit is cleared once byte 0 (TD0) is written to LINTD0. This bit is set after the data of the TX buffer are shifted into the SCITXSHF register. . This bit is set to 1 by: - RESET bit (SCIGCR0.0) - Setting the SWnRESET (SCIGCR1.7) - System reset Note: The TXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0/1 register. Note: The transmit interrupt request can be eliminated until the next series of data is written into the transmit buffers LINTD0 and LINTD1, by disaLINg the corresponding interrupt via the SCICLEARINT register or by disaLINg the transmitter via the TXENA bit (SCIGCR1.25=0). Reset type: SYSRSn 0h (R/W) = Compatible mode: SCITD is full. LIN mode: The multibuffers are full. 1h (R/W) = Compatible mode: SCITD is ready to receive the next character. LIN mode: The multibuffers are ready to receive the next character(s).
7	TOA3WUS	R/W1C	0h	Timeout After 3 Wakeup Signals flag. This bit is effective in LIN mode only. This flag is set if there is no Sync Break received after 3 wakeup signals and a period of 1.5 seconds have passed. Such expiration time is used before issuing another round of wakeup signals. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No timeout after 3 wakeup signals. 1h (R/W) = Timeout after 3 wakeup signals and 1.5s time.
6	TOAWUS	R/W1C	0h	Timeout After Wakeup Signal flag. This bit is effective in LIN mode only. This bit is set if there is no Sync Break received after a wakeup signal has been sent. A minimum of 150 ms expiration time is used before issuing another wakeup signal. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No timeout after one wakeup signal (150 ms). 1h (R/W) = Timeout after one wakeup signal.
5	RESERVED	R	0h	Reserved

Table 15-19. SCIFLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	TIMEOUT	R/W1C	0h	LIN Bus IDLE timeout flag. This bit is effective in LIN mode only. This bit is set if there is no LIN bus activity for at least 4 seconds. LIN bus activity being a transition from recessive to dominant. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = No bus idle detected. 1h (R/W) = LIN bus idle detected.
3	BUSY	R	0h	Bus BUSY flag. This bit is effective in LIN mode and SCI-compatible mode. This bit indicates whether the receiver is in the process of receiving a frame. As soon as the receiver detects the beginning of a start bit, the BUSY bit is set to 1. When the reception of a frame is complete, the BUSY bit is cleared. If SET WAKEUP INT is set and power down is requested while this bit is set, the SCI/LIN automatically prevents low-power mode from being entered and generates wakeup interrupt. The BUSY bit is controlled directly by the SCI receiver but can be cleared by: - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset. Reset type: SYSRSn 0h (R/W) = Receiver is not currently receiving a frame. 1h (R/W) = Receiver is currently receiving a frame.
2	IDLE	R	0h	SCI receiver in idle state. This bit is effective in SCI-compatible mode only. While this bit is set, the SCI looks for an idle period to resynchronize itself with the bit stream. The receiver does not receive any data while the bit is set. The bus must be idle for 11 bit periods to clear this bit. The SCI enters this state: - After a system reset - After a SCI software reset - After coming out of power down This bit is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = Idle period detected, the SCI is ready to receive. 1h (R/W) = Idle period not detected, the SCI will not receive any data.
1	WAKEUP	R/W1C	0h	Wake-up flag. This bit is effective in LIN mode only. This bit is set by the SCI/LIN when receiver or transmitter activity has taken the module out of power-down mode. An interrupt is generated if the SET WAKEUP INT bit (SCISETINT.1) is set. This bit is cleared by: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register. - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - Writing a 1 to this bit This field is writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = Do not wake up from power-down mode. 1h (R/W) = Wake up from power-down mode.

Table 15-19. SCIFLR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
0	BRKDT	R/W1C	0h	SCI break-detect flag. This bit is effective in SCI-compatible mode only. This bit is set when the SCI detects a break condition on the LINRX pin. A break condition occurs when the LINRX pin remains continuously low for at least 10 bits after a missing first stop bit, that is, after a framing error. Detection of a break condition causes the SCI to generate an error interrupt if the BRKDT INT ENA bit is set. The BRKDT bit is cleared by the following: - Reading the corresponding interrupt offset in the SCIINTVECT0/1 register. - Setting the SWnRESET bit (SCIGCR1.7) - RESET bit (SCIGCR0.0) - System reset - By writing a 1 to this bit This bit is writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = No break condition detected. 1h (R/W) = Break condition detected.

15.7.2.9 SCIINTVECT0 Register (Offset = 20h) [reset = 0h]

The SCIINTVECT0 register indicates the offset for the INT0 interrupt line.

Figure 15-34. SCIINTVECT0 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
R-0h															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED											INTVECT0				
R-0h											R-0h				

Table 15-20. SCIINTVECT0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-5	RESERVED	R	0h	Reserved
4-0	INTVECT0	R	0h	Interrupt vector offset for INT0. This register indicates the offset for interrupt line INT0. A read to this register updates its value to the next highest priority pending interrupt in SCIFLR and clears the flag corresponding to the offset that was read. Note: The flags for the receive (SCIFLR.9) and the transmit (SCIFLR.8) interrupts cannot be cleared by reading the corresponding offset vector in this register (see detailed description in SCIFLR register). Reset type: SYSRSn

15.7.2.10 SCIINTVECT1 Register (Offset = 24h) [reset = 0h]

The SCIINTVECT1 register indicates the offset for the INT1 interrupt line.

Figure 15-35. SCIINTVECT1 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
R-0h															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED											INTVECT1				
R-0h											R-0h				

Table 15-21. SCIINTVECT1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-5	RESERVED	R	0h	Reserved
4-0	INTVECT1	R	0h	Interrupt vector offset for INT1. This register indicates the offset for interrupt line INT1. A read to this register updates its value to the next highest priority pending interrupt in SCIFLR and clears the flag corresponding to the offset that was read. Note: The flags for the receive (SCIFLR.9) and the transmit (SCIFLR.8) interrupts cannot be cleared by reading the corresponding offset vector in this register (see detailed description in SCIFLR register). Reset type: SYSRSn

15.7.2.11 SCIFORMAT Register (Offset = 28h) [reset = 0h]

The SCIFORMAT register is used to set up the character and frame lengths.

Figure 15-36. SCIFORMAT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED													LENGTH		
R-0h													R/W-0h		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED													CHAR		
R-0h													R/W-0h		

Table 15-22. SCIFORMAT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-19	RESERVED	R	0h	Reserved
18-16	LENGTH	R/W	0h	Frame length control bits. In LIN mode, these bits indicate the number of bytes in the response field from 1 to 8 bytes. In buffered SCI mode, these bits indicate the number of characters. When these bits are used to indicate LIN response length (SCIGCR1[0] = 1), then when there is an ID RX match, this value should be updated with the expected length of the response. In buffered SCI mode, these bits indicate the number of characters with SCIFORMAT[2:0] bits per character, that is, these bits indicate the transmitter/receiver format for the number of characters: 1 to 8. There can be up to eight characters with eight bits each. Reset type: SYSRSn 0h (R/W) = The response field has 1 bytes/characters. 1h (R/W) = The response field has 2 bytes/characters. 2h (R/W) = The response field has 3 bytes/characters. 3h (R/W) = The response field has 4 bytes/characters. 4h (R/W) = The response field has 5 bytes/characters. 5h (R/W) = The response field has 6 bytes/characters. 6h (R/W) = The response field has 7 bytes/characters. 7h (R/W) = The response field has 8 bytes/characters.
15-3	RESERVED	R	0h	Reserved
2-0	CHAR	R/W	0h	Character length control bits. These bits are effective in SCI compatible mode only. These bits set the SCI character length from 1 to 8 bits. Note: In compatibility mode or buffered SCI mode, when data of fewer than eight bits in length is received, it is left justified in SCIRD/RDy and padded with trailing zeros. Data read from the SCIRD should be shifted by software to make the received data right justified. Note: Data written to the SCITD should be right justified but does not need to be padded with leading zeros. These bits are writable in SCI mode only. Reset type: SYSRSn 0h (R/W) = The character is 1 bits long. 1h (R/W) = The character is 2 bits long. 2h (R/W) = The character is 3 bits long. 3h (R/W) = The character is 4 bits long. 4h (R/W) = The character is 5 bits long. 5h (R/W) = The character is 6 bits long. 6h (R/W) = The character is 7 bits long. 7h (R/W) = The character is 8 bits long.

15.7.2.12 BRSR Register (Offset = 2Ch) [reset = 0h]

The BRSR register is used to configure the baud rate of the LIN module.

Figure 15-37. BRSR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED				M				SCI_LIN_PSH							
R-0h				R/W-0h				R/W-0h							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCI_LIN_PSL															
R/W-0h															

Table 15-23. BRSR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-28	RESERVED	R	0h	Reserved
27-24	M	R/W	0h	SCI/LIN 4-bit Fractional Divider Selection. (M) These bits are effective in LIN or SCI asynchronous mode. These bits are used to select a baud rate for the SCI/LIN module, and they are a fractional part for the baud rate specification. The M divider allows fine-tuning of the baud rate over the P prescaler with 15 additional intermediate values for each of the P integer values. Reset type: SYSRSn
23-16	SCI_LIN_PSH	R/W	0h	PRESCALER P (High Bits). SCI/LIN 24-bit Integer Prescaler Selection. These bits are used to select a baud rate for the SCI/LIN module. These bits are effective in LIN mode and SCI compatible mode. The SCI/LIN has an internally generated serial clock determined by the LIN module input clock and the prescalers P and M in this register. The SCI/LIN uses the 24-bit integer prescaler P value to select 1 of over 16,700,000 available baud rates. The additional 4-bit fractional prescaler M refines the baud rate selection. Reset type: SYSRSn
15-0	SCI_LIN_PSL	R/W	0h	PRESCALER P (Low Bits). SCI/LIN 24-bit Integer Prescaler Selection. These bits are used to select a baud rate for the SCI/LIN module. These bits are effective in LIN mode and SCI compatible mode. The SCI/LIN has an internally generated serial clock determined by the LIN module input clock and the prescalers P and M in this register. The SCI/LIN uses the 24-bit integer prescaler P value to select 1 of over 16,700,000 available baud rates. The additional 4-bit fractional prescaler M refines the baud rate selection. Reset type: SYSRSn

15.7.2.13 SCIED Register (Offset = 30h) [reset = 0h]

The SCIED register is a duplicate copy of SCIRD register that has no affect on the RXRDY flag for use with a debugger.

Figure 15-38. SCIED 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																								ED							
R-0h																								R-0h							

Table 15-24. SCIED Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	Reserved
7-0	ED	R	0h	Receiver Emulation Data. This bit is effective in SCI-compatible mode only. Reading SCIED(7-0) does not clear the RXRDY flag. This register should be used only by a debugger that must continually read the data buffer without affecting the RXRDY flag. Reset type: SYSRSn

15.7.2.14 SCIRD Register (Offset = 34h) [reset = 0h]

The SCIRD register is where received data is stored and can be read from.

Figure 15-39. SCIRD 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																								RD							
R-0h																								R-0h							

Table 15-25. SCIRD Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	Reserved
7-0	RD	R	0h	Received Data. This bit is effective in SCI-compatible mode only. When a frame has been completely received, the data in the frame is transferred from the receiver shift register SCIRXSHF to this register. As this transfer occurs, the RXRDY flag is set and a receive interrupt is generated if RX INT ENA (SCISETINT0.9) is set. When the data is read from SCIRD, the RXRDY flag is automatically cleared. When the SCI receives data that is fewer than eight bits in length, it loads the data into this register in a left justified format padded with trailing zeros. Therefore, your software should perform a logical shift on the data by the correct number of positions to make it right justified. Reset type: SYSRSn

15.7.2.15 SCITD Register (Offset = 38h) [reset = 0h]

The SCITD register is where data to be transmitted is written to by application software.

Figure 15-40. SCITD 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																								TD							
R-0h																								R/W-0h							

Table 15-26. SCITD Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	Reserved
7-0	TD	R/W	0h	Transmit data This bit is effective in SCI-compatible mode only. Data to be transmitted is written to this register. The transfer of data from this register to the transmit shift register SCITXSHF sets the TXRDY flag (SCIFLR.23), which indicates that SCITD is ready to be loaded with another byte of data. Note: If TX INT ENA (SCISETINT.8) is set, this data transfer also causes an interrupt. Note: Data written to the SCIRD register that is fewer than eight bits long must be right justified, but it does not need to be padded with leading zeros. Reset type: SYSRSn

15.7.2.16 SCIPIO2 Register (Offset = 44h) [reset = 0h]

The SCIPIO2 register indicates the current status of the LINTX and LINRX pins.

Figure 15-41. SCIPIO2 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED					TXIN	RXIN	RESERVED
R-0h					R-0h	R-0h	R-0h

Table 15-27. SCIPIO2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-3	RESERVED	R	0h	Reserved
2	TXIN	R	0h	Transmit data in. This bit is effective in LIN or SCI-compatible mode. This bit contains the current value on the LINTX pin. Reset type: SYSRSn
1	RXIN	R	0h	Receive data in. This bit is effective in LIN or SCI-compatible mode. This bit contains the current value on the LINRX pin. Reset type: SYSRSn
0	RESERVED	R	0h	Reserved

15.7.2.17 LINCOMP Register (Offset = 60h) [reset = 0h]

The LINCOMPARE register is used to configure the sync delimiter and sync break extension.

Figure 15-42. LINCOMP Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
R-0h															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED						SDEL		RESERVED					SBREAK		
R-0h						R/W-0h		R-0h					R/W-0h		

Table 15-28. LINCOMP Register Field Descriptions

Bit	Field	Type	Reset	Description
31-10	RESERVED	R	0h	Reserved
9-8	SDEL	R/W	0h	2-bit Sync Delimiter compare. These bits are effective in LIN mode only. These bits are used to configure the number of Tbit for the sync delimiter in the sync field. The time delay calculation for the synchronization delimiter is: TSDEL = (SDEL + 1)Tbit These bits are writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = The sync delimiter has 1 Tbit. 1h (R/W) = The sync delimiter has 2 Tbit. 2h (R/W) = The sync delimiter has 3 Tbit. 3h (R/W) = The sync delimiter has 4 Tbit.
7-3	RESERVED	R	0h	Reserved
2-0	SBREAK	R/W	0h	3-bit Sync Break extend. LIN mode only. These bits are used to configure the number of Tbits for the sync break to extend the minimum 13 Tbit in the Sync Field to a maximum of 20 Tbit. The time delay calculation for the sync break is: TSYNBRK = 13Tbit + SBREAK x Tbit These bits are writable in LIN mode only. Reset type: SYSRSn 0h (R/W) = The sync break has no additional Tbit. 1h (R/W) = The sync break has 1 additional Tbit. 2h (R/W) = The sync break has 2 additional Tbit. 3h (R/W) = The sync break has 3 additional Tbit. 4h (R/W) = The sync break has 4 additional Tbit. 5h (R/W) = The sync break has 5 additional Tbit. 6h (R/W) = The sync break has 6 additional Tbit. 7h (R/W) = The sync break has 7 additional Tbit.

15.7.2.18 LINRD0 Register (Offset = 64h) [reset = 0h]

The LINRD0 register contains the lower 4 bytes of the received LIN frame data.

Figure 15-43. LINRD0 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
RD0								RD1								RD2								RD3							
R-0h								R-0h								R-0h								R-0h							

Table 15-29. LINRD0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	RD0	R	0h	8-bit Receive Buffer 0 Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. A read of this byte clears the RXDY byte. Note: RD<x-1> is equivalent to Data byte <x> of the LIN frame. Reset type: SYSRSn
23-16	RD1	R	0h	8-bit Receive Buffer 1. Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. Reset type: SYSRSn
15-8	RD2	R	0h	8-bit Receive Buffer 2. Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. Reset type: SYSRSn
7-0	RD3	R	0h	8-bit Receive Buffer 3. Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. Reset type: SYSRSn

15.7.2.19 LINRD1 Register (Offset = 68h) [reset = 0h]

The LINRD1 register contains the upper 4 bytes of the received LIN frame data.

Figure 15-44. LINRD1 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
RD4								RD5								RD6								RD7							
R-0h								R-0h								R-0h								R-0h							

Table 15-30. LINRD1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	RD4	R	0h	8-bit Receive Buffer 4 Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. Reset type: SYSRSn
23-16	RD5	R	0h	8-bit Receive Buffer 5. Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. Reset type: SYSRSn
15-8	RD6	R	0h	8-bit Receive Buffer 6. Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. Reset type: SYSRSn
7-0	RD7	R	0h	8-bit Receive Buffer 7. Each response data-byte that is received in the SCIRXSHFT register is transferred to the corresponding RDy register according to the number of bytes received. Reset type: SYSRSn

15.7.2.20 LINMASK Register (Offset = 6Ch) [reset = 0h]

The LINMASK register is used to configure the masks used for filtering incoming ID messages for receive and transmit frames.

Figure 15-45. LINMASK 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								RXIDMASK								RESERVED								TXIDMASK							
R-0h								R/W-0h								R-0h								R/W-0h							

Table 15-31. LINMASK Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	RESERVED	R	0h	Reserved
23-16	RXIDMASK	R/W	0h	Receive ID mask. This field is effective in LIN mode only.This 8-bit mask is used for filtering an incoming ID message and compare it to the ID-byte. A compare match of the received ID with the RX ID mask will set the ID RX flag and trigger and ID interrupt if enabled. A 0 bit in the mask indicates that bit is compared to the ID-byte. A 1 bit in the mask indicates that bit is filtered and therefore not used in the compare. When HGENCTRL is set to 1, this field must be set to 0xFF. Reset type: SYSRSn
15-8	RESERVED	R	0h	Reserved
7-0	TXIDMASK	R/W	0h	Transmit ID mask. This field is effective in LIN mode only. This 8-bit mask is used for filtering an incoming ID message and compare it to the ID-byte. A compare match of the received ID with the TX ID Mask will set the ID TX flag and trigger an ID interrupt if enabled. A 0 bit in the mask indicates that bit is compared to the ID-byte. A 1 bit in the mask indicates that bit is filtered and therefore not used for the compare. When HGENCTRL is set to 1, this field must be set to 0xFF. Reset type: SYSRSn

15.7.2.21 LINID Register (Offset = 70h) [reset = 0h]

The LINID register contains the identification fields for LIN communication.
NOTE: For software compatibility with future LIN modules, the HGEN CTRL bit must be set to 1, the RX ID MASK field must be set to FFh, and the TX ID MASK field must be set to FFh.

Figure 15-46. LINID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED								RECEIVEDID							
R-0h								R-0h							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDSLAVETASKBYTE								IDBYTE							
R/W-0h								R/W-0h							

Table 15-32. LINID Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	RESERVED	R	0h	Reserved
23-16	RECEIVEDID	R	0h	Received ID. This bit is effective in LIN mode only. This byte contains the current message identifier. During header reception the received ID is copied from the SCIRXSHF register to this byte if there is no ID-parity error and there has been an RX/TX match. Note: If a framing error (FE) is detected during ID reception, the received ID will also not be copied to the LINID register. Reset type: SYSRSn
15-8	IDSLAVETASKBYTE	R/W	0h	ID Slave Task byte. This field is effective in LIN mode only. This byte contains the identifier to which the received ID of an incoming header will be compared in order to decide whether a RX response, a TX response, or no action needs to be done by the LIN node. These bits are writable in LIN mode only. Reset type: SYSRSn
7-0	IDBYTE	R/W	0h	ID byte. This field is effective in LIN mode only. This byte is the LIN mode message ID. On a master node, a write to this register by the CPU initiates a header transmission. For a slave task, this byte is used for message filtering when HGENCTRL (SCIGCR1.12) is '0'. These bits are writable in LIN mode only. Reset type: SYSRSn

15.7.2.22 LINTD0 Register (Offset = 74h) [reset = 0h]

The LINTD0 register contains the lower 4 bytes of the data to be transmitted.
NOTE: TD<x-1> is equivalent to Data byte <x> of the LIN frame.

Figure 15-47. LINTD0 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
TD0								TD1								TD2								TD3							
R/W-0h								R/W-0h								R/W-0h								R/W-0h							

Table 15-33. LINTD0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	TD0	R/W	0h	8-bit Transmit Buffer 0. Byte 0 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Once byte 0 is written in TDO buffer, transmission will be initiated. Reset type: SYSRSn
23-16	TD1	R/W	0h	8-bit Transmit Buffer 3. Byte 1 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Reset type: SYSRSn
15-8	TD2	R/W	0h	8-bit Transmit Buffer 2. Byte 2 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Reset type: SYSRSn
7-0	TD3	R/W	0h	8-bit Transmit Buffer 3. Byte 3 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Reset type: SYSRSn

15.7.2.23 LINTD1 Register (Offset = 78h) [reset = 0h]

The LINTD1 register contains the upper 4 bytes of the data to be transmitted.
NOTE: TD<x-1> is equivalent to Data byte <x> of the LIN frame.

Figure 15-48. LINTD1 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
TD4								TD5								TD6								TD7							
R/W-0h								R/W-0h								R/W-0h								R/W-0h							

Table 15-34. LINTD1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	TD4	R/W	0h	8-bit Transmit Buffer 4. Byte4 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Reset type: SYSRSn
23-16	TD5	R/W	0h	8-bit Transmit Buffer 5. Byte 5 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Reset type: SYSRSn
15-8	TD6	R/W	0h	8-bit Transmit Buffer 6. Byte 6 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Reset type: SYSRSn
7-0	TD7	R/W	0h	8-bit Transmit Buffer 7. Byte 7 to be transmitted is written into this register and then copied to SCITXSHF for transmission. Reset type: SYSRSn

15.7.2.24 MBRSR Register (Offset = 7Ch) [reset = 5DCh]

The MBRSR register is used to configure the expected maximum baud rate of the LIN network.

Figure 15-49. MBRSR 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																			MBR												
R-0h																			R/W-5DCh											

Table 15-35. MBRSR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-13	RESERVED	R	0h	Reserved
12-0	MBR	R/W	5DCh	Maximum Baud Rate Prescaler. This field is effective in LIN mode only. This 13-bit prescaler is used during the synchronization phase (see Section 15.3.1.5.2) of a slave module if the ADAPT bit is set. In this way, a SCI/LIN slave using an automatic or select bit rate modes detects any LIN bus legal rate automatically. The MBR value should be programmed to allow a maximum baud rate that is not more than 10% above the expected operating baud rate in the LIN network. Otherwise a s 0x00 data byte could mistakenly be detected as sync break. The default value is for a 30 MHz LINCLK (0x5DC). This MBR prescaler is used by the wake-up and idle time counters for a constant expiration time relative to a 20 kHz rate. Reset type: SYSRSn

15.7.2.25 IODFTCTRL Register (Offset = 90h) [reset = 0h]

The IODFTCTRL register is used to emulate various error and test conditions.

Figure 15-50. IODFTCTRL Register

31	30	29	28	27	26	25	24
BERRENA	PBERRENA	CERRENA	ISFERRENA	RESERVED	FERRENA	PERRENA	BRKDTERRENA
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h	R/W-0h	R/W-0h	R/W-0h
23	22	21	20	19	18	17	16
RESERVED			PINSAMPLEMASK		TXSHIFT		
R/W-0h			R/W-0h		R/W-0h		
15	14	13	12	11	10	9	8
RESERVED				IODFTENA			
R-0h				R/W-0h			
7	6	5	4	3	2	1	0
RESERVED						LPBENA	RXPENA
R-0h						R/W-0h	R/W-0h

Table 15-36. IODFTCTRL Register Field Descriptions

Bit	Field	Type	Reset	Description
31	BERRENA	R/W	0h	Bit Error Enable bit. This bit is effective in LIN mode only. This bit is used to create a Bit Error. When this bit is set, the bit received is ORed with 1 and passed to the Bit monitor circuitry. Reset type: SYSRSn
30	PBERRENA	R/W	0h	Physical Bus Error Enable bit. This bit is effective in LIN mode only. This bit is used to create a Physical Bus Error. When this bit is set, the bit received during Sync Break field transmission is ORed with 1 and passed to the Bit monitor circuitry Reset type: SYSRSn
29	CERRENA	R/W	0h	Checksum Error Enable bit. This bit is effective in LIN mode only. This bit is used to create a Checksum Error. When this bit is set, the polarity of the CTYPE (checksum type) in the receive checksum calculator is changed so that a checksum error is generated. Reset type: SYSRSn
28	ISFERRENA	R/W	0h	Inconsistent Sync Field Error Enable bit. This bit is effective in LIN mode only. This bit is used to create an ISF error. When this bit is set, the bit widths in the sync field are varied so that the ISF check fails and the error flag is set. Reset type: SYSRSn
27	RESERVED	R	0h	Reserved
26	FERRENA	R/W	0h	Frame Error Enable Bit. This bit is used to create a Frame Error. When this bit is set, the stop bit received is ANDed with '0' and passed to the stop bit check circuitry. Reset type: SYSRSn
25	PERRENA	R/W	0h	Parity Error Enable bit. This bit is effective in SCI-compatible mode only. This bit is used to create a Parity Error. When this bit is set, in SCI-compatible mode, the parity bit received is toggled so that a parity error occurs. Reset type: SYSRSn

Table 15-36. IODFTCTRL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
24	BRKDTERRENA	R/W	0h	Break Detect Error Enable bit. This bit is effective in SCI-compatible mode only. This bit is used to create BRKDT error (SCI mode only). When this bit is set, the stop bit of the frame is ANDed with '0' and passed to the RSM so that a frame error occurs. Then the RX Pin is forced to continuous low for 10 Tbits so that a BRKDT error occurs. Reset type: SYSRSn
23-21	RESERVED	R/W	0h	Reserved
20-19	PINSAMPLEMASK	R/W	0h	Pin sample mask. These bits define the sample number at which the TX Pin value that is being transmitted will be inverted to verify the receive pin samples correctly with the majority detection circuitry. Note: During IODFT mode testing for the pin sample mask, the prescalar P must be programmed to be greater than 2. Reset type: SYSRSn 0h (R/W) = No Mask 1h (R/W) = Invert the TX Pin value at TBIT_CENTER 2h (R/W) = Invert the TX Pin value at TBIT_CENTER + SCLK 3h (R/W) = Invert the TX Pin value at TBIT_CENTER + 2 SCLK
18-16	TXSHIFT	R/W	0h	Transmit shift. These bits define the delay by which the value on LINTX is delayed so that the value on LINRX is asynchronous. (Not applicable to Start Bit) Reset type: SYSRSn 0h (R/W) = No Delay 1h (R/W) = Delay by 1 SCLK 2h (R/W) = Delay by 2 SCLK 3h (R/W) = Delay by 3 SCLK 4h (R/W) = Delay by 4 SCLK 5h (R/W) = Delay by 5 SCLK 6h (R/W) = Delay by 6 SCLK 7h (R/W) = Delay by 7 SCLK
15-12	RESERVED	R	0h	Reserved
11-8	IODFTENA	R/W	0h	IO DFT Enable Key This field is used to enable the IODFT mode of the SCI/LIN module for testing. Reset type: SYSRSn 0h (R/W) = IODFT is disabled 1h (R/W) = IODFT is disabled 2h (R/W) = IODFT is disabled 3h (R/W) = IODFT is disabled 4h (R/W) = IODFT is disabled 5h (R/W) = IODFT is disabled 6h (R/W) = IODFT is disabled 7h (R/W) = IODFT is disabled 8h (R/W) = IODFT is disabled 9h (R/W) = IODFT is disabled Ah (R/W) = IODFT is enabled Bh (R/W) = IODFT is disabled Ch (R/W) = IODFT is disabled Dh (R/W) = IODFT is disabled Eh (R/W) = IODFT is disabled Fh (R/W) = IODFT is disabled
7-2	RESERVED	R	0h	Reserved

Table 15-36. IODFTCTRL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
1	LPBENA	R/W	0h	Module loopback enable. In analog loopback mode the complete communication path through the I/Os can be tested, whereas in digital loopback mode the I/O buffers are excluded from this path. Reset type: SYSRSn 0h (R/W) = Digital loopback is enabled. 1h (R/W) = Analog loopback is enabled in module I/O DFT mode (when IODFTENA = 1010)
0	RXPENA	R/W	0h	Module Analog loopback through receive pin enable. This bit defines whether the I/O buffers for the transmit or the receive pin are included in the communication path in analog loopback mode only. Reset type: SYSRSn 0h (R/W) = Analog loopback through the transmit pin is enabled. 1h (R/W) = Analog loopback through the receive pin is enabled.

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from December 19, 2018 to May 31, 2022 (from Revision * (December 2018) to Revision A (June 2022)) **Page**

- Chapter 10 **Control Law Accelerator (CLA):** 565
- Section 10.4.3.1: Removed "Legacy" from the title..... 575
- Deleted General Instructions table. Subsequent tables renumbered..... 586
- Section 10.6.3: Added an example and more description for the pipeline operation in MMOV32mem32,MSTF#MMOV32memMSTF instruction..... 586
- Chapter 11 **Serial Peripheral Interface (SPI):** 721
- Chapter 12 **Serial Communications Interface (SCI):** 761
- Changed Section 12.1 762
- Added Section 12.1.1 762
- Added Section 12.1.3 763
- Changed Figure 12-10 773
- Chapter 13 **Inter-Integrated Circuit Module (I2C):** 795
- Added Figure 13-6 801
- Added Figure 13-7. Subsequent figures renumbered.....801
- Moved Section 13.3.10 812
- Chapter 14 **Controller Area Network (CAN):** 839
- Changed last paragraph in Section 14.6.2 850
- Chapter 15 **Local Interconnect Network (LIN):** 895
- Section 15.1: Removed K-Line comment..... 896
- Added ".all" to LIN SCI FIFO/Data Buffer in Table 15-8.....936

This page intentionally left blank.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265