TMS320C55x DSP Peripherals Reference Guide

Preliminary Draft

This document contains preliminary data current as of the publication date and is subject to change without notice.

Literature Number: SPRU317B
May 2001

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

This manual describes the peripherals, interfaces, and related hardware that are available on TMS320C55x™ DSPs. It also describes how you can use software to turn on or off individual portions of the C55x™ DSP, so that you can manage power consumption (see Chapter 8, *Idle Configurations*).

The chapters of this document are in alphabetical order.

Notational Conventions

This document uses the following conventions.

- The device number TMS320C55x is often abbreviated as C55x.
- If an underscore is appended to the name of a signal (for example, RESET_), the signal is active low. Similarly, if an underscore is appended to the name of a register bit (for example, XEMPTY_), the bit has a negative polarity; that is, the bit is 0 rather than 1 when it is active.
- Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's.
- In most cases, hexadecimal numbers are shown with the suffix h. For example, the following number is a hexadecimal 40 (decimal 64):

  40h

  Similarly, binary numbers usually are shown with the suffix b. For example, the following number is the decimal number 4 shown in binary form:

  0100b

- Bits and signals are sometimes referenced with the following notations:

Notation	Description	Example
Register(n–m)	Bits n through m of Register	AC0(15–0) represents the 16 least significant bits of the register AC0.
Bus[n:m]	Signals n through m of Bus	A[21:1] represents signals 21 through 1 of the external address bus.

❑ The following terms are used to name portions of data:

Term	Description	Example
LSB	Least significant bit	In AC0(15–0), bit 0 is the LSB.
MSB	Most significant bit	In AC0(15–0), bit 15 is the MSB.
LSByte	Least significant byte	In AC0(15–0), bits 7–0 are the LSByte.
MSByte	Most significant byte	In AC0(15–0), bits 15–8 are the MSByte.
LSW	Least significant word	In AC0(31–0), bits 15–0 are the LSW.
MSW	Most significant word	In AC0(31–0), bits 31–16 are the MSW.

Related Documentation From Texas Instruments

The following books describe the C55x™ devices and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477-8924. When ordering, please identify the book by its title and literature number.

TMS320C55x Technical Overview (SPRU393). This overview is an introduction to the TMS320C55x digital signal processor (DSP). The TMS320C55x is the latest generation of fixed-point DSPs in the TMS320C5000 DSP platform. Like the previous generations, this processor is optimized for high performance and low-power operation. This book describes the CPU architecture, low-power enhancements, and embedded emulation features of the TMS320C55x.

TMS320C55x DSP CPU Reference Guide (literature number SPRU371) describes the architecture, registers, and operation of the CPU.

TMS320C55x DSP Mnemonic Instruction Set Reference Guide (literature number SPRU374) describes the mnemonic instructions individually. It also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the algebraic instruction set.

TMS320C55x DSP Algebraic Instruction Set Reference Guide (literature number SPRU375) describes the algebraic instructions individually. It also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the mnemonic instruction set.

TMS320C55x Optimizing C Compiler User's Guide (literature number SPRU281) describes the 'C55x C compiler. This C compiler accepts ANSI standard C source code and produces assembly language source code for TMS320C55x devices.

TMS320C55x Assembly Language Tools User's Guide (literature number SPRU280) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for TMS320C55x devices.

TMS320C55x DSP Programmer's Reference Guide (literature number SPRU376) describes ways to optimize C and assembly code for the TMS320C55x DSPs and includes application program examples.

Trademarks

Contents

Figures

Tables

Examples

Chapter 1

Analog-to-Digital Converter (ADC)

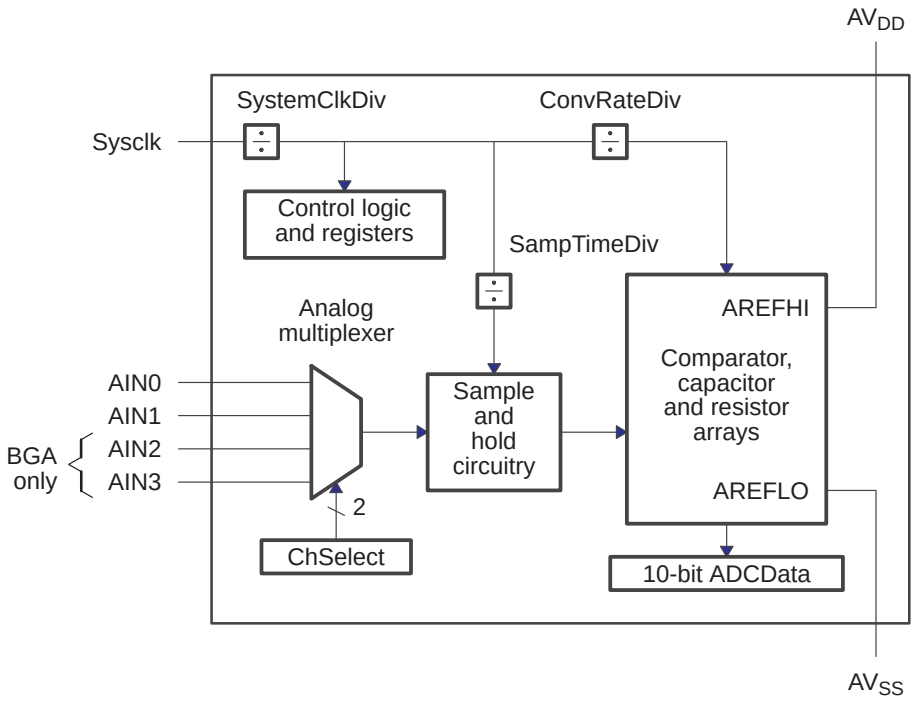
Some TMS320C55x™ (C55x™) DSPs include a 10-bit successive-approximation analog-to-digital converter (ADC). To determine whether a particular C55x DSP has an ADC, see the data sheet for that DSP.

Topic **Page**

1.1 Introduction to the ADC

The ADC module (Figure 1–1) converts an analog input signal to a digital value for use by the DSP. The ADC can sample one of up to four inputs (AIN0–AIN3) at a time, and generates a 10-bit digital representation (ADCData) of the samples. The maximum sampling rate of the ADC is 21.5 KHz. This performance makes the ADC suitable for sampling analog signals that change at a slow rate. For example, the ADC could be used to monitor the voltage drop across a potentiometer on a user interface panel or to sample the voltage on a battery monitoring circuit. The ADC is not intended to be used as the source of the main data stream for the DSP.

Figure 1–1. ADC Block Diagram



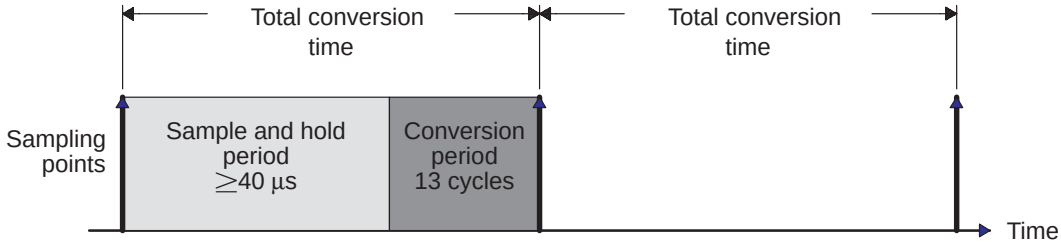
The ADC is based on a successive approximation architecture that achieves very low power consumption. A sample and hold feature is employed to help produce evenly spaced samples. The ADC uses external reference voltages to allow isolation of the conversion process from other system supply voltage planes. Three programmable clock dividers are included to allow flexibility in the choice of DSP input clock.

1.1.1 Total Conversion Time

The total conversion time of the ADC (Figure 1–2) has two components — the sample and hold period, and the conversion period.

- The sample and hold period is the time required for an analog sample to be acquired by the sample and hold circuitry; this time period is greater than or equal to 40 µs.
- The conversion time period is the time required for the successive approximation process to complete for one sample, producing the corresponding digital value. This conversion period requires 13 conversion clock cycles to complete. The internal conversion clock has a maximum frequency of 2 MHz.

Figure 1–2. ADC Total Conversion Time Diagram



The following equations describe the relationship between the ADC programmable clock dividers:

ADC Clock = (*System Clock*) / (*SystemClkDiv* + 1)

ADC Conversion Clock = (*ADC Clock*) / (2 × (*ConvRateDiv* + 1))
must be less than or equal to 2MHz

ADC Sample and Hold Period =
(1 / (*ADC Clock*)) / (2 × (*ConvRateDiv* + 1 + *SampTimeDiv*))
must be greater than or equal to 40 µs

ADC Total Conversion Time =
(*ADC Sample and Hold Period*) + (13 × (1 / (*ADC Conversion Clock*)))

1.1.2 Functional Overview

The ADC does not include a continuos mode; therefore, the DSP must initiate each conversion by writing a 1 to the ADCStart bit in the ADC control register (ADCR).

Once a conversion is initiated, the DSP must wait until the conversion completes before selecting another channel or initiating a new conversion. The ADC does not support interrupts to the DSP or DMA, so the DSP must poll the status of a conversion using the ADCBusy bit in the ADC data register (ADDR).

After the conversion process completes, the ADCBusy bit value changes from 1 to 0, indicating that the conversion data is available. The DSP can then read the data from the ADCData bits in ADDR. The value of the channel select (ChSelect) bits in ADCR is reproduced in ADDR, so that the DSP can identify which samples were acquired from which channel.

1.2 ADC Registers

The ADC memory-mapped registers are listed in Table 1–1.

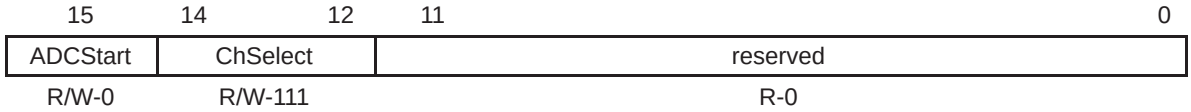
Table 1–1. ADC Memory-Mapped Registers

Address (Hex)	Name	Description
6800	ADCR	ADC Control Register
6801	ADDR	ADC Data Register
6802	ADCDR	ADC Clock Divider Register
6803	ADCCR	ADC Clock Control Register

1.2.1 ADC Control Register (ADCR)

The ADC control register (Figure 1–3) is a read/write register that resets the ADC, selects the analog input channel, and indicates to start a conversion.

Figure 1–3. ADC Control Register (ADCR)



Note: R/W-x = Read/Write-Reset value

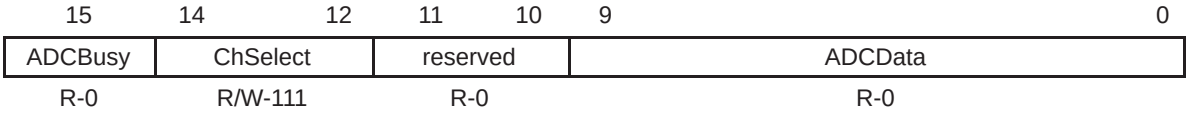
Table 1–2. ADC Control Register (ADCR) Field Values

Bit	*field*	*symval*	Value	Description
15	ADCStart			Start conversion bit.
			0	No effect.
			1	Start conversion cycle. After conversion is complete, the ADC automatically goes into power-down mode unless the ADCStart bit is high again.
14–12	ChSelect			Analog input channel select bits determine which one of four analog inputs to perform a conversion on.
			000	Analog input AIN0 is selected
			001	Analog input AIN1 is selected
			010	Analog input AIN2 is selected. (BGA package only.)
			011	Analog input AIN3 is selected. (BGA package only.)
			100–111	All analog switches are off.
11–0	reserved			Reserved. The reserved bit locations are always read as zero.

1.2.2 ADC Data Register (ADDR)

The ADC data register (Figure 1–4) is a read/write register that indicates if a conversion is in process, the actual digital data converted from the analog signal, and which channel the data came from.

Figure 1–4. ADC Data Register (ADDR)



Note: R/W-x = Read/Write-Reset value

Table 1–3. ADC Data Register (ADDR) Field Values

Bit	*field*	*symval*	Value	Description
15	ADCBusy			ADC busy bit.
			0	ADC data is available.
			1	ADC is busy performing a conversion. After ADCStart bit is high, the ADCBusy bit becomes high.
14–12	ChSelect			Channel select bits indicate which one of four analog inputs the conversion data in the ADCData bits is from.
			000	Analog input AIN0 is selected
			001	Analog input AIN1 is selected
			010	Analog input AIN2 is selected. (BGA package only.)
			011	Analog input AIN3 is selected. (BGA package only.)
			100–111	reserved
11–10	reserved			Reserved. The reserved bit locations are always read as zero.
9–0	ADCData			ADC data bits. The 10-bit digital data converted from the analog signal.

1.2.3 ADC Clock Divider Register (ADCDR)

The ADC clock divider register (Figure 1–5) is a read/write register that indicates the divider values for the conversion clock and the sample and hold time.

Figure 1–5. ADC Clock Divider Register (ADCDR)

15 8	7 4	3 0
SampTimeDiv	reserved	ConvRateDiv
R/W-0	R-0	R/W-1111

Note: R/W-x = Read/Write-Reset value

Table 1–4. ADC Clock Divider Register (ADCDR) Field Values

Bit	*field*	*symval*	Value	Description
15–8	SampTimeDiv		0–255	Sample and hold time divider bits. This 8-bit value in conjunction with the ConvRateDiv bits and the ADC clock period determines the sample and hold period.
				ADC Sample and Hold Period = (*ADC Clock Period*) × (2 × (*ConvRateDiv* + 1 + *SampTimeDiv*))
7–4	reserved			Reserved. The reserved bit locations are always read as zero.
3–0	ConvRateDiv			Conversion clock rate divider bits indicate the divider rate of the ADC clock. Also, this 4-bit value in conjunction with the SampTimeDiv bits and the ADC clock period determines the sample and hold period.
				ADC Conversion Clock = (*ADC Clock*) / (2 × (*ConvRateDiv* + 1))
			0000	Conversion clock = ADC clock divided by 2.
			0001	Conversion clock = ADC clock divided by 4.
			0010	Conversion clock = ADC clock divided by 6.
			0011	Conversion clock = ADC clock divided by 8.
			0100	Conversion clock = ADC clock divided by 10.
			0101	Conversion clock = ADC clock divided by 12.
			0110	Conversion clock = ADC clock divided by 14.
			0111	Conversion clock = ADC clock divided by 16.
			1000	Conversion clock = ADC clock divided by 18.
			1001	Conversion clock = ADC clock divided by 20.

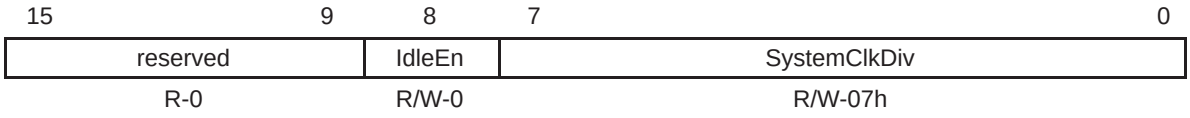
Table 1–4. ADC Clock Divider Register (ADCDR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
			1010	Conversion clock = ADC clock divided by 22.
			1011	Conversion clock = ADC clock divided by 24.
			1100	Conversion clock = ADC clock divided by 26.
			1101	Conversion clock = ADC clock divided by 28.
			1110	Conversion clock = ADC clock divided by 30.
			1111	Conversion clock = ADC clock divided by 32.

1.2.4 ADC Clock Control Register (ADCCR)

The ADC clock control register (Figure 1–6) is a read/write register that indicates the divider rate of the system clock and enables the ADC clock.

Figure 1–6. ADC Clock Control Register (ADCCR)



Note: R/W-x = Read/Write-Reset value

Table 1–5. ADC Clock Control Register (ADCCR) Field Values

Bit	*field*	*symval*	Value	Description
15–9	reserved			Reserved. The reserved bit locations are always read as zero.
8	IdleEn			ADC clock enable bit.
			0	ADC clock is active.
			1	ADC clock is disabled at execution of an IDLE instruction.
7–0	SystemClkDiv		0–255	System clock divider bits indicate the divider rate of the system clock.
				ADC Clock = (*System Clock*) / (*SystemClkDiv* + 1)

1.3 Conversion Example

The clock dividers are used to derive the total conversion time from the system clock within the DSP. The following example shows through the programming of these clock dividers how to obtain the maximum sampling frequency in a system where the DSP operates at 144 MHz. After the ADC sampling rate has been programmed, the DSP can begin using the ADC for sampling analog inputs.

1) Divide down the system clock to generate the main clock to the ADC (ADC clock). It is desirable to program the ADC clock to as low a frequency as possible to minimize the power consumption of the ADC.

   In this example, program the ADC clock to 4 MHz. To obtain the 4 MHz value, the system clock of 144 MHz must be divided by 36. An 8-bit divider, SystemClkDiv bits in ADCCR, is provided.

   ADC Clock = (*System Clock*) / (*SystemClkDiv* + 1)
   ADC Clock = (144 MHz) / (*SystemClkDiv* + 1)
   ADC Clock = (144 MHz) / (35 + 1) = 4 MHz

ADC Clock Control Register (ADCCR)

15 9	8	7 0
reserved	IdleEn	SystemClkDiv = 0010 0011

   The 4-MHz ADC clock is now divided to generate the two components of the total conversion time.

2) Divide down the 4-MHz ADC clock to generate the conversion clock.

   In this example, program the conversion clock rate divider to generate the maximum possible conversion clock frequency of 2 MHz. To obtain the 2-MHz conversion clock frequency, the ADC clock must be divided by the lowest value. A 5-bit divider, ConvRateDiv bits in ADCDR, is provided.

   ADC Conversion Clock = (*ADC Clock*) / (2 × (*ConvRateDiv* + 1))
   ADC Conversion Clock = (4 MHz) / (2 × (*ConvRateDiv* + 1))
   ADC Conversion Clock = (4 MHz) / (2 × (0 + 1)) = 2 MHz

ADC Clock Divider Register (ADCDR)

15 8	7 4	3 0
SampTimeDiv	reserved	ConvRateDiv = 0000

   The actual conversion time is 13 cycles of this clock, so the conversion time is 6.5 µs.

   ADC Conversion Time = 13 × (1 / *ADC Conversion Clock*)
   ADC Conversion Time = 13 × (1 / (2 MHz)) = 6.5 µs

3) Program the clock divider for the sample and hold time. The sample and hold time must be greater than or equal to 40 µs.

   In this example, program the sample and hold time to 40 µs. An 8-bit divider, SampTimeDiv bits in ADCDR, is used in conjunction with the conversion rate divider to obtain the sample and hold time from the ADC clock.

   ADC Sample and Hold Period =
   (1 / (*ADC Clock*)) / (2 × (*ConvRateDiv* + 1 + *SampTimeDiv*))
   = (1 / (4 MHz)) / (2 × (0 + 1 + *SampTimeDiv*))
   = 250 ns × (2 × (0 + 1 + 79)) = 40 µs

ADC Clock Divider Register (ADCDR)

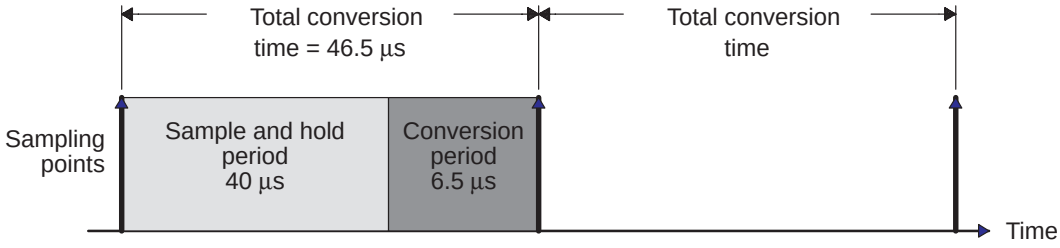
15 — 8	7 — 4	3 — 0
SampTimeDiv= 0100 1111	reserved	ConvRateDiv = 0000

4) The final results of this example are summarized (Figure 1–7). The total conversion time is composed of a 40-µs sample and hold time plus a 6.5-µs conversion time. A new conversion can begin every 46.5 µs, giving a maximum sampling rate of 21.5 KHz.

   Total Conversion Time = (*Sample and Hold Period*) + (*Conversion Period*)
   Total Conversion Time = 40 µs + 6.5 µs = 46.5 µs

   Sampling Frequency = 1 / (*Total Conversion Time*)
   Sampling Frequency = 1 / 46.5 µs = 21.5 KHz

Figure 1–7. Total Conversion Time for Example



Chapter 2

Clock Generator

This chapter describes the clock generator of the TMS320C55x™ DSP. The clock generator accepts an input clock at the CLKIN pin and enables you to modify that signal internally to produce an output clock with the desired frequency. The clock generator passes this output clock (the CPU clock) to the CPU, to peripherals, and to other modules inside the C55x™ DSP. The CPU clock is also passed through a programmable clock divider to the CLKOUT pin (see section 2.6).

Topic **Page**

2.1 Introduction to the DSP Clock Generator

The DSP clock generator supplies the DSP with a clock signal that is based on an input clock signal connected at the CLKIN pin. Included in the clock generator is a digital phase-lock loop (PLL), which can be enabled or disabled. You can configure the clock generator to create a CPU clock signal that has the desired frequency.

The clock generator has a clock mode register, CLKMD (see page 2-12), for controlling and monitoring the activity of the clock generator. For example, you can write to the PLL ENABLE bit in CLKMD to toggle between the two main modes of operation:

- In the bypass mode (see page 2-5), the PLL is bypassed, and the frequency of the output clock signal is equal to the frequency of the input clock signal divided by 1, 2, or 4. Because the PLL is disabled, this mode can be used to save power.
- In the lock mode (see page 2-6), the input frequency can be both multiplied and divided to produce the desired output frequency, and the output clock signal is phase-locked to the input clock signal. The lock mode is entered if the PLL ENABLE bit of the clock mode register is set and the phase-locking sequence is complete. (During the phase-locking sequence, the clock generator is kept in the bypass mode.)

The clock generator also has an idle mode (see page 2-8) for power conservation. You place the clock generator into its idle mode by turning off the CLKGEN idle domain. For information on turning on and off idle domains, see Chapter 8, *Idle Configurations*.

The output of the clock generator or a divided down version of that output can be seen on the CLKOUT pin. For details, see *The CLKOUT Pin and the Associated Clock Divider* on page 2-9.

2.2 Operational Flow of the DSP Clock Generator

Figure 2–1 and Table 2–1 describe the operational states (A–F) of the DSP clock generator. The clock mode register (CLKMD) is loaded by software or by a DSP reset. If the write to CLKMD enables the PLL, the PLL begins its phase-locking sequence (state A). If the write disables the PLL, the clock generator enters its bypass mode (state D).

Figure 2–1. Operational Flow of the DSP Clock Generator

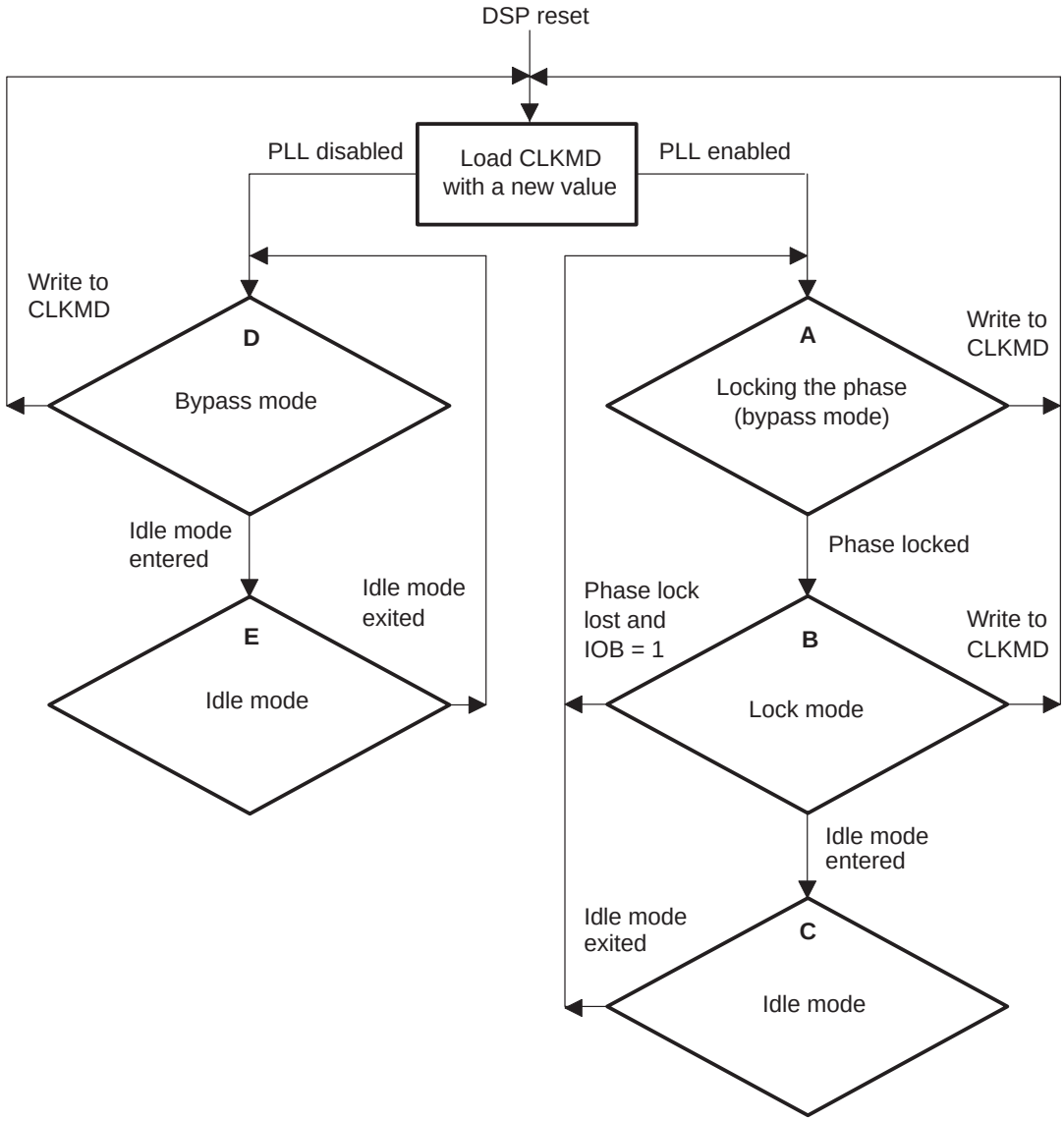


Table 2–1. Operational States Shown in Figure 2–1

State	Description
A	**Locking the phase.** The clock generator enters the bypass mode, and the PLL locks the phase of the output clock signal to that of the input clock signal. Once the phase is locked and the output signal is at the frequency defined by the PLL MULT bits and the PLL DIV bits of CLKMD (see page 2-12), the clock generator enters its lock mode (state B). You can reconfigure the clock generator by writing to CLKMD.
B	**Lock mode.** In the lock mode, the PLL is generating an output signal with the selected frequency. The output signal is phase-locked to the input signal. If the PLL loses the lock and the IOB bit of CLKMD is 1, the clock generator returns to the bypass mode and reacquires the lock (state A); if the IOB bit is 0, the clock generator does not reacquire the lock. An idle instruction can place the clock generator into its idle mode (state C). To change to the bypass mode or to reconfigure the clock generator in other ways, you can write to CLKMD. For more details about the lock mode, see section 2.4 page 2-6.
C	**Idle mode (entered from the lock mode).** An idle instruction has placed the clock generator into its idle mode. If the idle mode is properly exited, the clock generator starts again and reacquires the phase lock (state A). The method used to reacquire the lock depends on the IAI bit of CLKMD.
D	**Bypass mode.** The PLL is disabled, and the clock generator is in the bypass mode. The divider within the clock generator produces an output clock signal at the frequency defined by the BYPASS DIV bits of CLKMD. An idle instruction can place the clock generator into its idle mode (state E). To change to the lock mode or to reconfigure the clock generator in other ways, you can write to CLKMD. For more details about the bypass mode, see section 2.3 page 2-5.
E	**Idle mode (entered from the bypass mode).** An idle instruction has placed the clock generator into its idle mode. If the idle mode is properly exited, the clock generator starts again in the bypass mode.

2.3 Bypass Mode

When the DSP clock generator is in the bypass mode and the phase-lock loop (PLL) is disabled, the frequency of the output clock signal is equal to the frequency of the input clock signal divided by 1, 2, or 4.

2.3.1 Entering and Exiting the Bypass Mode

To enter the bypass mode, write a 0 to the PLL ENABLE bit in the clock mode register (CLKMD). The PLL will be disabled.

To exit the bypass mode, write a 1 to the PLL ENABLE bit. The PLL will start up and enter its phase-locking sequence. After the PLL is generating the configured output frequency and the phase of the output clock signal is locked to the phase of the input clock signal, the clock generator enters the lock mode. Until then, the clock generator stays in the bypass mode.

If the clock generator is in the lock mode and the PLL must reacquire its phase lock (IOB = 1), the clock generator enters the bypass mode until the phase is locked again.

2.3.2 CLKMD Bits Used in the Bypass Mode

Table 2–2 describes the bits of the clock mode register (CLKMD) that are used in the bypass mode. The reserved bits in CLKMD (Rsvd and TEST) should not be used in either the bypass mode or the lock mode. For a detailed description of CLKMD, see section 2.8 on page 2-12.

Table 2–2. CLKMD Bits Used in the Bypass Mode

CLKMD Bit Field	Role In The Bypass Mode
PLL ENABLE	Allows you to switch to the lock mode
BYPASS DIV	Determines how the input clock frequency is divided (if at all) to produce the output clock frequency
LOCK	Is 0 in the bypass mode

2.3.3 Setting the Output Frequency for the Bypass Mode

The output frequency is determined by the input frequency and the value in the BYPASS DIV bits. Load BYPASS DIV as required to divide the input frequency by 1, 2, or 4.

2.4 Lock Mode

In the lock mode, the input frequency can be both multiplied and divided to produce the desired output frequency, and the output clock signal is phase-locked to the input clock signal.

2.4.1 Entering and Exiting the Lock Mode

To enter the lock mode, write a 1 to the PLL ENABLE bit in the clock mode register (CLKMD). The PLL will start up and will enter its phase-locking sequence. After the PLL is generating the configured output frequency and the phase of the output clock signal is locked to the phase of the input clock signal, the clock generator enters the lock mode. Until then, the clock generator stays in the bypass mode.

If the clock generator is in the lock mode and the PLL must reacquire its phase lock (IOB = 1 in CLKMD), the clock generator will enter the bypass mode until the phase is locked again.

To exit the lock mode (enter the bypass mode), write a 0 to the PLL ENABLE bit. The PLL will be disabled.

2.4.2 CLKMD Bits Used in the Lock Mode

Table 2–3 describes the bits of the clock mode register (CLKMD) that are used in the lock mode. The reserved bits (Rsvd and TEST) in CLKMD should not be used in either the lock mode or the bypass mode. For a detailed description of CLKMD, see section 2.8 on page 2-12.

Table 2–3. CLKMD Bits Used in the Lock Mode

CLKMD Bit Field(s)	Role In The Lock Mode
PLL ENABLE	Allows you to switch to the bypass mode (disable the PLL)
PLL MULT and PLL DIV	Determine how the input clock frequency is modified (if at all) to produce the output clock frequency
IAI	Determines whether the PLL returns to the beginning of the phase-locking sequence when the clock generator exits its idle mode
BREAKLN	Indicates when the phase lock has been broken
IOB	Determines whether the PLL will reacquire a lost phase lock
LOCK	Is 1 in the lock mode

2.4.3 Setting the Output Frequency for the Lock Mode

The input frequency is multiplied by the PLL MULT value of CLKMD and is divided according to the PLL DIV value of CLKMD. PLL MULT can be a value from 2 to 31. PLL DIV can be a value from 0 (divide by 1) to 3 (divide by 4). The output frequency can be calculated with the following equation:

$$\text{Output frequency} = \frac{\text{PLL MULT}}{(\text{PLL DIV} + 1)} \times \text{Input frequency}$$

Table 2–4 shows some examples of using PLL MULT and PLL DIV to select an output frequency.

Table 2–4. Examples of Selecting a Lock Mode Frequency

PLL MULT	PLL DIV	Output Frequency
31	0 (divide by1)	31 × Input frequency (maximum frequency)
10	1 (divide by 2)	5 × Input frequency
2	2 (divide by 3)	2/3 × Input frequency
2	3 (divide by 4)	1/2 × Input frequency (minimum frequency)

2.4.4 Calculating the Lock Time

The time needed for phase locking (the lock time) depends on the PLL MULT and PLL DIV values and on the output clock frequency. The lock time in number of **input** clock cycles is given by Equation 2–1.

Equation 2–1. Lock Time for the PLL

$$\text{Lock time} = 4 \times (\text{PLL DIV} + 1) \times \left[10D + 30(\text{PLL MULT} + 1) + 2\right]$$

where $D = 1 + \log_2\left(\frac{200}{\text{PLL MULT} \times \text{Output frequency}}\right)$ rounded to the nearest integer.

2.5 Idle (Low-Power) Mode

To save power, you can put the DSP clock generator into its idle mode by loading an idle configuration that turns off the CLKGEN idle domain. When the clock generator is idle, the output clock is stopped and held high. For more details, see Chapter 8, *Idle Configurations*.

When the clock generator exits its idle mode, the reaction of the clock generator depends on several factors. If the clock generator was in its bypass mode before the idle instruction was executed, the PLL returns to the bypass mode. If the clock generator was in its lock mode before the idle instruction was executed, the clock generator switches to its bypass mode, reacquires the phase lock, and then returns to the lock mode. The method used for reacquiring the phase lock depends on the IAI bit of CLKMD:

IAI	Method Used For Reacquiring The Phase Lock
0	The PLL reacquires the phase lock using the same process that was underway before the idle mode was entered.
1	The PLL restarts the phase-locking sequence.

2.6 The CLKOUT Pin and the Associated Clock Divider

The DSP clock generator generates the CPU clock that is supplied to the CPU, to peripherals, and to other modules inside the DSP. As shown in Figure 2–2, the CPU clock is also passed to a clock divider that supplies a signal (CLKOUT) to the CLKOUT pin. The frequency of CLKOUT depends on the CLKDIV bits of the system register, SYSR (see Table 2–5). SYSR is an I/O-mapped register and is described in section 12.2 on page 12-3.

Figure 2–2. Dividing the CPU Clock for the CLKOUT Pin

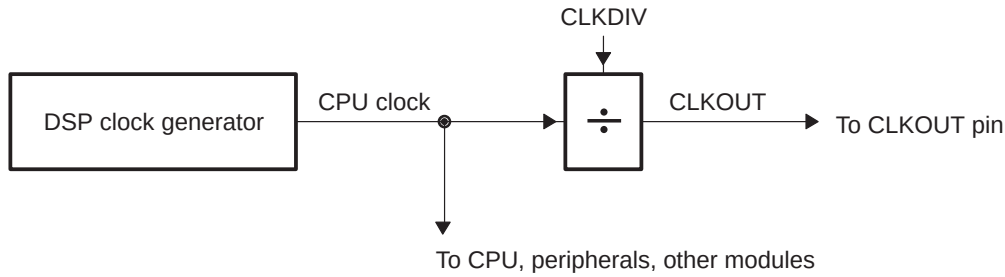


Table 2–5. Effect of CLKDIV Bits on CLKOUT Frequency

CLKDIV	Frequency of CLKOUT
000b	1/1 × CPU clock frequency
001b	1/2 × CPU clock frequency
010b	1/3 × CPU clock frequency
011b	1/4 × CPU clock frequency
100b	1/5 × CPU clock frequency
101b	1/6 × CPU clock frequency
110b	1/7 × CPU clock frequency
111b	1/8 × CPU clock frequency

2.7 DSP Reset Conditions of the DSP Clock Generator

The following sections describe the operation of the DSP clock generator when the DSP is held in its reset state and when the DSP is removed from its reset state.

2.7.1 Clock Generator During Reset

The DSP can make use of the output clock signal during reset. While the DSP reset signal is held low:

- ❏ The clock generator is in the bypass mode.
- ❏ The output clock frequency is determined by the level of the signal on the CLKMD input pin:

CLKMD Signal	Output Frequency
Low	Input frequency
High	1/2 × Input frequency

2.7.2 Clock Generator After Reset

On the rising edge of the DSP reset signal (when reset is deasserted), the clock mode register is loaded with a value determined by the level on the CLKMD pin:

CLKMD Signal	Output Frequency
Low	2002h
High	2006h

Table 2–6 summarizes the effects of this load to the clock mode register.

Table 2–6. Reset Values of CLKMD Bits and The Effects

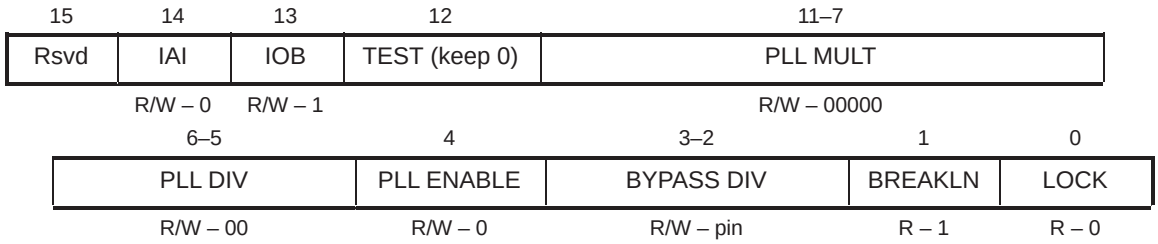
Reset Value	Effect
IAI = 0	Only applicable in the lock mode. Initialize-after-idle is not selected. After the idle mode is exited, the PLL reacquires the phase lock using the same process that was underway before the idle mode was entered (the phase-locking sequence is not restarted).
IOB= 1	Only applicable in the lock mode. Initialize-on-break is selected. Any time the PLL loses its phase lock, the clock generator switches to its bypass mode and starts a new phase-locking sequence.
PLL MULT= 00000b PLL DIV= 00b	Only applicable in the lock mode. The output frequency is equal to the input frequency.
PLL ENABLE = 0	The PLL is disabled. The clock generator is in its bypass mode.
If CLKMD signal is low BYPASS DIV= 00b If CLKMD signal is high BYPASS DIV= 01b	If CLKMD signal is low Output frequency = Input frequency If CLKMD signal is high Output frequency = 1/2 × Input frequency
BREAKLN = 1	The break-lock indicator is reset.
LOCK = 0	The lock-mode indicator reflects the fact that the clock generator is in the bypass mode.

2.8 Clock Mode Register

You control the DSP clock generator with the clock mode register, CLKMD. Figure 2–3 and Table 2–7 describe the contents of CLKMD, which is accessible in I/O space. After the DSP reset signal becomes inactive, the CLKMD register is initialized with a predetermined value dependent only upon the state of the CLKMD input pin (the difference is in the BYPASS DIV bits):

CLKMD Signal Level at Reset	CLKMD Register Reset Value
Low	2002h
High	2006h

Figure 2–3. Clock Mode Register (CLKMD)



Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset. X = pin indicates that the reset value depends on the signal level on the CLKMD pin.

Table 2–7. Bit Field Descriptions for the Clock Mode Register (CLKMD)

Bit(s)	Name	Description	Reset Value
15	Rsvd	This bit is reserved; it is not available for your use. This bit is always 0.	–
14	IAI	Initialize after idle bit. IAI determines how the PLL reacquires the phase lock after the clock generator exits its idle mode (when the CLKGEN idle domain is reactivated): 0 The PLL locks using the same process that was underway before the idle mode was entered. 1 The PLL restarts the phase-locking sequence.	0

Table 2–7. Bit Field Descriptions for the Clock Mode Register (CLKMD) (Continued)

Bit(s)	Name	Description	Reset Value
13	IOB	Initialize on break bit. IOB determines whether the clock generator initializes the PLL phase-locking sequence whenever the phase lock is broken. If the PLL indicates a break in the phase lock: 0 The clock generator does not interrupt the PLL. The clock generator stays in the lock mode, and the PLL continues to output the current clock signal. 1 The clock generator switches to its bypass mode and restarts the PLL phase-locking sequence.	1
12	TEST	This reserved test bit is cleared during a DSP reset and your program must be keep it 0 for proper operation of the clock generator. Make sure that whenever your program modifies CLKMD, it writes a 0 to bit 12.	0
11–7	PLL MULT	PLL multiply value. When the PLL is enabled (PLL ENABLE = 1), the frequency of the input clock signal is multiplied according to the value in PLL MULT. PLL MULT can be a value from 2 to 31. The input clock is multiplied by the unsigned integer in PLL MULT and is divided according to the value in the PLL DIV bits. The maximum frequency for the PLL output clock signal is 31 times the frequency of the input clock signal. To obtain this maximum frequency, load PLL MULT with 31 (multiply by 31), and load PLL DIV with 0 (divide by 1). The minimum frequency for the output clock signal is 1/2 the frequency of the input clock signal. To obtain this minimum frequency, load PLL MULT with 2 (multiply by 2) and load PLL DIV with 3 (divide by 4).	00000b

Table 2–7. Bit Field Descriptions for the Clock Mode Register (CLKMD) (Continued)

Bit(s)	Name	Description	Reset Value
6–5	PLL DIV	PLL divide value. When the PLL is enabled (PLL ENABLE = 1), the two PLL DIV bits select one of four divide options listed in the following table. The PLL also uses the multiply value supplied by the PLL MULT bits. To program the minimum or maximum frequency, see the description for the PLL MULT bits. 00b No division/divide by 1 The input frequency is not divided. 01b Divide by 2 The input frequency is divided by 2. 10b Divide by 3 The input frequency is divided by 3. 11b Divide by 4 The input frequency is divided by 4.	00b
4	PLL ENABLE	PLL enable bit. Write to PLL ENABLE to enable or disable the PLL. When you set PLL ENABLE, you request the clock generator to enter the lock mode. The clock generator does not enter the lock mode until the PLL is creating a phase-locked signal with the frequency selected by the PLL MULT bits and the PLL DIV bits. 0 Disable the PLL (enter the bypass mode). 1 Enable the PLL and, when the correct output clock signal is generated, enter the lock mode.	0
3–2	BYPASS DIV	Bypass-mode divide value. In the bypass mode, BYPASS DIV determines the frequency of the output clock signal. During a DSP reset, if the level on the CLKMD pin is low, BYPASS DIV is reset to 00b (no division). If the level on CLKMD is high, BYPASS DIV is reset to 01b (divide by 2). 00b No division/divide by 1 The frequency of the output clock signal is the same as the frequency of the input clock signal. 01b Divide by 2 The frequency of the output clock signal is 1/2 the frequency of the input clock signal. 10b or 11b Divide by 4 The frequency of the output clock signal is 1/4 the frequency of the input clock signal.	00b if CLKMD signal is low 01b if CLKMD signal is high

Table 2–7. Bit Field Descriptions for the Clock Mode Register (CLKMD) (Continued)

Bit(s)	Name	Description	Reset Value
1	BREAKLN	Break-lock indicator. BREAKLN indicates whether the PLL has broken the phase lock. In addition, if you write to CLKMD, BREAKLN is forced to 1. 0 The PLL has broken the phase lock. 1 The phase lock is restored, or a write to CLKMD has occurred.	1
0	LOCK	Lock-mode indicator. LOCK indicates whether the clock generator is in its lock mode: 0 The clock generator is in the bypass mode. The output clock signal has the frequency determined by the BYPASS DIV bits, or the PLL is in the process of getting a phase lock. 1 The clock generator is in the lock mode. The PLL has a phase lock, and the output clock has the frequency determined by the PLL MULT bits and the PLL DIV bits.	0

Chapter 3

DMA (Direct Memory Access) Controller

This chapter describes the DMA controller of the TMS320C55x™ (C55x™) DSP. The DMA controller allows movement of data among internal memory, external memory, peripherals, and the enhanced host port interface (EHPI) to occur without intervention from the CPU and in the background of CPU operation. The EHPI is described in Chapter 4.

Topic **Page**

3.1 Introduction to the DMA Controller

Acting in the background of CPU operation, the DMA controller can:

- ❏ Transfer data among internal memory, external memory, and on-chip peripherals.
- ❏ Transfer data between the enhanced host port interface (EHPI) and memory.

3.1.1 Key Features of the DMA Controller

The DMA controller has the following important features:

- ❏ Operation that is independent of the CPU.
- ❏ Four standard ports, one for each data resource: internal dual-access RAM (DARAM), internal single-access RAM (SARAM), external memory, and peripherals.
- ❏ An auxiliary port to enable certain transfers between the enhanced host port interface (EHPI) and memory.
- ❏ Six channels, which allow the DMA controller to keep track of the context of six independent block transfers among the standard ports.
- ❏ Bits for assigning each channel a low priority or a high priority. For details, see *Service Chain* on page 3-8.
- ❏ Event synchronization. DMA transfers in each channel can be made dependent on the occurrence of selected events. For details, see *Synchronizing Channel Activity* on page 3-17.
- ❏ An interrupt for each channel. Each channel can send an interrupt to the CPU on completion of certain operational events. See *Monitoring Channel Activity* on page 3-19.
- ❏ Software-selectable options for updating addresses for the sources and destinations of data transfers.
- ❏ A dedicated idle domain (see 8-2). You can put the DMA controller into a low-power state by turning off this domain. Each multichannel buffered serial port (McBSP) on the C55x DSP has the ability to temporarily take the DMA domain out of this idle state when the McBSP needs the DMA controller.

To read about the registers used to program the DMA controller, see section 3.12 on page 3-27.

3.1.2 Block Diagram of the DMA Controller

Figure 3–1 is a conceptual diagram of connections between the DMA controller and other parts of the DSP. The DMA controller ports in the diagram are:

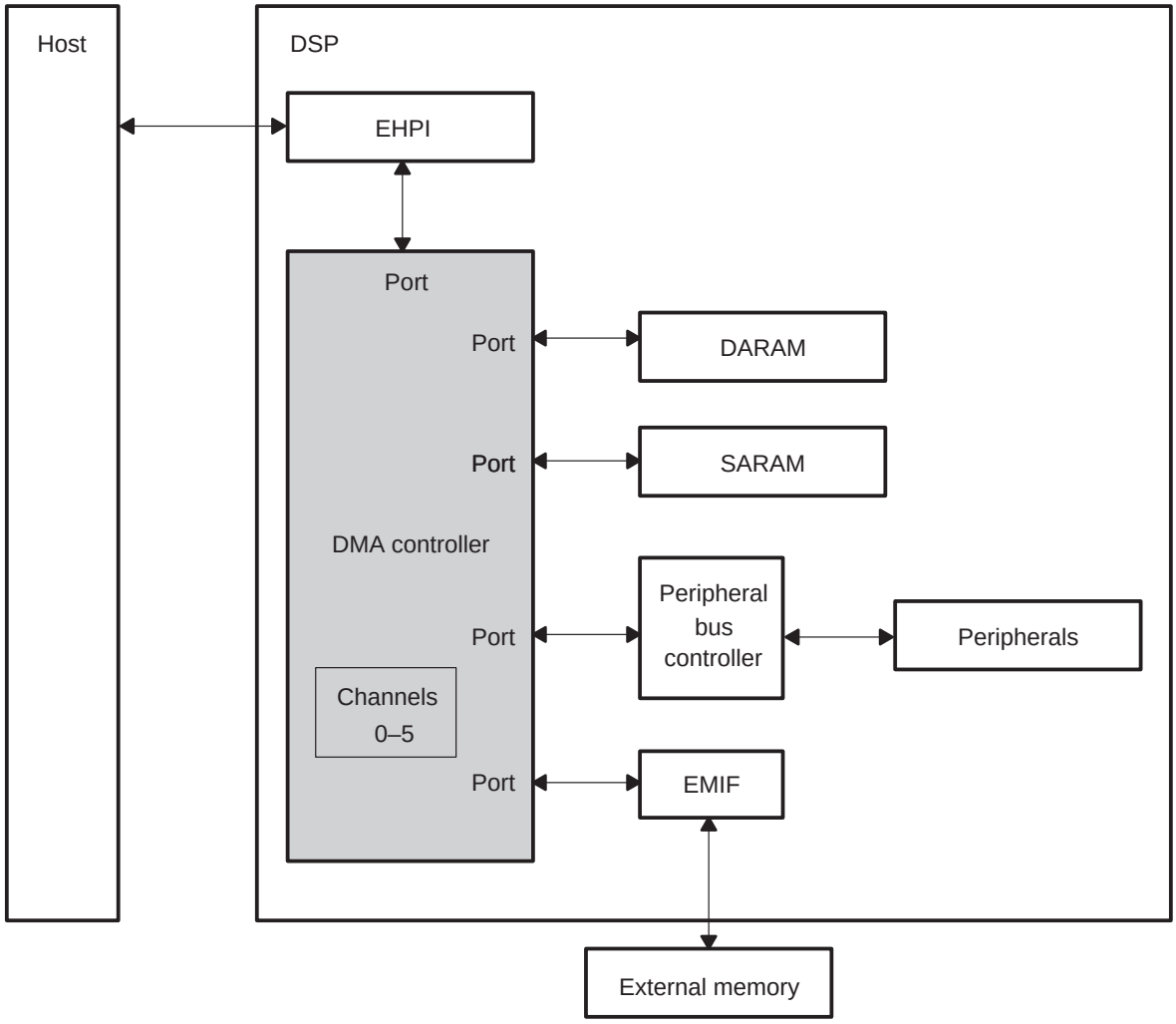
- Four standard ports. The DMA controller has a standard port for each of the following resources: internal dual-access RAM (DARAM), internal single-access RAM (SARAM), external memory, and peripherals. Data transfers among the standard ports occur in the six DMA channels. (The DMA channels are described on page 3-5.)

- Auxiliary port. A fifth port supports data transfers between memory and the enhanced host port interface (EHPI). The EHPI cannot access the peripherals port. If you want to transfer data from the EHPI to the peripherals port, you must use data memory as a temporary buffer. Transfers between the EHPI and the memory ports do not use a DMA channel.

  The EHPI shares the auxiliary port with the USB module (see Chapter 14). The USB module is given the higher priority at the port.

It is possible for multiple channels (or for one or more channels and the EHPI) to request access to the same standard port at the same time. To arbitrate simultaneous requests, the DMA controller has one programmable service chain that is used by each of the standard ports. For details on the service chain, see page 3-8.

Figure 3–1. Conceptual Block Diagram of the DMA Controller Connections



3.2 Channels and Port Accesses

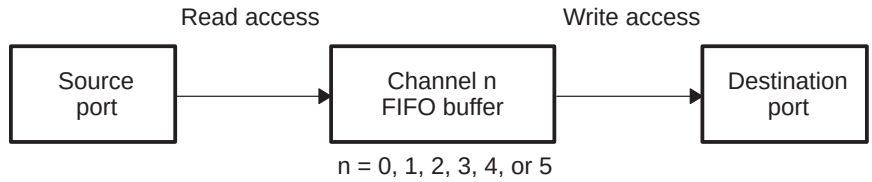
The DMA controller has six paths, called channels, to transfer data among the four standard ports (for DARAM, SARAM, external memory, and peripherals). Each channel reads data from one port (from the source) and writes data to that same port or another port (to the destination).

Each channel has a first in, first out (FIFO) buffer that allows the data transfer to occur in two stages (see Figure 3–2):

Port read access Transfer of data from the source port to the channel FIFO buffer.

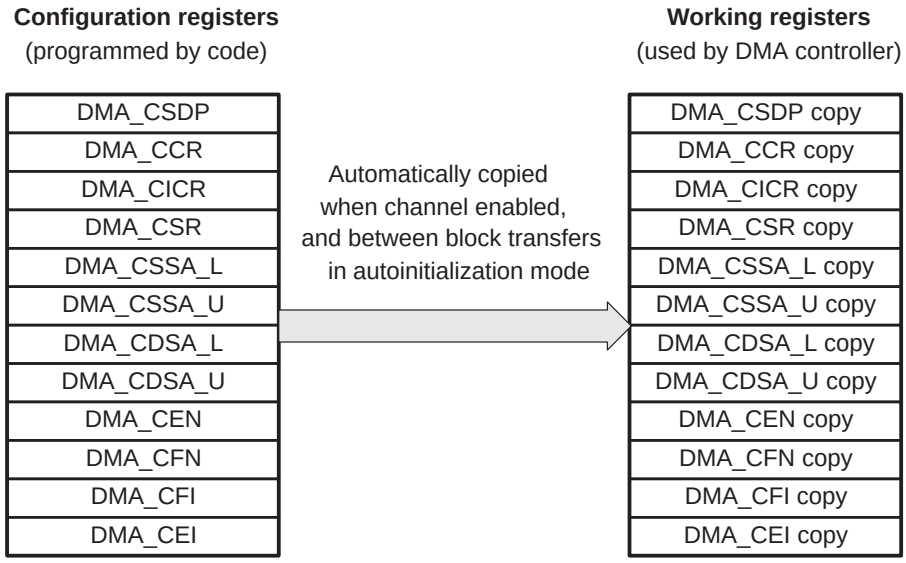
Port write access Transfer of data from the channel FIFO buffer to the destination port.

Figure 3–2. The Two Parts of a DMA Transfer



The set of conditions under which transfers occur in a channel is called the channel context. Each of the six channels contains a register structure for programming and updating the channel context (see Figure 3–3). Your code modifies the configuration registers. When it is time for data transferring, the contents of the configuration registers are copied to the working registers, and the DMA controller uses the working register values to control channel activity. The copy from the configuration registers to the working registers occurs whenever your code enables the channel (EN = 1 in DMA_CCR). In addition, if the autoinitialization mode is on (AUTOINIT = 1 in DMA_CCR), the copy occurs between block transfers. For more information about the DMA controller registers, see page 3-27.

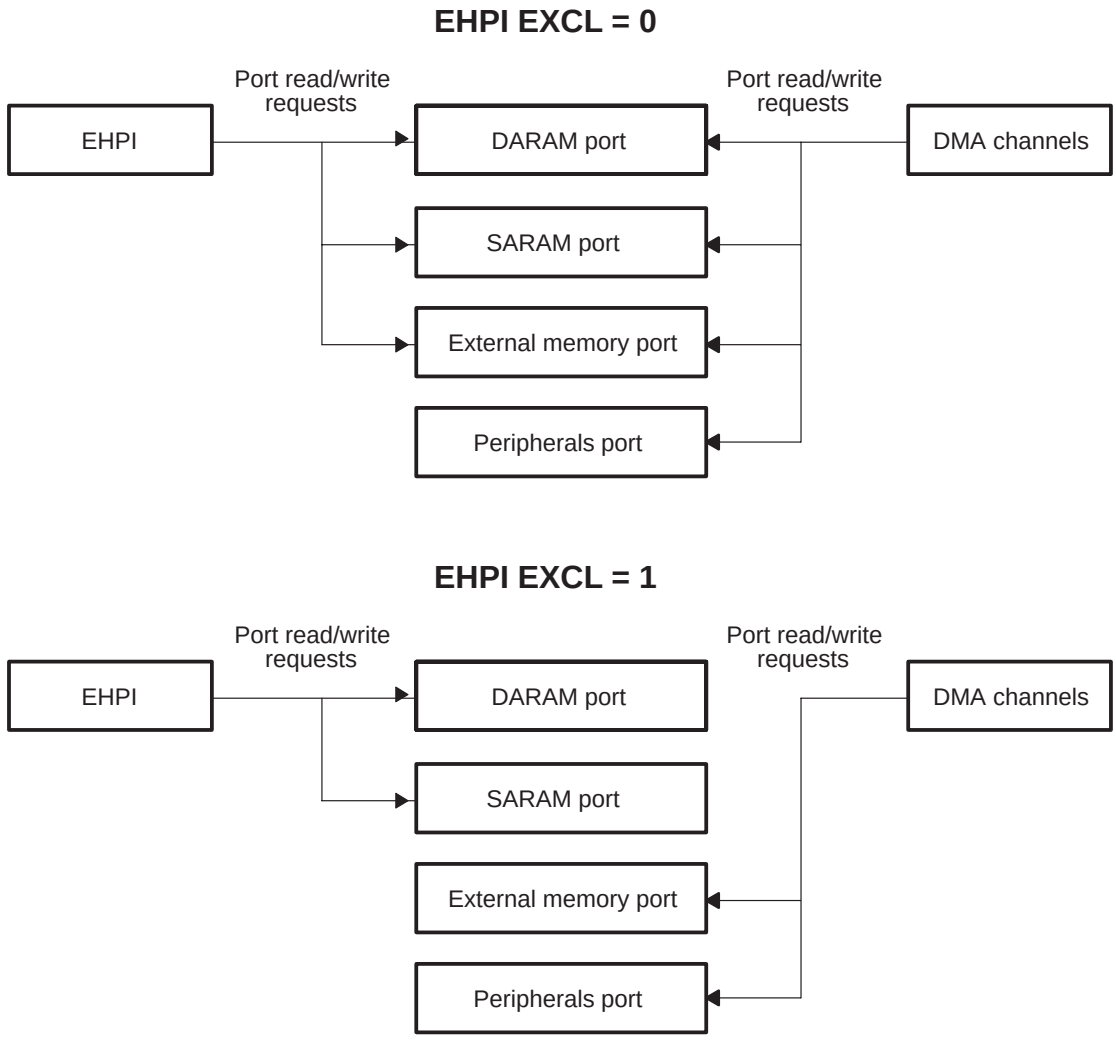
Figure 3–3. Registers for Controlling a Channel's Context



3.3 EHPI Access Configurations

As shown in Figure 3–4, the EHPI EXCL bit in DMA_GCR determines the relationship between the EHPI and the DMA channels. When EHPI EXCL = 0, the EHPI shares memory with the channels. When EHPI EXCL = 1, the EHPI cannot access external memory, but it can access internal RAM without interruptions from the channels. When EHPI EXCL = 1, the DARAM port and the SARAM port operate as if all the channels were disconnected from the service chain. (The service chain is described in section 3.4.)

Figure 3–4. EHPI Access Configurations

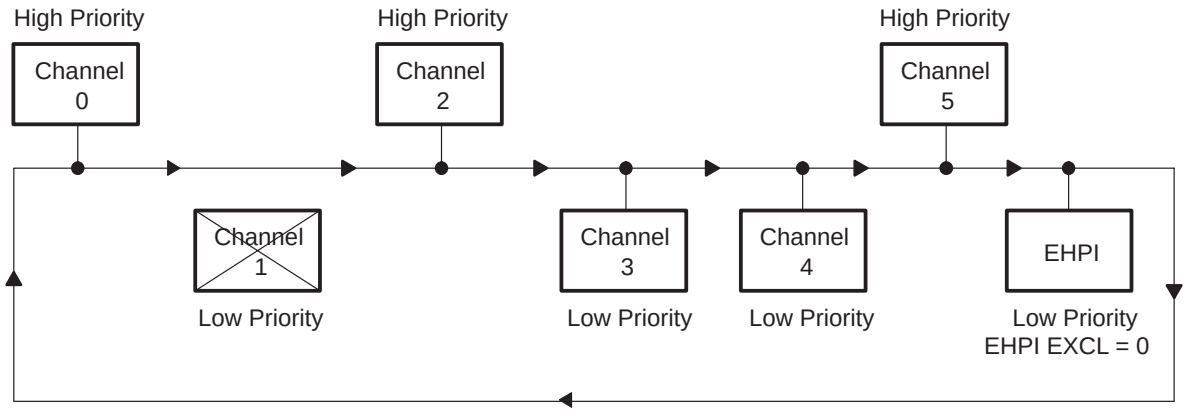


3.4 Service Chain

Each of the standard ports can arbitrate simultaneous access requests sent by the six DMA channels and the enhanced host port interface (EHPI). Each of the standard ports has an independently functioning service chain—a software- and hardware-controlled scheme for servicing access requests. Although the four service chains function independently, they share a common configuration. For example, if you disable channel 2, it is disabled in all four ports, and if you make channel 4 high-priority, it is high-priority in all four of the ports. One possible configuration for the service chains is shown in Figure 3–5. Important characteristics of the service chain are listed after the figure.

Section 3.4.1 contains an example that shows a service chain configuration applied to three ports.

Figure 3–5. One Possible Configuration for the Service Chains



- The channels and the EHPI have a programmable priority level. Each channel has a PRIO bit in DMA_CCR for selecting a high priority or a low priority. You assign the EHPI a high priority or low priority with the EHPI PRIO bit in DMA_GCR. The DMA controller only services the low-priority items when all the high-priority items are done or stalled. After a DSP reset, all channels and the EHPI are low priority.

  In the figure, channels 0, 2, and 5 are high-priority (in each of these channels, PRIO = 1). DMA channels 1, 3, and 4 and the EHPI are low priority (in each of these channels, PRIO = 0, and for the EHPI, EHPI PRIO = 0).

- The channels and the EHPI have fixed positions in the service chain. Regardless of the programmed priorities, the port checks the channels and the EHPI in a repeating circular sequence: 0, 1, 2, 3, 4, 5, EHPI, 0, 1, 2, 3, 4, 5, EHPI, and so on. At each position in the service chain, the port checks whether the channel/EHPI is ready and able to be serviced. If so, it is serviced; otherwise, the port skips to the next position. After a DSP reset, the port restarts its circular sequence, beginning with channel 0.

- The channels can be individually connected or disconnected from the service chain through software. If a channel is enabled (EN = 1 in DMA_CCR), it is connected to the service chain; if it is disabled (EN = 0), it is disconnected. After a DSP reset, all channels are disconnected.

  In the figure, only channel 1 is disconnected. As a port checks the channels and the EHPI in its repeating circular sequence, it will keep skipping channel 1 until the channel is reconnected.

- The EHPI cannot access the peripherals port. The peripherals port operates as if the EHPI is disconnected from the service chain.

- By writing a 1 to the EHPI EXCL bit in DMA_GCR, you can give the EHPI exclusive access to the DARAM and SARAM ports. Then the DARAM and SARAM ports operate as if only the EHPI is connected to the service chain (as if none of the channels are connected, regardless of whether the channels are enabled). For more details, see *EHPI Access Configurations* on page 3-7.

  In the figure, EHPI EXCL = 0. The EHPI shares the RAM ports with the channels.

- If a channel is tied to a synchronization event, the channel does not generate a DMA request (and, therefore, cannot be serviced) until the synchronization event occurs. To avoid long response times to synchronization events, always give synchronized channels a high priority.

3.4.1 Service Chain Example

Figure 3–6 shows a DMA service chain applied to the DARAM port, the external memory port, and the peripherals port. The service chain has the following programmed characteristics. A list of activity in the ports is provided after the figure.

- Channels 0, 2, and 5 are high-priority (PRIO = 1 in DMA_CCR). Channels 1, 3, and 4 are low-priority (PRIO = 0).
- Channels 1, 2, and 4 are enabled (EN = 1 in DMA_CCR). Channels 0, 3, and 5 are disabled (EN = 0).
- The EHPI is sharing the internal memory with the channels (EHPI EXCL = 0 in DMA_GCR) and is treated like a low-priority channel (EHPI PRIO = 0 in DMA_GCR). Notice that the EHPI is shown as disconnected in the peripherals port. This is because the EHPI cannot access the peripherals port.

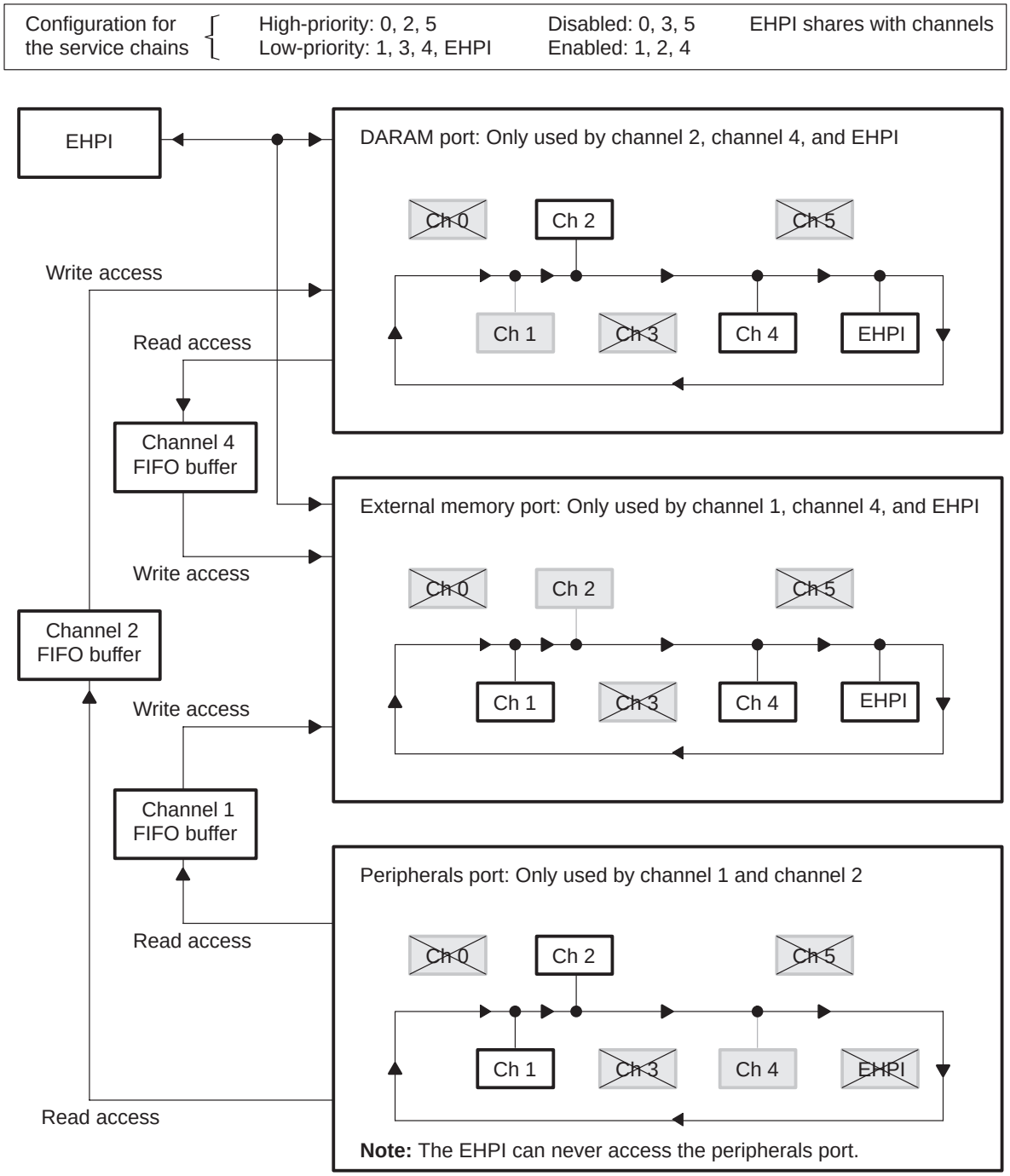
Table 3–1 summarizes the activity at the ports in the figure.

Table 3–1. Activity Shown in Figure 3–6

Port	**This Port Arbitrates** …
DARAM	Write access requests from channel 2 Read access requests from channel 4 Read or write access requests from the EHPI
External memory	Write access requests from channel 1 Write access requests from channel 4 Read or write access request from the EHPI
Peripherals	Read access requests from channel 1 Read access requests from channel 2

Finally, notice that for each port in the figure, there is a channel that is connected to the service chain but does not use the port. For example, the peripherals port is not used by channel 4. If channel 4 were redefined to include the peripherals port as source or destination, the port would handle channel 4 according to its position and priority in the service chain.

Figure 3–6. Service Chain Applied to Three DMA Ports



3.5 Units of Data: Byte, Element, Frame, and Block

This documentation on the DMA controller refers to data in four levels of granularity:

Byte An 8-bit value. A byte is the smallest unit of data transferred in a DMA channel.

Element One or more bytes to be transferred as a unit. Depending on the programmed data type, an element is an 8-bit, 16-bit, or a 32-bit value. An element transfer cannot be interrupted; all of its bytes are transferred to a port before another channel or the EHPI can take control of the port.

Frame One or more elements to be transferred as a unit. A frame transfer can be interrupted between element transfers.

Block One or more frames to be transferred as a unit. Each channel can transfer one block of data (once or multiple times). A block transfer can be interrupted between frame transfers and element transfers.

For each of the six DMA channels, you can define the number of frames in a block (with DMA_CFN), the number of elements in a frame (with DMA_CEN), and the number of bytes in an element (with the DATA TYPE bits in DMA_CSDP). For descriptions of DMA_CFN, DMA_CEN, DMA_CSDP, and other registers of the DMA controller, see section 3.12 on page 3-27.

3.6 Start Addresses in a Channel

During a data transfer in a DMA channel, the first address at which data is read is called the source start address. The first address to which the data is written is called the destination start address. These are byte addresses. From the standpoint of the DMA controller, every 8 bits in memory or I/O space has its own address. Each channel contains the following registers for specifying the start addresses:

Table 3–2. Registers Used to Define the Start Addresses for a DMA Transfer

Register	Load With …
DMA_CSSA_L	Source start address (lower part)
DMA_CSSA_U	Source start address (upper part)
DMA_CDSA_L	Destination start address (lower part)
DMA_CDSA_U	Destination start address (upper part)

The following sections explain how to load the start address registers for memory accesses and I/O accesses. The DMA controller can access all of the internal and external memory and all of I/O space (which contains registers for the DSP peripherals).

3.6.1 Start Address in Memory

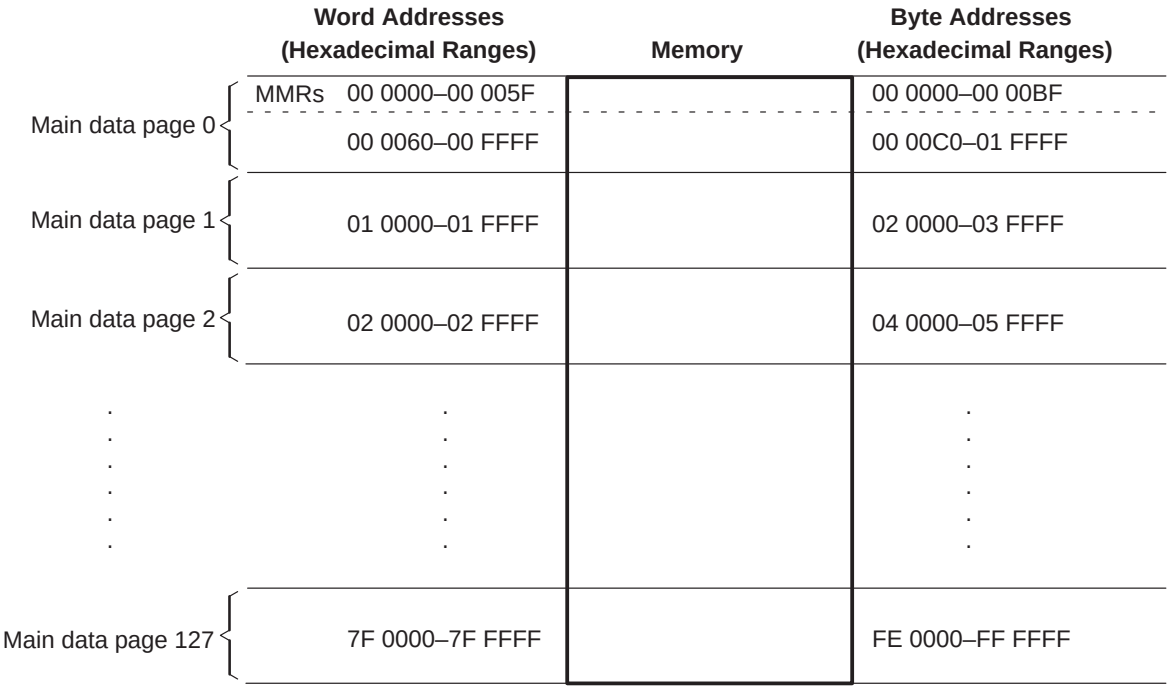
Figure 3–7 is a high-level memory map for TMS320C55x DSPs. The diagram shows both the word addresses (23-bit addresses) used by the CPU and byte addresses (24-bit addresses) used by the DMA controller. To load the source/destination start address registers:

1) Identify the correct start address. Check for any alignment constraint for the data type; see the description for the DATA TYPE bits in section 3.12.4 (page 3-40). If you have a word address, shift it left by 1 bit to form a byte address with 24 bits. For example, word address 02 4000h should be converted to byte address 04 8000h.
2) Load the 16 least significant bits (LSBs) of the byte address into DMA_CSSA_L (for source) or DMA_CDSA_L (for destination).
3) Load the 8 most significant bits (MSBs) of the byte address into the 8 LSBs of DMA_CSSA_U (for source) or DMA_CDSA_U (for destination).

Note:

Word addresses 00 0000h–00 005Fh (which correspond to byte addresses 00 0000h–00 00BFh) are reserved for the memory-mapped registers (MMRs) of the DSP CPU.

Figure 3–7. High-Level Memory Map for TMS320C55x DSPs



3.6.2 Start Address in I/O Space

Figure 3–8 is an I/O space map for TMS320C55x DSPs. The diagram shows both the word addresses (16-bit addresses) used by the CPU and byte addresses (17-bit addresses) used by the DMA controller. To load the source/destination start address registers:

1) Identify the correct start address. Check for any alignment constraint for the data type; see the description for the DATA TYPE bits in section 3.12.4 (page 3-40). If you have a word address, shift it left by 1 bit to form a byte address with 17 bits. For example, word address 8000h should be converted to byte address 1 0000h.

2) Load the 16 least significant bits (LSBs) of the byte address into DMA_CSSA_L (for source) or DMA_CDSA_L (for destination).

3) Load the most significant bit (MSB) of the byte address into the LSB of DMA_CSSA_U (for source) or DMA_CDSA_U (for destination).

Figure 3–8. High-Level I/O Map for TMS320C55x DSPs

Word Addresses (Hexadecimal Range)	I/O Space	Byte Addresses (Hexadecimal Range)
0000–FFFF		0 0000–1 FFFF

3.7 Updating Addresses in a Channel

During data transfers in a DMA channel, the DMA controller begins its read and write accesses at the start addresses you specify (as described in section 3.6). In many cases, after a data transfer has begun, these addresses must be updated so that data is read and written at consecutive or indexed locations. You can configure address updates at two levels:

- Block-level address updates. In the autoinitialization mode (AUTOINIT = 1 in DMA_CCR), block transfers can occur one after another until you turn off autoinitialization or disable the channel. If you want different start addresses for the block transfers, you can update the start addresses between the block transfers.

- Element-level address updates. You can have the DMA controller update the source address and/or the destination address after each element transfer. You can make sure the source address points to the start of the next element, and you can make sure the element will be precisely positioned at the destination. Choose an addressing mode for the source with the SRC AMODE bits in DMA_CCR. Choose an addressing mode for the destination with the DST AMODE bits in DMA_CCR.

3.8 Synchronizing Channel Activity

Activity in a channel can be synchronized to an event in a DSP peripheral or to an event signaled by the driving of an external interrupt pin. Using the SYNC bits of DMA_CCR, you can specify which synchronization event (if any) triggers activity.

Each channel also has an FS bit in DMA_CCR that allows you to choose among two synchronization modes:

- Element synchronization mode (FS = 0) requires one event per element transfer. When the selected synchronization event occurs, a read access request is sent to the source port and then a write access request is sent to the destination port. When all the bytes of the current element are transferred, the channel makes no more requests until the next occurrence of the synchronization event.

- Frame synchronization mode (FS = 1) requires one event to trigger an entire frame of elements. When the event occurs, the channel sends a read access request and a write access request for each element in the frame. When all the elements are transferred, the channel makes no more requests until the next occurrence of the event.

If you specify a synchronization event, the source port does not receive an access request until the event occurs. Once the request is received, it is handled according to the predefined position and the programmed priority of the channel in the DMA service chain (see page 3-8). To avoid long delays, it is best to give all synchronized channels a high priority.

If you choose not to synchronize the channel (SYNC = 00000b), the channel sends an access request to the source port as soon as the channel is enabled (EN = 1 in DMA_CCR).

3.8.1 Checking the Synchronization Status

Each channel has a synchronization flag (SYNC) in its status register, DMA_CSR. When the synchronization event occurs, the DMA controller sets the flag (SYNC = 1). The flag is cleared (SYNC = 0) when the DMA controller has completed the first read access (transfer from source port to channel buffer) after receiving synchronization.

3.8.2 Dropped Synchronization Events

If a synchronization event occurs before the DMA controller is done servicing the previous one (before the DMA controller clears the SYNC bit in DMA_CSR), a synchronization event has been dropped. The DMA controller responds to an event drop in the following manner:

- After the current element transfer, the DMA controller disables the channel (EN = 0 in DMA_CCR); activity in the channel stops after the current element transfer.
- If the corresponding interrupt enable bit is set (DROP IE = 1 in DMA_CICR), the DMA controller also sets the event drop status bit (DROP = 1 in DMA_CSR) and sends an interrupt request to the CPU. For more details, see *Monitoring Channel Activity* on page 3-19.

3.9 Monitoring Channel Activity

The DMA controller can send an interrupt to the CPU in response to the operational events listed in the following table. Each channel has interrupt enable (IE) bits in the interrupt control register (DMA_CICR) and some corresponding status bits in the status register (DMA_CSR). (DMA_CICR and DMA_CSR are described in section 3.12.3 on page 3-36.) If one of the operational events in the table occurs, the DMA controller checks the corresponding IE bit and acts accordingly:

- ❏ If the IE bit is 1 (the interrupt is enabled), the DMA controller sets the corresponding status bit and sends the associated interrupt request to the CPU. DMA_CSR is automatically cleared if your program reads the register.
- ❏ If the IE bit is 0, no interrupt is sent and the status bit is not affected.

DMA_CSR also has a SYNC bit that is used if you choose a synchronization event for the channel. SYNC indicates when the selected synchronization event has occurred (SYNC = 1) and when it has been serviced (SYNC = 0). For more details about synchronization events, see *Synchronizing Channel Activity* on page 3-17.

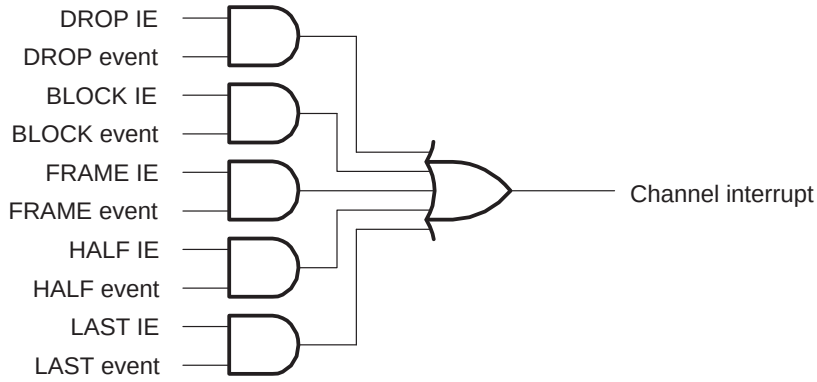
Table 3–3. DMA Controller Operational Events and Their Associated Bits and Interrupts

Operational Event	Interrupt Enable Bit	Status Bit	Associated Interrupt
Block transfer is complete	BLOCK IE	BLOCK	Channel interrupt
Last frame transfer has started	LAST IE	LAST	Channel interrupt
Frame transfer is complete	FRAME IE	FRAME	Channel interrupt
First half of current frame has been transferred	HALF IE	HALF	Channel interrupt
Synchronization event has been dropped	DROP IE	DROP	Channel interrupt
Time-out error has occurred	TIMEOUT IE	TIMEOUT	Bus-error interrupt

3.9.1 Channel Interrupt

Each of the six channels has its own interrupt. As shown Figure 3–9, the channel interrupt is the logical OR of all the enabled operational events except the time-out event (the time-out event generates a bus-error interrupt request). You can choose any combination of these five events by setting or clearing the appropriate interrupt enable (IE) bits in the interrupt control register (DMA_CICR) for the channel. You can determine which event(s) caused the interrupt by reading the bits in the status register (DMA_CSR) for the channel.

Figure 3–9. Triggering a Channel Interrupt Request



For example, suppose you are monitoring activity in channel 1. In DMA_CICR:

DROP IE = 1
HALF IE = 0
FRAME IE = 1
LAST IE = 0
BLOCK IE = 0

If a synchronization event is dropped (see 3.8.2 on page 3-18) or if the current frame transfer is done, the channel 1 interrupt request is sent to the CPU. No other event can generate the channel 1 interrupt. To determine whether one or both of the events triggered the interrupt, you can read the DROP and FRAME bits in DMA_CSR.

The channel 1 interrupt sets its corresponding flag bit in an interrupt flag register of the CPU. The CPU can respond to the interrupt or ignore the interrupt.

For more details about DMA_CICR and DMA_CSR, see section 3.12.3 on page 3-36.

3.9.2 Time-Out Conditions

A time-out condition exists when a memory access has been stalled for too many cycles. Each of the four standard ports of the DMA controller is supported by hardware to detect a time-out condition:

- DARAM port: A time-out counter in the DARAM port keeps track of how many cycles have passed since a request was made to access the DARAM. When the counter reaches 255, the DARAM port generates a time-out signal.
- SARAM port: A time-out counter in the SARAM port keeps track of how many cycles have passed since a request was made to access the SARAM. When the counter reaches 255, the SARAM port generates a time-out signal.
- External memory port: A time-out counter in the external memory interface (EMIF) keeps track of how many cycles the external ready pin (ARDY) has been sampled low. The external memory map is divided into four memory spaces, each of which has a programmable time-out value up to 255 cycles. When the counter reaches the time-out value, the EMIF sends a time-out signal to the DMA controller. For more details about the effects of a time-out condition in the EMIF, see *Configuring the EMIF for Asynchronous Accesses* on page 5-28.
- Peripherals port: A time-out counter in the peripheral bus controller counts how many cycles have passed since a request was made to access a peripheral. When the counter reaches 127, the peripheral bus controller sends a time-out signal to the DMA controller.

In response to a time-out signal, the DMA controller disables the channel (EN = 0 in DMA_CCR); activity in the channel stops after the current element transfer. If the corresponding interrupt enable bit is set (TIMEOUT IE = 1 in DMA_CICR), the DMA controller also sets the time-out status bit (TIMEOUT = 1 in DMA_CSR) and sends the time-out signal to the CPU as an interrupt request. The interrupt request sets the bus-error interrupt (BERRINT) flag bit in the CPU. The CPU can respond to the interrupt request or ignore the interrupt request.

3.10 Latency in DMA Transfers

Each element transfer in a channel is composed of a read access (a transfer from the source location to the channel buffer) and a write access (a transfer from the channel buffer to the destination location). The time to complete this activity depends on factors such as:

- ❏ The selected frequency of the CPU clock signal. This signal, as propagated to the DMA controller, determines the timing for all DMA transfers.
- ❏ Wait states or other extra cycles added by or resulting from an interface
- ❏ Competition from other channels. The DMA controller divides cycles among all enabled channels according to their position and priority level in the service chain (see section 3.4 on page 3-8). If fewer channels are enabled, more cycles are allotted per channel during a given interval of time.
- ❏ Competition from the enhanced host port interface (EHPI). If the EHPI is sharing internal RAM with the channels, the DMA controller allocates cycles to the EHPI like it does to channels. If you give the EHPI exclusive access to the internal RAM, no channels can access the internal RAM until you change the EHPI access configuration (see section 3.3 on page 3-7).
- ❏ The timing of synchronization events (if the channel is synchronized). The DMA controller cannot service a synchronized channel until the synchronization event has occurred. For more details on synchronization, see *Synchronizing Channel Activity* on page 3-17.

3.11 Power, Emulation, and Reset Considerations

The following sections describe how to put the DMA controller into a low-power state, how to program the response of the DMA controller to debugger breakpoints, and what values the DMA controller registers have after a DSP reset.

3.11.1 Reducing Power Consumed By the DMA Controller

The DSP is divided into idle domains that can be programmed to be idle or active. The state of all domains is called the idle configuration. Any idle configuration that disables the clock generator (CLKGEN) domain and/or the DMA domain stops the DMA clock and, therefore, stops activity in the DMA controller. For more details, see Chapter 8, *Idle Configurations*.

3.11.2 Emulation Modes of the DMA Controller

The FREE bit of DMA_GCR controls the behavior of the DMA controller when a breakpoint is encountered in the debugger software. If FREE = 0 (the reset value), a breakpoint suspends DMA transfers. If FREE = 1, DMA transfers are not interrupted by a breakpoint.

3.11.3 DMA Controller after DSP Reset

Table 3–4 shows the effects of a DSP reset on the DMA controller registers, and the resulting effects on the DSP. A number of registers are undefined and must be initialized/reinitialized.

Table 3–4. Effects of Resetting the DMA Controller Registers

Register	Reset Value	Key Effects on the DMA Controller
DMA_GCR	0000h	FREE = 0: A breakpoint in the software debugger suspends DMA transfers. EHPI EXCL = 0: The enhanced host port interface (EHPI) shares the internal RAM with the DMA channels. EHPI PRIO = 0: The EHPI has a low priority in the DMA service chain.
DMA_CSDP in every channel: DMA_CSDP0 DMA_CSDP1 DMA_CSDP2 DMA_CSDP3 DMA_CSDP4 DMA_CSDP5	0000h	In every channel: SRC BEN = DST BEN = 00b: Burst transfers are disabled at both the source port and the destination port. SRC PACK = DST PACK = 0: Data packing is disabled at both the source port and the destination port. SRC = DST = 0000b: The SARAM port is both the source port and the destination port. DATA TYPE = 00b: An 8-bit data type is selected.

Table 3–4. Effects of Resetting the DMA Controller Registers (Continued)

Register	Reset Value	Key Effects on the DMA Controller
DMA_CCR in every channel: DMA_CCR0 DMA_CCR1 DMA_CCR2 DMA_CCR3 DMA_CCR4 DMA_CCR5	0000h	In every channel: SRC AMODE = 00b: Each element transfer uses the same source address. DST AMODE = 00b: Each element transfer uses the same destination address. AUTOINIT = 0: The autoinitialization mode is off. The channel will stop after completing a block transfer. EN = 0: The channel is disabled. No data transfers can occur in the channel until EN = 1. PRIO = 0: The channel has a low priority in the DMA service chain. SYNC = 00000b: The channel does not wait for a synchronization event to trigger data transfers. Data transfers begin as soon as the channel is enabled (EN = 1).
DMA_CICR in every channel: DMA_CICR0 DMA_CICR1 DMA_CICR2 DMA_CICR3 DMA_CICR4 DMA_CICR5	0003h	In every channel: The channel interrupt can be triggered only by a dropped synchronization event. The bus-error interrupt can be triggered by a time-out condition.
DMA_CSR in every channel: DMA_CSR0 DMA_CSR1 DMA_CSR2 DMA_CSR3 DMA_CSR4 DMA_CSR5	0000h	In every channel: All of the channel status bits have been cleared.

Table 3–4. Effects of Resetting the DMA Controller Registers (Continued)

Register	Reset Value	Key Effects on the DMA Controller
DMA_CSSA_L and DMA_CSSA_U in every channel: DMA_CSSA_L0 DMA_CSSA_U0 DMA_CSSA_L1 DMA_CSSA_U1 DMA_CSSA_L2 DMA_CSSA_U2 DMA_CSSA_L3 DMA_CSSA_U3 DMA_CSSA_L4 DMA_CSSA_U4 DMA_CSSA_L5 DMA_CSSA_U5	Undefined	In every channel: You must load these registers to provide a known source start address for the channel.
DMA_CDSA_L and DMA_CDSA_U in every channel: DMA_CDSA_L0 DMA_CDSA_U0 DMA_CDSA_L1 DMA_CDSA_U1 DMA_CDSA_L2 DMA_CDSA_U2 DMA_CDSA_L3 DMA_CDSA_U3 DMA_CDSA_L4 DMA_CDSA_U4 DMA_CDSA_L5 DMA_CDSA_U5	Undefined	In every channel: You must load these registers to provide a known destination start address for the channel.
DMA_CEN in every channel: DMA_CEN0 DMA_CEN1 DMA_CEN2 DMA_CEN3 DMA_CEN4 DMA_CEN5	Undefined	In every channel: You must load this register to provide a known number of elements for the channel.

Table 3–4. Effects of Resetting the DMA Controller Registers (Continued)

Register	Reset Value	Key Effects on the DMA Controller
DMA_CFN in every channel: DMA_CFN0 DMA_CFN1 DMA_CFN2 DMA_CFN3 DMA_CFN4 DMA_CFN5	Undefined	In every channel: You must load this register to provide a known number of frames for the channel.
DMA_CFI in every channel: DMA_CFI0 DMA_CFI1 DMA_CFI2 DMA_CFI3 DMA_CFI4 DMA_CFI5	Undefined	In every channel: You must load this register to provide a known frame index for the channel.
DMA_CEI in every channel: DMA_CEI0 DMA_CEI1 DMA_CEI2 DMA_CEI3 DMA_CEI4 DMA_CEI5	Undefined	In every channel: You must load this register to provide a known element index for the channel.

3.12 DMA Controller Registers

Table 3–5 lists the types of registers in the direct memory access (DMA) controller. There is one global control register (DMA_GCR). In addition, for each of the DMA channels, there are 12 channel configuration registers. For the I/O address of each register, see the data sheet for your TMS320C55x DSP.

Table 3–5. Registers of the DMA Controller

Register Type	Description	For Details, See ...
DMA_GCR	Global control register (only one)	Page 3-28
DMA_CCR	Channel control register (one for each channel)	Page 3-29
DMA_CICR	Interrupt control register (one for each channel)	Page 3-36
DMA_CSR	Status register (one for each channel)	Page 3-36
DMA_CSDP	Source and destination parameters register (one for each channel)	Page 3-40
DMA_CSSA_L	Source start address (lower part) register (one for each channel)	Page 3-44
DMA_CSSA_U	Source start address (upper part) register (one for each channel)	Page 3-44
DMA_CDSA_L	Destination start address (lower part) register (one for each channel)	Page 3-45
DMA_CDSA_U	Destination start address (upper part) register (one for each channel)	Page 3-45
DMA_CEN	Element number register (one for each channel)	Page 3-45
DMA_CFN	Frame number register (one for each channel)	Page 3-45
DMA_CEI	Element index register (one for each channel)	Page 3-46
DMA_CFI	Frame index register (one for each channel)	Page 3-46

3.12.1 Global Control Register (DMA_GCR)

The global control register (see Figure 3–10) is a 16-bit read/write register used to configure global conditions in the DMA controller. Use this I/O-mapped register to set the emulation mode of the DMA controller (FREE) and to define how the DMA controller treats the enhanced host port interface (EHPI EXCL and EHPI PRIO).

Figure 3–10. Global Control Register (DMA_GCR)

DMA_GCR

15–3	2	1	0
Reserved	FREE	EHPI EXCL	EHPI PRIO
	R/W – 0	R/W – 0	R/W – 0

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 3–6. DMA_GCR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–3	Reserved	These bits are not available for your use.	–
2	FREE	Emulation mode bit. FREE controls the behavior of the DMA controller when a breakpoint is encountered in the software debugger: 0 A breakpoint suspends DMA transfers. 1 DMA transfers continue uninterrupted when a breakpoint occurs.	0
1	EHPI EXCL	EHPI exclusive access bit. EHPI EXCL determines whether the enhanced host port interface (EHPI) has exclusive access to the internal RAM of the DSP: 0 The EHPI shares the internal RAM with the DMA channels. The EHPI can access any internal and external memory in its address reach. 1 The EHPI has exclusive access to the internal RAM. If any channels must access the DARAM port or the SARAM port, activity in these channels is suspended. In this EHPI access configuration, the EHPI can only access the DARAM port and the SARAM port. It cannot access the external memory port. **Note:** Regardless of the value of EHPI EXCL, the EHPI cannot access the peripherals port.	0

Table 3–6. DMA_GCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
0	EHPI PRIO	EHPI priority bit. EHPI PRIO assigns the EHPI a high or low priority level in the service chain of the DMA controller: 0 Low priority level 1 High priority level **Note:** When the EHPI has exclusive access to the DARAM and SARAM ports (EHPI EXCL = 1), the EHPI priority is irrelevant at these ports because none of the DMA channels can access the DARAM and SARAM ports.	0

3.12.2 Channel Control Register (DMA_CCR)

Each channel has a channel control register of the form shown in the following figure. This I/O-mapped register enables you to:

- ❏ Choose how the source and destination addresses are updated (SRC AMODE and DST AMODE)
- ❏ Enable and control repeated DMA transfers (AUTOINIT, REPEAT, and END PROG)
- ❏ Enable or disable the channel (EN)
- ❏ Choose a low or high priority level for the channel (PRIO)
- ❏ Select element synchronization or frame synchronization (FS)
- ❏ Determine what synchronization event (if any) initiates a transfer in the channel (SYNC)

Figure 3–11. Channel Control Register (DMA_CCR)

DMA_CCR

15–14	13–12	11	10	9	8
DST AMODE	SRC AMODE	END PROG	Rsvd (keep 0)	REPEAT	AUTOINIT
R/W – 00	R/W – 00	R/W – 0		R/W – 0	R/W – 0

7	6	5	4–0
EN	PRIO	FS	SYNC
R/W – 0	R/W – 0	R/W – 0	R/W – 00000

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 3–7. DMA_CCR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–14	DST AMODE	Destination addressing mode. DST AMODE determines the addressing mode used by the DMA controller when it writes to the destination port of the channel: 00b Constant address The same address is used for each element transfer. 01b Automatic post increment After each element transfer, the address is incremented according to the selected data type: If data type is 8-bit Address = Address + 1 If data type is 16-bit Address = Address + 2 If data type is 32-bit Address = Address + 4 10b Single index After each element transfer, the address is incremented by the programmed element index amount: Address = Address + element index 11b Double index (sort) After each element transfer, the address is incremented by the appropriate index amount: If there are more elements to transfer in the current frame Address = Address + element index If the last element in the frame has been transferred Address = Address + frame index	00b

Table 3–7. DMA_CCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
13–12	SRC AMODE	Source addressing mode. SRC AMODE determines the addressing mode used by the DMA controller when it reads from the source port of the channel: 00b Constant address The same address is used for each element transfer. 01b Automatic post increment After each element transfer, the address is incremented according to the selected data type: If data type is 8-bit Address = Address + 1 If data type is 16-bit Address = Address + 2 If data type is 32-bit Address = Address + 4 10b Single index After each element transfer, the address is incremented by the programmed element index amount: Address = Address + element index 11b Double index (sort) After each element transfer, the address is incremented by the appropriate index amount: If there are more elements to transfer in the current frame Address = Address + element index If the last element in the frame has been transferred Address = Address + frame index	00b

Table 3–7. DMA_CCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
11	END PROG	End-of-programmation bit. Each DMA channel has two sets of registers: configuration registers and working registers. When block transfers occur repeatedly because of auto-initialization (AUTOINIT = 1), you can change the context for the next DMA transfer by writing to the configuration registers during the current block transfer. At the end of the current transfer, the contents of the configuration registers are copied into the working registers, and the DMA controller begins the next transfer using the new context. For proper auto-initialization, the CPU must finish programming the configuration registers before the DMA controller copies their contents. To make sure auto-initialization waits for the CPU, follow this procedure: 1) Make auto-initialization wait for END PROG = 1 by clearing the REPEAT bit (REPEAT = 0) 2) Clear END PROG (END PROG = 0) to indicate that programmation of the configuration registers is in process. 3) Program the configuration registers. 4) Set END PROG (END PROG = 1) to indicate the end of programmation.	0
10	Rsvd	This is a reserved bit (not available for your use) and must be kept 0. Make sure that whenever your program modifies DMA_CCR, it writes a 0 to bit 10.	–
9	REPEAT	Repeat condition bit. If autoinitialization is selected for a channel (AUTOINIT = 1), REPEAT specifies one of two special repeat conditions: 0 Repeat only if END PROG = 1 Once the current DMA transfer is complete, auto-initialization only occurs if the end-of-programmation bit (END PROG) is set. 1 Repeat regardless of END PROG Once the current DMA transfer is complete, auto-initialization occurs regardless of whether END PROG is 0 or 1.	0

Table 3–7. DMA_CCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
8	AUTOINIT	Auto-initialization bit. The DMA controller supports auto-initialization, which is the automatic reinitialization of the channel between DMA transfers. Use AUTOINIT to enable or disable this feature: 0 Auto-initialization is disabled Activity in the channel stops at the end of the current block transfer. To stop a transfer immediately, clear the channel enable bit (EN). 1 Auto-initialization is enabled Once the current block transfer is complete, the DMA controller reinitializes the channel and starts a new block transfer. To stop activity in the channel you have two options: ❑ To stop activity immediately, clear the channel enable bit (EN = 0). ❑ To stop activity after the current block transfer, clear AUTOINIT (AUTOINIT= 0)	0
7	EN	Channel enable bit. Use EN to enable or disable transfers in the channel. The DMA controller clears EN once a block transfer in the channel is complete. 0 Channel is disabled The channel cannot be serviced by the DMA controller. If a DMA transfer is already active in the channel, the DMA controller stops the transfer and resets the channel. 1 Channel is enabled The channel can be serviced by the DMA controller at the next available time slot. **Note:** If the CPU attempts to write to EN at the same time that the DMA controller must clear EN, the DMA controller is given higher priority. EN is cleared, and the value from the CPU is discarded.	0
6	PRIO	Channel priority bit. All six of the DMA channels are given a fixed position and programmable priority level on the service chain of the DMA controller. PRIO determines whether the associated channel has a high priority or a low priority. High-priority channels are serviced before low-priority channels. 0 Low priority 1 High priority	0

Table 3–7. DMA_CCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
5	FS	Frame/element synchronization bit. You can use the SYNC bits of DMA_CCR to specify a synchronization event for the channel. The FS bit determines whether the synchronization event initiates the transfer of an element or an entire frame of data: 0 Element synchronization When the selected synchronization event occurs, one element is transferred in the channel. Each element transfer waits for the synchronization event. 1 Frame synchronization When the selected synchronization event occurs, an entire frame is transferred in the channel. Each frame transfer waits for the synchronization event.	0
4–0	SYNC	Synchronization control bits. SYNC in DMA_CCR determines which event in the DSP (for example, a timer countdown) initiates a DMA transfer in the channel. Multiple channels can have the same SYNC value; in other words, one synchronization event can initiate activity in multiple channels. A DSP reset selects SYNC = 00000b (no synchronization event). When SYNC = 00000b, the DMA controller does not wait for a synchronization event before beginning a DMA transfer in the channel; channel activity begins as soon as the channel is enabled (EN = 1). To see what events are tied to values of SYNC other than 00000b, see the data sheet for your TMS320C55x DSP. As an example, Table 3–8 shows the event mapping for the TMS320VC5510 DSP.	00000b

Table 3–8. Synchronization Event Mapping for the TMS320VC5510 DSP

SYNC	Synchronization Event For The Channel (TMS320VC5510 DSP)
00000b	No synchronization event
00001b	McBSP 0 receive event (REVT0)
00010b	McBSP 0 transmit event (XEVT0)
00011b	McBSP 0 A-bis mode receive event (REVTA0)
00100b	McBSP 0 A-bis mode transmit event (XEVTA0)
00101b	McBSP 1 receive event (REVT1)
00110b	McBSP 1 transmit event (XEVT1)
00111b	McBSP 1 A-bis mode receive event (REVTA1)
01000b	McBSP 1 A-bis mode transmit event (XEVTA1)
01001b	McBSP 2 receive event (REVT2)
01010b	McBSP 2 transmit event (XEVT2)
01011b	McBSP 2 A-bis mode receive event (REVTA2)
01100b	McBSP 2 A-bis mode transmit event (XEVTA2)
01101b	Timer 1 event (TEVT1)
01110b	Timer 2 event (TEVT2)
01111b	External interrupt 0
10000b	External interrupt 1
10001b	External interrupt 2
10010b	External interrupt 3
10011b	External interrupt 4
10100b	External interrupt 5
Other values	Reserved (do not use these values)

3.12.3 Interrupt Control Register (DMA_CICR) and Status Register (DMA_CSR)

Each channel has an interrupt control register (DMA_CICR) and a status register (DMA_CSR). DMA_CICR and DMA_CSR are I/O-mapped registers. Their bits are shown in Figure 3–12 and described in Table 3–9 and Table 3–10.

Use DMA_CICR to specify that one or more DMA controller events will trigger an interrupt. If an event occurs and its interrupt enable (IE) bit is 1, an interrupt request is sent to the DSP CPU, where it can be serviced or ignored. Each channel has its own interrupt line to the CPU and one set of flag and enable bits in the CPU. In addition the DMA controller can send a bus-error interrupt request to the CPU in response to a time-out error. The bus-error interrupt also has a set of flag and enable bits in the CPU.

To see which event or events have occurred in the DMA controller, your program can read DMA_CSR. The SYNC bit is always set when its event occurs (when the DMA controller responds to the chosen synchronization event). The other status bits are not necessarily set when their events occur; the DMA controller sets one of these bits only if the event occurs and the associated interrupt enable bit is set in DMA_CICR. After your program reads DMA_CSR, all of its bits are cleared automatically.

Figure 3–12. Interrupt Control Register (DMA_CICR) and Status Register (DMA_CSR)

DMA_CICR

15–6	5	4	3	2	1	0
Reserved	BLOCK IE	LAST IE	FRAME IE	HALF IE	DROP IE	TIMEOUT IE
	R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 1	R/W – 1

DMA_CSR

15–7	6	5	4	3	2	1	0
Reserved	SYNC	BLOCK	LAST	FRAME	HALF	DROP	TIMEOUT
	R – 0	R – 0	R – 0	R – 0	R – 0	R – 0	R – 0

Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset.

Table 3–9. DMA_CICR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–6	Reserved	These bits are not available for your use.	–
5	BLOCK IE	Whole block interrupt enable bit. BLOCK IE determines how the DMA controller responds when all of the current block has been transferred from the source port to the destination port: 0 Do not record the event. 1 Set the BLOCK bit and send the channel interrupt request to the CPU.	0
4	LAST IE	Last frame interrupt enable bit. LAST IE determines how the DMA controller responds when the DMA controller starts transferring the last frame from the source port to the destination port: 0 Do not record the event. 1 Set the LAST bit and send the channel interrupt request to the CPU.	0
3	FRAME IE	Whole frame interrupt enable bit. FRAME IE determines how the DMA controller responds when the all of the current frame has been transferred from the source port to the destination port: 0 Do not record the event. 1 Set the FRAME bit and send the channel interrupt request to the CPU.	0
2	HALF IE	Half frame interrupt enable bit. HALF IE determines how the DMA controller responds when the first half of the current frame has been transferred from the source port to the destination port: 0 Do not record the event. 1 Set the HALF bit and send the channel interrupt request to the CPU.	0

Table 3–9. DMA_CICR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
1	DROP IE	Synchronization event drop interrupt enable bit. If a DMA synchronization event occurs again before the DMA controller is done servicing the previous DMA request, an error has occurred—a synchronization event drop. DROP IE determines how the DMA controller responds when a synchronization event drop occurs in the channel: 0 Do not record the drop. 1 Set the DROP bit and send the channel interrupt request to the CPU.	1
0	TIMEOUT IE	Timeout interrupt enable bit. TIMEOUT IE determines how the DMA controller responds to a timeout error at the source port or the destination port of the channel: 0 Do not record the timeout error. 1 Set the TIMEOUT bit and send the bus-error interrupt request to the CPU.	1

Table 3–10. DMA_CSR Bit Descriptions

Bit	Name	Description	Reset Value
15–7	Reserved	These bits are not available for your use.	–
6	SYNC	Synchronization event status bit. The DMA controller updates SYNC to indicate when the synchronization event for the channel has occurred and when the synchronized channel has been serviced: 0 The DMA controller has finished servicing the previous access request. 1 The synchronization event has occurred. In response to the event, the synchronized channel submits an access request to its source port. **Note 1:** If a synchronization event occurs again before the DMA controller is done servicing the previous DMA request, an error has occurred—a synchronization event drop. You can track this type of error using the DROP IE bit and the DROP bit. **Note 2:** To select a synchronization event for a channel, use the SYNC bits of DMA_CCR.	0

Table 3–10. DMA_CSR Bit Descriptions (Continued)

Bit	Name	Description	Reset Value
5	BLOCK	Whole block status bit. The DMA controller sets BLOCK only if BLOCK IE = 1 in DMA_CICR and all of the current block has been transferred from the source port to the destination port: 0 The whole-block event has not occurred yet, or BLOCK has been cleared. 1 The whole block has been transferred. A channel interrupt request has been sent to the CPU.	0
4	LAST	Last frame status bit. The DMA controller sets LAST only if LAST IE = 1 in DMA_CICR and the DMA controller has started transferring the last frame from the source port to the destination port: 0 The last-frame event has not occurred yet, or LAST has been cleared. 1 The DMA controller has started transferring the last frame. A channel interrupt request has been sent to the CPU.	0
3	FRAME	Whole frame status bit. The DMA controller sets FRAME only if FRAME IE = 1 in DMA_CICR and all of the current frame has been transferred from the source port to the destination port: 0 The whole-frame event has not occurred yet, or FRAME has been cleared. 1 The whole frame has been transferred. A channel interrupt request has been sent to the CPU.	
2	HALF	Half frame status bit. The DMA controller sets HALF only if HALF IE = 1 in DMA_CICR and the first half of the current frame has been transferred from the source port to the destination port: 0 The half-frame event has not occurred yet, or HALF has been cleared. 1 The first half of the frame has been transferred. A channel interrupt request has been sent to the CPU.	0

Table 3–10. DMA_CSR Bit Descriptions (Continued)

Bit	Name	Description	Reset Value
1	DROP	Synchronization event drop status bit. If a DMA synchronization event occurs again before the DMA controller is done servicing the previous DMA request, an error has occurred—a synchronization event drop. The DMA controller sets DROP only if DROP IE = 1 in DMA_CICR and a synchronization event drop has occurred in the channel: 0 A synchronization event drop has not occurred, or DROP has been cleared. 1 A synchronization event drop has occurred. A channel interrupt request has been sent to the CPU.	0
0	TIMEOUT	Timeout status bit. The DMA controller sets TIMEOUT only if TIMEOUT IE = 1 in DMA_CICR and a time-out error has occurred at the source port or the destination port of the channel: 0 A time-out error has not occurred, or TIMEOUT has been cleared. 1 A time-out error has occurred. A bus-error interrupt request has been sent to the CPU.	0

3.12.4 Source and Destination Parameters Register (DMA_CSDP)

Each channel has a source and destination parameters register of the form shown in Figure 3–13. This I/O-mapped register enables you to choose a source port (SRC) and a destination port (DST), specify a data type (DATA TYPE) for port accesses, enable or disable data packing (SRC PACK and DST PACK), and enable or disable burst transfers (SRC BEN and DST BEN).

Figure 3–13. Source and Destination Parameters Register (DMA_CSDP)

DMA_CSDP

15–14	13	12–9	8–7	6	5–2	1–0
DST BEN	DST PACK	DST	SRC BEN	SRC PACK	SRC	DATA TYPE
R/W – 00	R/W – 0	R/W – 0000	R/W – 00	R/W – 0	R/W – 0000	R/W – 00

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 3–11. Source and Destination Parameters Register (DMA_CSDP)

Bit(s)	Name	Description	Reset Value
15–14	DST BEN	Destination burst enable bit. A burst in the DMA controller is four consecutive 32-bit accesses at a DMA port. DST BEN determines whether the DMA controller performs a burst at the destination port of the channel. 00b Bursting disabled (single access enabled) at the destination 01b Bursting disabled (single access enabled) at the destination 10b Bursting enabled at the destination When writing to the destination, the DMA controller performs four consecutive 32-bit accesses. 11b Reserved (do not use)	00b
13	DST PACK	Destination packing enable bit. The DMA controller can perform data packing to double or quadruple the amount of data passed to the destination. For example, if an 8-bit data type is selected and the destination port has a 32-bit data bus, four 8-bit pieces of data can be packed into 32 bits before being sent to the destination. DST PACK determines whether data packing is used at the destination port. 0 Packing disabled at the destination 1 Packing enabled at the destination Where possible, the DMA controller packs data before each write to the destination. Table 3–12 (page 3-44) shows the instances where data packing is performed.	0
12–9	DST	Destination selection bit. DST selects which DMA port is the destination for data transfers in the channel (a bit shown as X can be 0 or 1): XX00b SARAM (single-access RAM inside the DSP) XX01b DARAM (dual-access RAM inside the DSP) XX10b External memory (via the external memory interface, EMIF) XX11b Peripherals (via the peripheral bus controller)	0000b

Table 3–11. Source and Destination Parameters Register (DMA_CSDP) (Continued)

Bit(s)	Name	Description	Reset Value
8–7	SRC BEN	Source burst enable bit. A burst in the DMA controller is four consecutive 32-bit accesses at a DMA port. SRC BEN determines whether the DMA controller performs a burst at the source port of the channel. 00b Bursting disabled (single access enabled) at the source 01b Bursting disabled (single access enabled) at the source 10b Bursting enabled at the source When reading from the source, the DMA controller performs four consecutive 32-bit accesses. 11b Reserved (do not use)	00b
6	SRC PACK	Source packing enable bit. The DMA controller can perform data packing to double or quadruple the amount of data gathered at the source. For example, if an 8-bit data type is selected and the source port has a 32-bit data bus, four 8-bit pieces of data can be packed into 32 bits before being sent through the channel. SRC PACK determines whether data packing is used at the source port. 0 Packing disabled at the source 1 Packing enabled at the destination Where possible, the DMA controller packs data from the source before beginning a data transfer in the channel. Table 3–12 (page 3-44) shows the instances where data packing is performed.	0
5–2	SRC	Source selection bit. SRC selects which DMA port is the source for data transfers in the channel (a bit shown as X can be 0 or 1): XX00b SARAM (single-access RAM inside the DSP) XX01b DARAM (dual-access RAM inside the DSP) XX10b External memory (via the external memory interface, EMIF) XX11b Peripherals (via the peripheral bus controller)	0000b

Table 3–11. Source and Destination Parameters Register (DMA_CSDP) (Continued)

Bit(s)	Name	Description	Reset Value
1–0	DATA TYPE	Data type bit. DATA TYPE indicates how data is to be accessed at the source and at the destination of the channel. Note that the DMA controller uses **byte addresses** for its accesses; each byte in data space or I/O space has its own address. For information on how addresses are updated between element transfers, see the descriptions for the SRC AMODE bits and the DST AMODE bits in DMA_CCR (page 3-29). 00b 8-bit The DMA controller makes 8-bit accesses at the source and at the destination of the channel. The source and destination start addresses have no alignment constraint: Start address: XXXX XXXX XXXX XXXXb (X can be 0 or 1) If you choose the automatic post increment addressing mode at the source or the destination, the corresponding address is updated by an increment of 1 after each element transfer. 01b 16-bit The DMA controller makes 16-bit accesses at the source and at the destination. The source and destination start addresses must each be on an even 2-byte boundary; the least significant bit (LSB) must be 0: Start address: XXXX XXXX XXXX XXX0b (X can be 0 or 1) If you choose the automatic post increment addressing mode at the source or the destination, the address is updated by an increment of 2 after each element transfer. 10b 32-bit The DMA controller makes 32-bit accesses at the source and at the destination. The source and destination start addresses must be on an even 4-byte boundary; the 2 LSBs must be 0: Start address: XXXX XXXX XXXX XX00b (X can be 0 or 1) If you choose the automatic post increment addressing mode at the source or the destination, the address is updated by an increment of 4 after each element transfer. 11b Reserved (do not use)	00b

Table 3–12. Data Packing Performed by the DMA Controller

Data Type	Port Bus Size	Data Packing
8-bit	16-bit	Two data values packed into 16 bits.
8-bit	32-bit	Four data values packed into 32 bits.
16-bit	32-bit	Two data values packed into 32 bits.

3.12.5 Source Start Address Registers (DMA_CSSA_L and DMA_CSSA_U)

Each channel has two source start address registers, which are shown in Figure 3–14. For the first access to the source port of the channel, the DMA controller generates a byte address by concatenating the contents of the two I/O-mapped registers. DMA_CSSA_U supplies the upper bits, and DMA_CSSA_L supplies the lower bits:

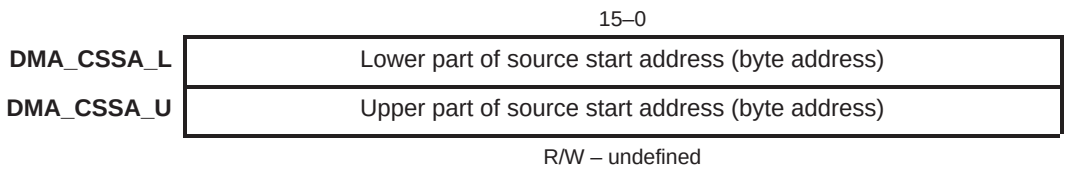
Source start address: DMA_CSSA_U:DMA_CSSA_L

Notes:

1) You must load the source start address registers with a byte address. If you have a word address, shift it left by 1 before loading the registers.

2) If you have a 16-bit or 32-bit data type, the start address must be aligned properly. See the description for the DATA TYPE bits of DMA_CSDP (page 3-40).

Figure 3–14. Source Start Address Registers (DMA_CSSA_L and DMA_CSSA_U)



Legend:

R/W Read/write access

– undefined The contents of the registers is undefined after a DSP reset.

The destination start address is supplied by DMA_CDSA_L and DMA_CDSA_U, which are described in section 3.12.6.

3.12.6 Destination Start Address Registers (DMA_CDSA_L and DMA_CDSA_U)

Each channel has two destination start address registers, which are shown in Figure 3–15. For the first access to the destination port of the channel, the DMA controller generates a byte address by concatenating the contents of the two I/O-mapped registers. DMA_CDSA_U supplies the upper bits, and DMA_CDSA_L supplies the lower bits:

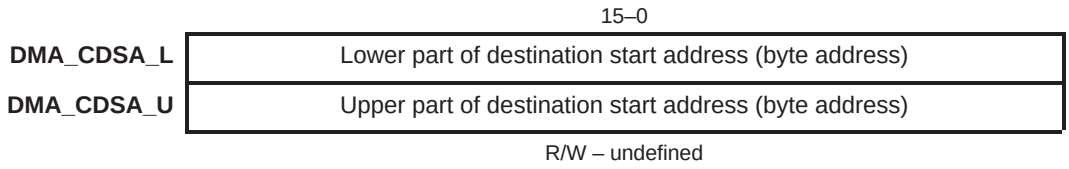
Destination start address: DMA_CDSA_U:DMA_CDSA_L

Notes:

1) You must load the destination start address registers with a byte address. If you have a word address, shift it left by 1 before loading the registers.

2) If you have a 16-bit or 32-bit data type, the start address must be aligned properly. See the description for the DATA TYPE bits of DMA_CSDP (page 3-40).

Figure 3–15. Destination Start Address Registers (DMA_CDSA_L and DMA_CDSA_U)



Legend:

R/W	Read/write access
– undefined	The contents of the registers is undefined after a DSP reset.

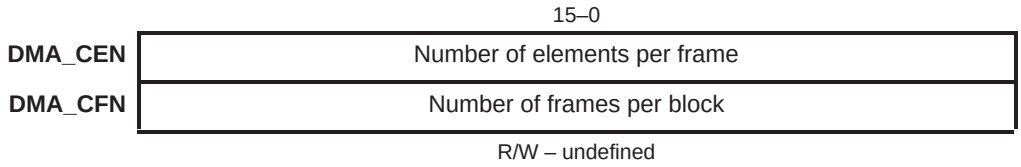
The source start address is supplied by DMA_CSSA_L and DMA_CSSA_U, which are described in section 3.12.5.

3.12.7 Element Number Register (DMA_CEN) and Frame Number Register (DMA_CFN)

Each channel has an element number register and a frame number register, which are shown in Figure 3–16. Load DMA_CFN with the number of frames you want in the channel. Load DMA_CEN with the number of elements you want in each frame. You must have at least one frame and one element, and you can have as many as 65535 of each:

$1 \leq$ frame number ≤ 65535
$1 \leq$ element number ≤ 65535

Figure 3–16. Element Number Register (DMA_CEN) and Frame Number Register (DMA_CFN)



Legend:

R/W Read/write access
– undefined The contents of the registers is undefined after a DSP reset.

3.12.8 Element Index Register (DMA_CEI) and Frame Index Register (DMA_CFI)

Each channel has an element index register and a frame index register, which are shown in Figure 3–17. Load DMA_CFI with the frame index you want to use for the double-index addressing mode. Load DMA_CEI with the element index you want to use for the single- or double-index addressing mode. You select the single- or double-index addressing mode separately for the source and destination ports by using the SRC AMODE bits and the DST AMODE bits, respectively, in DMA_CCR (page 3-29).

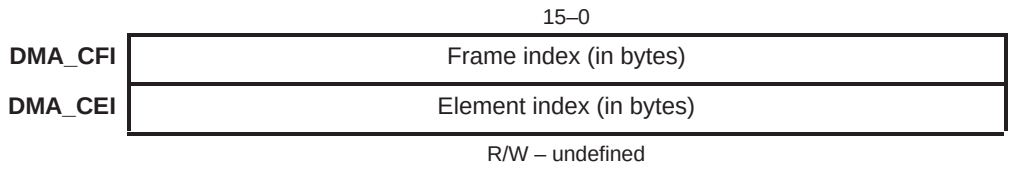
Both indexes are 16-bit signed numbers, providing the following range:

–32768 bytes ≤ frame index ≤ 32767 bytes
–32768 bytes ≤ element index ≤ 32767 bytes

Note:

If you use an index addressing mode, make sure that all the addresses computed by the DMA controller will match the alignment constraint for the chosen data type. For more details, see the description for the DATA TYPE bits of DMA_CSDP (page 3-40).

Figure 3–17. Element Index Register (DMA_CEI) and Frame Index Register (DMA_CFI)



Legend:

R/W	Read/write access
– undefined	The contents of the registers is undefined after a DSP reset.

Chapter 4

Enhanced Host Port Interface (EHPI)

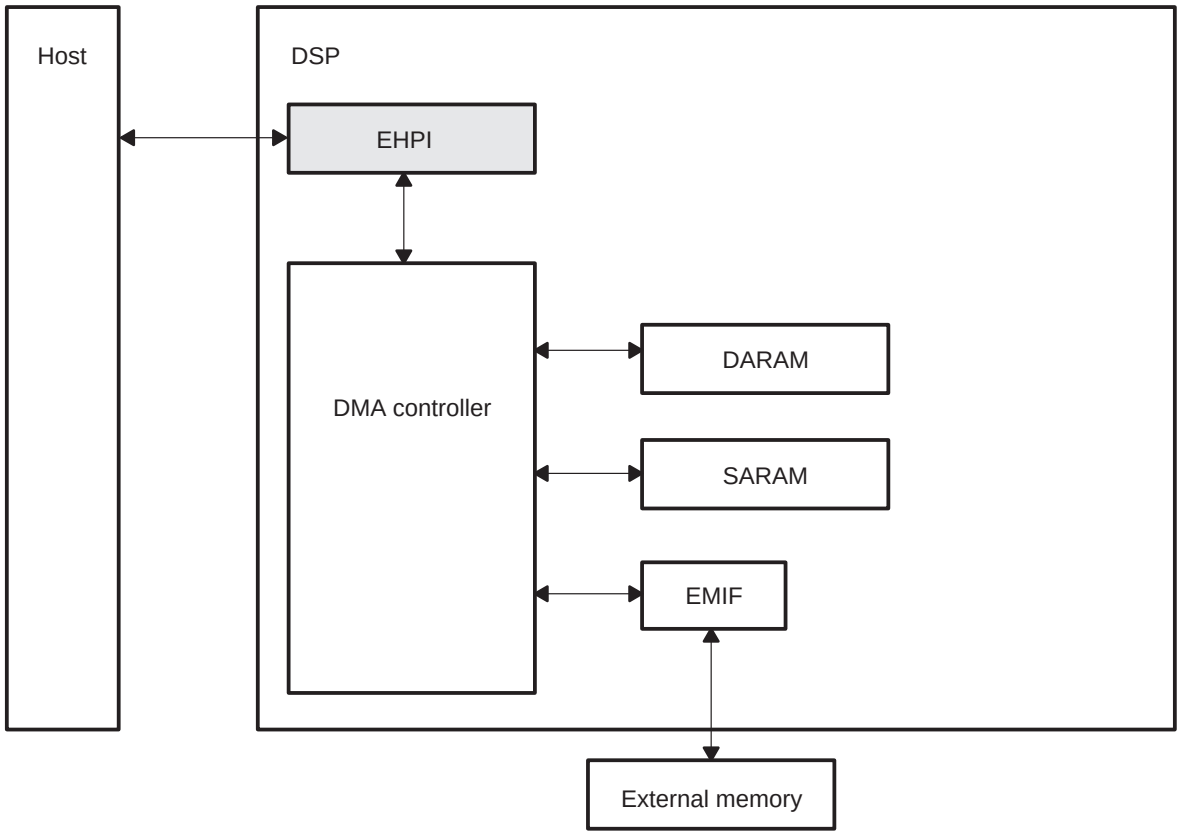
This chapter describes the enhanced host port interface (EHPI) of the TMS320C55x™ DSP. The EHPI enables an external host processor (host) to directly access a portion of the memory in the memory map of the C55x™ DSP.

Topic **Page**

4.1 Introduction to the EHPI

The enhanced host port interface (EHPI) provides a 16-bit-wide parallel port through which a host processor (host) can directly access the memory of the DSP. The host and the DSP can exchange information via memory internal or external to the DSP and within the address reach of the EHPI. The EHPI uses 23-bit addresses, where each address is assigned to a 16-bit word in memory. Figure 4–1 is a conceptual diagram of the connections between the EHPI and other components of a host-DSP system.

Figure 4–1. The Position of the EHPI in a Host-DSP System



The DMA controller (see Chapter 3) handles all EHPI accesses. Through the DMA controller, you can choose one of two EHPI access configurations (see page 3-7). In one configuration, the EHPI shares internal memory with the DMA channels. In the other configuration, the EHPI has exclusive access to the internal memory.

The EHPI cannot directly access the peripherals of the DSP. Data from peripherals must be transferred to memory before being transferred to the host. Likewise, data from the host must be transferred to memory before being transferred to peripherals.

When the DSP reset signal is low, the EHPI can access only the internal single-access RAM (SARAM). The EHPI can prolong the DSP reset process so that the host can load code into the SARAM before the CPU fetches the DSP reset vector. For more details, see section 4.8 on page 4-20.

To provide flexibility in the choice of a host, the EHPI allows two modes for passing data and addresses. The nonmultiplexed mode (see page 4-7) provides the host processor with separate address and data buses. The multiplexed mode (see page 4-12) provides a single bus to transport address and data information. The different modes require some different connections to EHPI signals. There are three EHPI registers for data, addresses, and control information (see page 4-23).

4.2 EHPI Signals

Table 4–1 provides a summary of the signals. In the Type column, Z refers to the high impedance state. There are some differences in the signal connections between the two modes of the EHPI: Nonmultiplexed mode (see page 4-7) and multiplexed mode (see page 4-12).

Table 4–1. Signals of the EHPI

Signal(s)	Type	Description
HD[15:0]	Input/Output/Z	Host data bus. HD in the nonmultiplexed mode: These signals are used to carry data only. HD in the multiplexed mode: These 16 signals are used to carry both addresses and data. The EHPI places HD in the high impedance state whenever HD is not being used for a read operation or a write operation.
HA[19:0]	Input	Host address bus. HA in the nonmultiplexed mode: These signals transfer addresses from the host processor to the EHPI. HA in the multiplexed mode: Pin HA[1] is used for the signal HCNTL1. Pin HA[2] is available for the signal HAS_. The other pins are not used.
HBE[1:0]	Input	Host byte-enable signals. These two signals determine whether the host processor is accessing the whole word, the least significant byte, or the most significant byte at the addressed memory location. The effect of the HBE pins is shown in the following table. In the first table column, 0 indicates a low signal level, and 1 indicates a high signal level. **HBE[1:0] Access** 00 Word 01 MSByte 10 LSByte 11 Reserved (do not use)
HCS_	Input	Chip-select signal. This input is used to indicate when the address and control input lines are valid. It serves as the enable input of the EHPI, and must be low during an access.
HR/W_	Input	Read/write signal. This input indicates the direction of the host access. When high, HR/W_ indicates a read from the DSP memory. When low, HR/W_ indicates a write to the DSP memory. This signal is valid at the same time the address bus is valid.

Table 4–1. Signals of the EHPI (Continued)

Signal(s)	Type	Description
HDS1_, HDS2_	Input	Data strobe signals. The exclusive-NOR of HDS1_ and HDS2_ forms a strobe signal for controlling data transfers during host-access cycles. The width of the strobe signal must be a minimum of 2 CPU clock cycles. Connections to HDS1_ and HDS2_ depend on the host's strobe signal(s): **Available Host Data Strobe Pins** / **Connections to EHPI Data Strobe Pins** Host has separate read and write strobe pins, both active-low — Connect one strobe to HDS1_ and the other to HDS2_. Host has one active-low strobe pin — Connect the strobe to HDS1_ or HDS2_, and connect the other pin to logic level 1. Host has one active-high strobe pin — Connect the strobe to HDS1_ or HDS2_ and connect the other pin to logic level 0.
HRDY	Output	EHPI ready signal. This signal tells the host whether the EHPI is ready for an access. When low, HRDY indicates that the EHPI is busy and the host should extend the current transfer cycle. When high, HRDY indicates that the EHPI has completed the data transfer and is ready for the host to continue. HRDY is always high (EHPI ready) when HCS_ is inactive (high).
HCNTL0, HCNTL1	Input	EHPI control signals. HCNTL signals in the nonmultiplexed mode: HCNTL0 determines whether the EHPI accesses the DSP data memory (HCNTL0 high) or the EHPI control register (HCNTL0 low). HCNTL1 is not used. HCNTL signals in the multiplexed mode: HCNTL1 and HCNTL0 together select the type of register access. HCNTL1 is multiplexed with line 1 of the address bus, HA[1], but this creates no conflict because HA is not used in the multiplexed mode.

Table 4–1. Signals of the EHPI (Continued)

Signal(s)	Type	Description
HAS_	Input	Address strobe signal. This signal is used only in the multiplexed mode. This address strobe signal allows HCNTL[1:0] and HR/W_ to be removed earlier in an access cycle, which allows more time to switch bus states from address to data information. HAS_ facilitates interfacing to multiplexed address and data type buses. An address latch enable (ALE) signal is often provided and would normally be the signal connected to HAS_. HAS_ is multiplexed with line 2 of the address bus, HA[2], but this creates no conflict because HA is not used in the multiplexed mode.
HMODE	Input	EHPI mode signal. When held high, HMODE selects the nonmultiplexed mode. When held low, HMODE selects the multiplexed mode.
RST_MODE	Input	Reset mode signal. If RST_MODE is high: When the DSP reset signal is asserted at the pin, the RESET bit of HPIC is cleared and remains 0 after the reset signal returns to the high level. At this time, the EHPI can access the single-access RAM (SARAM) inside the DSP. The DSP CPU does not start running until the host sets the RESET bit. If RST_MODE is low: The RESET bit is ignored. When the DSP reset signal is asserted, the DSP CPU starts running immediately after the reset signal goes high.
HINT_	Output	DSP-to-host interrupt signal. HINT_ enables you to send an interrupt pulse to the host processor. The signal level is controlled by the HINT bit in status register ST3_55 (HINT = 0 means HINT_ low; HINT = 1 means HINT_ high).

4.3 Nonmultiplexed Mode

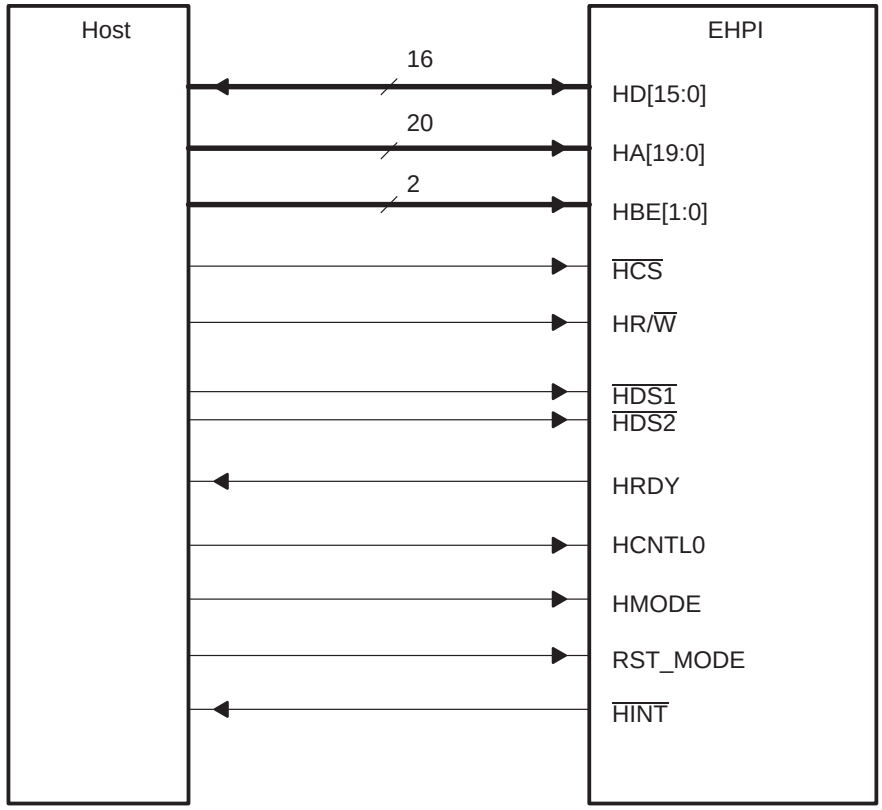
In its nonmultiplexed mode, the EHPI uses **separate buses for addresses and data**. To select this mode, hold the HMODE signal high. This section covers the following topics:

- Signal connections (this page)
- Transferring addresses and data (page 4-11)
- Using HPIC (EHPI control register) (page 4-11)

4.3.1 Signal Connections in the Nonmultiplexed Mode

Figure 4–2 shows signal connections for the nonmultiplexed mode, and details about these connections follow the figure.

Figure 4–2. Host-EHPI Connections in the Nonmultiplexed Mode



HD[15:0] HD[15:0] is a parallel, bidirectional, 3-state data bus. The EHPI places HD in the high impedance state whenever HD is not being used for a read operation or a write operation. In this EHPI mode (the nonmultiplexed mode), HD is used for data only.

HA[19:0] HA[19:0] is a parallel, unidirectional address bus that carries 20-bit addresses from the host processor to the EHPI. The 20 lines of this bus allow the addressing of 1024K words of the DSP memory.

HBE[1:0] Byte-enable signals. These two signals determine whether the EHPI accesses the whole word, the least significant byte (LSByte), or the most significant byte (MSByte) at the addressed location in the DSP memory.

Table 4–2 shows the available HBE signal combinations and describes their effects. In the first table column, 0 indicates a low signal level, and 1 indicates a high signal level.

Table 4–2. Effect of Driving the EHPI Byte-Enable Signals in the Nonmultiplexed Mode

HBE[1:0]	Access	For Reading ...	For Writing ...
00	Word	The EHPI reads a word at the address specified on HA[19:0] and sends the word out on HD[15:0].	The EHPI accepts a word from HD[15:0] and writes the word to the address specified on HA[19:0].
01	MSByte	The EHPI reads the 8 MSBs at the address specified on HA[19:0] and sends the byte out on HD[15:8].	The EHPI accepts a byte from HD[15:8] and writes the byte to the 8 MSBs at the address specified on HA[19:0]. The 8 LSBs are not modified.
10	LSByte	The EHPI reads the 8 LSBs at the address specified on HA[19:0] and sends the byte out on HD[7:0].	The EHPI accepts a byte from HD[7:0] and writes the byte to the 8 LSBs at the address specified on HA[19:0]. The 8 MSBs are not modified.
11	Reserved	–	–

HCS_ Chip-select signal. This input is used to indicate when the address and control input lines are valid. It serves as the enable input of the EHPI, and must be low during an access.

HCS_ high: Address and control input lines not valid
HCS_ low: Address and control input lines valid

HR/W_ Read/write signal. This input indicates the direction of the host access:

HR/W_ high: Read from DSP memory
HR/W_ low: Write to DSP memory

This signal is valid at the same time the address bus is valid.

HDS1_, HDS2_ Data strobe signals. The exclusive-NOR of HDS1_ and HDS2_ forms a strobe signal for controlling data transfers during host-access cycles. The width of the strobe signal must be a minimum of 2 CPU cycles. Connections to HDS1_ and HDS2_ depend on the host's strobe signal(s), as shown in Table 4–3.

Table 4–3. Host-EHPI Data Strobe Connections

Available Host Data Strobe Pins	Connections to EHPI Data Strobe Pins
Host has separate read and write strobe pins, both active-low	Connect one strobe to HDS1_ and the other to HDS2_.
Host has one active-low strobe pin	Connect the strobe to HDS1_ or HDS2_, and connect the other pin to logic level 1.
Host has one active-high strobe pin	Connect the strobe to HDS1_ or HDS2_, and connect the other pin to logic level 0.

HRDY This ready signal tells the host whether the EHPI is ready for an access:

HRDY low: The EHPI is busy. The host should extend the current transfer cycle.
HRDY high: The EHPI has completed the data transfer and is ready for the host to continue.

HRDY is always high (EHPI ready) when HCS_ is high (inactive).

HCNTL0 This signal indicates whether the EHPI accesses the DSP data memory or the EHPI control register (HPIC):

HCNTL0 high: EHPI accesses DSP data memory
HCNTL0 low: EHPI accesses HPIC

HMODE This signal must be held high to enable the nonmultiplexed mode of the EHPI.

HMODE high: Nonmultiplexed mode (separate address and data buses)
HMODE low: Multiplexed mode (data bus carries addresses and data)

RST_MODE If you want the host to prolong the DSP reset process, hold this signal high. Otherwise, hold RST_MODE low.

RST_MODE high: When the DSP reset signal is asserted at the pin, the RESET bit of HPIC is cleared and remains 0 after the reset signal returns to the high level. At this time, the EHPI can access the SARAM inside the DSP. The DSP CPU does not start running until the host sets the RESET bit.
RST_MODE low: The RESET bit is ignored. When the DSP reset signal is asserted, the DSP CPU starts running immediately after the reset signal goes high.

HINT_ This is the DSP-to-host interrupt signal, which enables you to interrupt the host processor. The signal level is controlled by the HINT bit in status register ST3_55 of the CPU (HINT = 0 means HINT_ is low; HINT = 1 means HINT_ is high).

4.3.2 Transferring Addresses and Data in the Nonmultiplexed Mode

In this mode, the EHPI receives 20-bit addresses via the address bus (HA). For each data transfer, an address must be driven on HA. The EHPI address register (HPIA) is not used.

The EHPI data register (HPID) acts as a temporary holding place for data to be transferred through the EHPI. If the current access is a read, HPID contains the data that was read from the DSP memory. If the current access is a write, HPID contains the data that will be written to the DSP memory. The DSP CPU cannot access HPID.

The EHPI registers are described in section 4.12 on page 4-23.

4.3.3 Using HPIC in the Nonmultiplexed Mode

HPIC provides the host with important options for controlling data transfers. By writing to HPIC, you can send interrupt requests to the DSP and control the timing of the DSP reset process.

To access HPIC in the nonmultiplexed mode, the host must drive the HCNTL0 signal low. The DSP CPU cannot access HPIC.

HPIC and the other EHPI registers are described in section 4.12 on page 4-23.

4.4 Multiplexed Mode

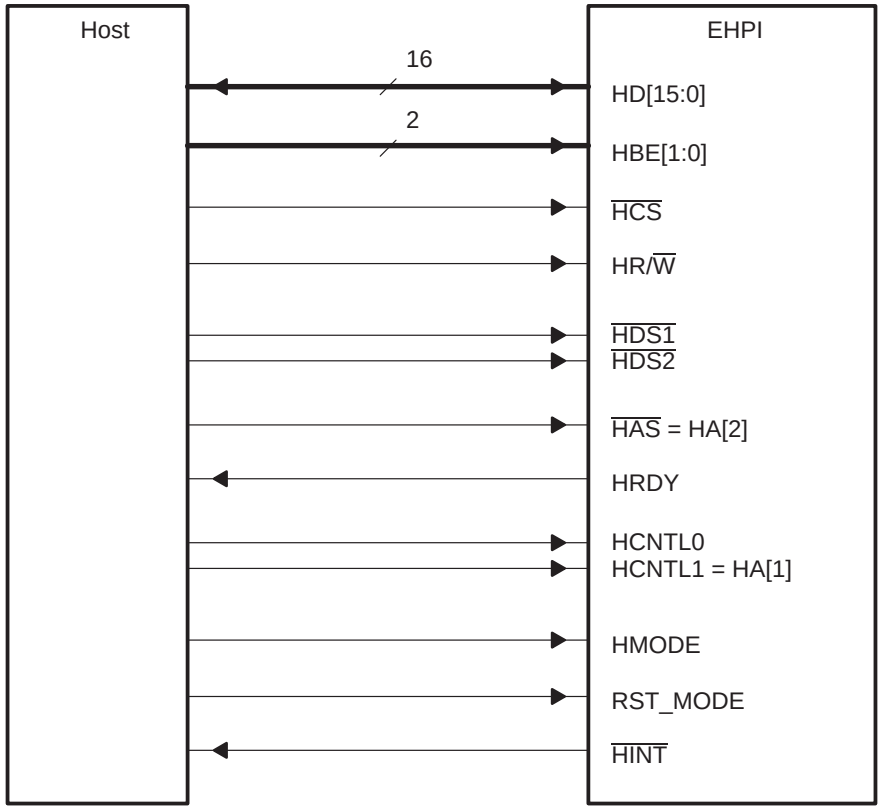
In the multiplexed mode of the EHPI, **addresses and data are carried on the same bus** (the EHPI data bus, HD[15:0]). To select this mode, hold the HMODE signal low. This section covers the following topics:

- ❏ Signal connections (this page)
- ❏ Transferring addresses and data (page 4-16)
- ❏ Autoincrement mode (for automatic address incrementing) (page 4-17)
- ❏ Using HPIC (EHPI control register) (page 4-17)

4.4.1 Signal Connections in the Multiplexed Mode

Figure 4–3 shows signal connections for the multiplexed mode, and details about these connections follow the figure. HAS_ and HCNTL1 each use one of the lines of the address bus (HA), which is not otherwise used in the multiplexed mode. The other lines of HA are pulled up internally.

Figure 4–3. Host-EHPI Connections in the Multiplexed Mode



HD[15:0] HD[15:0] is a parallel, bidirectional, 3-state data bus. The EHPI places HD in the high impedance state whenever HD is not being used for a read operation or a write operation. In this EHPI mode (the multiplexed mode), HD is used for addresses and data.

HBE[1:0] Byte-enable signals. These two signals determine whether the EHPI accesses the whole word, the least significant byte (LSByte), or the most significant byte (MSByte) at the addressed location in the DSP memory.

Table 4–4 shows the available HBE signal combinations and describes their effects. In the first table column, 0 indicates a low signal level, and 1 indicates a high signal level.

Table 4–4. Effect of Driving the EHPI Byte-Enable Signals in the Multiplexed Mode

HBE[1:0]	Access	For Reading ...	For Writing ...
00	Word	The EHPI reads a word at the address specified in HPIA and sends the word out on HD[15:0].	The EHPI accepts a word from HD[15:0] and writes the word to the address specified in HPIA.
01	MSByte	The EHPI reads the 8 MSBs at the address specified in HPIA and sends the byte out on HD[15:8].	The EHPI accepts a byte from HD[15:8] and writes the byte to the 8 MSBs at the address specified in HPIA. The 8 LSBs are not modified.
10	LSByte	The EHPI reads the 8 LSBs at the address specified in HPIA and sends the byte out on HD[7:0].	The EHPI accepts a byte from HD[7:0] and writes the byte to the 8 LSBs at the address specified in HPIA. The 8 MSBs are not modified.
11	Reserved	–	–

HCS_ Chip-select signal. This input is used to indicate when the address and control input lines are valid. It serves as the enable input of the EHPI, and must be low during an access.

HCS_ high: Address and control input lines not valid
HCS_ low: Address and control input lines valid

HR/W_ Read/write signal. This input indicates the direction of the host access:

HR/W_ high: Read from DSP memory
HR/W_ low: Write to DSP memory

This signal is valid at the same time the address bus is valid.

HDS1_, HDS2_ Data strobe signals. The exclusive-NOR of HDS1_ and HDS2_ forms a strobe signal for controlling data transfers during host-access cycles. The width of the strobe signal must be a minimum of 2 CPU cycles. Connections to HDS1_ and HDS2_ depend on the host's strobe signal(s), as shown in Table 4–5.

Table 4–5. Host-EHPI Data Strobe Connections

Available Host Data Strobe Pins	Connections to EHPI Data Strobe Pins
Host has separate read and write strobe pins, both active-low	Connect one strobe to HDS1_ and the other to HDS2_.
Host has one active-low strobe pin	Connect the strobe to HDS1_ or HDS2_, and connect the other pin to logic level 1.
Host has one active-high strobe pin	Connect the strobe to HDS1_ or HDS2_, and connect the other pin to logic level 0.

HAS_ This address strobe signal allows HCNTL[1:0] and HR/W_ to be removed earlier in an access cycle, which allows more time to switch bus states from address to data information. HAS_ facilitates interfacing to multiplexed address and data type buses. An address latch enable (ALE) signal is often provided and would normally be the signal connected to HAS_. HAS_ is multiplexed with line 2 of the address bus, HA[2], but this creates no conflict because HA is not used in this EHPI mode (the multiplexed mode).

HRDY This ready signal tells the host whether the EHPI is ready for an access:

HRDY low: The EHPI is busy. The host should extend the current transfer cycle.
HRDY high: The EHPI has completed the data transfer and is ready for the host to continue.

HRDY is always high (EHPI ready) when HCS_ is high (inactive).

HCNTL[1:0] These control signals determine which EHPI register is being accessed. You can select the EHPI address register (HPIA), the EHPI data register (HPID), or the EHPI control register (HPIC) as shown in the following table (0 indicates a low signal level, and 1 indicates a high signal level). Notice that for accesses to HPID, HCNTL[1:0] also determine whether the address is automatically incremented by 1 during each access. HCNTL1 is multiplexed with line 1 of the address bus, HA[1], but this creates no conflict because HA is not used in the multiplexed mode.

Table 4–6. Effect of Driving the EHPI Control Signals in the Multiplexed Mode

HCNTL[1:0]	Register Access Type
00	HPIC read or write
01	HPID read/write with address autoincrement by 1
10	HPIA read/write
11	HPID read/write without address autoincrement by 1

HMODE This signal must be held low to enable the multiplexed mode of the EHPI:

HMODE high: Nonmultiplexed mode (separate address and data buses)
HMODE low: Multiplexed mode (data bus carries addresses and data)

RST_MODE If you want the host to prolong the DSP reset process, hold this signal high. Otherwise, hold RST_MODE low.

RST_MODE high: When the DSP reset signal is asserted at the pin, the RESET bit of HPIC is cleared and remains 0 after the reset signal returns to the high level. At this time, the EHPI can access the SARAM inside the DSP. The DSP CPU does not start running until the host sets the RESET bit.
RST_MODE low: The RESET bit is ignored. When the DSP reset signal is asserted, the DSP CPU starts running immediately after the reset signal goes high.

HINT_ This is the DSP-to-host interrupt signal, which enables you to interrupt the host processor. The signal level is controlled by the HINT bit in status register ST3_55 of the CPU (HINT = 0 means HINT_ is low; HINT = 1 means HINT_ is high.)

4.4.2 Transferring Addresses and Data in the Multiplexed Mode

In the multiplexed mode, the data bus (HD) is used for both addresses and data. Thus, an address register (HPIA) is needed to store an address while the bus is carrying data. HPIA is 20 bits wide in order to support up to 1024K words of accessible memory. The multiplexing of addresses and data means that the host must load HPIA *before* performing reads and writes to the DSP memory.

When reading from or writing to the DSP memory, the EHPI uses its data register (HPID) as a temporary holding place for the data. HPID contains the data that was read from the DSP memory (for a host read operation) or the data that will be written to the DSP memory (for a host write operation). The DSP CPU cannot access HPID.

It is important to keep in mind that the host must drive the signals HCNTL1 and HCNTL0 to the appropriate levels to indicate whether the host is accessing HPIA or HPID, and whether address autoincrementing is used. Autoincrementing is explained in section 4.4.3.

Loading HPIA With an Address:

1) Drive both HCNTL1 and HCNTL0 low to indicate an HPIC access.

2) Clear the XADD bit in HPIC (XADD = 0), so that writes to HPIA go to HPIA(15–0).

3) Drive HCNTL1 high and keep HCNTL0 low to indicate an HPIA access.

4) Send the 16 LSBs of the memory address on the data bus (HD). The 16 LSBs fill HPIA(15–0).

5) Drive HCNTL1 low and keep HCNTL0 low to indicate an HPIC access.

6) Set the XADD bit (XADD = 1), so that writes to HPIA go to HPIA(19–16).

7) Drive HCNTL1 high and keep HCNTL0 low to indicate an HPIA access.

8) Send the 4 MSBs of the memory address on HD[3:0]. Because of the XADD setting, the 4 MSBs fill HPIA(19–16). Now HPIA contains the full 20-bit address.

Note:

If you are making a series of accesses within one main data page, you need to load HPIA(19–16) only one time, before the first access. For each subsequent access within that page, a change to HPIA(15–0) is sufficient.

4.4.3 Autoincrement Mode: Automatic Address Increment Between Transfers (Multiplexed Mode Only)

If the host is reading and/or writing at random addresses, it must write to HPIA before each data transfer. If the host performs accesses at sequential addresses and the EHPI is in its multiplexed mode, it can reduce the required number of cycles by using the address autoincrement mode: Drive HCNTL1 low and HCNTL0 high. In the autoincrement mode, the host needs to write only the start address to HPIA; for each subsequent HPID access, the address in HPIA is automatically incremented by 1.

4.4.4 Using HPIC in the Multiplexed Mode

HPIC provides the host with important options for controlling data transfers. By writing to HPIC, you can select which portion of HPIA is being loaded, send interrupt requests to the DSP, and control the timing of the DSP reset process. To access HPIC in the multiplexed mode, the host must drive the signals HCNTL1 and HCNTL0 low. The DSP CPU cannot access HPIC.

4.5 Interrupts Between Host and DSP

By modifying special interrupt bits, the host and the DSP can send interrupt requests to each other.

4.5.1 Sending an Interrupt Request from the Host to the DSP

You can have the host send an interrupt request to the DSP as follows:

1) Make sure the EHPI is configured to write to the EHPI control register (HPIC).

   In the nonmultiplexed mode of the EHPI, the HCNTL0 signal must be held low to select HPIC. In the multiplexed mode of the EHPI, HCNTL1 and HCNTL0 must both be held low to select HPIC.

2) Write a 1 to bit 1 (DSPINT) of HPIC.

   Setting this bit causes the DSP to set the DSPINT flag bit in the CPU. If this maskable interrupt is properly enabled in the CPU, the CPU will fetch the DSPINT interrupt vector and branch to the corresponding interrupt service routine.

4.5.2 Sending an Interrupt Request from the DSP to the Host

The DSP can send an interrupt request to the host by clearing and then setting the HINT bit in status register ST3_55 of the CPU. A change to the HINT bit changes the level of the HINT_ output signal of the EHPI. If the DSP writes a 0 to the HINT bit, HINT_ goes low (active). If the DSP writes a 1 to the HINT bit, HINT_ goes high (inactive). Thus, the interrupt pulse width is managed by software.

There is no direct acknowledging path from the host to the HINT bit. You can select a space in the memory shared by the host and the DSP to create an acknowledging path to the interrupt.

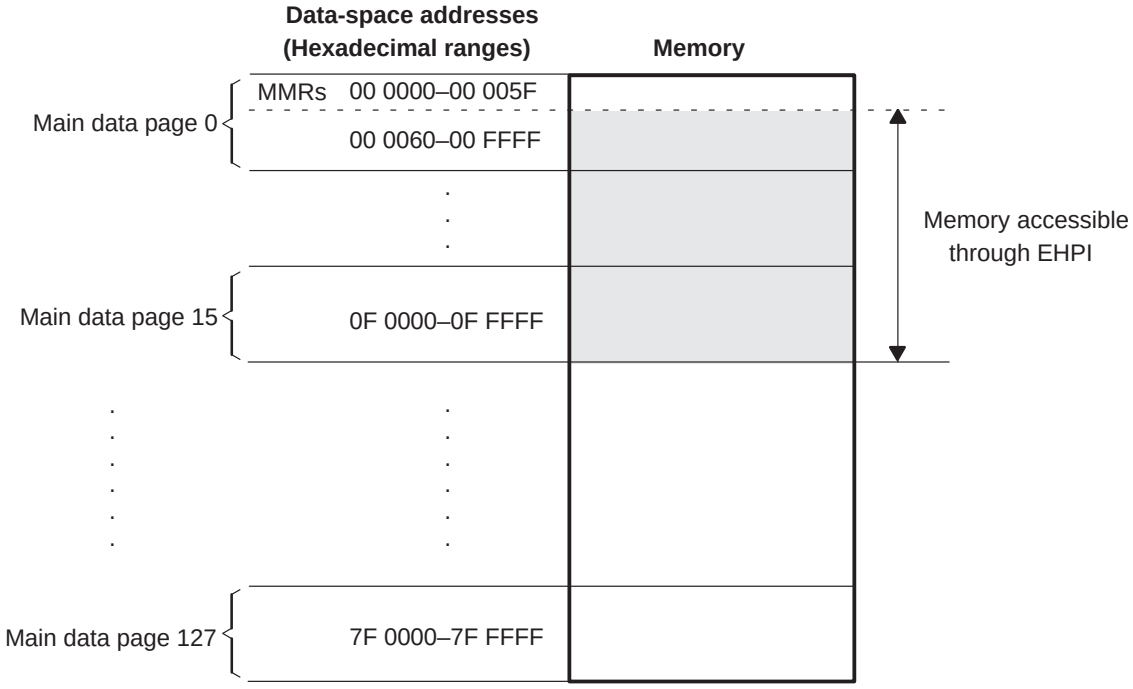
During a DSP reset, the CPU sets the HINT bit and HINT_ goes high (inactive).

4.6 Memory Accessible Through the EHPI

Addresses driven by a host on the external address lines of the EHPI are treated as word addresses, not byte addresses. Each EHPI address corresponds to a 16-bit word in data space.

Figure 4–4 shows a map of the memory available on the DSP and highlights the portion of data space that is accessible to a host through the EHPI. Data space is divided into 128 main data pages (0 through 127). The 20 address lines of the EHPI enable the host to access internal and external memory on main data pages 0 through 15 (at addresses 00 0060h–0F FFFFh). On main data page 0, the first 96 addresses (00 0000h–00 005Fh) are reserved for the memory-mapped registers (MMRs) of the CPU and are not accessible through the EHPI.

Figure 4–4. Memory Accessible Through EHPI



4.7 Boot Loading with the EHPI

When the DSP reset signal is low, the EHPI can access the single-access RAM (SARAM) internal to the DSP. During this time, you can load boot code from the host to the SARAM. More details are in section 4.8.

4.8 Changing the DSP Reset Process to Accommodate a Host

The DSP reset process is controlled by the reset input signal of the DSP and the reset mode signal (RST_MODE). When RST_MODE is high, the RESET bit of the EHPI control register (HPIC) determines how long the DSP stays in reset.

- **RST_MODE is high.** The DSP assumes there is a host interfaced to the EHPI. The reset process is controlled by both the reset pin on the DSP and by the RESET bit located in HPIC.

  When the reset signal is driven low, the RESET bit is cleared and remains 0 after the reset signal returns to its high level. If the host then sets the RESET bit, the DSP CPU starts running and fetches its reset vector.

  During the time when the reset signal is high and the RESET bit is 0, the host can access the SARAM internal to the DSP. This is a good time for the host to download code to the DSP RAM. When this download is complete, the host can set the RESET bit to start the DSP CPU.

- **RST_MODE is low.** The DSP assumes there is no host interfaced to the EHPI and initiates a standard reset process: When the reset signal goes high, the DSP CPU fetches its reset vector and starts running.

4.9 EHPI Affected by Certain Idle Configurations

To reduce power consumption, you might want to turn off certain idle domains within the DSP (for details, see Chapter 7, *Idle Configurations*). The EHPI does not belong to any of the idle domains, but it is affected by the clock generator (CLKGEN) and DMA idle domains. If the CLKGEN domain is idle, the host cannot access the DSP memory. If the DMA domain is idle, the EHPI cannot make any accesses that depend on the DMA controller.

4.10 EHPI Emulation Mode

The FREE bit of the DMA controller determines how the DMA controller reacts to a breakpoint in the debugger software:

- If FREE = 0 (the reset value), a breakpoint suspends DMA transfers. If the EHPI is using the DMA controller at the time, the suspension interrupts the EHPI data transfer.
- If FREE = 1, DMA transfers are not interrupted by a breakpoint.

4.11 EHPI and DSP Reset

If you drive the DSP reset signal low, the DSP undergoes a reset. Table 4–7 describes the effects of a DSP reset on the EHPI registers, and the resulting effects on the DSP. The EHPI registers are described in section 4.12 on page 4-23.

While the reset signal is low, if the RST_MODE signal is high, the EHPI can access the internal SARAM of the DSP. This time could be used to move program code to the SARAM. For more details, see section 4.8 on page 4-20.

Table 4–7. Affect of a DSP Reset on the EHPI

Register	Reset Value	Effect on the DSP
HPIC	0000h	XADD = 0: If the multiplexed mode is selected (HMODE signal is low), host address writes go to the 16 LSBs of the address register, HPIA(15–0). RESET = 0: If EHPI is enabled, the DSP can be held in the reset state until RESET is changed to 1.
HPIA	Not affected	
HPID	Not affected	

4.12 EHPI Registers

The enhanced host port interface (EHPI) contains three registers (see the following list) that a host processor can use to access the memory of the DSP. The registers share a data bus; therefore, the host must drive the signals HCNTL1 and/or HCNTL0 to the appropriate levels to indicate which EHPI register host is to access. The DSP can neither read from nor write to these registers.

- EHPI data register (HPID). This 16-bit register acts as a temporary holding place for data to be transferred through the EHPI. HPID contains the data that was read from the DSP memory if the current access is a read, or the data that will be written to the DSP memory if the current access is a write.
- EHPI address register (HPIA). In the multiplexed mode of the EHPI (see page 4-12), this 20-bit register acts as a temporary holding place for a 16- or 20-bit address for a read or write operation. In the nonmultiplexed mode of the EHPI (see page 4-7), HPIA is not needed because the address is directly available on input signals HA[19:0].
- EHPI control register (HPIC). HPIC provides important options for controlling data transfers. The fields of HPIC are shown in Figure 4–5 and described in Table 4–8.

Figure 4–5. EHPI Control Register (HPIC)

HPIC

15–6	5	4–2	1	0
Reserved	XADD	Reserved	DSPINT	RESET
	H_R/W – 0		H_W – 0	H_W – 0

Legend:

H_W	Write-only access for host. DSP cannot access HPIC.
H_R/W	Read/write access for host. DSP cannot access HPIC.
– X	X is the value after a DSP reset.

Table 4–8. HPIC Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–6	Reserved	These bits are not available for your use.	–
5	XADD	Extended address enable bit. In the multiplexed mode of the EHPI, the EHPI address register (HPIA) is loaded via the 16-bit data bus, HD[15:0]. When a 20-bit address is used, HPIA must be loaded with two transfers across HD[15:0]. To load bits 15–0 of HPIA, clear XADD beforehand. To load bits 19–16, set XADD beforehand. 0 Values written to HPIA go to HPIA(15–0). 1 Values written to HPIA go to HPIA(19–16). In the nonmultiplexed mode of the EHPI, XADD is ignored because HPIA is not used; instead, the address comes directly from the 20-bit address bus, HA[19:0].	0
4–2	Reserved	These bits are not available for your use.	–
1	DSPINT	Host-to-DSP interrupt request bit. The host can send a maskable interrupt request to the DSP CPU by writing a 1 to DSPINT. If the interrupt is properly enabled, the CPU will respond to the interrupt request; otherwise, the CPU will ignore it. 0 Clear DSPINT. 1 Send an interrupt request to the DSP CPU.	0

Table 4–8. HPIC Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
0	RESET	Reset exit bit. If RST_MODE signal low: The RESET bit is ignored. When the DSP reset signal is asserted, the DSP CPU starts running immediately after the reset signal goes high. If RST_MODE signal high: The RESET bit allows the host to prolong the DSP reset process. When the DSP reset signal is asserted at the pin, the RESET bit is cleared and remains 0 after the reset signal returns to the high level. The DSP CPU does not start running until the host sets the RESET bit. During the time when the DSP reset signal is high and the RESET bit is 0, the host can access the single-access RAM (SARAM) inside the DSP. This is a good time for the host to download code to the SARAM. When the download is complete, the host can set the RESET bit to start the CPU. 0 Clear RESET. 1 Start the DSP./Stop holding the DSP in reset. **Note:** If the host clears RESET and then sets RESET while the CPU is running, the CPU fetches the reset vector and switches execution to the reset interrupt service routine.	0

Chapter 5

External Memory Interface (EMIF)

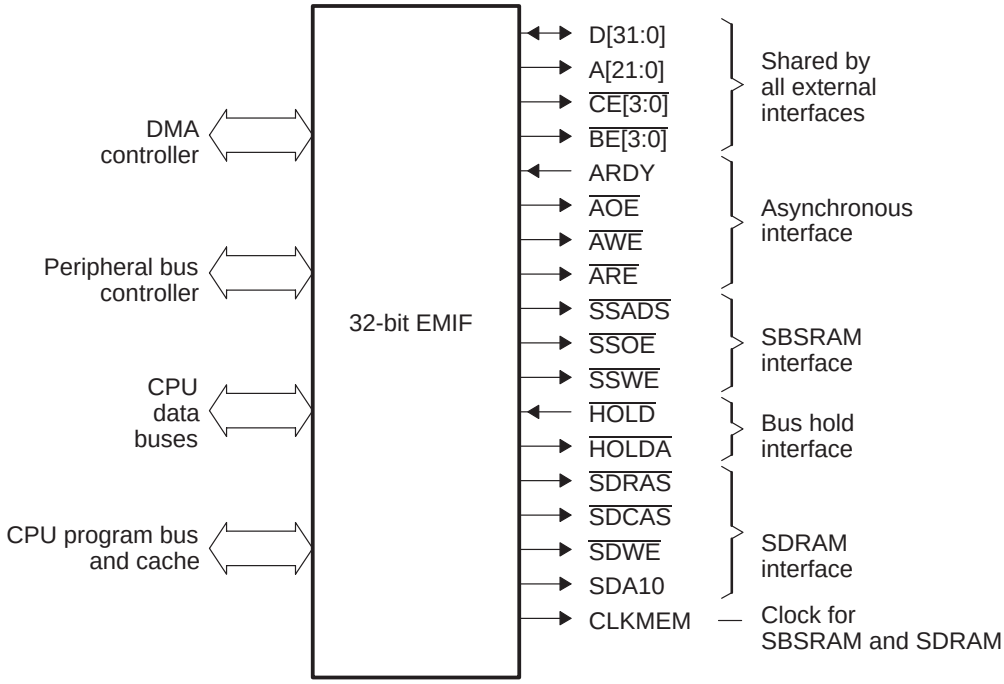
This chapter describes the external memory interface (EMIF) of the TMS320C55x™ DSP. The EMIF controls all data transfers between the C55x™ DSP and external memory.

Topic **Page**

5.1 Introduction to the EMIF

Figure 5–1 illustrates how the EMIF is interconnected with other parts of the DSP and with external memory devices. The connection to the peripheral bus controller allows the CPU to access the EMIF registers.

Figure 5–1. Diagram of EMIF Inputs and Outputs



The EMIF provides a glueless interface to three types of memory devices:

- Asynchronous devices, including ROM, flash memory, and asynchronous SRAM. For details on using asynchronous memory, see section 5.7 on page 5-27.
- Synchronous burst SRAM (SBSRAM) running at 1/2 or 1 times the CPU clock rate. For details on using SBSRAM, see section 5.8 on page 5-36.
- Synchronous DRAM (SDRAM) running at either 1/2 or 1 times the CPU clock rate. For details on using SDRAM, see section 5.9 on page 5-41.

The EMIF supports the following types of accesses:

- Program accesses (see page 5-13)
- 32-bit data accesses (see page 5-16)
- 16-bit data accesses (see page 5-21)
- 8-bit data accesses (see page 5-23)

To see what parts of the DSP can send external-memory requests to the EMIF and the order in which the EMIF services simultaneous requests, see *EMIF Request Priorities* on page 5-8.

If you want the DSP to share memory chips with an external device, see *HOLD Requests* on page 5-41.

If you would like to buffer CPU write operations to reduce delays, see *Write Posting* on page 5-42.

5.2 EMIF Signals

Table 5–1 provides a summary of the signals. In the Type column, I = Input, O = Output, and Z = High impedance state.

Related topics:

- *Signal Connections for External Asynchronous Memory* on page 5-27
- *Signal Connections for External SBSRAM* on page 5-36
- *HOLD Requests* on page 5-41

Table 5–1. EMIF Signals

Signal(s)	Type	Description
D[31:0]	I/O/Z	32-bit EMIF data bus The EMIF drives 32-bit, 16-bit, or 8-bit data on these pins. The pins you need to connect to the memory chip depend on the type of access and the width of the memory. For more details, see the following topics: *Program Accesses* on page 5-13 *32-Bit Data Accesses* on page 5-16 *16-Bit Data Accesses* on page 5-21 *8-Bit Data Accesses* on page 5-23 D is placed in the high impedance state when D is not carrying data or when the EMIF acknowledges a HOLD request from an external device.
A[21:0]	O/Z	22-bit EMIF address bus **A[21:0] used for asynchronous memory:** Which of these pins you connect to the asynchronous memory chip depends on the width of the memory (8-bit, 16-bit, or 32-bit). **A[21:0] used for SBSRAM:** The SBSRAM is assumed to be 32 bits wide. You use pins A[(N+2):2], where N is the number of the most significant address pin of the SBSRAM. The A pins are placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
CE0_ CE1_ CE2_ CE3_	O/Z	Chip enable pins, one for each CE space Connect these active-low pins to the chip select pins of the appropriate memory chips. For example, if an SBSRAM chip is to be in space CE2, connect CE2_ to the chip select pin of that SBSRAM chip. When the EMIF makes an access in space CE2, that particular chip is enabled. The CE_ pins are each placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.

Table 5–1. EMIF Signals (Continued)

Signal(s)	Type	Description
BE[3:0]_	O/Z	Byte enable pins The EMIF drives signal combinations on these active-low pins to indicate the size of the data being accessed. In some cases, the signal combination also indicates which portion of the EMIF data bus, D[31:0], carries the data. More details are in the following sections: *Program Accesses* on page 5-13 *32-Bit Data Accesses* on page 5-16 *16-Bit Data Accesses* on page 5-21 *8-Bit Data Accesses* on page 5-23 The BE_ pins are each placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
ARDY	I	Asynchronous ready pin For details, see *Inserting Extra Cycles with the Ready (ARDY) Signal* on page 5-35.
AOE_	O/Z	Asynchronous output enable pin For details, see *Signal Connections for External Asynchronous Memory* on page 5-27. AOE_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
AWE_	O/Z	Asynchronous write strobe pin For details, see *Signal Connections for External Asynchronous Memory* on page 5-27. AWE_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
ARE_	O/Z	Asynchronous read strobe pin For details, see *Signal Connections for External Asynchronous Memory* on page 5-27. ARE_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
SSADS_	O/Z	Address strobe/enable pin for SBSRAM For details, see *Signal Connections for External SBSRAM* on page 5-36. SSADS_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.

Table 5–1. EMIF Signals (Continued)

Signal(s)	Type	Description
SSOE_	O/Z	Output buffer enable pin for SBSRAM For details, see *Signal Connections for External SBSRAM* on page 5-36. SSOE_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
SSWE_	O/Z	Write enable pin for SBSRAM For details, see *Signal Connections for External SBSRAM* on page 5-36. SSWE_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
SDRAS_	O/Z	Row strobe pin for SDRAM SDRAS_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
SDCAS_	O/Z	Column strobe pin for SDRAM SDCAS_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
SDWE_	O/Z	Write enable pin for SDRAM SDWE_ is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
SDA10	O/Z	A10 address line/autoprecharge disable for SDRAM This pin serves as a row address bit (logically equivalent to A12) during ACTV commands and also disables the autoprecharging function of the SDRAM during read or write operations. SDA10 is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.
CLKMEM	O/Z	Memory clock pin for SBSRAM and SDRAM This pin is locked high or reflects the memory clock, depending on the value of the MEMCEN bit. If the pin reflects the clock, the frequency of the clock (equal to or 1/2 the frequency of the CPU clock) depends on the MEMFREQ bits. MEMCEN and MEMFREQ are in the EMIF global control register, which is described beginning on page 5-43. CLKMEM is placed in the high impedance state when the EMIF acknowledges a HOLD request from an external device.

Table 5–1. EMIF Signals (Continued)

Signal(s)	Type	Description
HOLD_	I	HOLD request pin To request the DSP to release control of external memory chips, an external device can drive this active-low signal. For details, see *HOLD Requests* on page 5-41.
HOLDA_	O	HOLD acknowledge pin When the EMIF receives a HOLD request from an external device (on the HOLD_ pin), the EMIF completes any current activity. Then it places the external bus pins in the high impedance state and sends acknowledgement on the HOLDA_ pin. The external device should wait until HOLDA_ is driven low before accessing external memory chips. For details, see *HOLD Requests* on page 5-41.

5.3 EMIF Request Priorities

The EMIF services the requests shown in Table 5–2. If multiple requests arrive simultaneously, the EMIF prioritizes them as shown in the Priority column.

Table 5–2. EMIF Request Priorities

EMIF Request Type	Priority	Description
HOLD	1 (highest)	A request made by an external device to take control of external memory. This request is initiated when the external device drives the HOLD_ pin low. For details, see *HOLD Requests* on page 5-41.
Urgent refresh	2	A request from synchronous DRAM that needs an immediate refresh.
E bus	3	A request from the E bus to write to external memory. The E bus is one of the data-write data buses of the DSP CPU.
F bus	4	A request from the F bus to write to external memory. The F bus is one of the data-write data buses of the DSP CPU.
D bus	5	A request from the D bus to read from external memory. The D bus is one of the data-read data buses of the DSP CPU.
C bus	6	A request from the C bus to read from external memory. The C bus is one of the data-read data buses of the DSP CPU.
P bus	7	A request from the P bus to read code from external memory. The P bus is the program-read program bus of the DSP CPU.
Cache	8	A line fill request from the DSP instruction cache.
DMA controller	9	A request from the DSP DMA controller to read from or write to external memory.
Trickle refresh	10 (lowest)	A request from synchronous DRAM that needs the next periodic refresh.

5.4 Memory Considerations

When programming the EMIF, you must understand how the external memory addresses are divided into chip enable (CE) spaces (see section 5.4.1), what type of memory can be attached in each CE space (see section 5.4.2), and what register bits are used to configure the CE spaces (see section 5.4.3).

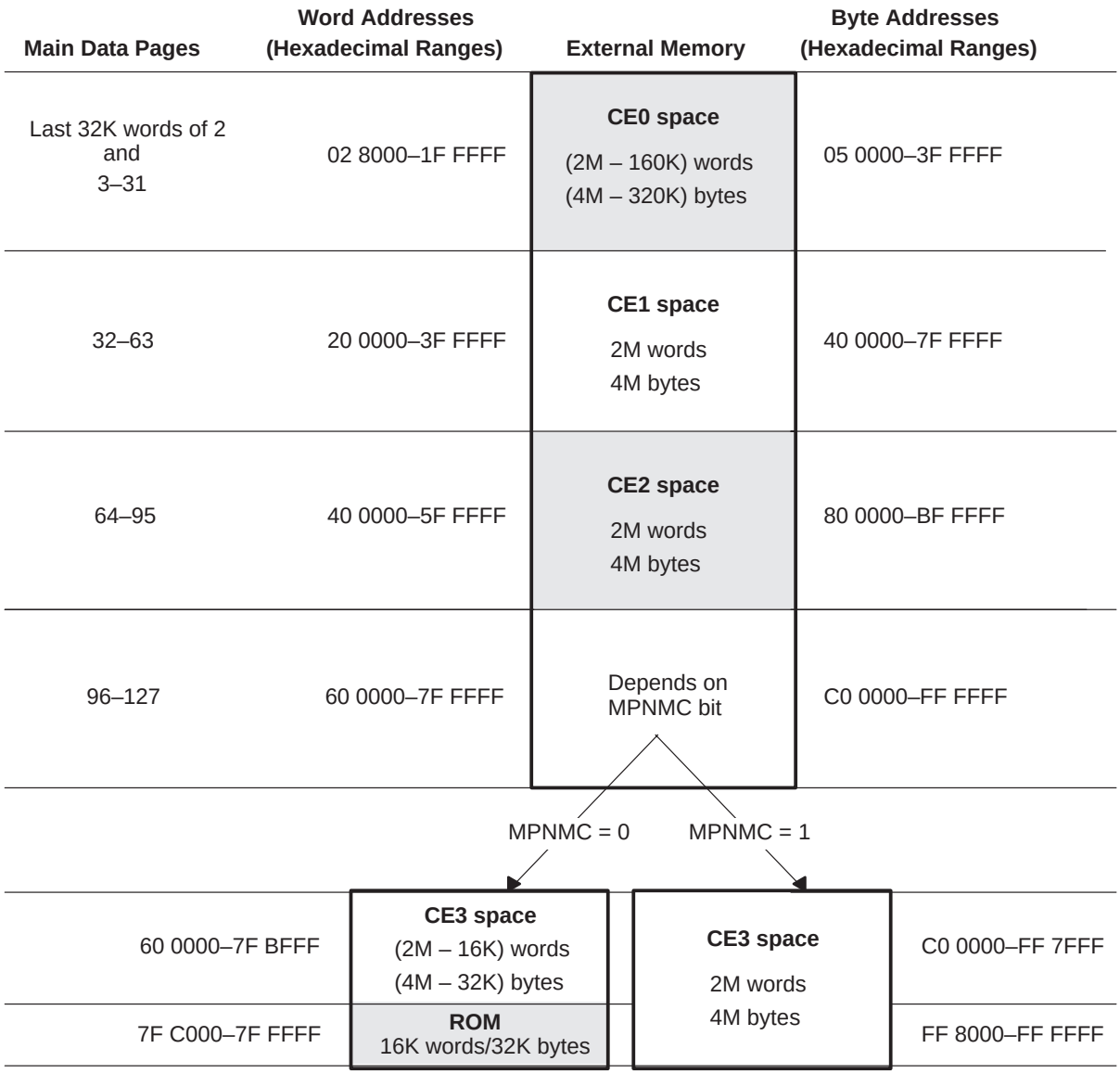
5.4.1 Memory Map and CE Spaces

The portion of the TMS320C55x memory map that is available for external memory is divided into spaces that correspond to the chip enable signals of the EMIIF. For example, a memory chip in the CE1 space must have its chip select pin connected to the CE1_ pin of the EMIF. When the EMIF performs an access in the CE1 space, it drives CE1_ low. To illustrate the concept of the CE spaces, Figure 5–2 shows the external memory map of the TMS320VC5510 DSP.

Notice that both word addresses and byte addresses are given in the figure. When the TMS320C55x CPU is accessing data, it uses 23-bit addresses to access words (16-bit values). The 7 most significant bits of the 23-bit address specify one of the main data pages (0 through 127), each of which has 64K addresses. When the CPU is accessing program code, it uses 24-bit addresses to access bytes (8-bit values). When the DMA controller accesses memory, it always uses byte addresses.

The highest addresses in the TMS320VC5510 external memory map can be assigned solely as the CE3 space or can be shared by memory in the CE3 space and the internal DSP ROM. The assignment of these addresses is determined by the MPNMC bit in CPU status register ST3_55. You can change the bit any time, but during a DSP reset, the value in MPNMC depends on the signal level on the MP/MC_ pin of the DSP. If the signal is low, MPNMC is cleared (MPNMC = 0); if the signal is high, MPNMC is set (MPNMC = 1).

Figure 5–2. TMS320VC5510 External Memory Address Map



5.4.2 Supported Memory and Access Types

Each CE space of the external memory map can use any one of the memory types listed in the following table. You select the memory types by writing to each CE space's MTYPE bits. In addition to the memory types, Table 5–3 shows the types of accesses the EMIF can perform for each memory type.

For details on how the EMIF performs the types of accesses, see the following topics:

- *Program Accesses* on page 5-13
- *32-Bit Data Accesses* on page 5-16
- *16-Bit Data Accesses* on page 5-21
- *8-Bit Data Accesses* on page 5-23

Table 5–3. Available Memory Types and the Allowable Access Types For Each

Memory Type	Allowable Access Types
Asynchronous, 8 bits wide (MTYPE = 000b)	Program
Asynchronous, 16 bits wide (MTYPE = 001b)	Program 32-bit data 16-bit data 8-bit data
Asynchronous, 32 bits wide (MTYPE = 010b)	Program 32-bit data 16-bit data 8-bit data
SDRAM, 32 bits wide (MTYPE = 011b)	Program 32-bit data 16-bit data 8-bit data
SBSRAM, 32 bits wide (MTYPE = 100b)	Program 32-bit data 16-bit data 8-bit data

Table 5–4 shows one possible MTYPE configuration for the CE spaces:

Table 5–4. One Possible MTYPE Configuration for the CE Spaces

CE Space	Memory Type
CE0	MTYPE = 100b: 32-bit-wide SBSRAM
CE1	MTYPE = 001b: 16-bit-wide asynchronous memory
CE2	MTYPE = 001b: 16-bit-wide asynchronous memory
CE3	MTYPE = 011b: 32-bit-wide SDRAM

5.4.3 Configuring the CE Spaces

You configure the CE spaces by using the global control register (EGCR) and the CE space control registers (three for each CE space). EGCR is described beginning on page 5-43. For a description of the CE space registers, see page 5-50.

For each CE space, you must load the following bit field in control register 1:

MTYPE Specifies the memory type

If you choose an **asynchronous memory type**, you must initialize access parameters in the other bits of the CE space control registers (see *Configuring the EMIF for Asynchronous Accesses* on page 5-28).

If you choose a **synchronous memory type**, MTYPE is the only field you need to initialize in the CE space control registers. However, you must load two fields in the global control register:

MEMFREQ Determines the frequency of the memory clock signal (1 or 1/2 times the frequency of the CPU clock signal)

MEMCEN Determines whether the signal on the CLKMEM pin reflects the memory clock signal or is held high

Regardless of the memory type in each CE space, make sure you write to the following control bits in the global control register. These bits affect all CE spaces as a group:

WPE Enables or disables write posting for all CE spaces

NOHOLD Enables or disables HOLD requests for all CE spaces

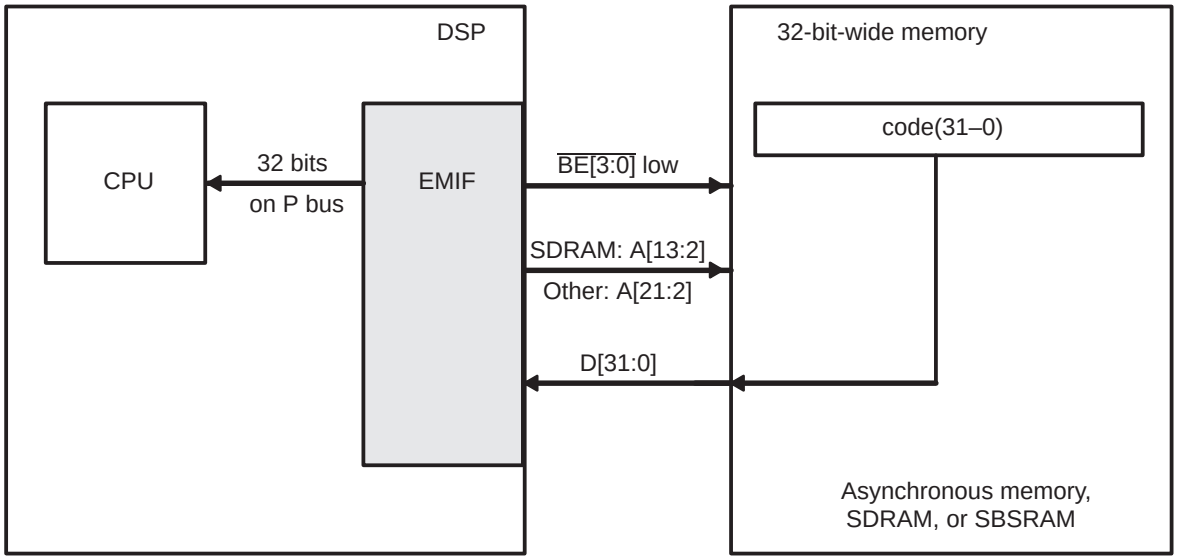
5.5 Program Accesses

When fetching instruction code from external memory, the CPU sends an access request to the EMIF. The EMIF must read 32 bits from the external memory and then pass all 32 bits to the program-read data bus (P bus) of the CPU. The EMIF can manage the 32-bit access for three memory widths: 32 bits, 16 bits, and 8 bits.

5.5.1 Program Access of 32-Bit-Wide Memory

Figure 5–3 shows how the EMIF behaves when reading program code from 32-bit-wide external memory. The least significant line of the external address bus that is required by 32-bit-wide memory is A2. The external address lines A[21:2] of the EMIF correspond to bits 21–2 of the internal program address. The whole external data bus, D[31:0], is used to transport the data from external memory to the DSP. During an access, the EMIF drives low all four of the byte enable signals, BE[3:0]_. After the access, the EMIF passes all 32 bits to the P bus, which then carries them to the CPU.

Figure 5–3. Program Access of 32-Bit-Wide External Memory



5.5.2 Program Access of 16-Bit-Wide Memory

Figure 5–4 illustrates the data transfers involved in a program access of 16-bit-wide external memory. The EMIF places a word address on address lines A[21:1]. The 32-bit access is performed as two 16-bit transfers, in two consecutive cycles. During the second cycle, the EMIF automatically increments the first address by 1 to create the second address value.

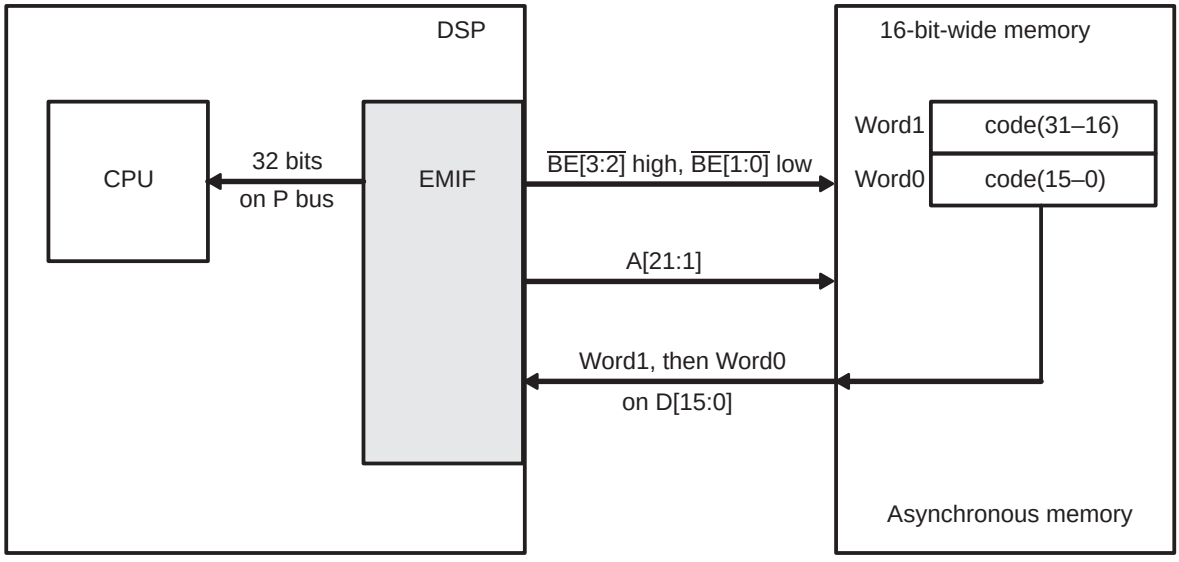
For both 16-bit accesses, the EMIF uses data lines D[15:0]. The 32-bit code block is transferred in the following manner:

1) Bits 31 through 16 of the code block are read at the first address.

2) Bits 15 through 0 are read at the second address.

During an access, BE3_ and BE2_ stay high (inactive), and BE1_ and BE0_ are driven low. After an access, the EMIF passes all 32 bits of the code to the P bus of the CPU.

Figure 5–4. Program Access of 16-Bit-Wide External Memory



5.5.3 Program Access of 8-Bit-Wide Memory

As shown in Figure 5–5, when the EMIF performs a program access of 8-bit-wide memory, the EMIF places a byte address on address lines A[21:0]. The 32-bit access is performed as four 8-bit transfers, in four consecutive cycles. During the second, third, and fourth cycles, the EMIF automatically generates a new address by incrementing the previous address by 1.

For all four 8-bit accesses, the EMIF uses data lines D[7:0]. As shown in the figure, the 32-bit code block is transferred in the following manner:

1) Bits 31 through 24 of the code block are read at the first address.

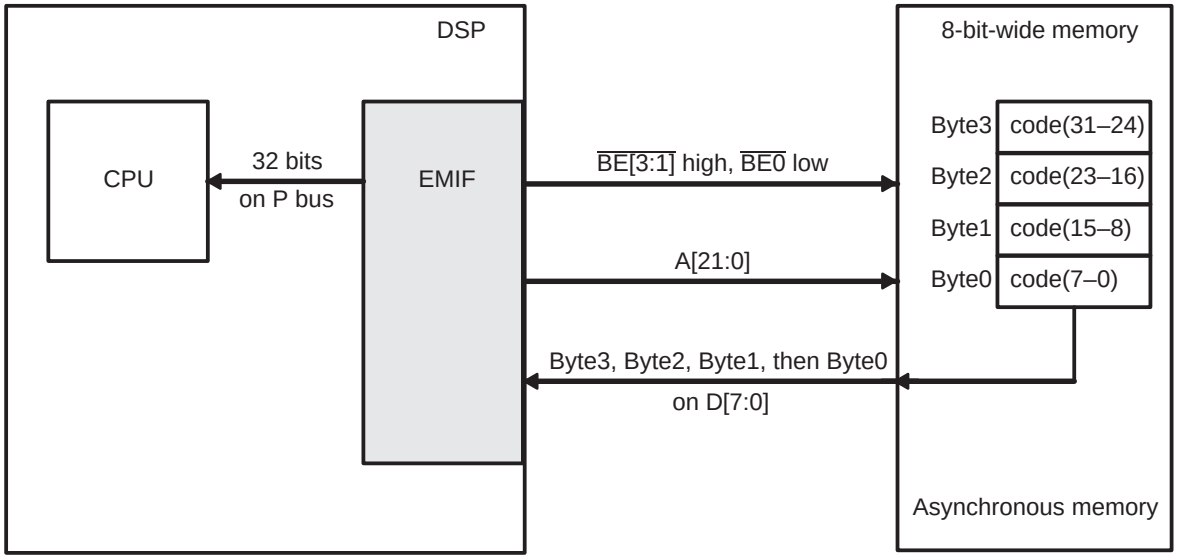
2) Bits 23 through 16 are read at the second address.

3) Bits 15 through 8 are read at the third address.

4) Bits 7 through 0 are read at the fourth address.

During an access, BE3_, BE2_, and BE1_ stay high (inactive), and BE0_ is driven low. After an access, the EMIF passes all 32 bits of the code to the P bus of the CPU.

Figure 5–5. Program Access of 8-Bit-Wide External Memory



5.6 Data Accesses

The EMIF supports data accesses for:

- ❏ 32-bit data (see section 5.6.1)
- ❏ 16-bit data (see section 5.6.2 on page 5-21)
- ❏ 8-bit data (see section 5.6.3 on page 5-23)

5.6.1 32-Bit Data Accesses

A 32-bit data access is generated by a CPU instruction or a DMA controller operation that reads or writes a 32-bit value. For each 32-bit read or write operation, the EMIF must communicate with two 16-bit CPU buses. For read operations, the C and D buses carry the data from the EMIF to the CPU. For write operations, the E and F buses carry the data from the CPU to the EMIF. The EMIF can manage the 32-bit access for two memory widths: 32 bits and 16 bits.

5.6.1.1 32-Bit Data Access of 32-Bit-Wide Memory

The process for accessing 32-bit data from 32-bit-wide external memory is illustrated in the two parts of Figure 5–6. The least significant line of the external address bus that is required by 32-bit-wide memory is A2. The external address lines A[21:2] of the EMIF correspond to bits 21–2 of the internal data address. The whole external data bus, D[31:0], is used to transport the data between the DSP and the external memory. All four of the byte enable signals, BE[3:0]_, are forced low (active) during an access.

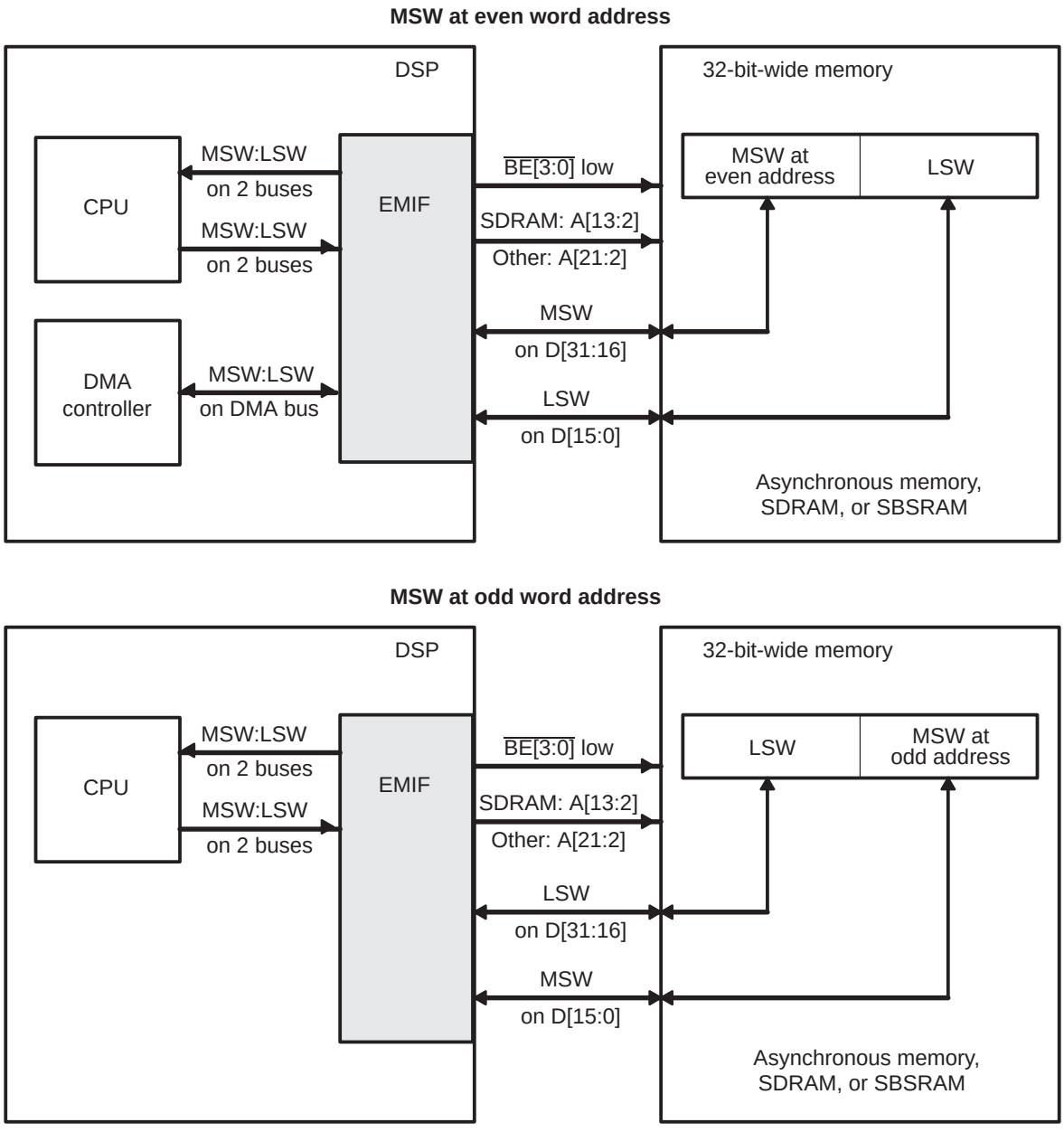
Although only bits 21–2 of the internal address are used by the external memory chip, the EMIF uses bit 1 to determine the positions of the most significant word (MSW) and least significant word (LSW) of the 32-bit value (see Table 5–5). The relative positions of the MSW and LSW determine how the EMIF data lines are used. D[31:16] carry the word to/from the even word address. Lines D[15:0] carry the word to/from the odd word address.

If the DMA controller is to make a 32-bit data access, the MSW must be at an even address.

Table 5–5. The Role of Internal Address Bit 1 During a 32-Bit Data Access of 32-Bit-Wide External Memory

Internal Address Bit 1	MSW and LSW Positions	Use of D[31:0]
0	The MSW is at an even word address, and the LSW is at the following odd word address.	D[31:16] carry the MSW. D[15:0] carry the LSW.
1	The MSW is at an odd word address, and the LSW is at the previous even word address.	D[31:16] carry the LSW. D[15:0] carry the MSW.

Figure 5–6. 32-Bit Data Access of 32-Bit-Wide External Memory



5.6.1.2 32-Bit Data Access of 16-Bit-Wide Memory

The two parts of Figure 5–7 illustrate the data transfers involved in a 32-bit data access of 16-bit-wide external memory. The least significant address line required by 16-bit-wide memory is A1. The external address lines A[21:1] of the EMIF correspond to bits 21–1 of the internal data address. Data bus lines D[15:0] are used to transport the data between the DSP and the external memory. During an access, BE3_ and BE2_ stay high (inactive), and BE1_ and BE0_ are driven low.

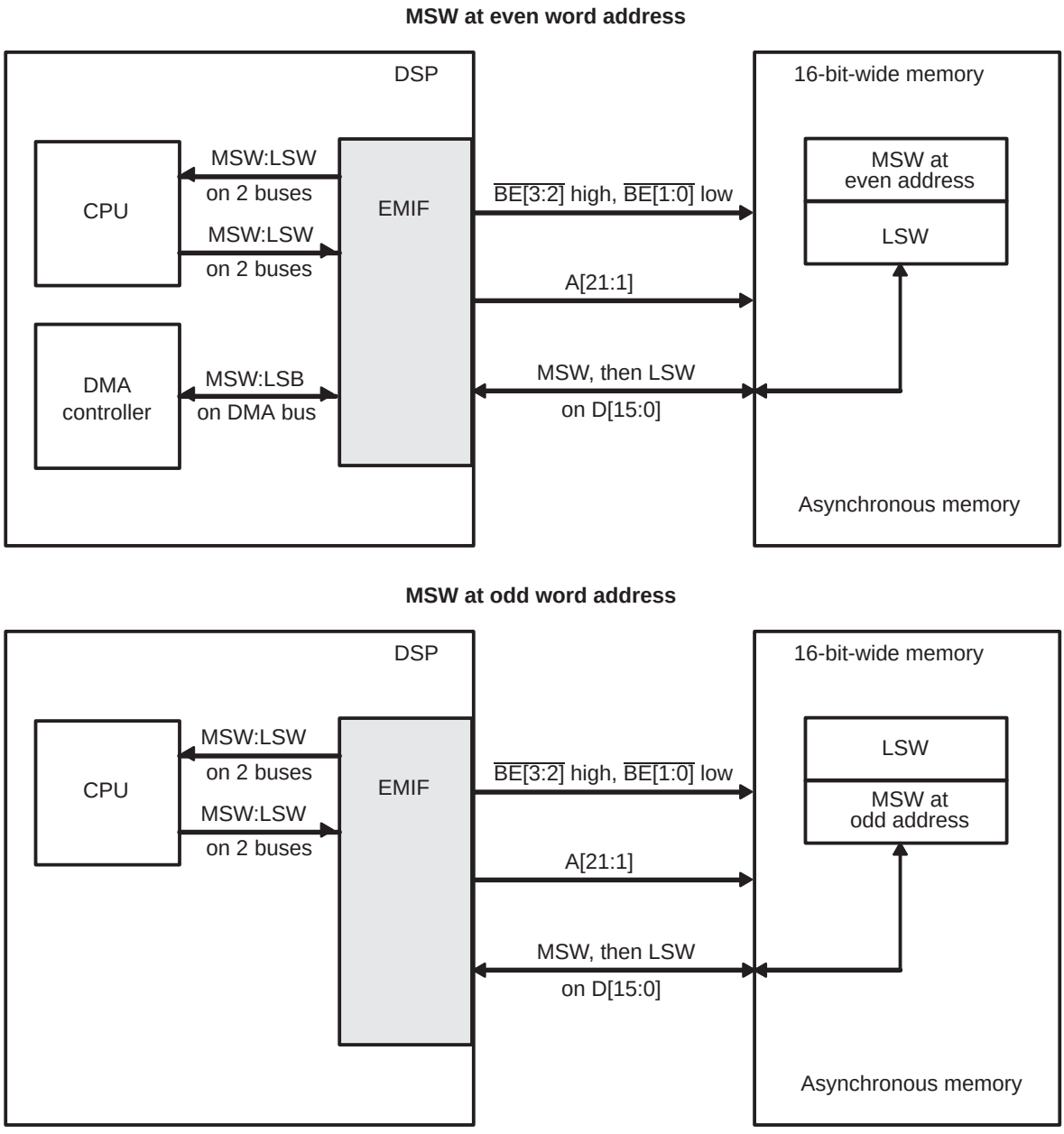
The 32-bit access is performed as two 16-bit transfers across data bus lines D[15:0]. The transfers are performed in two consecutive cycles. During the second cycle, the EMIF automatically generates the second address as described in Table 5–6.

If the DMA controller is to make a 32-bit data access, the MSW must be at an even address.

Table 5–6. The Role of Internal Address Bit 1 During a 32-Bit Data Access of 16-Bit-Wide External Memory

Internal Address Bit 1	MSW and LSW Positions	Generation of LSW Address
0	The MSW is at an even word address, and the LSW is at the following odd word address.	After the first access, the EMIF adds 1 to the MSW address to generate the LSW address.
1	The MSW is at an odd word address, and the LSW is at the previous even word address.	After the first access, the EMIF subtracts 1 from the MSW address to generate the LSW address.

Figure 5–7. 32-Bit Data Access of 16-Bit-Wide External Memory



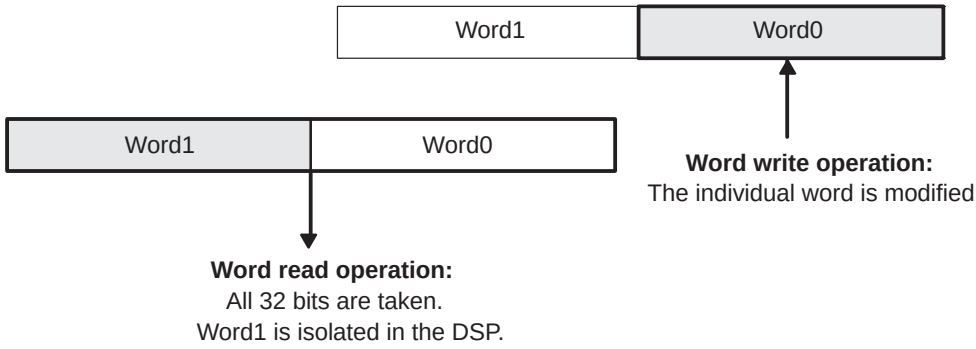
5.6.2 16-Bit Data Accesses

A 16-bit data access is generated by a CPU instruction or a DMA controller operation that reads or writes a 16-bit value. For CPU operations, the D bus carries read data, and the E bus carries write data. The EMIF manages the 16-bit access for two memory widths: 32 bits and 16 bits.

5.6.2.1 16-Bit Data Access of 32-Bit-Wide Memory

As shown in Figure 5–8, when the EMIF makes 16-bit accesses in 32-bit-wide external memory, the actual width of the access is different for read operations and write operations. When a word is written to external memory, the EMIF modifies an individual word. However, when the EMIF reads a word from external memory, the EMIF reads the full width of the memory, and the desired word is isolated in the DSP.

Figure 5–8. Accessing 16-Bit Data in 32-Bit-Wide Memory

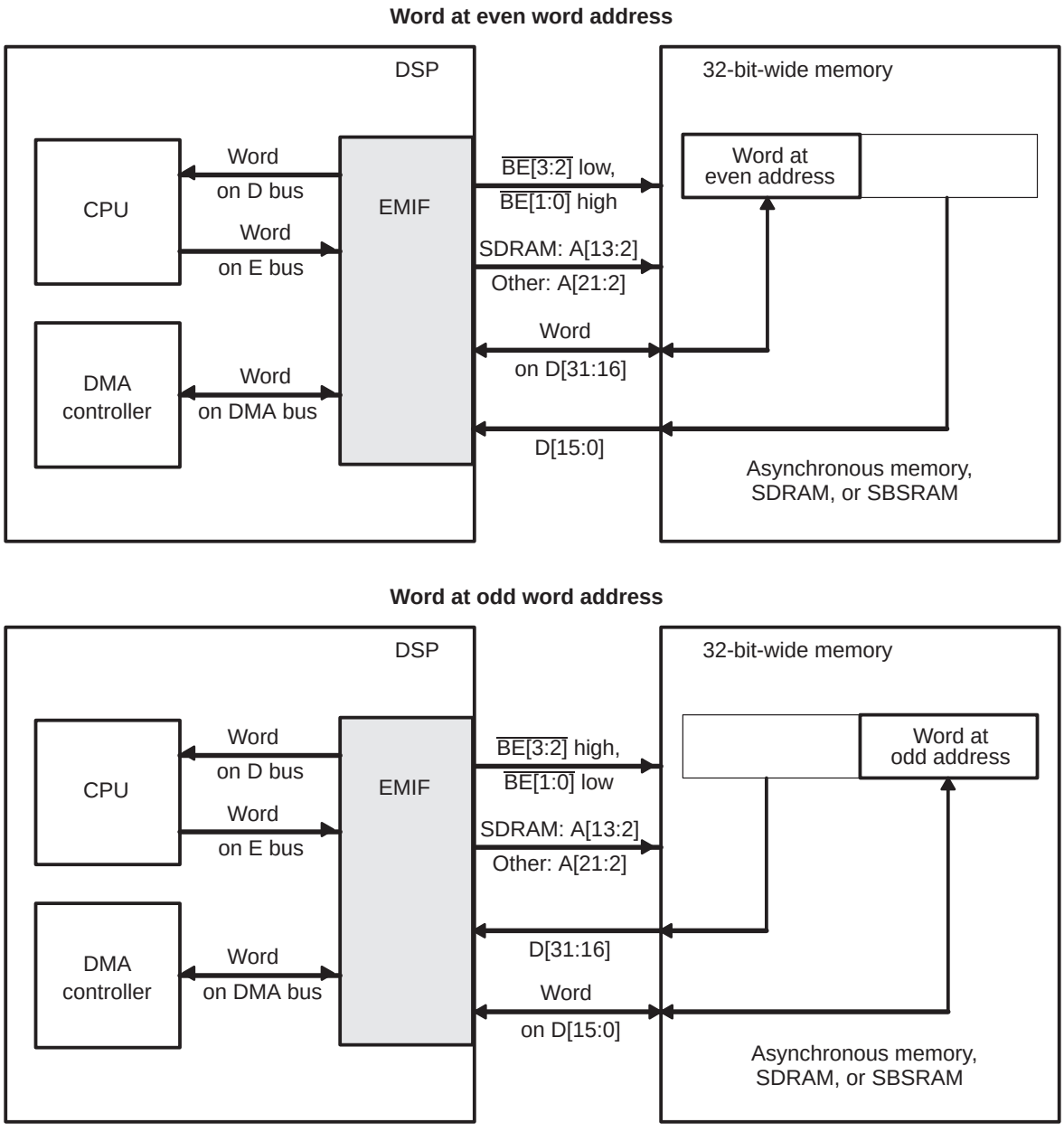


The process for accessing 16-bit data from 32-bit-wide external memory is illustrated in the two parts of Figure 5–9. The least significant address line required by 32-bit-wide memory is A2. The external address lines A[21:2] of the EMIF correspond to bits 21–2 of the internal data address. The EMIF uses bit 1 of the internal address to determine which half of the data bus is used and which byte enable signals are active (see Table 5–7).

Table 5–7. The Role of Internal Address Bit 1 During a 16-Bit Data Access of 32-Bit-Wide External Memory

Internal Address Bit 1	Word Is At an …	Data Lines Used	Byte Enable Signal Levels
0	Even word address	D[31:16]	BE[3:2]_ low (active) BE[1:0]_ high
1	Odd word address	D[15:0]	BE[3:2]_ high BE[1:0]_ low (active)

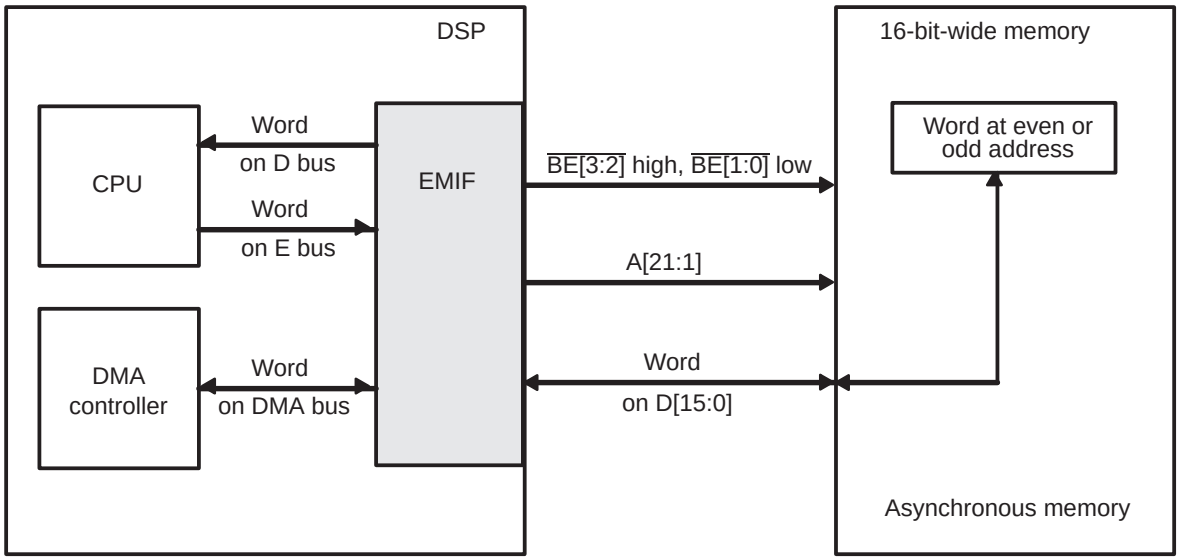
Figure 5–9. 16-Bit Data Accesses of 32-Bit-Wide External Memory



5.6.2.2 16-Bit Data Access of 16-Bit-Wide Memory

Figure 5–10 illustrates the data transfers involved in a 16-bit data access of 16-bit-wide external memory. The least significant address line required by 16-bit-wide memory is A1. The external address lines A[21:1] of the EMIF correspond to bits 21–1 of the internal data address. Data bus lines D[15:0] are used to transport the data between the DSP and the external memory. During an access, BE3_ and BE2_ stay high (inactive), and BE1_ and BE0_ are driven low.

Figure 5–10. 16-Bit Data Accesses of 16-Bit-Wide External Memory



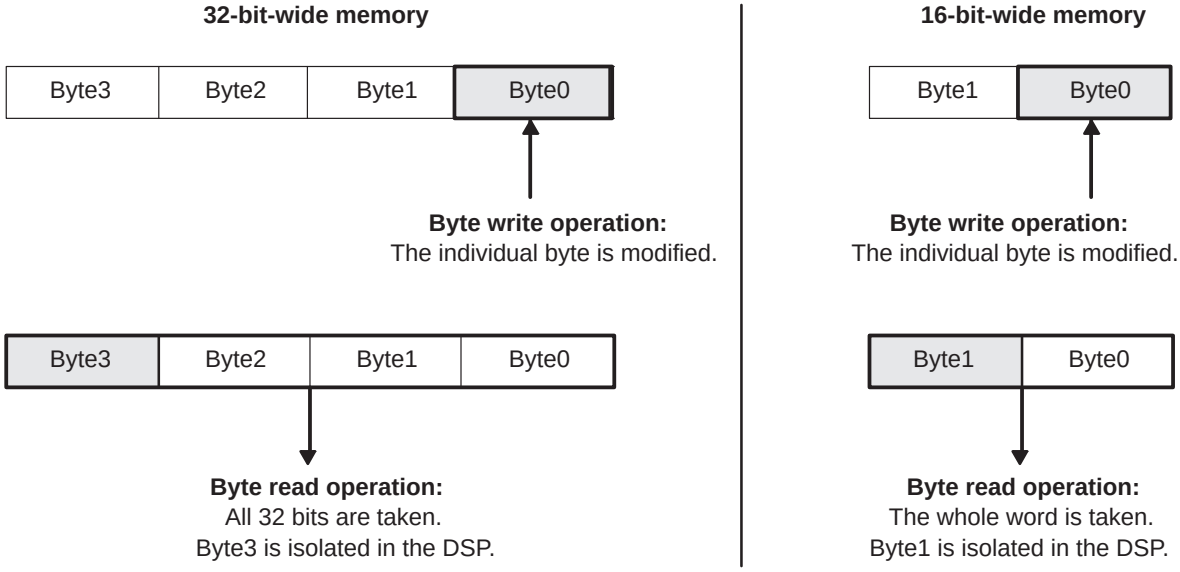
5.6.3 8-Bit Data Accesses

Some CPU instructions and DMA controller operations access 8-bit data (bytes). These byte accesses can be done in 16-bit-wide or 32-bit-wide memory. The CPU buses used are the D bus (for byte read operations) and the E bus (for byte write operations).

As shown in Figure 5–11, the actual width of the memory access is different for byte read operations and byte write operations. When a byte is written to external memory, the EMIF modifies an individual byte. However, when the EMIF reads a byte from external memory, the EMIF reads the full width of the memory, and the desired byte is isolated in the DSP.

The EMIF manages 8-bit accesses for two memory widths: 32 bits and 16 bits.

Figure 5–11. Accessing 8-Bit Data in 32-Bit-Wide and 16-Bit-Wide External Memory



5.6.3.1 8-Bit Data Access of 32-Bit-Wide Memory

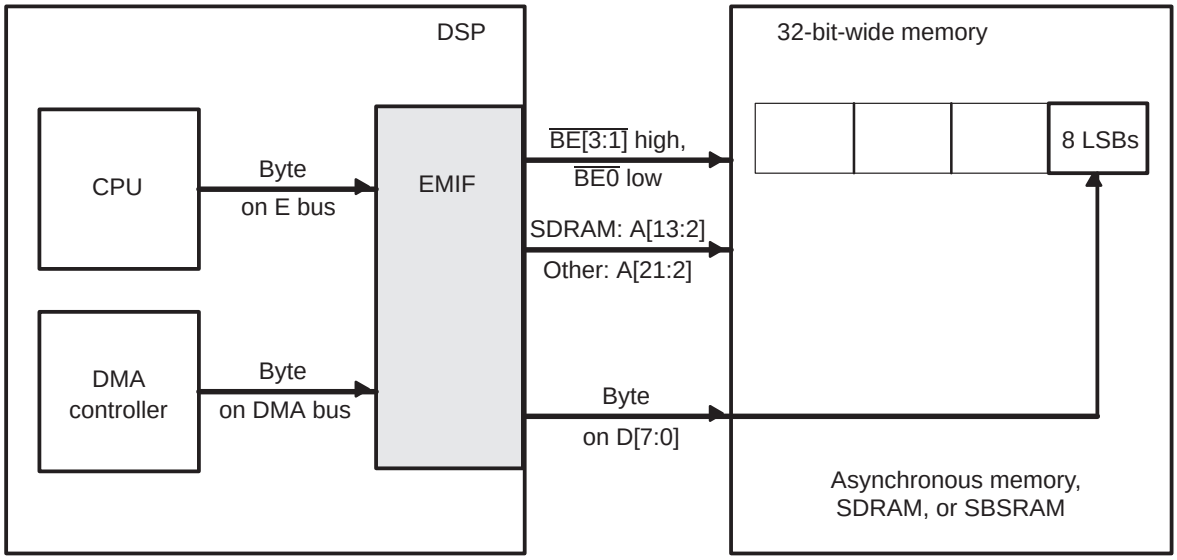
Reading 8-bit data. The EMIF reads 8-bit data from 32-bit-wide memory the same way that it reads 16-bit data from 32-bit-wide memory (see *16-Bit Data Access of 32-Bit-Wide Memory* on page 5-21). The EMIF reads the full width of the memory, and the desired byte is isolated in the DSP.

Writing 8-bit data. The way the EMIF writes 8-bit data to 32-bit-wide external memory is illustrated by Figure 5–12. Specifically, this figure shows the CPU or the DMA controller modifying the 8 LSBs (bits 7–0) of a memory location. External address lines A[21:2] correspond to bits 21–2 of the internal data address. The EMIF uses bits 1 and 0 of the internal address to determine which byte is loaded, which data lines carry the data, and which byte enable signal is active (see Table 5–8).

Table 5–8. The Role of Internal Address Bits 1–0 During an 8-Bit Data Write to 32-Bit-Wide External Memory

Internal Address Bits 1–0	Bits Loaded At Memory Location	Data Lines Used	Byte Enable Signal Levels
00	31–24 (the 8 MSBs)	D[31:24]	BE3_ low (active), others high
01	23–16	D[23:16]	BE2_ low (active), others high
10	15–8	D[15:8]	BE1_ low (active), others high
11	7–0 (the 8 LSBs)	D[7:0]	BE0_ low (active), others high

Figure 5–12. Writing to the 8 LSBs of a 32-Bit-Wide External Memory Location



5.6.3.2 8-Bit Data Access of 16-Bit-Wide Memory

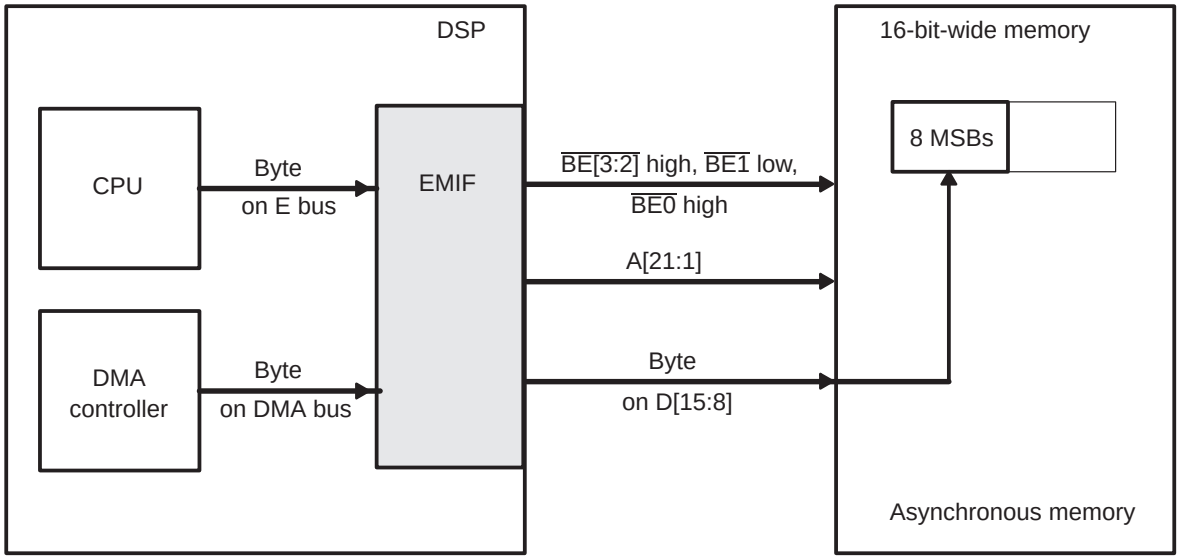
Reading 8-bit data. The EMIF reads 8-bit data from 16-bit-wide memory the same way that it reads 16-bit data from 16-bit-wide memory (see *16-Bit Data Access of 16-Bit-Wide Memory* on page 5-23). The EMIF reads the full width of the memory, and the desired byte is isolated in the DSP.

Writing 8-bit data. Figure 5–13 illustrates the data transfers involved in an 8-bit write operation that uses 16-bit-wide external memory. Specifically, the figure shows the CPU or the DMA controller modifying the 8 MSBs (bits 15–8) of a memory location. External address lines A[21:1] correspond to bits 21–1 of the internal data address. The EMIF uses bit 0 of the internal address to determine which byte is loaded, which data lines carry the data, and which byte enable signal is active (see Table 5–9).

Table 5–9. The Role of Internal Address Bit 0 During an 8-Bit Data Write to 16-Bit-Wide External Memory

Internal Address Bit 0	Bits Loaded At Memory Location	Data Lines Used	Byte Enable Signal Levels
0	15–8 (the 8 MSBs)	D[15:8]	BE1_ low (active), others high
1	7–0 (the 8 LSBs)	D[7:0]	BE0_ low (active), others high

Figure 5–13. 8-Bit Write Operation Using 16-Bit-Wide External Memory



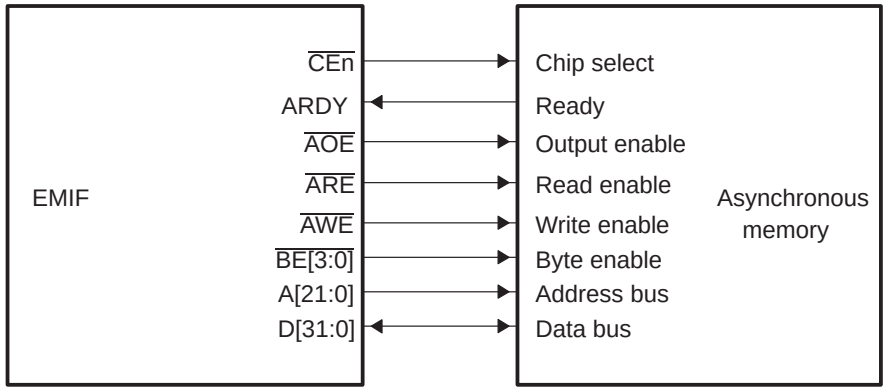
5.7 Using Asynchronous Memory

The EMIF offers configurable timing parameters so that you can interface the DSP to a variety of asynchronous memory types, including flash memory, SRAM, and EPROM. This section includes the following topics:

5.7.1 Signal Connections for External Asynchronous Memory

Figure 5–14 shows generalized connections between the EMIF and an asynchronous memory chip. Descriptions of the EMIF pins follow the figure.

Figure 5–14. EMIF Connected to an Asynchronous Memory Chip



CEn_ Chip enable pin for space CEn (n can be 0, 1, 2, ...). Connect this active-low pin to the chip select pin of the memory chip, so that when the EMIF makes an access in CE space n, the memory chip is enabled.

ARDY Asynchronous ready pin. The memory chip can drive this active-high signal low whenever it needs to delay the asynchronous accesses of the EMIF. For details, see *Inserting Extra Cycles with the Ready (ARDY) Signal* on page 5-35.

AOE_ Asynchronous output enable pin. Connect this active-low pin to the output enable pin of the memory chip.

ARE_ Asynchronous read strobe pin. Connect this active-low pin to the read enable pin of the memory chip. ARE_ defines the boundaries of a memory read access.

AWE_ Asynchronous write strobe pin. Connect this active-low pin to the write enable pin of the memory chip. AWE_ defines the boundaries of a memory write access.

BE[3:0]_ Byte enable pins. The EMIF drives signal combinations on these active-low pins to indicate the size of the data being accessed. In some cases, the signal combination also indicates which portion of the EMIF data bus, D[31:0], carries the data. For more details, see the following topics:
Program Accesses on page 5-13
32-Bit Data Accesses on page 5-16
16-Bit Data Accesses on page 5-21
8-Bit Data Accesses on page 5-23

A[21:0] Address bus pins. For 32-bit-wide memory, connect pins A[21:2] to the memory chip. For 16-bit-wide memory, connect pins A[21:1]. For 8-bit-wide memory, connect A[21:0].

D[31:0] Data bus pins. The EMIF drives 32-bit, 16-bit, or 8-bit data on these pins. The pins you need to connect to the memory chip depend on the type of access and the width of the memory. For more details, see the following topics:
Program Accesses on page 5-13
32-Bit Data Accesses on page 5-16
16-Bit Data Accesses on page 5-21
8-Bit Data Accesses on page 5-23

5.7.2 Configuring the EMIF for Asynchronous Accesses

The external memory is divided into CE spaces (see section 5.4.1 on page 5-9). For asynchronous accesses, first configure the CE spaces that must support asynchronous memory (see section 5.4.3 on page 5-12). For each CE space, program the parameters described in Table 5–10. Each CE space has control registers 1, 2, and 3 (see page 5-50), which together contain all the bit fields for the programmable parameters. These parameters are ignored unless the MTYPE bits in CE space control register 1 indicate asynchronous memory.

Note:

The SETUP and STROBE fields have a minimum count of 1, and because of this, a 0 in one of these fields is interpreted by the DSP as a 1. For the first access (even if there is only one), the setup period will have a minimum of 2. Table 5–11 shows examples to illustrate the constraints on the setup behavior.

Table 5–10. Parameters for an Access of External Asynchronous Memory

Parameters	Control Bits	Definition
Setup periods	READ SETUP WRITE SETUP	A setup period is the time in CPU clock cycles given to setup the address, chip enable (CE_), and byte enable (BE_) signals before the read strobe signal (ARE_) or write strobe signal (AWE_) falls. For an asynchronous read operation, this is also the setup period for the output enable signal (AOE_) before ARE_ falls.
Strobe periods	READ STROBE WRITE STROBE	A strobe period is the time in CPU clock cycles between the falling (activation) and rising (deactivation) of the read or write strobe signal.
Hold periods	READ HOLD WRITE HOLD	A hold period is the time in CPU clock cycles during which the address and byte enable lines are held active after the read or write strobe signal rises. For an asynchronous read operation, this is also the hold period for the output enable signal after ARE_ rises.

Table 5–10. Parameters for an Access of External Asynchronous Memory (Continued)

Parameters	Control Bits	Definition
Extended hold periods	READ EXT HOLD WRITE EXT HOLD	An extended hold period is the number of additional CPU cycles inserted when (a) the EMIF must switch to a different CE space before performing the next access, or (b) the next access requires a change in the data direction (for example, the EMIF has completed a read access and must now perform a write access). All chip enable signals are inactive during this period. Whenever the EMIF must change to another CE space between accesses, the EMIF automatically adds 1 cycle in addition to any cycles you have programmed. For example, if WRITE EXT HOLD = 0 and the EMIF must switch CE spaces, the extended hold period is 1 cycle.
Time-out value	TIMEOUT	A single time-out value applies to both read operations and write operations. During an operation, an internal counter counts the number of CPU clock cycles that the ARDY signal is sampled low (indicating that the memory is not ready for an access). If the counter reaches the time-out value, the EMIF records an error in the bus error status register (see page 5-47). If a CPU bus requested the memory access, the EMIF sends a bus-error interrupt request to the CPU. If the DMA controller requested the memory access, the EMIF sends a time-out signal to the DMA controller. The DMA controller can ignore the signal or send a bus-error interrupt request to the CPU. The bus error interrupt is maskable; the CPU ignores it or services it depending on whether the interrupt is properly enabled.

Table 5–11. Examples to Illustrate the Constraints on Setup Behavior

SETUP	Setup Period For First Access (CPU Clock Cycles)	Setup Period For Following Accesses (CPU Clock Cycles)
0	2	1
1	2	1
2	2	2
3	3	3
4	4	4

5.7.3 Asynchronous Read Operations of the EMIF

The descriptions in Table 5–12 and the timing diagram in Example 5–1 explain the signal activity of the EMIF when it reads from an asynchronous memory chip. The write strobe signal (AWE_) is driven high (inactive) during a read operation. For more detailed timing information, see the data sheet for your TMS320C55x DSP.

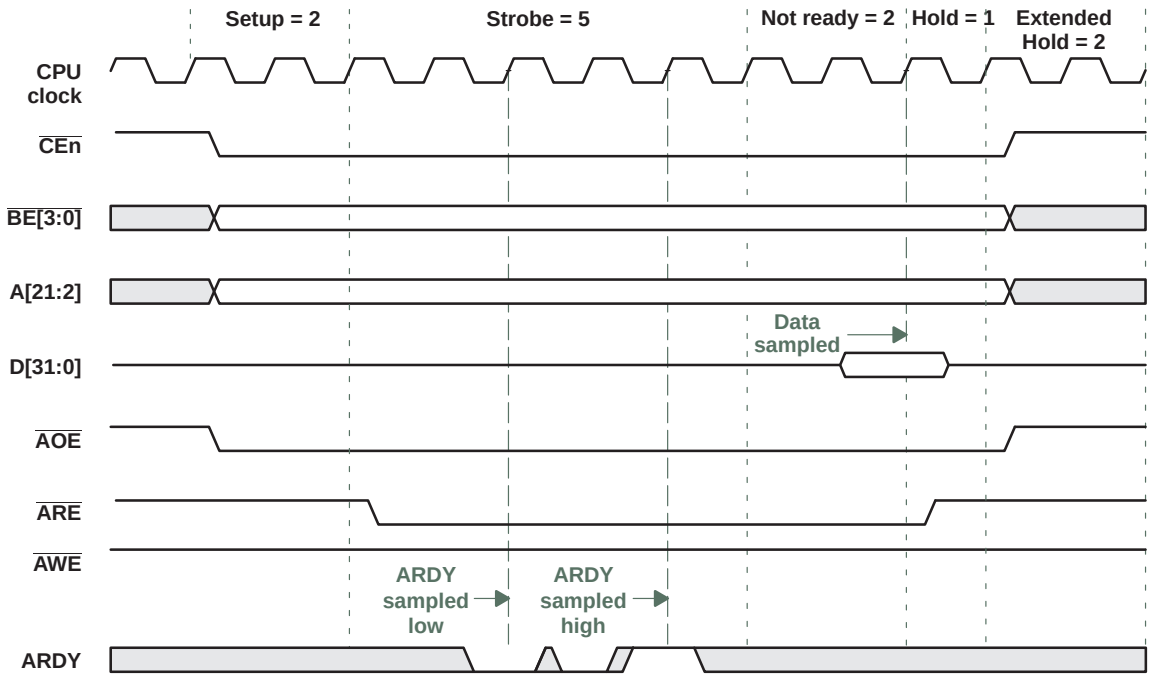
Table 5–12. EMIF Signal Activity During an Asynchronous Read Operation

Time Interval	Signal Activity
Start of setup period	**If this is the first memory read cycle in the selected CE space,** the following signal changes occur simultaneously (n is the number of the selected CE space): ❑ CEn_ falls to enable the memory chip. ❑ The byte enable lines become valid to indicate the number and positions of the bytes being accessed. ❑ The address lines become valid to carry the address to the memory chip. ❑ AOE_ falls to activate the output enable function of the memory chip. **If this is a subsequent read cycle,** CEn_ and AOE_ stay low, while new values become valid on the byte enable and address lines. **Note:** In any asynchronous read operation, the setup period is a minimum of 2 cycles for the first memory access. After the first access, the setup period is a minimum of 1 cycle.
Strobe period (may be extended with ARDY)	1) ARE_ falls to mark the start of the strobe period. 2) The memory chip can drive ARDY low to request additional cycles if it needs more time to provide the data. If ARDY is low on the third rising edge of the CPU clock before the end of the programmed strobe period, the strobe period is extended by 1 CPU clock cycle. For each subsequent CPU clock rising edge that ARDY is sampled low, the strobe period is extended by 1 CPU clock cycle. More details are in section 5.7.5 on page 5-35. 3) Data is sampled by the EMIF on the CPU clock rising edge that is concurrent with the end of the strobe period, just prior to the rising of ARE_. 4) ARE_ rises.

Table 5–12. EMIF Signal Activity During an Asynchronous Read Operation (Continued)

Time Interval	Signal Activity
End of hold period	If there is another read access in the same CE space, new values become valid on the byte enable and address lines to begin a new memory cycle. Otherwise, the following signal changes occur simultaneously: ❑ The byte enable lines become invalid. ❑ The address lines become invalid. ❑ AOE_ rises to deactivate the output enable function of the memory chip.
Extended hold period	All chip enable (CE_) signals are deactivated, to prevent accidental contention while the active memory chip is disabled and another memory chip is enabled. The extended hold period is only inserted if the EMIF receives no new access request in the same CE space and with the same data direction (read, in this case). If a request of the same type occurs during this period, the extended hold is aborted and a new memory cycle begins.

Example 5–1. Timing Diagram of an EMIF Asynchronous Read Operation



5.7.4 Asynchronous Write Operations of the EMIF

For an explanation of signal activity during a write to asynchronous memory, see Table 5–13 and Example 5–2. The output enable signal (AOE_) and the read strobe signal (ARE_) are driven high (inactive) during a write operation. For more detailed timing information, see the data sheet for your TMS320C55x DSP.

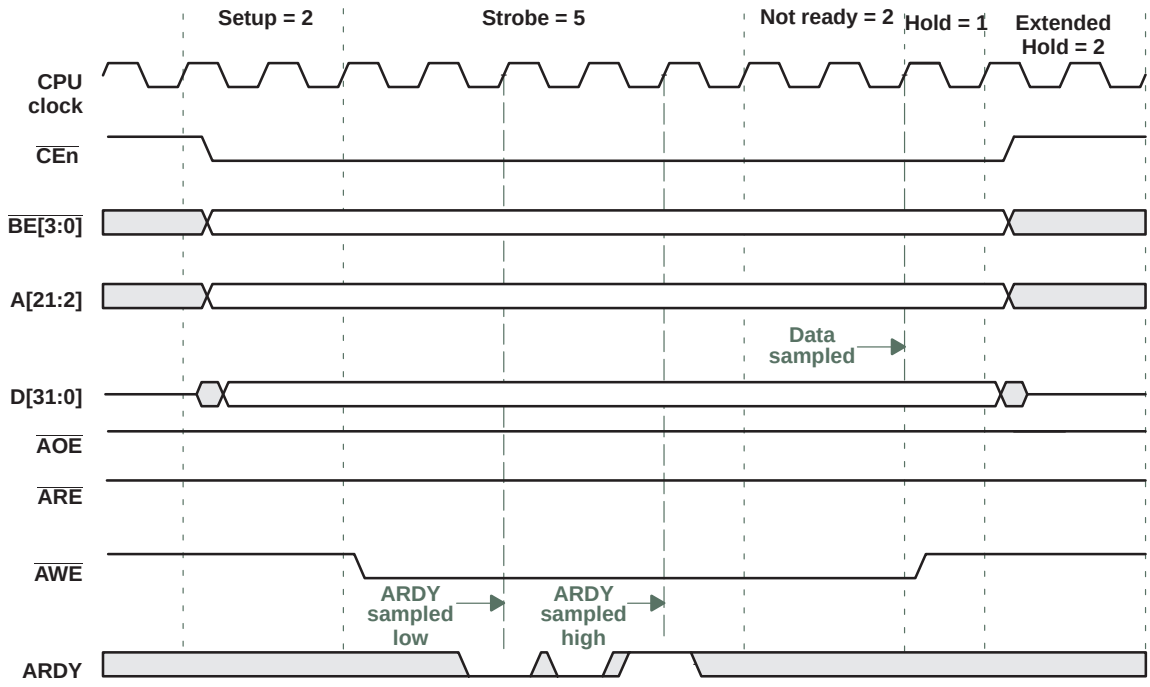
Table 5–13. EMIF Signal Activity During an Asynchronous Write Operation

Time Interval	Signal Activity
Start of setup period	**If this is first memory write cycle in the selected CE space,** the following signal changes occur (n is the number of the selected CE space): ❑ CEn_ falls to enable the memory chip. ❑ The byte enable lines become valid to indicate the number and positions of the bytes being accessed. ❑ The address lines become valid to carry the address to the memory chip. ❑ The data lines become valid to carry data to the memory chip. **If this is a subsequent write cycle,** CEn_ stays low, while new values become valid on the byte enable, address, and data lines. **Note:** In any asynchronous write operation, the setup period is a minimum of 2 cycles for the first memory access. After the first access, the setup period is a minimum of 1 cycle.
Strobe period (may be extended with ARDY)	1) AWE_ falls to mark the start of the strobe period. 2) The memory chip can drive ARDY low to request additional cycles if it needs more time to accept the data. If ARDY is low on the third rising edge of the CPU clock before the end of the programmed strobe period, the strobe period is extended by 1 CPU clock cycle. For each subsequent CPU clock rising edge that ARDY is sampled low, the strobe period is extended by 1 CPU clock cycle. More details are in section 5.7.5 on page 5-35. 3) Data is sampled by the memory chip on the CPU clock rising edge that is concurrent with the end of the strobe period, just prior to the rising of AWE_. 4) AWE_ rises.

Table 5–13. EMIF Signal Activity During an Asynchronous Write Operation (Continued)

Time Interval	Signal Activity
End of hold period	If there is another write access in the same CE space, new values become valid on the byte enable, address, and data lines to begin a new memory cycle. Otherwise, the following signal changes occur: ❑ The byte enable lines become invalid. ❑ The address lines become invalid. ❑ The data lines become invalid.
Extended hold period	All chip enable (CE_) signals are deactivated, to prevent accidental contention while the active memory chip is disabled and another memory chip is enabled. The extended hold period is only inserted if the EMIF receives no new access request in the same CE space and with the same data direction (write, in this case). If a request of the same type occurs during this period, the extended hold is aborted and a new memory cycle begins.

Example 5–2. Timing Diagram of an EMIF Asynchronous Write Operation



5.7.5 Inserting Extra Cycles with the Ready (ARDY) Signal

In addition to programmable access shaping, you can insert extra cycles into the strobe period by activating the ARDY input signal. The ready input signal is internally synchronized to the CPU clock. Ready operation is as follows:

If ARDY is low on the third rising edge of the CPU clock before the end of the programmed strobe period, the strobe period is extended by 1 CPU clock cycle. For each subsequent CPU clock rising edge that ARDY is sampled low, the strobe period is extended by 1 CPU clock cycle. Thus, the minimum value of STROBE should be 4. If STROBE is less than 4, ARDY must be held high; otherwise unexpected behavior may occur in the EMIF. The maximum number of cycles that ARDY can be used depends on the time-out value in the TIMEOUT bits. A time-out counter in the EMIF starts to count on the third rising edge of the CPU clock cycle before the end of the programmed strobe period.

5.8 Using SBSRAM (Synchronous Burst SRAM)

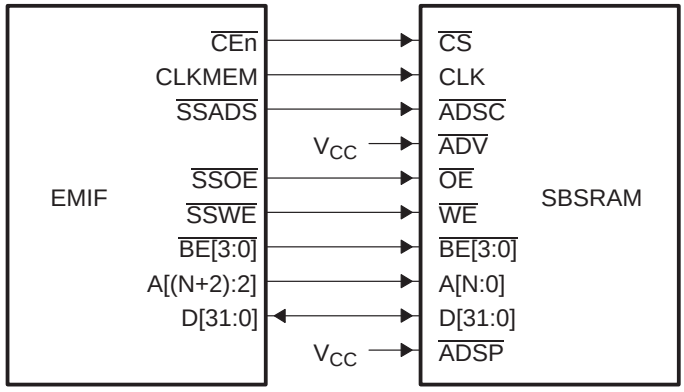
The EMIF interfaces directly to 32-bit-wide industry standard synchronous burst SRAMs. SBSRAMs are available in both Flow Through and Pipeline types; however, the EMIF supports only Pipeline SBSRAM, which has the capability to operate at higher frequencies with sustained throughput. The SBSRAM interface runs at the speed of the CPU clock or at half the speed of the CPU clock. This section includes the following topics:

5.8.1 Signal Connections for External SBSRAM

Figure 5–15 shows connections between the EMIF and an SBSRAM chip. Descriptions of the EMIF pins follow the figure.

The three SBSRAM control signals are latched by the SBSRAM on the rising CLKMEM edge to determine the current operation. These signals are only valid if the chip select line for the SBSRAM is low. The ADV_ signal of the SBSRAM allows the SBSRAM device to generate addresses internally for controllers that cannot provide addresses quickly enough. The EMIF generates the addresses at the required rate, and thus does not use ADV_.

Figure 5–15. EMIF Connected to an SBSRAM Chip



CEn_ Chip enable pin for space CEn (n can be 0, 1, 2, ...). Connect this active-low pin to the chip select pin of the SBSRAM chip, so that when the EMIF makes an access in CE space n, the SBSRAM chip is enabled.

CLKMEM Memory interface clock pin. Connect CLKMEM to the clock input of the memory. CLKMEM is controlled by the MEMFREQ bits and the MEMCEN bits of EGCR (see page 5-43). Make sure that you write to the MEMFREQ bits to select the frequency of this clock, and make sure there is a 1 in the MEMCEN bit of EGCR. If MEMCEN = 1, the signal on the CLKMEM pin does not reflect the memory interface clock; instead, it stays high.

SSADS_ Address strobe/enable pin. Connect this active-low pin to the active-low ADSC_ pin of the SBSRAM chip. Driving SSADS_ low then causes a new address to be registered.

SSOE_ Output buffer enable pin. Connect this active-low pin to the output enable pin of the SBSRAM chip.

SSWE_ Write enable pin. Connect this active-low pin to the write enable pin of the SBSRAM chip.

BE[3:0]_ Byte enable pins. The EMIF drives signal combinations on these active-low pins to indicate the size of the data being accessed. In some cases, the signal combination also indicates which portion of the EMIF data bus, D[31:0], carries the data. For more details, see the following topics:
Program Accesses on page 5-13
32-Bit Data Accesses on page 5-16
16-Bit Data Accesses on page 5-21
8-Bit Data Accesses on page 5-23

A[(N+2):2] Address bus pins. N is the number of the most significant address pin on the SBSRAM chip. As an example, suppose the SBSRAM chip has 20 address pins, 19 through 0. Connect these pins with EMIF pins A[21:2].

D[31:0] Data bus pins. The EMIF drives 32-bit, 16-bit, or 8-bit data on these pins. The pins you need to connect to the memory chip depend on the type of access and the width of the memory. For more details, see the following topics:
Program Accesses on page 5-13
32-Bit Data Accesses on page 5-16
16-Bit Data Accesses on page 5-21
8-Bit Data Accesses on page 5-23

5.8.2 Configuring the EMIF for SBSRAM Accesses

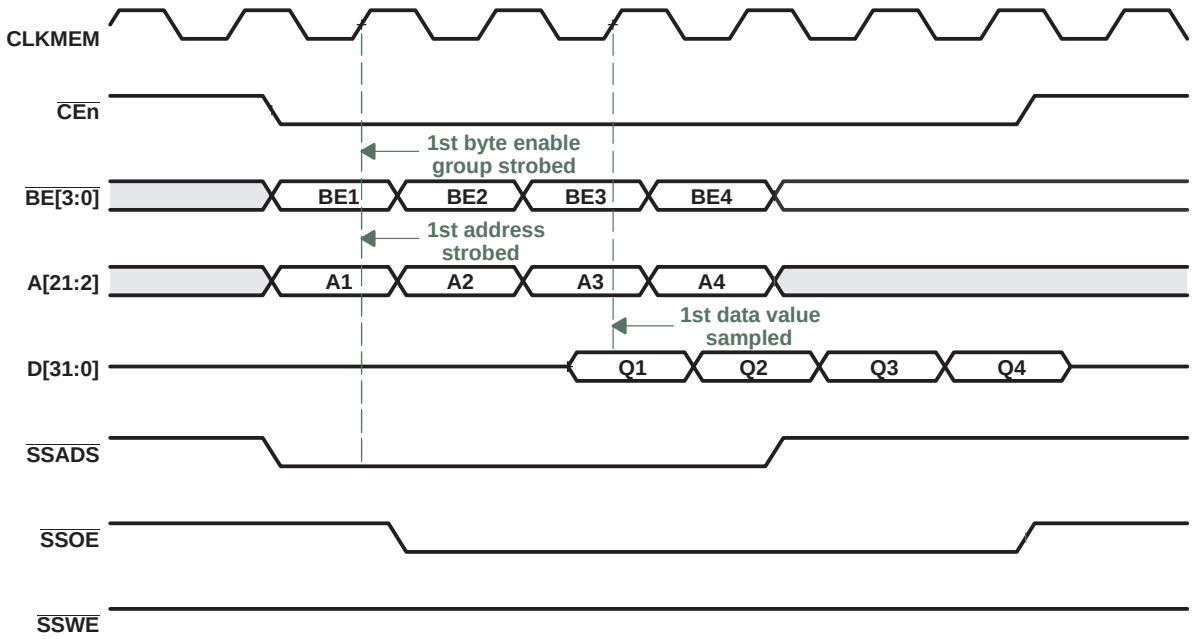
If any of the CE spaces in the external memory map will contain SBSRAM, you must make sure each of those CE spaces is configured for SBSRAM, and you must configure the MEMCLK frequency and pin. You must also decide whether you want write posting and HOLD requests enabled for those CE spaces. For details, see *Configuring the CE Spaces* on page 5-12.

5.8.3 SBSRAM Read Operations of the EMIF

Example 5–3 shows signal activity involved when the EMIF performs a DMA burst to read four 32-bit values from an SBSRAM. Every access strobes a new address into the SBSRAM. The period during which the addresses are driven is indicated by the address strobe signal (SSADS_) being low. Pipelined SBSRAM has a read latency of two cycles. The address is strobed into the SBSRAM on the first rising CLKMEM edge after SSADS_ goes active, and data is latched by the EMIF 2 cycles later. Although the first access requires a delay of 2 cycles before the data is present on the bus, each subsequent data transfer has single-cycle throughput.

For more detailed timing information, see the data sheet for your TMS320C55x DSP.

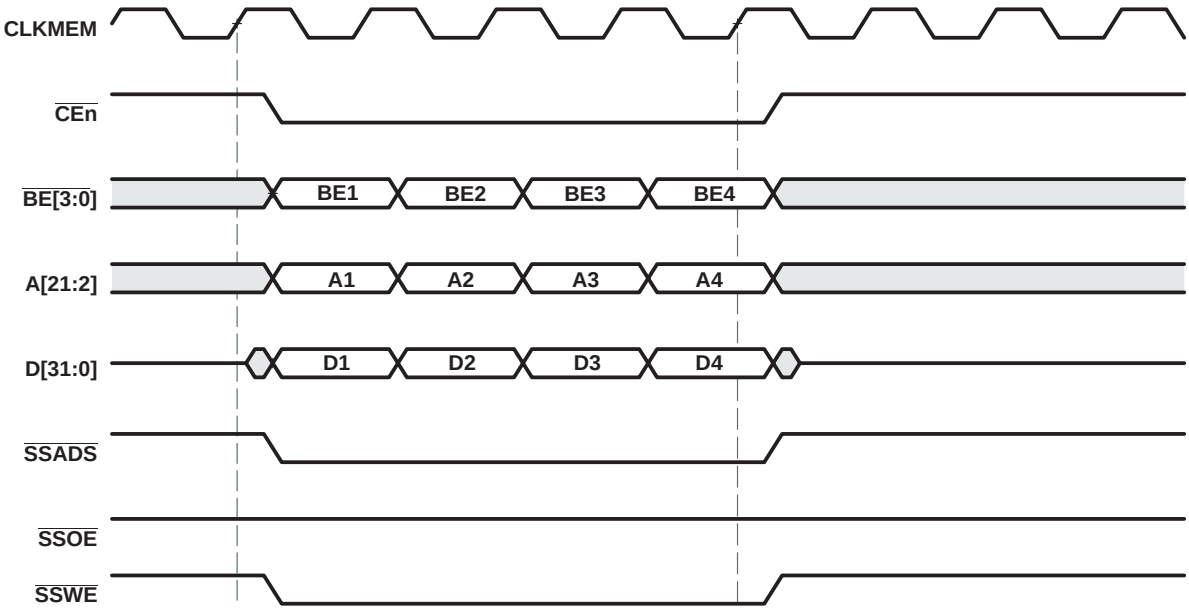
Example 5–3. Timing Diagram of an EMIF SBSRAM Read Operation



5.8.4 SBSRAM Write Operations of the EMIF

Example 5–4 shows signal activity involved when the EMIF performs a DMA burst to write four 32-bit values to an SBSRAM. For more detailed timing information, see the data sheet for your TMS320C55x DSP.

Example 5–4. Timing Diagram of an EMIF SBSRAM Write Operation



5.9 Using SDRAM (Synchronous DRAM)

The EMIF features for interfacing to external SDRAM are to be documented in a future draft of this guide.

5.10 HOLD Requests: Sharing External Memory

The EMIF provides the following two signals to allow an external device and the EMIF to share the external memory:

- HOLD_ (HOLD request pin): The external device drives this active-low pin to request exclusive access to memory that it is sharing with the EMIF. A HOLD request is the highest priority request that the EMIF can receive during active operation. When it receives the HOLD request, the EMIF stops accessing the external memory at the earliest possible moment, which may entail completion of the current accesses and memory device deactivation. The external device must continue to drive HOLD low for as long as it wants to use the bus.

- HOLDA_ (HOLD acknowledge pin): Once HOLD_ has been asserted, the EMIF places all of its other output pins in the high impedance state and then asserts HOLDA_ by driving it low. The external device may then use the pins of the external memory without the chance of interference from the EMIF.

You can block HOLD requests by setting the NOHOLD bit in the EMIF global control register (see page 5-43).

While the external device has control of the memory, the CPU can continue to execute instructions (from internal memory) or stop. You choose one of these options by writing to the HOLD mode bit (HM) of status register ST1_55. ST1_55 is a register in the CPU.

5.11 Write Posting: Buffering Writes to External Memory

The EMIF has two write post registers and a write posting enable bit (WPE). WPE is a bit in the EMIF global control register (see page 5-43).

If write posting is enabled (WPE = 1), the write posting registers are used to store the write address and data, so that the EMIF can acknowledge the CPU with zero wait states. The CPU is free to begin the next access while the EMIF assumes control of the posted write operations. The EMIF runs the posted write operations externally as time slots become available. If the next access is not for the EMIF and is for internal memory, that access is able to run concurrently with a slow external write operation.

The write post registers may be freely associated with either of the two data-write data buses of the CPU (the E bus and the F bus). A section of code that just comprises, for example, E bus writes, benefits from two levels of write posting.

When write posting has been disabled (WPE = 0), a request from the E bus/F bus is acknowledged as the write data is driven onto the external bus. It might be useful during debugging to disable write posting.

5.12 EMIF Registers

For the EMIF, the DSP contains the registers listed in Table 5–14. For the I/O address of each register, see the data sheet for your TMS320C55x DSP.

Table 5–14. Registers of the External Memory Interface (EMIF)

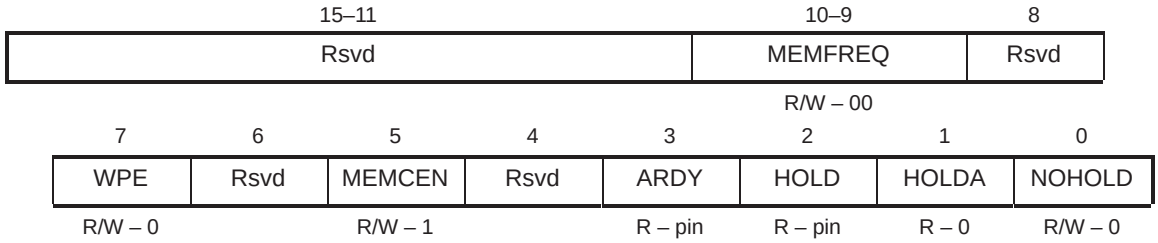
Register(s)	Description	For Details, See ...
EGCR	EMIF global control register	Page 5-43
EMI_RST	EMIF global reset register	Page 5-46
EMI_BE	EMIF bus error status register	Page 5-47
CE0_1–CEx_1 (x = number of CE spaces – 1)	CE space control registers 1 (one for each CE space)	Page 5-50
CE0_2–CEx_2 (x = number of CE spaces – 1)	CE space control registers 2 (one for each CE space)	Page 5-50
CE0_3–CEx_3 (x = number of CE spaces – 1)	CE space control registers 3 (one for each CE space)	Page 5-50
SDC1	SDRAM control register 1	Details to be in a future draft of this document
SDC2	SDRAM control register 2	Details to be in a future draft of this document
SDPER	SDRAM period register	Details to be in a future draft of this document
SDCNT	SDRAM counter register	Details to be in a future draft of this document
INIT	SDRAM initialization register	Details to be in a future draft of this document

5.12.1 EMIF Global Control Register (EGCR)

The global control register (see Figure 5–16) is a 16-bit I/O-mapped register used to configure and monitor global conditions in the EMIF. Use this register to set up the clock for synchronous memory chips (MEMFREQ and MEMCEN), to enable or disable write posting (WPE), to monitor certain EMIF pins (ARDY, HOLD, and HOLDA), and to allow or disallow HOLD requests (NOHOLD). Table 5–15 describes the bit fields of EGCR.

Figure 5–16. EMIF Global Control Register (EGCR)

EGCR



Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset. X = pin indicates that the reset value depends on the signal level on the associated pin.

Table 5–15. EGCR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–11	Rsvd	These are reserved bits (not available for your use). They are read-only bits and return 0s when read.	–
10–9	MEMFREQ	Memory clock frequency bit. The CLKMEM pin of the EMIF provides the clock signal for synchronous memory (SBSRAM or SDRAM) chips. MEMFREQ determines the relationship between the frequency of the CPU clock signal and the frequency of the signal on the CLKMEM pin: 00b The CLKMEM frequency is equal to the CPU clock frequency. 01b The CLKMEM frequency is 1/2 the CPU clock frequency. other Reserved (do not use) **Note:** When CLKMEM is needed for synchronous memory accesses, make sure that the memory clock is enabled at the CLKMEM pin (MEMCEN = 1).	00b
8	Rsvd	This is a reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.	–

Table 5–15. EGCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
7	WPE	Write posting enable bit. Use WPE to enable or disable the write posting feature of the EMIF (see page 5-42). WPE affects all of the CE spaces. 0 Disabled 1 Enabled	0
6	Rsvd	This is a reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.	–
5	MEMCEN	Memory clock enable bit. MEMCEN determines whether the memory clock is enabled at the CLKMEM pin: 0 Disabled The signal on the CLKMEM pin is held high. 1 Enabled The memory clock is reflected on the CLKMEM pin. The frequency of the clock depends on the MEMFREQ bits.	1
4	Rsvd	This is a reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.	–
3	ARDY	ARDY signal status bit. The ARDY bit reflects the signal level on the asynchronous ready (ARDY) pin: 0 ARDY signal is low External memory is not ready to accept or present data. 1 ARDY signal is high External memory is ready to accept or present data.	Reflects signal
2	HOLD	HOLD_ signal status bit. The HOLD bit reflects the signal level on the active-low HOLD request (HOLD_) pin: 0 HOLD_ signal is low An external device is driving a HOLD request (a request for the DSP to hold its external memory accesses and allow the external device to access external memory chips). 1 HOLD_ signal is high No HOLD request	Reflects signal

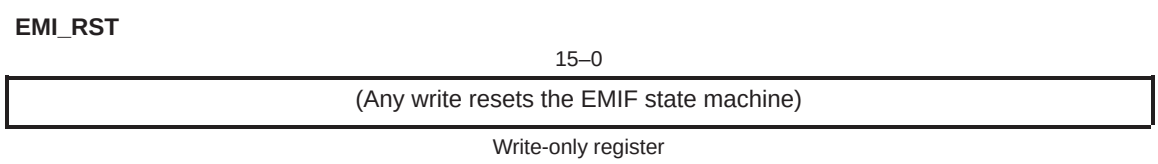
Table 5–15. EGCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
1	HOLDA	HOLDA_ signal status bit. The HOLDA bit reflects the signal level on the active-low HOLD acknowledge (HOLDA_) pin: 0 HOLDA_ signal is low The EMIF has acknowledged a HOLD request (see page 5-41). The EMIF has relinquished its control over external memory chips, so that an external device can access the chips. 1 HOLDA_ signal is high The EMIF is in control of the external memory (it is not in the HOLD state).	0
0	NOHOLD	HOLD_ disable bit. Use NOHOLD to enable or disable the HOLD_ pin and thus allow or disallow HOLD requests (see page 5-41). 0 HOLD_ enabled HOLD requests are accepted by the EMIF. 1 HOLD_ disabled No HOLD requests are accepted by the EMIF.	0

5.12.2 EMIF Global Reset Register (EMI_RST)

Any write to this register (see Figure 5–17) resets the EMIF state machine but does not change the current configuration values. This register cannot be read.

Figure 5–17. EMIF Global Reset Register (EMI_RST)



5.12.3 EMIF Bus Error Status Register (EMI_BE)

The bus error status register (see Figure 5–18) is a 16-bit I/O-mapped register used to record errors that occur during accesses to external memory. For each bus error that is recognized by the EMIF, the EMIF sets at least two bits in EMI_BE:

- One of the CE bits (bits 10 through 7), to identify which CE space was being accessed when the error occurred.
- One of the requester bits (6 through 2 and 0), to identify which DSP resource requested the external memory access.

In addition, if the error is the result of a time-out during an access to asynchronous memory, the EMIF sets the TIME bit.

After EMI_BE is read, it is automatically cleared. The bits of EMI_BE are described in Table 5–16.

EMIF bus errors also have interrupt activity associated with them. If the requester was a CPU bus, the EMIF sends a bus-error interrupt request to the CPU. If the requester was the DMA controller and a time-out error has occurred, the EMIF sends a time-out signal to the DMA controller. The DMA controller can ignore the signal or send a bus-error interrupt request to the CPU. The bus-error interrupt is maskable; the CPU ignores it or services it depending on whether the interrupt is properly enabled.

Figure 5–18. EMIF Bus Error Status Register (EMI_BE)

EMI_BE

15–13	12	11	10	9	8
Rsvd	TIME	Rsvd	CE3	CE2	CE1
	R – 0		R – 0	R – 0	R – 0

7	6	5	4	3	2	1	0
CE0	DMA	FBUS	EBUS	DBUS	CBUS	Rsvd	PBUS
R – 0	R – 0	R – 0	R – 0	R – 0	R – 0		R – 0

Legend:

R Read-only access
– 0 0 is the value after a DSP reset.

Table 5–16. EMI_BE Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–13	Rsvd	These are reserved bits (not available for your use). They are read-only bits and return 0s when read.	–
12	TIME	Time-out error status bit. The EMIF sets TIME when a time-out error occurs during an access to asynchronous memory. 0 No error 1 Error	0
11	Rsvd	This is a reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.	–
10	CE3	CE3 error status bit. The EMIF sets CE3 when an error occurs during an access to memory in the address range defined as the CE3 space. 0 No error 1 Error	0
9	CE2	CE2 error status bit. The EMIF sets CE2 when an error occurs during an access to memory in the address range defined as the CE2 space. 0 No error 1 Error	0
8	CE1	CE1 error status bit. The EMIF sets CE1 when an error occurs during an access to memory in the address range defined as the CE1 space. 0 No error 1 Error	0
7	CE0	CE0 error status bit. The EMIF sets CE0 when an error occurs during an access to memory in the address range defined as the CE0 space. 0 No error 1 Error	0
6	DMA	DMA error status bit. The EMIF sets DMA when an error occurs during an access requested by the DMA controller. 0 No error 1 Error	0

Table 5–16. EMI_BE Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
5	FBUS	F bus error status bit. The EMIF sets FBUS when an error occurs during an access requested by the F bus (one of the data-write data buses of the CPU). 0 No error 1 Error	0
4	EBUS	E bus error status bit. The EMIF sets EBUS when an error occurs during an access requested by the E bus (one of the data-write data buses of the CPU). 0 No error 1 Error	0
3	DBUS	D bus error status bit. The EMIF sets DBUS when an error occurs during an access requested by the D bus (one of the data-read data buses of the CPU). 0 No error 1 Error	0
2	CBUS	C bus error status bit. The EMIF sets CBUS when an error occurs during an access requested by the C bus (one of the data-read data buses of the CPU). 0 No error 1 Error	0
1	Rsvd	This is a reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.	–
0	PBUS	P bus error status bit. The EMIF sets PBUS when an error occurs during an access requested by the P bus (the program-read data bus of the CPU). 0 No error 1 Error	0

5.12.4 CE Space Control Registers (CEn_1, CEn_2, CEn_3)

The external memory map is divided into CE spaces (see section 5.4.1 on page 5-9). Each CE space has three CE space control registers of the form shown in Figure 5–19. These are 16-bit I/O-mapped registers used primarily for configuring accesses to asynchronous memory. With the MTYPE bit, select the memory type for the given CE space.

If you choose an asynchronous memory type, use the other bits in the CE space control registers to define the access parameters. For details about the asynchronous access parameters, including setup periods, see *Configuring the EMIF for Asynchronous Accesses* on page 5-28.

If you choose a synchronous memory type, the EMIF ignores all of the bits but MTYPE.

Table 5–17 describes the bit fields of the CE space control registers.

Note:

The SETUP and STROBE fields have a minimum count of 1, and because of this, a 0 in one of these fields is interpreted by the DSP as a 1. For the first access (even if there is only one), the setup period will have a minimum of 2. Table 5–18 shows examples to illustrate the constraints on the setup behavior.

Figure 5–19. CE Space Control Registers (CEn_1, CEn_2, CEn_3) for Each CE Space

CEn_1

15	14–12	11–8	7–2	1–0
Rsvd	MTYPE	READ SETUP	READ STROBE	READ HOLD
	R/W – 010	R/W – 1111	R/W – 111111	R/W – 11

CEn_2

15–14	13–12	11–8	7–2	1–0
READ EXT HOLD	WRITE EXT HOLD	WRITE SETUP	WRITE STROBE	WRITE HOLD
R/W – 01	R/W – 01	R/W – 1111	R/W – 111111	R/W – 11

CEn_3

15–8	7–0
Rsvd	TIMEOUT
	R/W – 00000000

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 5–17. CEn_1, CEn_2, and CEn_3 Bit Descriptions

Register	Bit(s)	Name	Description		Reset Value
CEn_1	15	Rsvd	This is a reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.		–
CEn_1	14–12	MTYPE	Memory type bits. Each of the CE spaces has MTYPE bits. For each CE space, use MTYPE to select one of the following memory types:		010b
			000b	8-bit-wide asynchronous	
			001b	16-bit-wide asynchronous	
			010b	32-bit-wide asynchronous	
			011b	32-bit-wide synchronous DRAM (SDRAM)	
			100b	32-bit-wide synchronous burst SRAM (SBSRAM)	
			other	Reserved (do not use)	

Table 5–17. CEn_1, CEn_2, and CEn_3 Bit Descriptions (Continued)

Register	Bit(s)	Name	Description	Reset Value
CEn_1	11–8	READ SETUP	Read setup period bits. For each CE space that contains asynchronous memory, load the associated READ SETUP bits with a setup period for asynchronous read operations performed in that CE space. Choose a value in this range: 1 ≤ READ SETUP ≤ 15 (CPU clock cycles) If synchronous memory is chosen, the READ SETUP bits are ignored.	1111b (15)
CEn_1	7–2	READ STROBE	Read strobe period bits. For each CE space that contains asynchronous memory, load the associated READ STROBE bits with a strobe period for asynchronous read operations performed in that CE space. Choose a value in this range: 1 ≤ READ STROBE ≤ 63 (CPU clock cycles) If synchronous memory is chosen, the READ STROBE bits are ignored.	111111b (63)
CEn_1	1–0	READ HOLD	Read hold period bits. For each CE space that contains asynchronous memory, load the associated READ HOLD bits with a hold period for asynchronous read operations performed in that CE space. The 2 bits allow you to choose a value from 0 to 3: 0 ≤ READ HOLD ≤ 3 (CPU clock cycles) If synchronous memory is chosen, the READ HOLD bits are ignored.	11b
CEn_2	15–14	READ EXT HOLD	Read extended hold period bits. For each CE space that contains asynchronous memory, load the associated READ EXT HOLD bits with an extended hold period for asynchronous read operations performed in that CE space. The 2 bits allow you to choose a value from 0 to 3: 0 ≤ READ EXT HOLD ≤ 3 (CPU clock cycles) **Note:** Whenever the EMIF must change to another CE space between accesses, the EMIF automatically adds 1 cycle in addition to any cycles you have programmed. For example, if READ EXT HOLD = 0 and the EMIF must switch CE spaces, the extended hold period is 1 cycle. If synchronous memory is chosen, the READ EXT HOLD bits are ignored.	01b

Table 5–17. CEn_1, CEn_2, and CEn_3 Bit Descriptions (Continued)

Register	Bit(s)	Name	Description	Reset Value
CEn_2	13–12	WRITE EXT HOLD	Write extended hold period bits. For each CE space that contains asynchronous memory, load the associated WRITE EXT HOLD bits with an extended hold period for asynchronous write operations performed in that CE space. The 2 bits allow you to choose a value from 0 to 3: 0 ≤ WRITE EXT HOLD ≤ 3 (CPU clock cycles) **Note:** Whenever the EMIF must change to another CE space between accesses, the EMIF automatically adds 1 cycle in addition to any cycles you have programmed. For example, if WRITE EXT HOLD = 0 and the EMIF must switch CE spaces, the extended hold period is 1 cycle. If synchronous memory is chosen, the WRITE EXT HOLD bits are ignored.	01b
CEn_2	11–8	WRITE SETUP	Write setup period bits. For each CE space that contains asynchronous memory, load the associated WRITE SETUP bits with a setup period for asynchronous write operations performed in that CE space. Choose a value in this range: 1 ≤ WRITE SETUP ≤ 15 (CPU clock cycles) If synchronous memory is chosen, the WRITE SETUP bits are ignored.	1111b (15)
CEn_2	7–2	WRITE STROBE	Write strobe period bits. For each CE space that contains asynchronous memory, load the associated WRITE STROBE bits with a strobe period for asynchronous write operations performed in that CE space. Choose a value in this range: 1 ≤ WRITE STROBE ≤ 63 (CPU clock cycles) If synchronous memory is chosen, the WRITE STROBE bits are ignored.	111111b (63)
CEn_2	1–0	WRITE HOLD	Write hold period bits. For each CE space that contains asynchronous memory, load the associated WRITE HOLD bits with a hold period for asynchronous read operations performed in that CE space. The 2 bits allow you to choose a value from 0 to 3: 0 ≤ WRITE HOLD ≤ 3 (CPU clock cycles) If synchronous memory is chosen, the WRITE HOLD bits are ignored.	11b

Table 5–17. CEn_1, CEn_2, and CEn_3 Bit Descriptions (Continued)

Register	Bit(s)	Name	Description	Reset Value
CEn_3	15–8	Rsvd	These are reserved bits (not available for your use). They are read-only bits and return 0s when read.	–
CEn_3	7–0	TIMEOUT	Time-out bits. For each CE space that contains asynchronous memory, load the associated TIMEOUT bits with a time-out value (N) for all asynchronous operations performed in that CE space, or disable the time-out feature by clearing TIMEOUT: 0 Time-out feature disabled 1 ≤ N ≤ 255 An internal counter counts the number of cycles that the asynchronous ready signal (ARDY) is sampled low (indicating that the memory is not ready for an access). If ARDY is sampled low for N CPU clock cycles, the EMIF signals a time-out error. If synchronous memory is chosen, the TIMEOUT bits are ignored.	00000000b (disabled)

Table 5–18. Examples to Illustrate the Constraints on Setup Behavior

SETUP	Setup Period For First Access (CPU Clock Cycles)	Setup Period For Following Accesses (CPU Clock Cycles)
0	2	1
1	2	1
2	2	2
3	3	3
4	4	4

Chapter 6

General-Purpose I/O Port (GPIO)

The TMS320C55x™ DSP has a general-purpose I/O port (GPIO) consisting of eight individually programmable pins, IO0–IO7. By configuring the GPIO, you can have input signals of your choice affect your program, and you can have output signals of your choice affected by your program.

You control and monitor the GPIO with the bits in the I/O direction register (IODIR) and in the I/O data register (IODATA). The bits are shown in Figure 6–1 and described in Table 6–1. IODIR and IODATA are accessible to the CPU and to the DMA controller at addresses in I/O space. For the I/O addresses, see the data sheet for your TMS320C55x DSP.

The GPIO has eight pins, IO0–IO7. For a given pin (IOx), use its direction bit (IOxDIR) to configure the pin as an input (IOxDIR = 0) or output (IOxDIR = 1). Then you use its data bit (IOxD) to read from or write to the pin. For example, to read input from pin IO4, clear IO4DIR and read IO4D. IIf a pin is configured as an input, its data bit reflects the signal level on the pin. If a pin is configured as an output, the value written to its data bit affects the signal level on the pin.

Figure 6–1. GPIO Registers (IODIR and IODATA)

IODIR

15–8	7	6	5	4	3	2	1	0
Reserved	IO7DIR	IO6DIR	IO5DIR	IO4DIR	IO3DIR	IO2DIR	IO1DIR	IO0DIR
	R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0

IODATA

15–8	7	6	5	4	3	2	1	0
Reserved	IO7D	IO6D	IO5D	IO4D	IO3D	IO2D	IO1D	IO0D
	R/W – pin	R/W – pin	R/W – pin	R/W – pin	R/W – pin	R/W – pin	R/W – pin	R/W – pin

Legend:

R/W Read/write access

– X X is the value after a DSP reset. X = pin indicates that the reset value depends on the signal level on the corresponding I/O pin.

Table 6–1. IODIR and IODATA Bit Descriptions

Register	Bit(s)	Name(s)	Description	Reset Value
IODIR	15–8	Reserved	These bits are not available for your use.	–
IODIR	7–0	IO7DIR–IO0DIR	The functions of these bits can be summarized as follows, where x is a number from 0 to 7: **IOxDIR** → **IOx Direction** 0 → Input 1 → Output On the falling edge of the DSP reset signal, the pins are made inputs.	0000 0000b
IODATA	15–8	Reserved	These bits are not available for your use.	–
IODATA	7–0	IO7D–IO0D	The functions of these bits can be summarized as follows, where x is a number from 0 to 7: **IOx Direction** / **IOxD** / **Description** Input / 0 / The signal on the IOx pin is low. Input / 1 / The signal on the IOx pin is high. Output / 0 / Drive the signal on the IOx pin low. Output / 1 / Drive the signal on the IOx pin high. On the falling edge of the DSP reset signal, the pins are inputs and the values in IO7D–IO0D depend on the levels on pins IO7–IO0.	Reflects the pins

Chapter 7

I²C Module

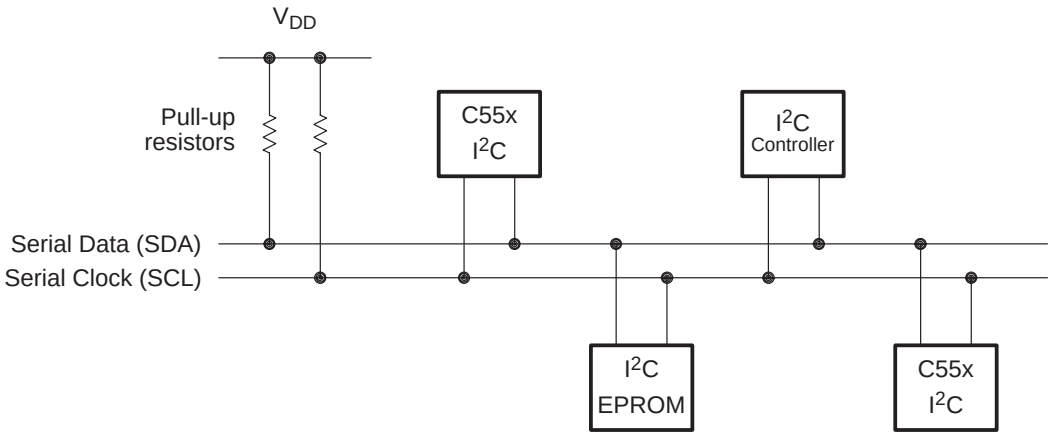
The inter-IC control (I²C) module provides an interface between a TMS320C55x™ (C55x™) DSP and I²C-compatible devices connected by way of the I²C serial bus. External components attached to the I²C bus serially transmit/receive up to 8-bit data to/from the C55x DSP through the 2-wire I²C interface. To determine whether a particular C55x DSP has an I²C module, see the data sheet for that DSP.

Topic **Page**

7.1 Introduction to the I²C Module

The I²C module supports any slave or master I²C-compatible device. Figure 7–1 shows an example of multiple I²C serial ports connected for a two-way transfer from one device to other devices.

Figure 7–1. Multiple I²C Modules Connection Diagram



7.1.1 Features

The I²C module has the following features:

- ❑ Compliance to the Philips Semiconductors I²C specification (v2.1)
  - ■ Bit/Byte format transfer
  - ■ 7-bit and 10-bit device addressing modes
  - ■ General call
  - ■ Start byte
  - ■ Free data format
  - ■ Multi-master transmitter/slave receiver mode
  - ■ Multi-master receiver/slave transmitter mode
  - ■ Combined master transmit/receive and receive/transmit mode
  - ■ I²C data transfer rate of from 10 kbps up to 400 kbps (Philips I²C rate)
- ❑ One read and one write DMA event that can be used by the DMA
- ❑ One read/write and one ‘illegal operation’ interrupt that can be used by the CPU
- ❑ Operate with DSP core frequency from 12 Mhz up
- ❑ Operate with module frequency of 12 Mhz
- ❑ Module enable/disable capability

7.1.2 Features Not Supported

The I2C module does not support:

- ❑ High-speed (HS) mode
- ❑ C-bus compatibility mode

7.1.3 Functional Overview

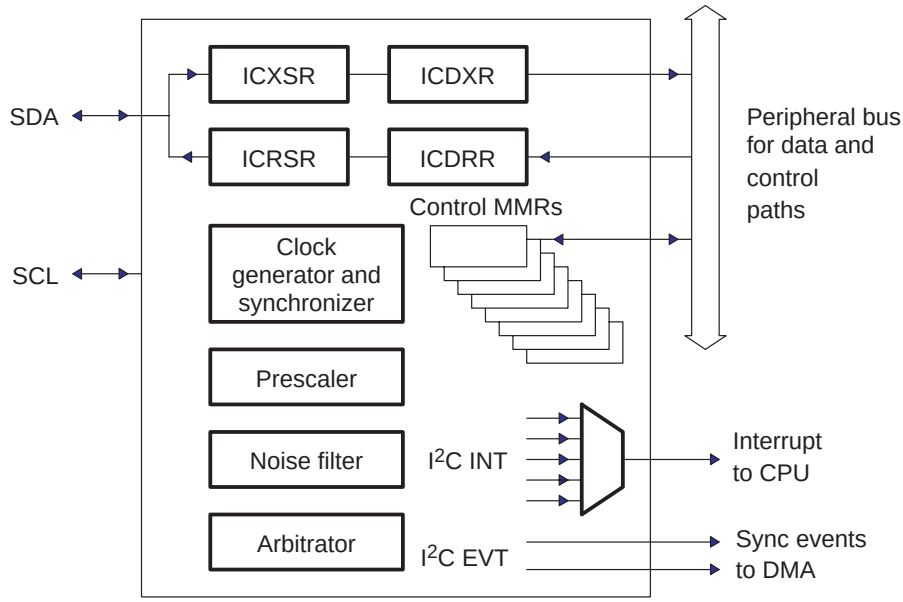
The I2C module is a serial bus that supports the multimaster mode in which one or more devices, capable of controlling the bus, can be connected to the same bus. Including the C55x DSP, each I2C device is recognized by a unique address and can operate as either a transmitter or a receiver depending on the function of the device. In addition to being a transmitter or receiver, a device connected to the I2C bus can also be considered as the master or the slave when performing data transfers. Note that 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.

Data is communicated to devices interfacing to the I2C module using the serial data pin (SDA) and the serial clock pin (SCL), shown in Figure 7–2. These two wires carry information between the C55x device and other devices connected to the I2C bus. Both SDA and SCL pins on the C55x device are bidirectional. They 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.

The I2C module consists of the following primary blocks:

- ❑ Serial interface
- ❑ DSP register interface
- ❑ Prescaler
- ❑ Peripheral bus interface
- ❑ Control/Status
- ❑ Data/Address
- ❑ I2C clock generator

Figure 7–2. I2C Module Internal Block Diagram



7.2 I2C Module Operational Details

7.2.1 I2C Module Reset

The I2C module can be reset in the following two ways:

- The C55x device V-bus reset (VBUS_RESET_ = 0) places the I2C module in reset. A device reset causes the V-bus reset.
- The I2C module can be reset by clearing the $\overline{\text{IRS}}$ bit in the I2C mode register (ICMDR). When the V-bus reset is removed (VBUS_RESET_ = 1), the IRS bit is cleared to 0 keeping the I2C module in reset.

During an I2C module reset, both the SDA and SCL pins are in a high-impedance state.

7.2.2 I2C Module Bit Transfer

One clock pulse is generated by the master device for each data bit transferred. Due to a variety of different technology devices (CMOS, NMOC, Bipolar) 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}. Refer to Table 7–1 for the electrical specification.

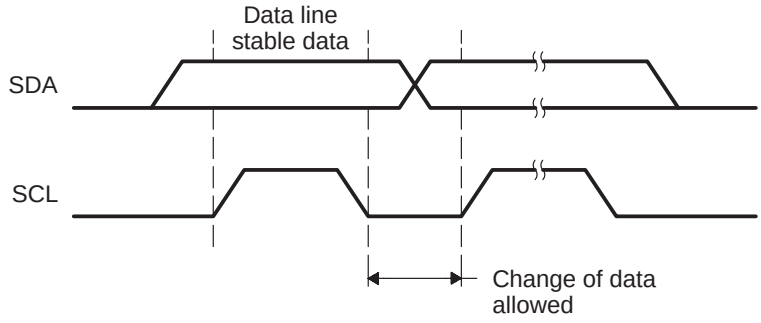
Table 7–1. Electrical Specification of the Input/Outputs for the I2C Module

		Standard Mode Devices		Fast Mode Devices		
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Unit**
	Low-level input voltage:					
VIL	Fixed-input levels	– 0.5	1.5	– 0.5	1.5	V
	V_{DD}-related input levels	– 0.5	0.3 V_{DD}	– 0.5	0.3 V_{DD}	
	High-level input voltage:					
VIH	Fixed-input levels	3.0	V_{DD}max + 0.5	3.0	V_{DD}max + 0.5	V
	V_{DD}-related input levels	0.7 V_{DD}	V_{DD}max + 0.5	0.7 V_{DD}	V_{DD}max + 0.5	
	Low-level output voltage:					
VOL1	At 3 mA sink current	0	0.4	0	0.4	V
VOL2	At 6 mA sink current	N/A	N/A	0	0.6	

7.2.3 I2C Module Data Validity

The data on SDA must be stable during the high period of the clock, see Figure 7–3. The high and low state of the data line, SDA, can only change when the clock signal on SCL is low.

Figure 7–3. Bit Transfer on the I2C Bus

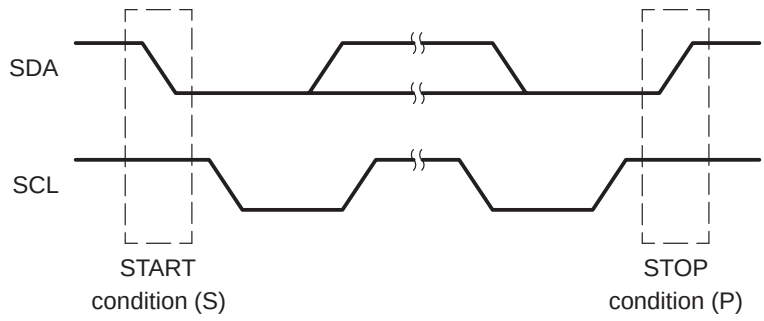


7.2.4 I2C *Module START and STOP Conditions*

START and STOP conditions are generated by a master I2C module. See Figure 7–4.

- START condition is defined as a high-to-low transition on the SDA line while SCL is high. The bus busy bit (BB) in ICSTR is set to 1 (the I2C bus is considered to be busy).
- STOP condition is defined as a low-to-high transition on the SDA line while SCL is high. The bus busy bit (BB) in ICSTR is cleared to 0 (the I2C bus is considered to be free).

Figure 7–4. I2C Module START and STOP Conditions

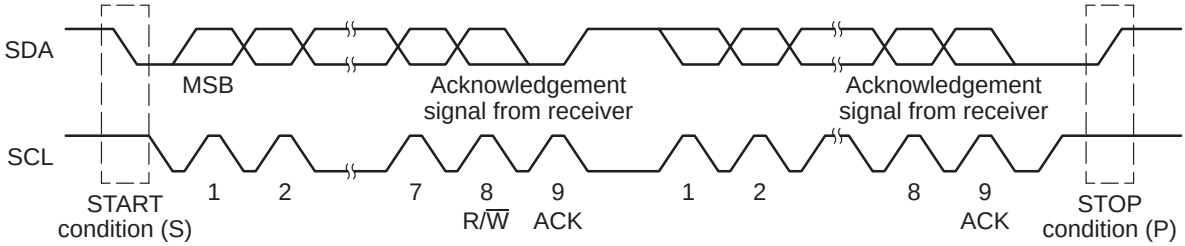


7.2.5 I2C Module Operation

7.2.5.1 Serial Data Formats

The I2C module is operated in byte data format. Each byte put on the SDA line equates to 8 pulses on the SCL. The number of bytes that can be transmitted or received is unrestricted. The data is transferred with the most significant bit (MSB) first (Figure 7–5). The I2C module does not support endianess.

Figure 7–5. I2C Module Data Transfer



The first byte after a START condition (S) always consists of 8 bits that comprises either a 7-bit slave address and the R/$\overline{W}$ bit or 8-bit data. The eighth bit, the R/$\overline{W}$, in the first byte determines the direction of the data. When the R/$\overline{W}$ bit is 0, the master writes (transmits) data to a selected slave device; when the R/$\overline{W}$ bit is 1, the master reads (receives) data from the slave device. In the acknowledge mode, an extra bit dedicated for the acknowledgement (ACK) bit is inserted after each byte.

The I2C module supports the following data formats:

- ❏ 7-bit addressing format (Figure 7–6)
- ❏ 10-bit addressing format (Figure 7–7)
- ❏ 7-bit/10-bit addressing format with repeated START condition (Figure 7–8)
- ❏ Free-data format (Figure 7–9)

In the 7-bit addressing format (Figure 7–6), the first byte is the 7-bit slave address bits (MSB) and the R/$\overline{W}$ bit (LSB). In the acknowledge mode, the ACK bit is inserted after each byte, followed by the 8-bit data.

In the 10-bit addressing format (Figure 7–7), the first byte is 11110b, the two MSBs of the 10-bit slave address, and the R/$\overline{W}$ bit. In the acknowledge mode, the ACK bit is inserted after each byte. The next byte is the remaining 8 bits of the 10-bit slave address, followed by the ACK bit and the 8-bit data.

In the free-data format (Figure 7–9), the direction of data (transmit or receive) remains constant throughout the transfer. The first byte after a START condition (S) is the 8-bit data. In addition, the ACK bit is inserted after each byte, followed by another 8-bit data.

Figure 7–6. I²C Module 7-Bit Addressing Format

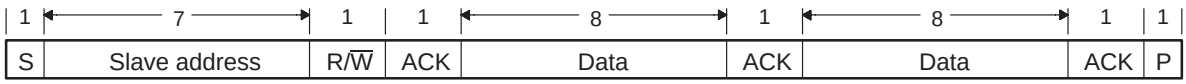


Figure 7–7. I²C Module 10-Bit Addressing Format

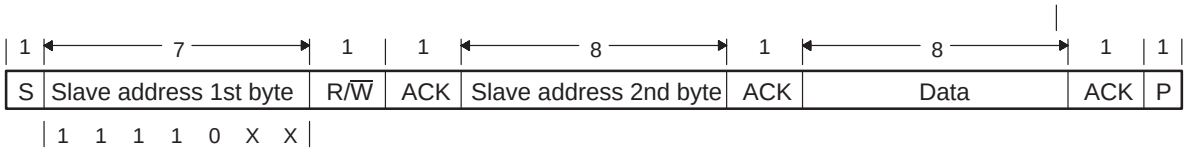


Figure 7–8. I²C Module Addressing Format with Repeated START Conditions

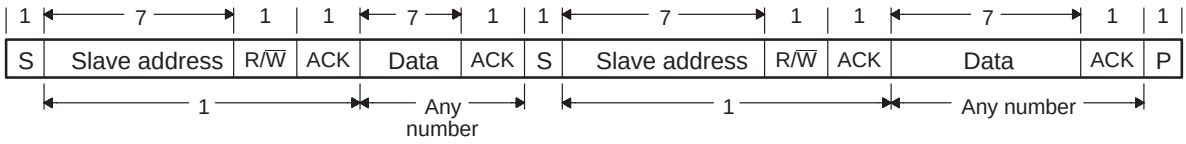
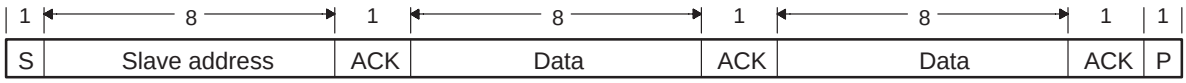


Figure 7–9. I²C Module Free-Data Format



7.2.5.2 Master Transmitter Mode

In this mode, data assembled in any of the addressing formats (Figure 7–6, Figure 7–7, or Figure 7–8) is shifted out on SDA, synchronized with the self-generated clock pulses on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the DSP is required ($\overline{\text{XSMT}}$) after a byte has been transmitted.

7.2.5.3 Master Receiver Mode

This mode can only be entered from the master transmitter mode. In any of the addressing formats (Figure 7–6, Figure 7–7, or Figure 7–8), the master receiver is entered after the slave address byte and the R/$\overline{\text{W}}$ bit has been transmitted (if the R/$\overline{\text{W}}$ bit is 1). Serial data bits received on SDA are shifted in with the self-generated clock pulses on SCL. The clock pulses are inhibited and SCL is held low when the intervention of the DSP is required (RSFULL) after a byte has been received. At the end of a transfer, a STOP condition is generated.

7.2.5.4 Slave Transmitter Mode

This mode can only be entered from the slave receiver mode. In any of the addressing formats (Figure 7–6, Figure 7–7, or Figure 7–8), the slave transmitter is entered if the slave address byte is the same as its own address and the R/$\overline{W}$ bit has been transmitted (if the R/$\overline{W}$ bit is 1). The slave transmitter shifts the serial data out on SDA with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low while intervention of the DSP is required (XSMT) after a byte has been transmitted.

7.2.5.5 Slave Receiver Mode

In this mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low while intervention of the DSP is required (RSFULL) after a byte has been received.

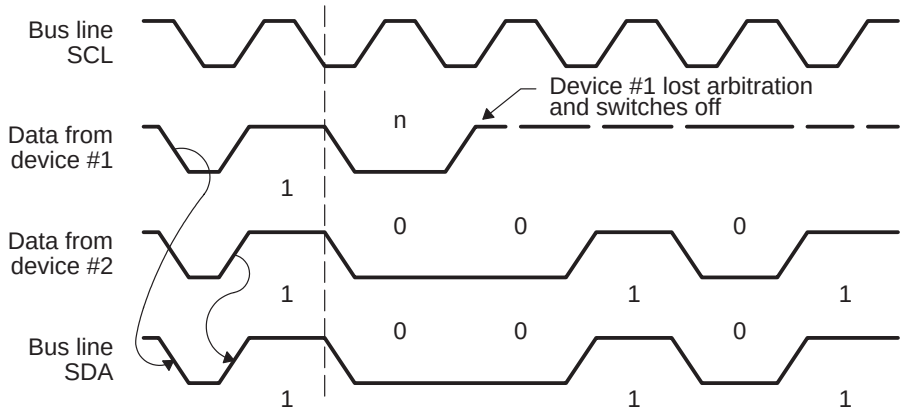
7.2.6 Arbitration

If two or more master transmitters simultaneously start a transmission on the same bus, an arbitration procedure is invoked. Figure 7–10 illustrates the arbitration procedure between two devices. The arbitration procedure uses the data presented on the serial data bus (SDA) by the competing transmitters. The first master transmitter that generates a high is overruled by the other master transmitter that generates a low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. The master transmitter that lost arbitration switches to the slave receiver mode, sets the arbitration lost (AL) flag, and generates the arbitration-lost interrupt. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If during a serial transfer and the arbitration procedure is still in progress when a repeated START condition or STOP condition is transmitted to SDA, the master transmitters involved must send the repeated START condition or STOP condition at the same position in the format frame. Arbitration is not allowed between:

- a repeated START condition and a data bit
- a STOP condition and a data bit
- a repeated START condition and a STOP condition

Figure 7–10. Arbitration Procedure Between Two Master Transmitters

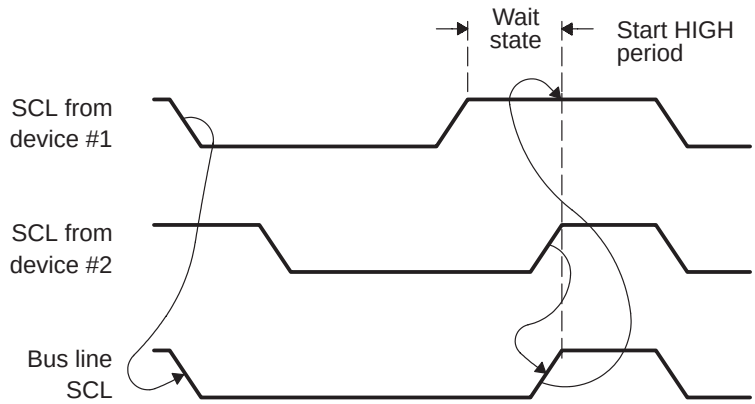


7.2.7 I²C Clock Generation and I²C Clock Synchronization

Under normal conditions, only one master device generates the clock signal, SCL. During the arbitration procedure, however, there are two or more master devices and the clock must be synchronized so that the data output can be compared. Figure 7–11 illustrates the clock synchronization. The wired-AND property of SCL means that a device that first generates a low period on SCL overrules the other devices. At this high-to-low transition, the clock generators of the other devices are forced to start their own low period. The SCL is held low by the device with the longest low period. The other devices that finish their low periods must wait for SCL to be released, before starting their high periods. A synchronized signal on SCL is obtained, where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

If a device pulls down the clock line for a longer time, the result is that all clock generators must enter the wait state. In this way, a slave slows down a fast master and the slow device creates enough time to store a received byte or to prepare a byte to be transmitted.

Figure 7–11. Synchronization of Two I2C Clock Generators During Arbitration



7.2.8 Prescaler (SYSCLK/MCLK)

The I2C module is operated with an approximately 12 MHz clock. This clock is generated by way of the I2C prescaler block. The prescaler block consists of a 16-bit register, ICPSC, used for dividing down the system clock (SYSCLK) to obtain the approximately 12 MHz clock for the I2C module.

7.2.9 Noise Filter

The noise filter is used to suppress any noise that is 50 ns or less. It is designed to suppress noise with one MCLK, assuming the lower and upper limits of MCLK are 8 Mhz and 16 Mhz, respectively.

7.3 I2C Module Interrupts

The I2C module generates five types of interrupts. These five interrupts are accompanied with five interrupt mask bits in the interrupt mask register (ICIMR) and with five interrupt flag bits in the interrupt status register (ICSTR).

- Arbitration-lost interrupt (AL) is generated when the I2C arbitration procedure is lost.
- No-acknowledge interrupt (NACK) is generated when the master I2C does not receive an acknowledge from the receiver.
- Register-access-ready interrupt (ARDY) is generated when the previously programmed address, data, and command have been performed and the status bits have been updated. This interrupt is used to notify the DSP that the I2C registers are ready to be accessed.
- Receive-data-ready interrupt (ICRRDY) is generated when the LSB of the received data in the receive-shift register (ICRSR) has been copied into the data receive register (ICDRR). The ICRRDY bit can also be polled by the DSP to read the received data in ICDRR. The I2C module sends a receive interrupt (ICRINT) to the CPU.
- Transmit-data-ready interrupt (ICXRDY) is generated when the LSB of the transmitted data has been copied from the data transmit register (ICDXR) into the transmit-shift register (ICXSR) and shifted out from the SDA pin. The ICXRDY bit can also polled by the DSP to write the next transmitted data into ICDXR. The I2C module sends a transmit interrupt (ICXINT) to the CPU.

The interrupt vector register (ICIVR) contains the binary-coded-interrupt vector that indicates which interrupt has occurred. Reading ICIVR clears the interrupt flag in ICSTR; if other interrupts are pending, a new interrupt is generated. If there is more than one interrupt flag, reading ICIVR clears the highest priority interrupt flag.

The I2C interrupt signal is a one-clock wide active-high signal.

7.3.1 DMA Controller Events

The I2C module has two events that use the DMA controller to synchronously read received data (ICREVNT) from ICDRR, and synchronously write transmitted data (ICWEVNT) to ICDXR. The read and write events have the same timing as ICRRDY (ICRINT) and ICXRDY (ICXINT), respectively.

The CPU or the DMA controller reads the received data from ICDRR and writes the data to be transmitted to ICDXR. Data written to ICDXR is copied to ICXSR and shifted out from the SDA pin when the I2C module is configured as a transmitter. When the I2C module is configured as a receiver, receive data is shifted into ICRSR and copied to ICDRR that can be read by the CPU or the DMA controller.

The CPU or the DMA controller writes the address of the I2C slave device that it wants to communicate with into ICSAR and its own address into ICOAR to identify its own slave address when it is in slave mode.

7.3.2 I2C Enable/Disable

The I2C module can be enabled/disabled with the I2C reset enable bit (IRS) in the I2C mode register (ICMDR).

7.4 I2C Module Registers

The I2C module registers are listed in Table 7–2.

Table 7–2. I2C Module Registers

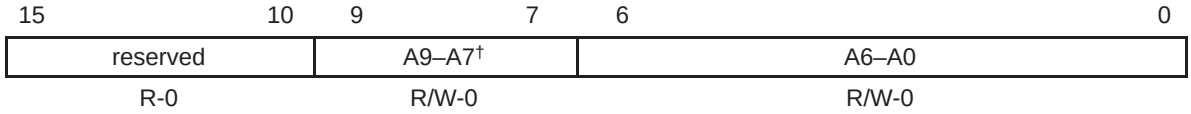
Address (Hex)	Name	Description
3C00h	ICOAR	I2C Own Address Register
3C01h	ICIMR	I2C Interrupt Mask/Status Register
3C02h	ICSTR	I2C Interrupt Status Register
3C03h	ICCLKL	I2C Clock Divider Low Register
3C04h	ICCLKH	I2C Clock Divider High Register
3C05h	ICCNT	I2C Data Count Register
3C06h	ICDRR	I2C Data Receive Register
3C07h	ICSAR	I2C Slave Address Register
3C08h	ICDXR	I2C Data Transmit Register
3C09h	ICMDR	I2C Mode Register
3C0Ah	ICIVR	I2C Interrupt Vector Register
3C0Bh	ICGPIO	I2C GPIO Register
3C0Ch	ICPSC	I2C Prescaler Register
†	ICRSR	I2C Data Receive Shift Register
†	ICXSR	I2C Data Transmit Shift Register

† This register is not accessible by the DSP.

7.4.1 *I2C Own Address Register (ICOAR)*

The I2C own address register (ICOAR) is a 16-bit memory-mapped register used to specify its own address that distinguishes it from other peripherals connected to the I2C bus.

Figure 7–12. I2C Own Address Register (ICOAR)



† Only available when in expanded address mode; otherwise, reserved.

Note: R/W-x = Read/Write-Reset value

Table 7–3. I2C Own Address Register (ICOAR) Field Values

Bit	*field*	*symval*	Value	Description
				In normal address mode (XA = 0 in ICMDR):
15–7	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
6–0	A6–A0		0–7Fh	This 7-bit value is used as the slave address.
				In expanded address mode (XA = 1 in ICMDR):
15–10	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
9–0	A9–A0		0–3FFh	This 10-bit value is used as the slave address.

7.4.2 I2C Interrupt Mask Register (ICIMR)

The I2C interrupt mask register (ICIMR) is a 16-bit memory-mapped register used by the DSP to enable/disable the interrupts.

Figure 7–13. I2C Interrupt Mask Register (ICIMR)

15	5	4	3	2	1	0
reserved		ICXRDY	ICRRDY	ARDY	NACK	AL
R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Note: R/W-x = Read/Write-Reset value

Table 7–4. I2C Interrupt Mask Register (ICIMR) Field Values

Bit	*field*	*symval*	Value	Description
15–5	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
4	ICXRDY			Transmit-data-ready interrupt mask enable bit.
			0	Transmit-data-ready interrupt is masked.
			1	Transmit-data-ready interrupt is unmasked.
3	ICRRDY			Receive-data-ready interrupt mask enable bit.
			0	Receive-data-ready interrupt is masked.
			1	Receive-data-ready interrupt is unmasked.
2	ARDY			Register-access-ready interrupt mask enable bit.
			0	Register-access-ready interrupt is masked.
			1	Register-access-ready interrupt is unmasked.
1	NACK			No-acknowledgement interrupt mask enable bit.
			0	No-acknowledgement interrupt is masked.
			1	No-acknowledgement interrupt is unmasked.
0	AL			Arbitration-lost interrupt mask enable bit.
			0	Arbitration-lost interrupt is masked.
			1	Arbitration-lost interrupt is unmasked.

7.4.3 I2C Interrupt Status Register (ICSTR)

The I2C interrupt status register (ICSTR) is a 16-bit memory-mapped register used by the DSP to determine which interrupt has occurred and to provide I2C status.

Figure 7–14. I2C Status Register (ICSTR)

15 13	12	11	10	9	8	7 5	4	3	2	1	0
reserved	BB	RSFULL	XSMT	AAS	AD0	reserved	ICXRDY	ICRRDY	ARDY	NACK	AL
R-0	R-0	R-0	R-1	R-0	R-0	R-0	R-1	R/W-0	R-0	R-0	R-0

Note: R/W-x = Read/Write-Reset value

Table 7–5. I2C Status Register (ICSTR) Field Values

Bit	*field*	*symval*	Value	Description
15–13	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
12	BB			Bus busy bit indicates the state of the serial bus. On reception of a START condition, the device sets the BB bit to 1. The BB bit is cleared to 0 after reception of a STOP condition. In the master mode, the BB bit is controlled by software. To start a transmission with a START condition, the MST, TRX and STT bits in ICMDR must be set to 1. To end a transmission with a STOP condition, the STP bit must be set to 1. When the BB bit is set to 1 and the STT bit is set to 1, a restart condition is generated.
			0	Serial data bus is free.
			1	Serial data bus is occupied.
11	RSFULL			Receive-shift register full bit indicates that the receiver has experienced overrun. Overrun occurs when the receive-shift register (ICRSR) is full and the data receive register (ICDRR) has not been read since the ICRSR-to-ICDRR transfer. The FSM is holding for ICDRR read access. The RSFULL bit is cleared when reading the ICDRR, resetting the I2C module (IRS = 0), or resetting the device.
			0	ICRSR is not in overrun condition.
			1	ICRSR has recognized an overrun. The contents of ICDRR are lost in this case.

Table 7–5. I2C Status Register (ICSTR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
10	XSMT			Transmit-shift register empty bit indicates that the transmitter has experienced underflow. Underflow occurs when the transmit-shift register (ICXSR) is empty and the data transmit register (ICDXR) has not been loaded since the last ICDXR-to-ICXSR transfer. The FSM is holding for ICXDR write access. The XSMT bit is cleared when underflow has occurred, resetting the I2C module (IRS = 0), or resetting the device. The XSMT bit is set to 1 as a result of writing to ICDXR.
			0	Transmitter-empty condition.
			1	No transmitter-empty condition.
9	AAS			Address as slave bit.
			0	The AAS bit is cleared by restart or STOP condition. The device remains selected until the STOP condition.
			1	The device has recognized its own slave address or an address of all zeros (general call). The AAS bit is also set if the first byte has been received in the free data format (ALS = 1).
8	AD0			Address 0 status bit.
			0	A START or STOP condition is detected.
			1	An address of all zeros (general call) is detected.
7–5	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
4	ICXRDY			Transmit-data-ready interrupt flag bit indicates that the LSB of the transmitted data has been copied from the data transmit register (ICDXR) into the transmit-shift register (ICXSR) and shifted out from the SDA pin. The ICXRDY bit can also polled by the DSP to write the next transmitted data into ICDXR.
			0	ICDXR is written.
			1	The transmitted data LSB has been shifted out of ICDXR.
3	ICRRDY			Receive-data-ready interrupt flag bit indicates that the LSB of the received data in the receive-shift register (ICRSR) has been copied into the data receive register (ICDRR). The ICRRDY bit can also be polled by the DSP to read the received data in ICDRR.
			0	ICDRR is read.
			1	The received data LSB has been shifted into ICDRR.

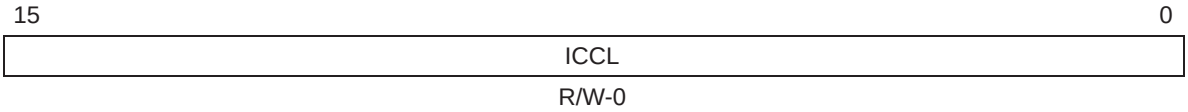
Table 7–5. I2C Status Register (ICSTR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
2	ARDY			Register-access-ready interrupt flag bit indicates that the previously programmed address, data, and command have been performed and the status bits have been updated. This interrupt is used to notify the DSP that the I2C registers are ready to be accessed.
				In nonrepeat mode (RM = 0 in ICMDR):
			0	Registers are not ready to be accessed.
			1	ICCNT passes 0 (if STP bit has not been set).
				In repeat mode (RM = 1 in ICMDR):
			0	Registers are not ready to be accessed.
			1	End of each byte transmitted from ICDXR.
1	NACK			No-acknowledgement interrupt flag bit indicates when the master I2C does not receive an acknowledge from the receiver.
			0	The hardware detects that an acknowledge (ACK) bit has been received.
			1	The hardware detects that no acknowledge (ACK) bit has been received or the I2C is operating in the general call, even though an acknowledgement is received.
0	AL			Arbitration-lost interrupt flag bit indicates when an arbitration is lost.
			0	The device in the master transmitter mode has not lost an arbitration.
			1	The device in the master transmitter mode senses it has lost an arbitration because two or more transmitters start a transmission almost simultaneously or when the I2C attempts to start a transfer while the BB bit is set to 1. The device becomes a slave receiver. The MST and STP bits in ICSTR are cleared to 0.

7.4.4 I2C Clock Divider Low Register (ICCLKL)

The I2C clock divider low register (ICCLKL) is a 16-bit memory-mapped register used to divide the master clock down to obtain the I2C serial clock low time.

Figure 7–15. I2C Clock Divider Low Register (ICCLKL)



Note: R/W-x = Read/Write-Reset value

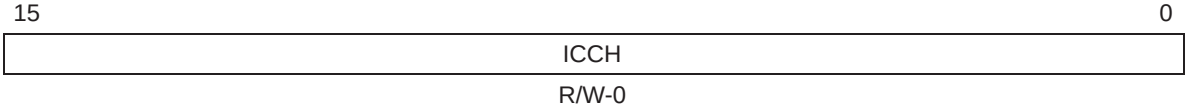
Table 7–6. I2C Clock Divider Low Register (ICCLKL) Field Values

Bit	*field*	*symval*	Value	Description
15–0	ICCL		0–FFFFh	Low-time I2C SCL clock division factor is used to divide down the master clock to create the SCL low-time transition frequency.

7.4.5 I2C Clock Divider High Register (ICCLKH)

The I2C clock divider high register (ICCLKH) is a 16-bit memory-mapped register used to divide the master clock down to obtain the I2C serial clock high time.

Figure 7–16. I2C Clock Divider High Register (ICCLKH)



Note: R/W-x = Read/Write-Reset value

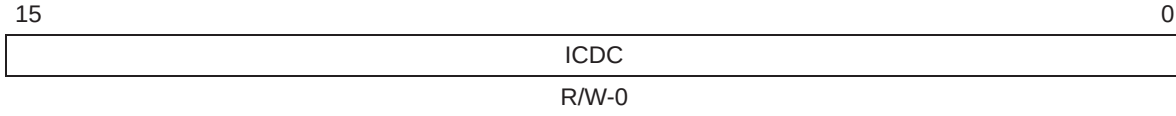
Table 7–7. I2C Clock Divider High Register (ICCLKH) Field Values

Bit	*field*	*symval*	Value	Description
15–0	ICCH		0–FFFFh	High-time I2C SCL clock division factor is used to divide down the master clock to create the SCL high-time transition frequency.

7.4.6 I2C Data Count Register (ICCNT)

The I2C data count register (ICCNT) is a 16-bit memory-mapped register used to generate the STOP condition to terminate the transfer.

Figure 7–17. I2C Data Count Register (ICCNT)



Note: R/W-x = Read/Write-Reset value

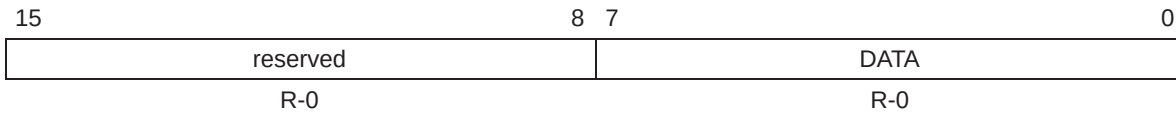
Table 7–8. I2C Data Count Register (ICCNT) Field Values

Bit	*field*	*symval*	Value	Description
15–0	ICDC			The countdown counter value is used to generate a STOP condition if a STOP condition is specified (STP = 1 in ICMDR). If the data counter is set to all 1s, the counter is ignored. In this mode, the I2C continues sending or receiving data until the STP bit is set to 1 by the DSP. Note that the value in ICCNT is a don't care when the RM bit in ICMDR is set to 1.
			0000	Data counter is 65536
			0001h–FFFFh	Data counter is 1–65535

7.4.7 I2C Data Receive Register (ICDRR)

The I2C data receive register (ICDRR) is a 16-bit memory-mapped register used by the DSP to read the receive data.

Figure 7–18. I2C Data Receive Register (ICDRR)



Note: R/W-x = Read/Write-Reset value

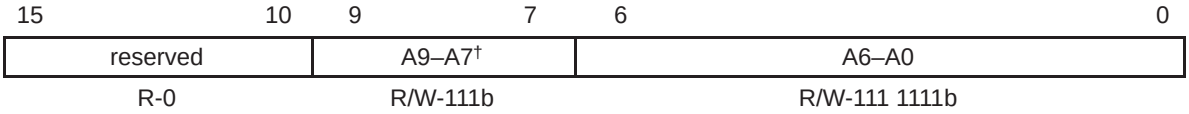
Table 7–9. I2C Data Receive Register (ICDRR) Field Values

Bit	*field*	*symval*	Value	Description
15–8	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
7–0	DATA			Receive data.

7.4.8 *I2C Slave Address Register (ICSAR)*

The I2C slave address register (ICSAR) is a 16-bit memory-mapped register used to specify the address of the communicating slave device connected to the I2C bus.

Figure 7–19. I2C Slave Address Register (ICSAR)



† Only available when in expanded address mode; otherwise, reserved.

Note: R/W-x = Read/Write-Reset value

Table 7–10. I2C Slave Address Register (ICSAR) Field Values

Bit	*field*	*symval*	Value	Description
				In normal address mode (XA = 0 in ICMDR):
15–7	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
6–0	A6–A0		0–7Fh	This 7-bit value is used as the slave address.
				In expanded address mode (XA = 1 in ICMDR):
15–10	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
9–0	A9–A0		0–3FFh	This 10-bit value is used as the slave address.

7.4.9 I2C Data Transmit Register (ICDXR)

The I2C data transmit register (ICDXR) is a 16-bit memory-mapped register used by the DSP to write the transmit data.

Figure 7–20. I2C Data Transmit Register (ICDXR)

15	8	7	0
reserved		DATA	
R-0		R-0	

Note: R/W-x = Read/Write-Reset value

Table 7–11. I2C Data Transmit Register (ICDXR) Field Values

Bit	*field*	*symval*	Value	Description
15–8	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
7–0	DATA			Transmit data.

7.4.10 I2C Mode Register (ICMDR)

The I2C mode register (ICMDR) is a 16-bit memory-mapped register that contains the control bits of the I2C module.

In slave mode, the start condition bit (STT) is not required to be set; however, it is recommended that the STT bit is set to 1 to transmit or receive data. The I2C slave recognizes a START condition and slave address, and then the transmit or receive data.

Figure 7–21. I2C Mode Register (ICMDR)

15	14	13	12	11	10	9	8
reserved	FREE	STT	IDLEEN	STP	MST	TRX	XA
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	0
RM	DLB	IRS	STB	FDF	BC	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	

Note: R/W-x = Read/Write-Reset value

Table 7–12. I2C Mode Register (ICMDR) Field Values

Bit	*field*	*symval*	Value	Description
15	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
14	FREE			Free running bit is used to determine the state of the I2C when a breakpoint is encountered in the HLL debugger.
			0	If SCL is low, the I2C stops immediately and keeps driving SCL low whether the I2C is the master transmitter or receiver. If SCL is high, the I2C waits until SCL becomes low and then stops. If the I2C is a slave, it stops when the transmission/ receiving completes.
			1	I2C runs free.
13	STT			Start condition bit works in conjunction with the STP bit (master only mode). The STT and STP bits are configured to generate different transfer formats (see Table 7–13). Note that the STT and STP bits can be used to terminate the repeat mode.
			0	STT is reset to 0 by the hardware after the START condition has been generated.
			1	STT is set to 1 by the DSP to generate a START condition. In master mode, setting STT to 1 generates a START condition. In slave mode, setting STT to 1 enables the I2C.
12	IDLEEN			Idle enable bit.
			0	The I2C does not go into an IDLE state when the DSP executes the IDLE instruction.
			1	The I2C does go into an IDLE state when the DSP executes the IDLE instruction.
11	STP			Stop condition bit works in conjunction with the STT bit (master only mode). The STT and STP bits are configured to generate different transfer formats (see Table 7–13). Note that the STT and STP bits can be used to terminate the repeat mode.
			0	STP is reset to 0 by the hardware after the STOP condition has been generated. The STOP condition is generated when ICCNT passes 0.
			1	STP is set to 1 by the DSP to generate a STOP condition.

Table 7–12. I2C Mode Register (ICMDR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
10	MST			Master bit determines if the I2C is in the slave mode or master mode. See Table 7–14. This bit is cleared when the I2C master detects a STOP condition.
			0	The I2C is in the slave mode and the clock is received from the master device.
			1	The I2C is in the master mode and it generates the clock. The MST bit works in conjunction with the TRX bit to determine the direction of data transmission of the I2C.
9	TRX			Transmitter bit determines the direction of data transmission of the I2C. See Table 7–14.
			0	I2C is in receiver mode and data on data line SDA is shifted into ICDRR.
			1	I2C is in the transmitter mode and the data in ICDXR is shifted out on data line SDA.
8	XA			Expanded address enable bit.
			0	7-bit address mode (normal address mode)
			1	10-bit address mode (expanded address mode)
7	RM			Repeat mode enable bit. This bit is a don't care if the I2C is configured in slave mode (MST = 0). See Table 7–13.
			0	I2C is not in the repeat mode.
			1	I2C is in the repeat mode. Data is continuously transmitted out of ICDXR until the STP bit is set to 1, regardless of the value in ICCNT.
6	DLB			Digital loop back mode enable bit disables or enables the digital loopback mode of the I2C.
			0	Digital loop back mode is disabled.
			1	Digital loop back mode is enabled in master transmit mode only. In this mode, data transmitted out of ICDXR is received in ICDRR after ((DSP freq/I2C freq) $\times$ 8) DSP cycles by an internal path. The address of ICOAR is output on SDA.

Table 7–12. I2C Mode Register (ICMDR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
5	IRS			I2C reset enable bit. When this bit is cleared to 0, all status bits are set to default values.
			0	I2C is in reset.
			1	I2C is out of reset.
4	STB			START byte mode enable bit (master only mode).
			0	I2C is not in START byte mode.
			1	I2C is in START byte mode (ICSAR = 0000 0001).
3	FDF			Free data format enable bit (both master and slave modes). See Table 7–14.
			0	I2C is not in free data format mode.
			1	I2C is in free data format mode.
2–0	BC		0–7	Bit count bits define the number of bits starting from the LSB (excluding the acknowledge bit) of the next byte that is to be received or transmitted.

Table 7–13. I2C Module Condition, Bus Activity, and Mode

ICMDR Bit					
RM	STT	STP	Condition	Bus Activities†	Mode
0	0	0	Idle	None	NA
0	0	1	Stop	P	NA
0	1	0	(Re)Start	S-A-D..(n)..D	Repeat n
0	1	1	(Re)Start-Stop	S-A-D..(n)..D-P	Repeat n
1	0	0	Idle	none	NA
1	0	1	Stop	P	NA
1	1	0	(Re)Start	S-A-D-D-D…..	Continuous
1	1	1	Reserved	None	NA

† P = STOP condition; S = START condition; A = Acknowledge bit; D = data

Table 7–14. I2C Module Operating Modes

ICMDR Bit			
FDF	**MST**	**TRX**	**Operating Mode**
0	0	x	Slave not in free data format mode
0	1	0	Master receive not in free data format mode
0	1	1	Master transmit not in free data format mode
1	0	0	Slave receiver in free data format mode
1	0	1	Slave transmitter in free data format mode
1	1	0	Master receiver in free data format mode
1	1	1	Master transmitter in free data format mode

7.4.11 *I2C Interrupt Vector Register (ICIVR)*

The I2C interrupt vector register (ICIVR) is a 16-bit memory-mapped register used by the DSP to determine which interrupt has occurred.

Figure 7–22. I2C Interrupt Vector Register (ICIVR)

15 12	11 8	7 3	2 0
reserved	TESTMD	reserved	INTCODE
R-0	R/W-0	R-0	R-0

Note: R/W-x = Read/Write-Reset value

Table 7–15. I2C Interrupt Vector Register (ICIVR) Field Values

Bit	*field*	*symval*	Value	Description
15–12	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
11–8	TESTMD			Reserved for internal testing.
7–3	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.

Table 7–15. I2C Interrupt Vector Register (ICIVR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
2–0	INTCODE			Interrupt code bits. The binary-coded-interrupt vector indicates which interrupt has occurred. Reading ICIVR clears the interrupt flag; if other interrupts are pending, a new interrupt is generated. If there is more than one interrupt flag, reading ICIVR clears the highest priority interrupt flag.
			000	None
			001	Arbitration-lost interrupt
			010	No-acknowledgement interrupt
			011	Register-access-ready interrupt
			100	Receive-data-ready interrupt
			101	Transmit-data-ready interrupt
			110–111	Reserved

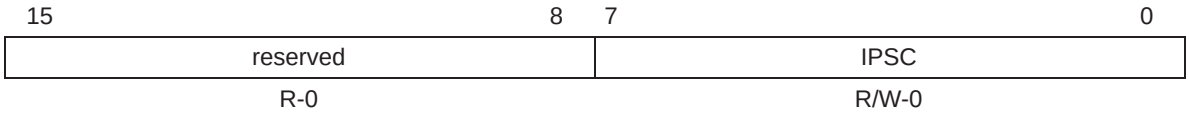
7.4.12 *I2C General-Purpose I/O Register (ICGPIO)*

The I2C general-purpose I/O register (ICGPIO) is a 16-bit memory-mapped register used by the I2C to control the SDA and SCL pins when the module is not operating in I2C mode.

7.4.13 *I2C Prescaler Register (ICPSC)*

The I2C prescaler register (ICPSC) is a 16-bit memory-mapped register used for dividing down the system clock (SYSCLK) to obtain an approximately 12 MHz clock for the I2C module.

Figure 7–23. I2C Prescaler Register (ICPSC)



Note: R/W-x = Read/Write-Reset value

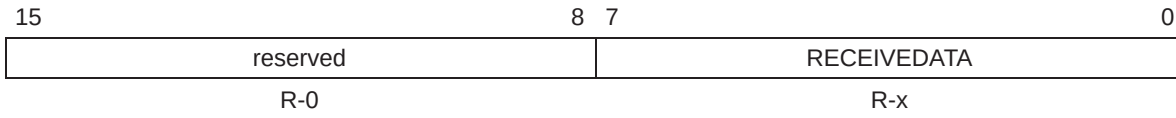
Table 7–16. I2C Prescaler Register (ICPSC) Field Values

Bit	*field*	*symval*	Value	Description
15–8	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
7–0	IPSC			I2C prescaler counter bits.

7.4.14 *I2C Data Receive Shift Register (ICRSR)*

The I2C data receive shift register (ICRSR) is a 16-bit memory-mapped register used by the I2C module to shift in the receive data. This register is not accessible by the DSP.

Figure 7–24. I2C Data Receive Shift Register (ICRSR)

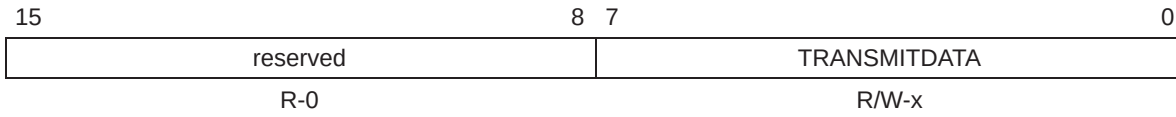


Note: R/W-x = Read/Write-Reset value

7.4.15 *I2C Data Transmit Shift Register (ICXSR)*

The I2C data transmit shift register (ICXSR) is a 16-bit memory-mapped register used by the I2C module to shift out the transmit data. This register is not accessible by the DSP.

Figure 7–25. I2C Data Transmit Shift Register (ICXSR)



Note: R/W-x = Read/Write-Reset value

7.5 I2C Module Programming Examples

7.5.1 Main Program

7.5.1.1 State After Reset

1) Program the prescaler to obtain an approximately 12 Mhz I2C module clock (ICPSC = x; this value is to be calculated and is dependent on the CPU frequency).
2) Take the I2C module out of reset (IRS = 1).
   a) If using interrupt for transmit/receive data, enable interrupt masks.
   b) If using DMA for transmit/receive data, enable the DMA and program the DMA controller.

7.5.1.2 Initialization Procedure

Configure the I2C mode register (ICMDR) bits.

7.5.1.3 Program Clock Control Registers (ICCLKL and ICCLKH)

Program the I2C clock to obtain 100 Kbps or 400 Kbps (ICCLKL = x and ICCLKH = x; these values are to be calculated and are dependent on the CPU frequency).

7.5.1.4 Configure Address Registers

1) Configure its own address (ICOAR = x)
2) Configure the slave address (ICSAR = x)

7.5.1.5 Program Transmit Data Register (ICDXR)

If in master transmitter mode, program the data transmit register (ICDXR = x).

7.5.1.6 Configure Status and Mode Register (ICSTR)

Poll the bus busy (BB) bit in the I2C status register (ICSTR), if it is cleared to 0 (bus not busy), configure START/STOP condition to initiate a transfer.

7.5.1.7 Poll Receive Data

Poll the receive data ready interrupt flag bit (ICRRDY) in the I2C status register (ICSTR), use the ICRRDY interrupt, or use the DMA to read the receive data in the data receive register (ICDRR).

7.5.1.8 Poll Transmit Data

Poll the transmit data ready interrupt flag bit (ICXRDY) in the I2C status register (ICSTR), use the ICXRDY interrupt, or use the DMA to write data into the data transmit register (ICDXR).

7.5.2 Interrupt Subroutines

1) Test for arbitration lost and resolve accordingly.
2) Test for no-acknowledge and resolve accordingly.
3) Test for register access ready and resolve accordingly.
4) Test for receive data and resolve accordingly.
5) Test for transmit data and resolve accordingly.

7.5.3 Flow Diagrams

Figure 7–26. Setup Procedure

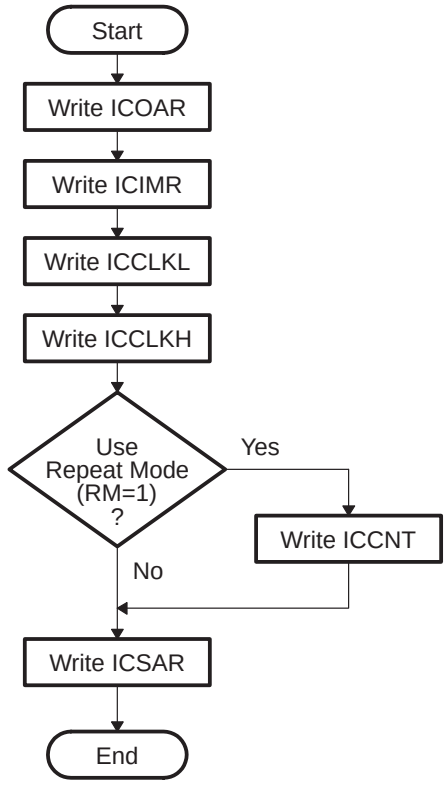


Figure 7–27. Master Transmitter Mode, RM = 1

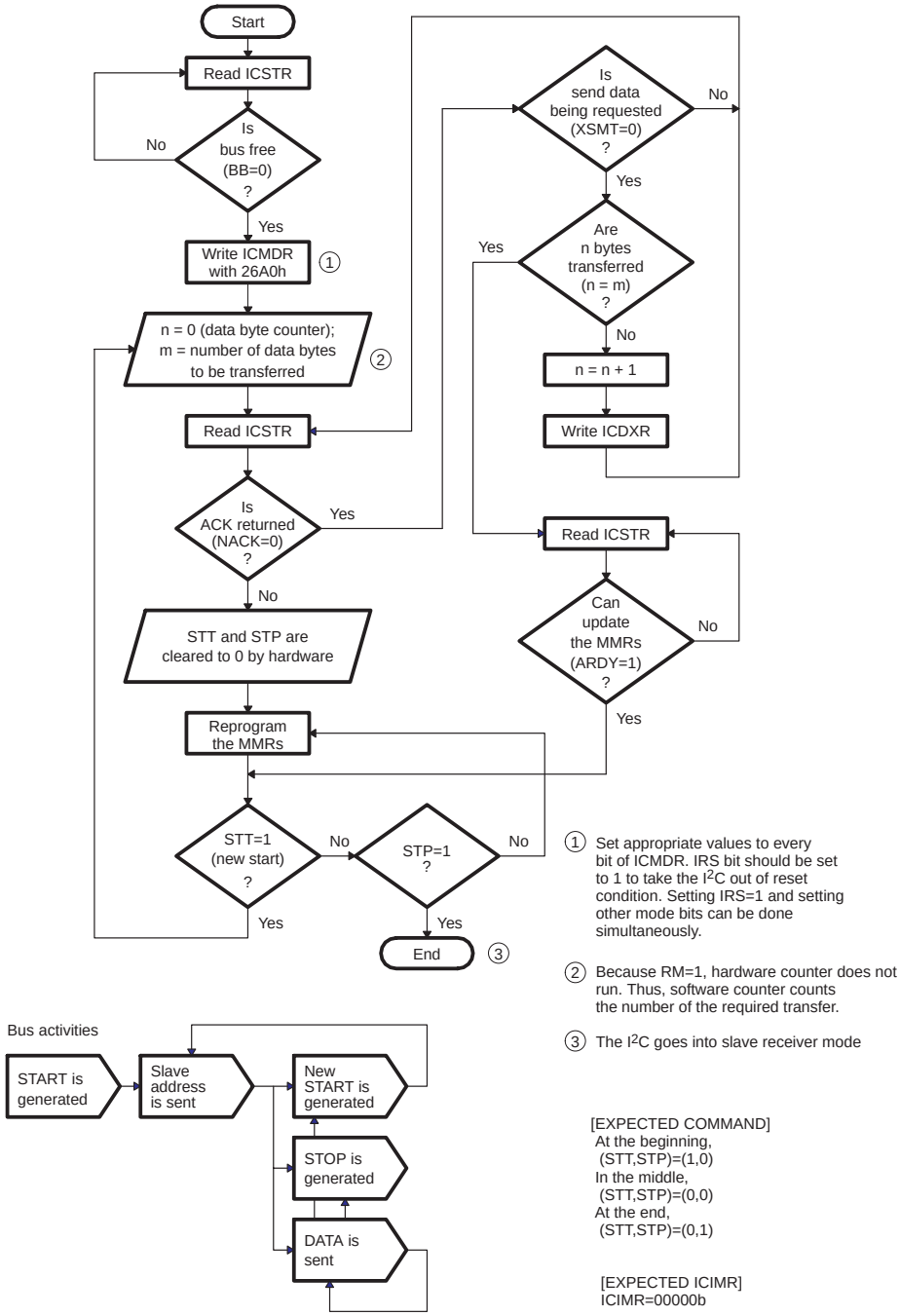
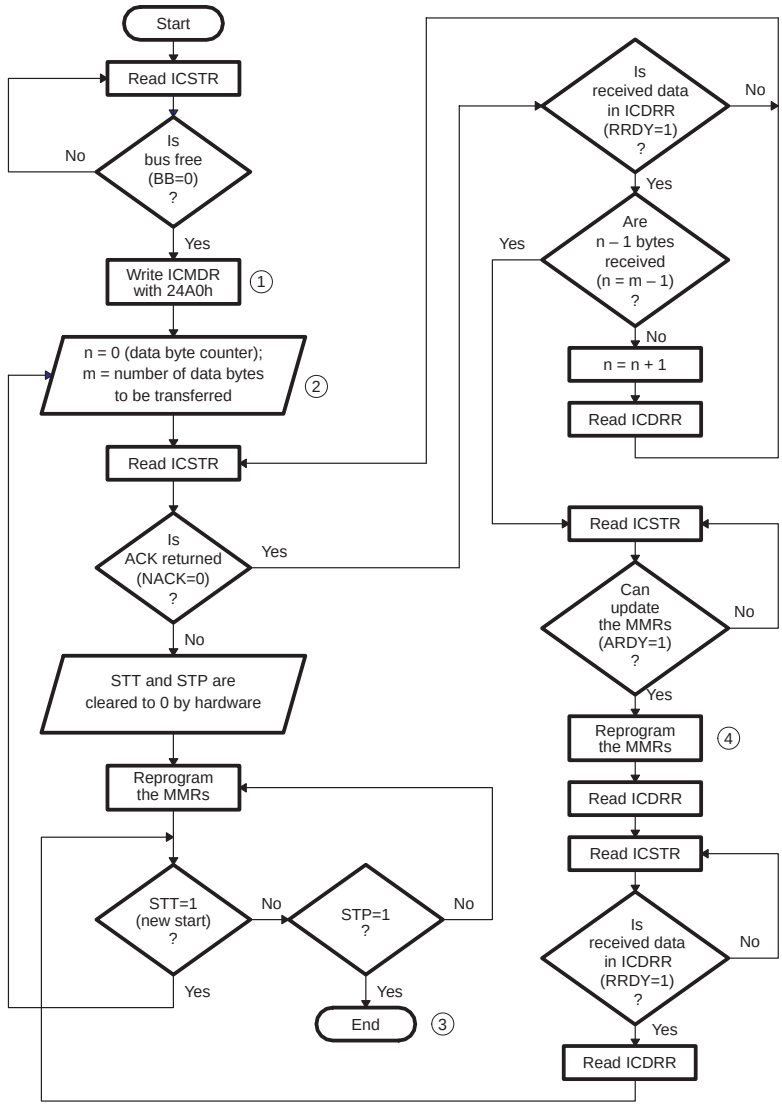


Figure 7–28. Master Receiver Mode, RM = 1, Polling 1 (Using Software Counter, Number of the Receive Data is Fixed)

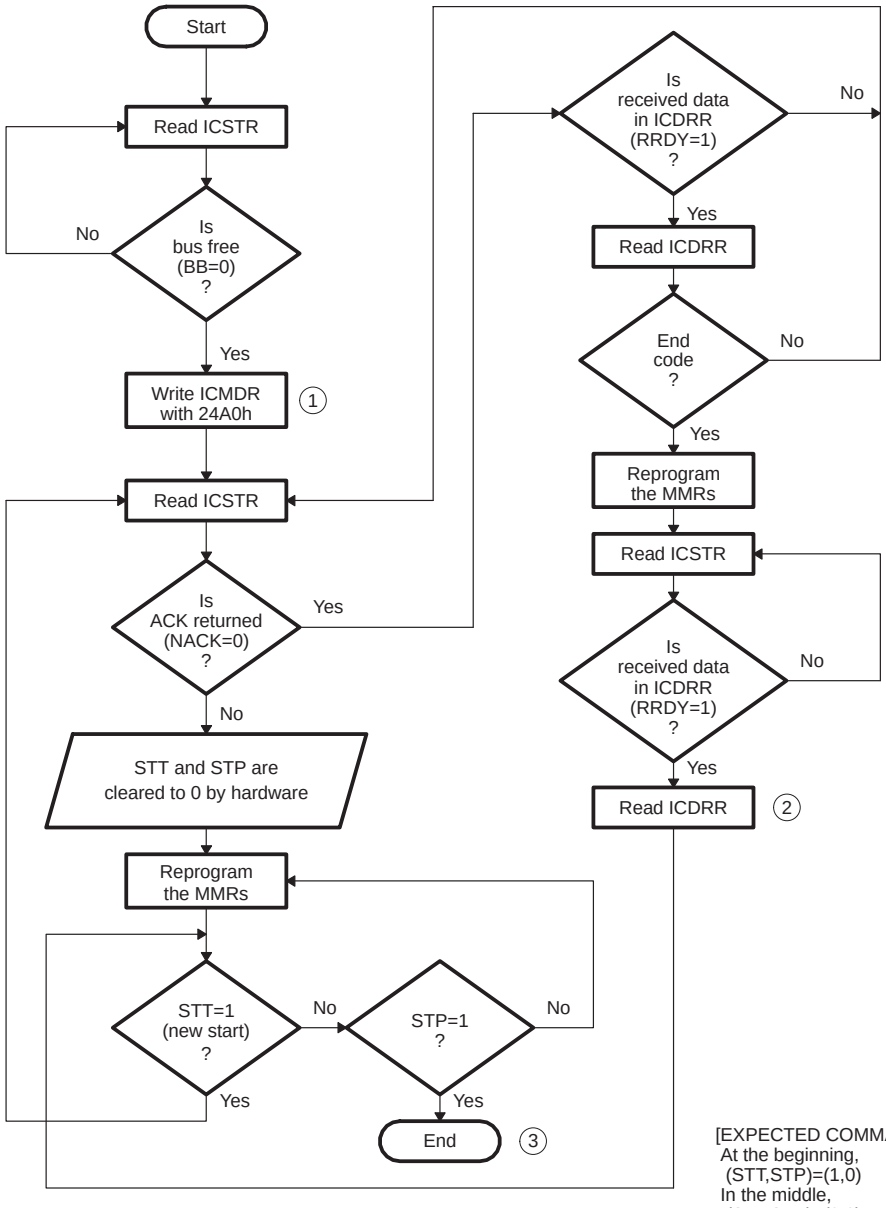


1. Set appropriate values to every bit of ICMDR. IRS bit should be set to 1 to take the I²C out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.
2. Because RM=1, hardware counter does not run. Thus, software counter counts the number of the required transfer.
3. The I²C goes into slave receiver mode
4. Set STP=1

[EXPECTED COMMAND]
At the beginning,
(STT,STP)=(1,0)
In the middle,
(STT,STP)=(0,0)
At the end,
(STT,STP)=(0,1)

[EXPECTED ICIMR]
ICIMR=00000b

Figure 7–29. Master Receiver Mode, RM = 1, Polling 2 (Number of the Receive Data is Variable, Data Contents Dependent)



(1) Set appropriate values to every bit of ICMDR. IRS bit should be set to 1 to take the I²C out of reset condition. Setting IRS=1 and setting other mode bits can be done simultaneously.

(2) Dummy read. The contents of this read data has no meaning.

(3) The I²C goes into slave receiver mode

Figure 7–30. Master Transmitter Mode, RM = 0, Polling

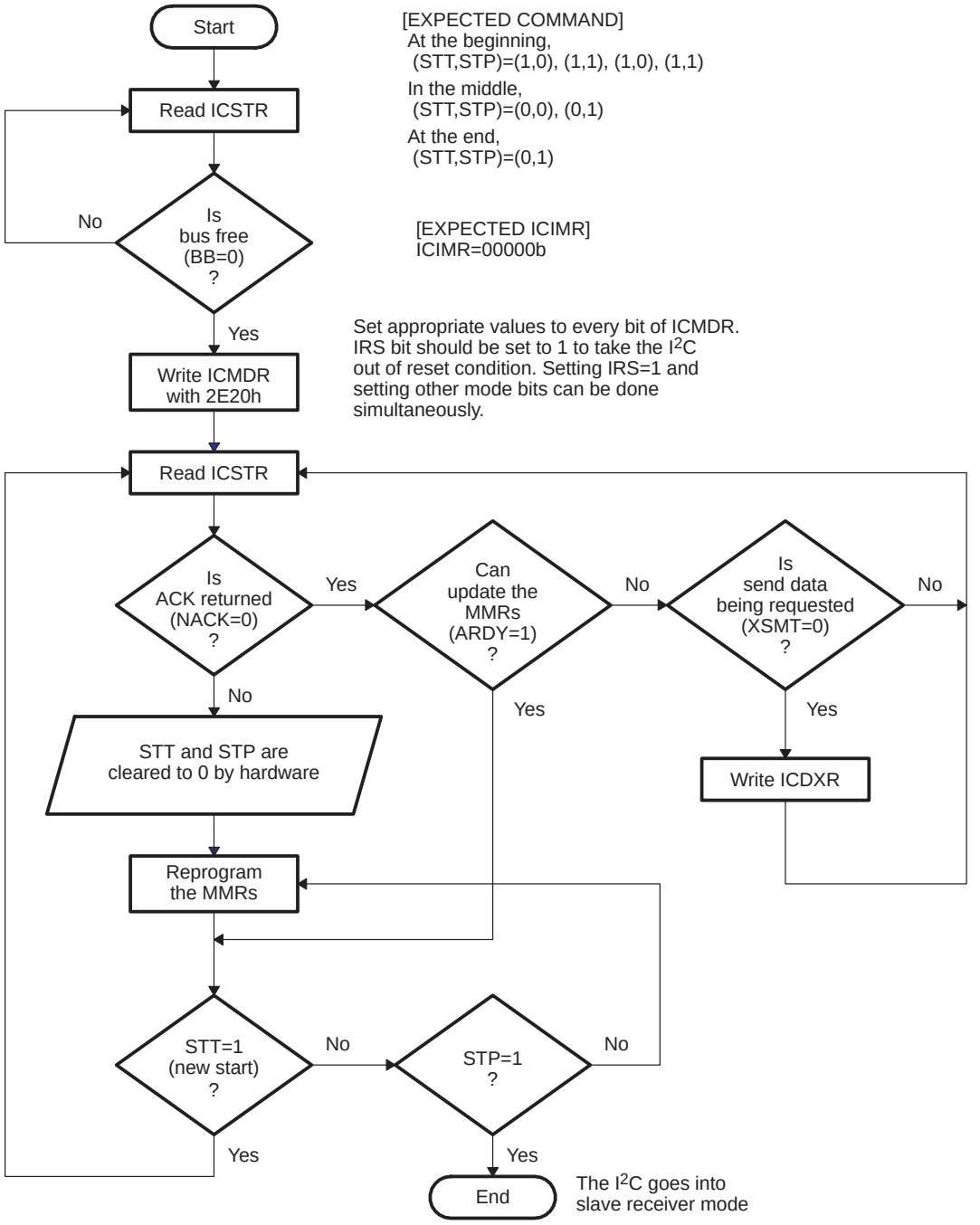


Figure 7–31. Master Receiver Mode, RM = 0, Polling

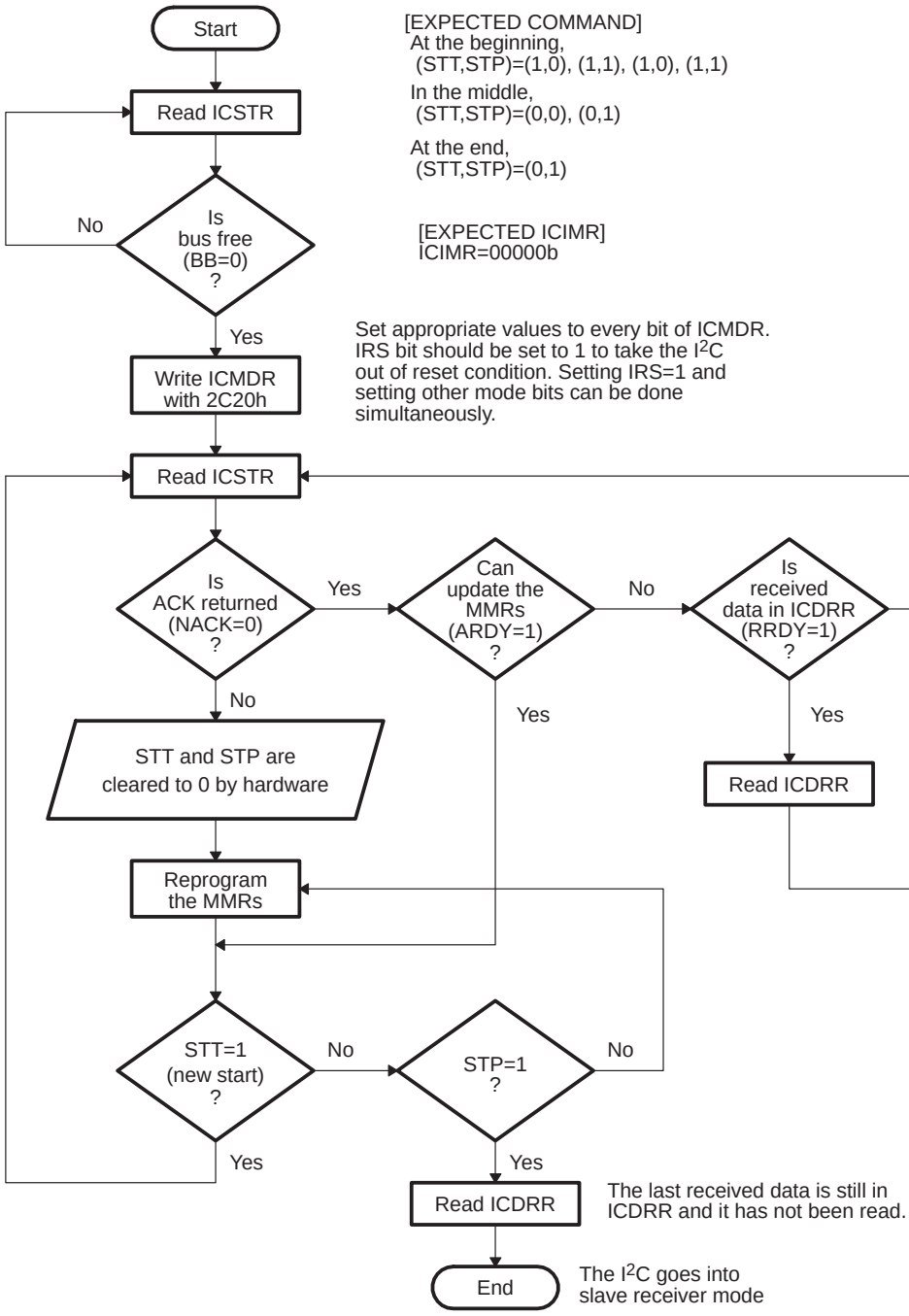


Figure 7–32. Master Transmitter Mode, RM = 0, Interrupt

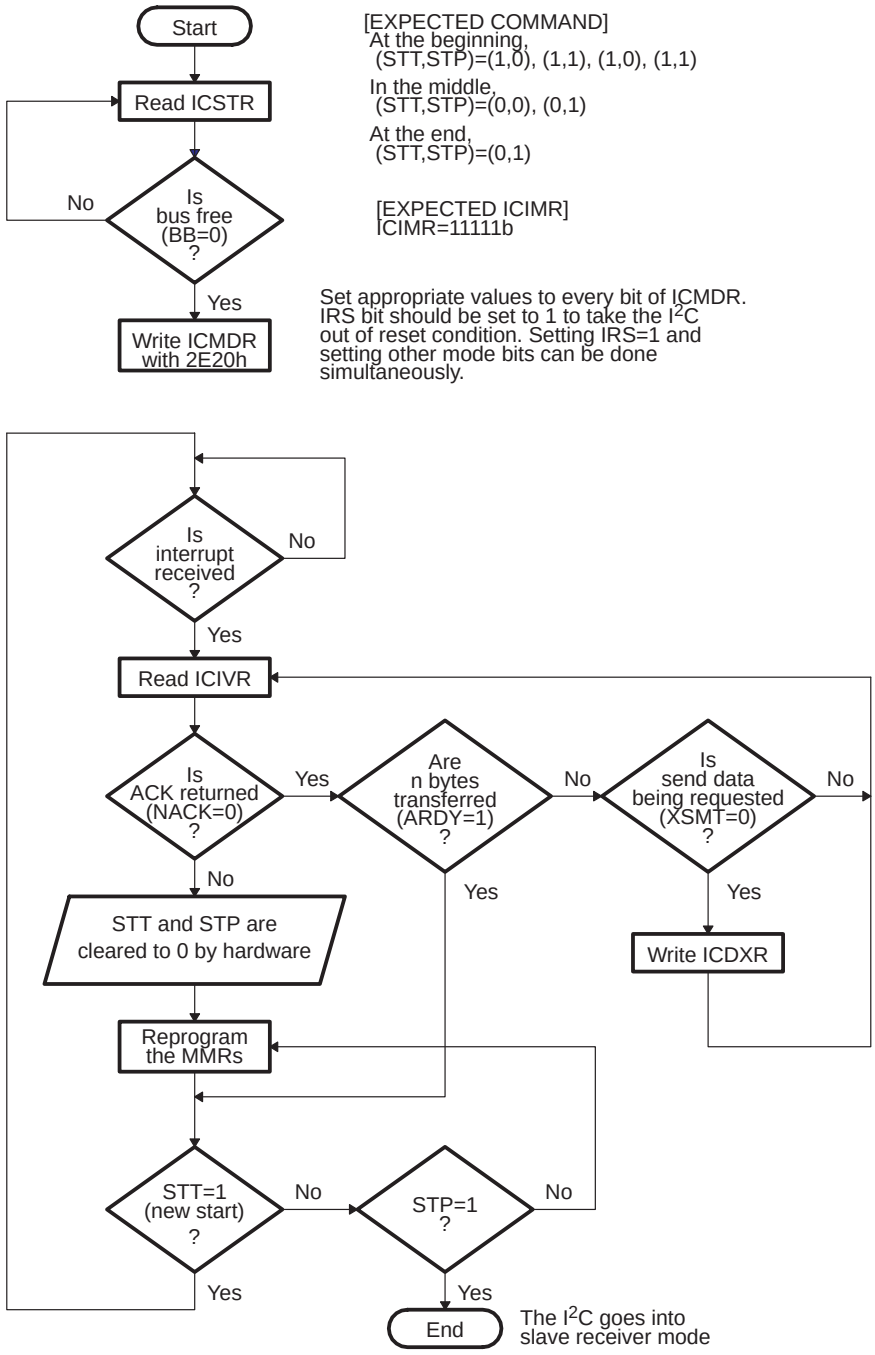


Figure 7–33. Master Receiver Mode, RM = 0, Interrupt

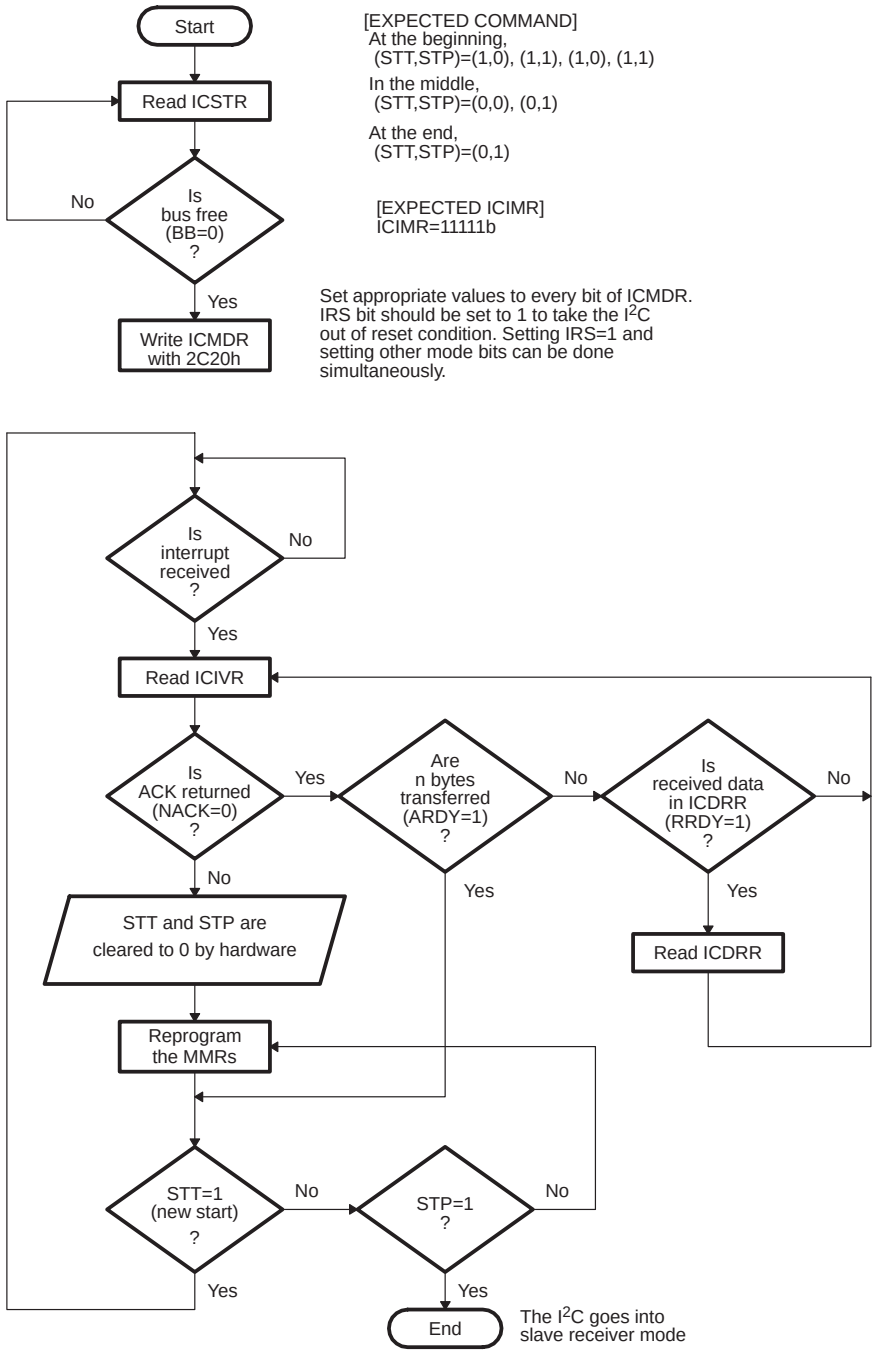


Figure 7–34. Master Transmitter/Receiver Mode, RM = 1, Interrupt

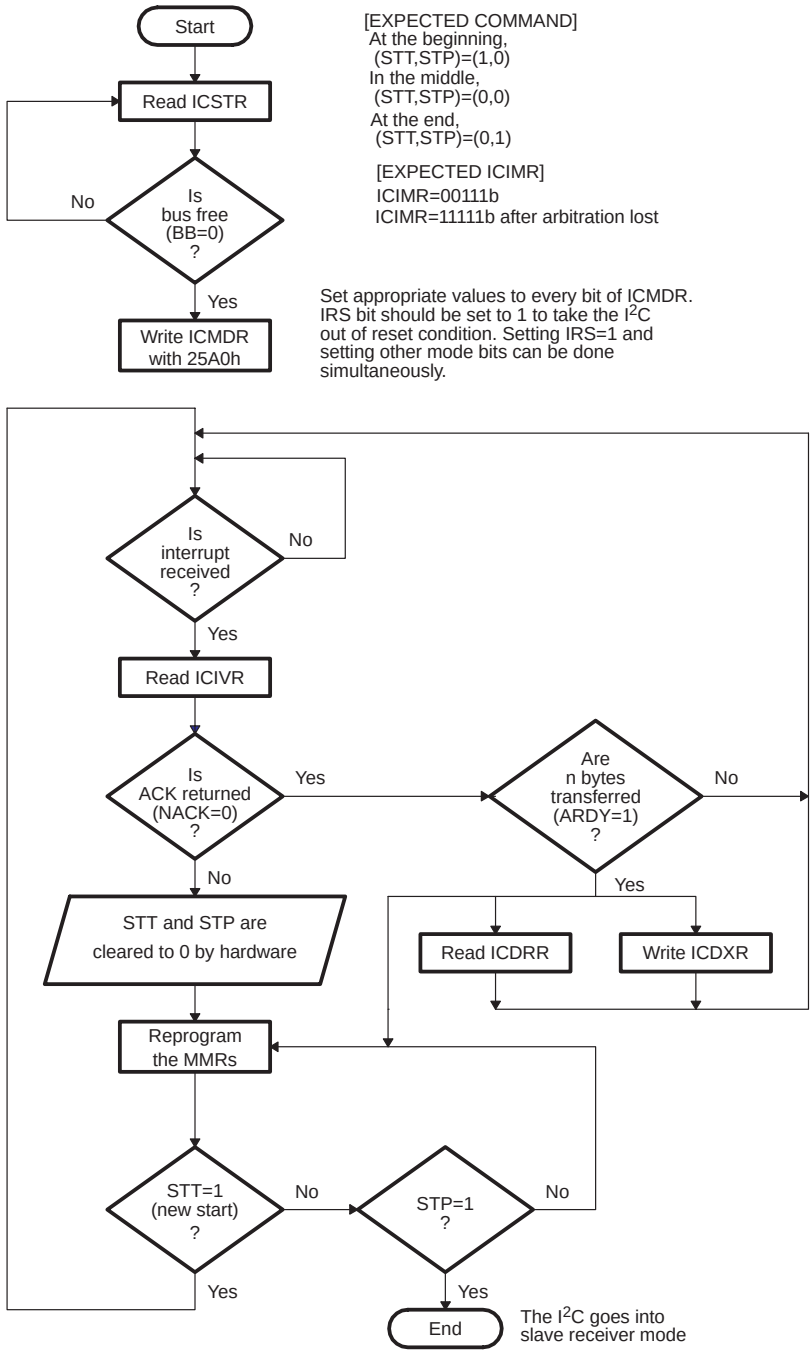


Figure 7–35. Master Transmitter Mode, RM = 0, DMA

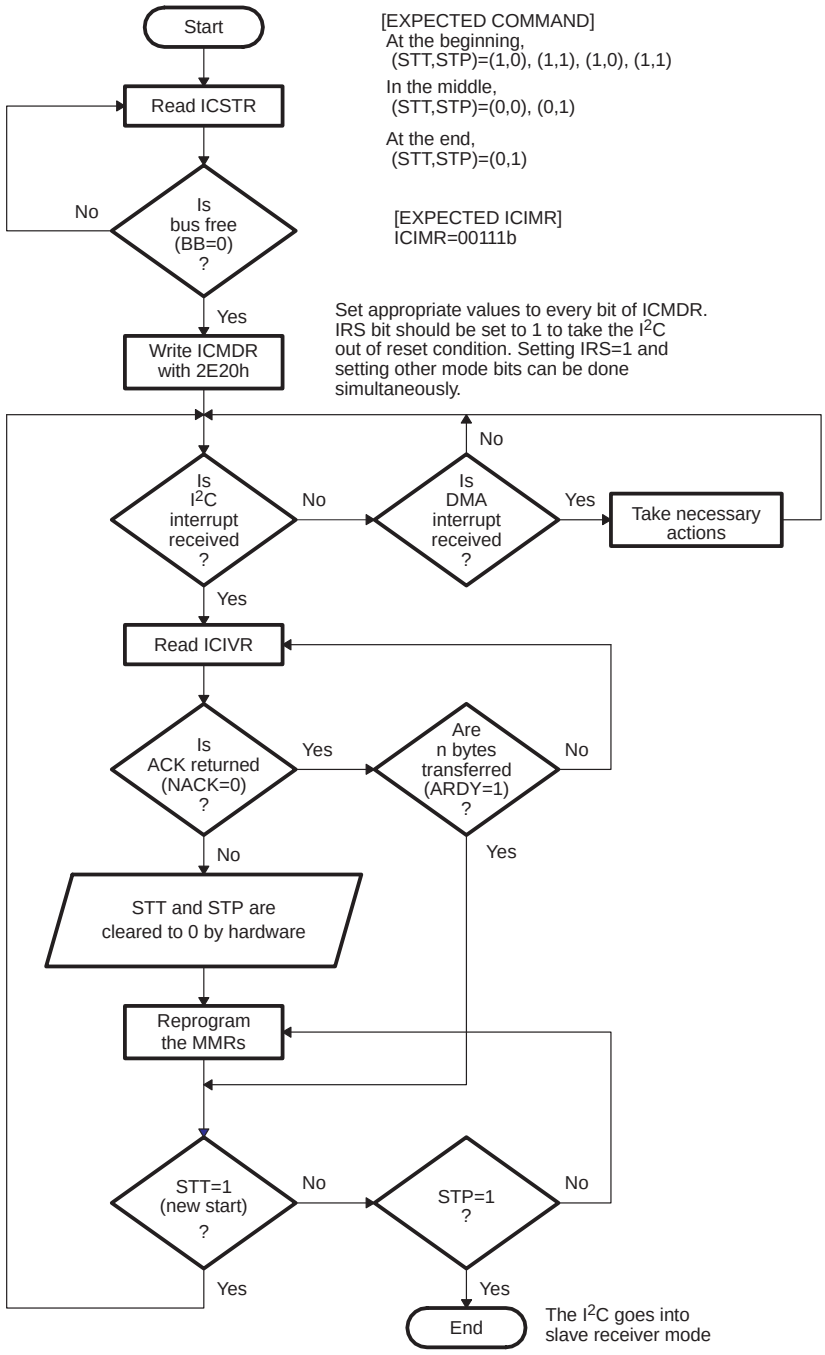


Figure 7–36. Master Receiver Mode, RM = 0, DMA

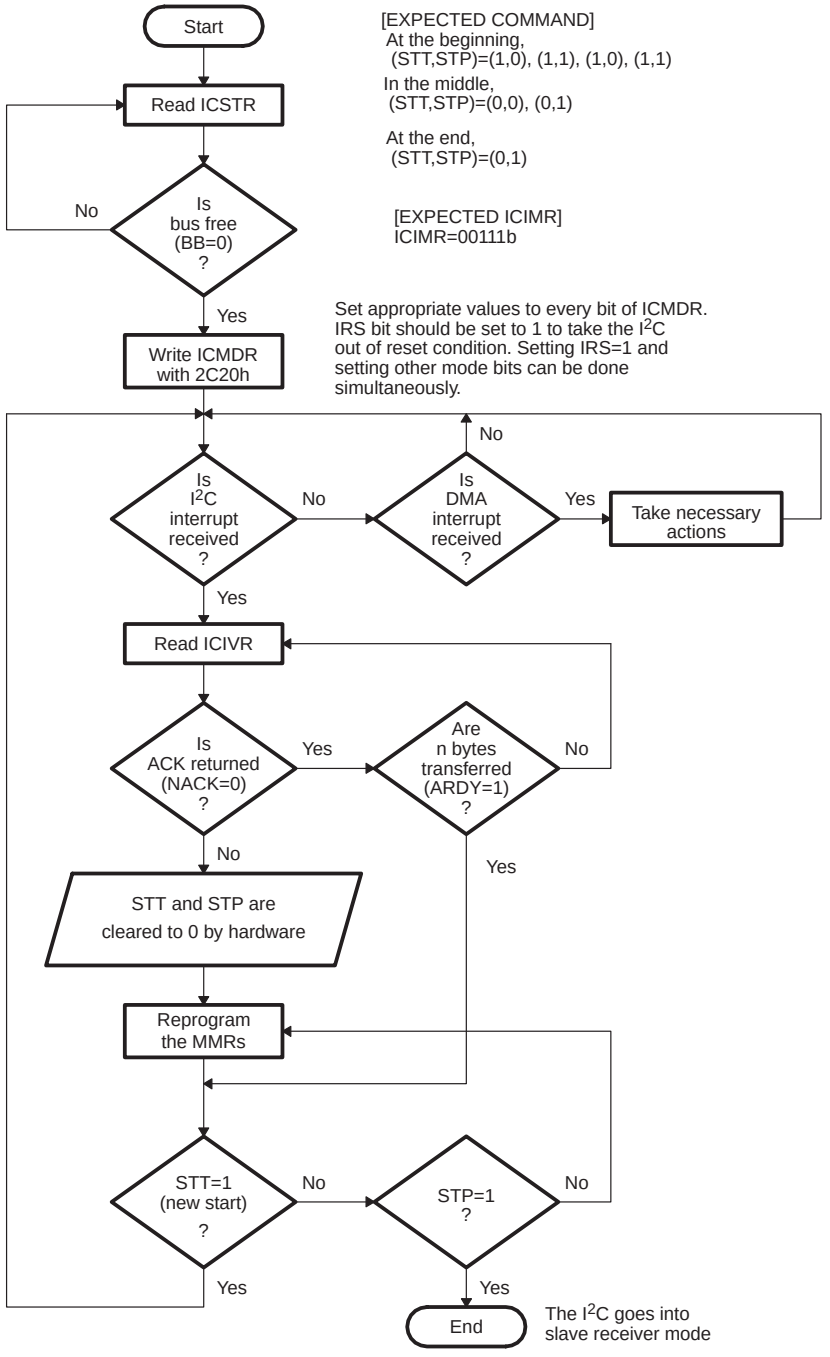


Figure 7–37. Slave Transmitter/Receiver Mode, RM = 1, Polling

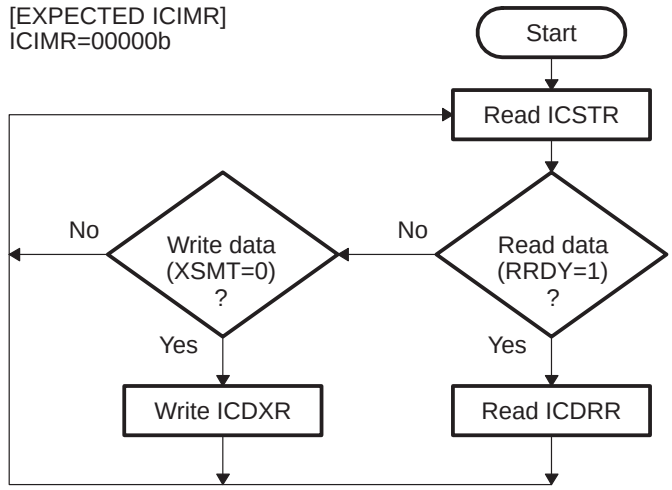
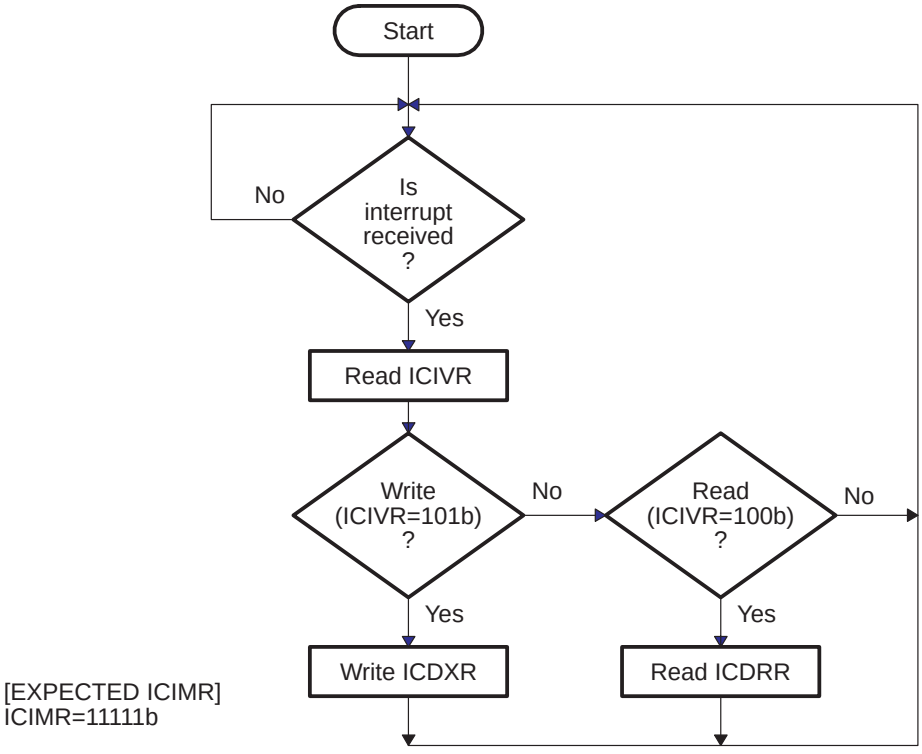


Figure 7–38. Slave Transmitter/Receiver Mode, RM = 1, Interrupt



Chapter 8

Idle Configurations

The TMS320C55x™ (C55x™) DSP is divided into the idle domains described in section 8.1. To minimize power consumption, you can choose which domains are active and which domains are idle at any given time. The current state of all domains is collectively called the *idle configuration*.

Topic **Page**

8.1 Idle Domains

The DSP is divided into the idle domains described in Table 8–1. You can control which of these idle domains are active and which are idle at any given time, as described in section 8.2.

Notes:

1) The peripheral bus controller and the enhanced host port interface (EHPI) on the DSP are not part of any idle domain. The only way to turn these modules off is to put the clock generator into its idle mode (make the CLKGEN domain idle).

2) The internal memory blocks (SARAM and DARAM) and the external memory are shared by two domains (CPU and DMA). When both domains are idle, memory accesses are disabled.

Table 8–1. Idle Domains in the DSP

Domain	Contents of the Domain	Configurability
CPU	CPU and CPU buses	When the idle instruction is executed, the CPU remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.
DMA	DMA controller and DMA buses	When the idle instruction is executed, the DMA (direct memory access) controller remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.
CACHE	Instruction cache	When the idle instruction is executed, the instruction cache remains active or becomes idle, depending on the chosen idle configuration. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.
PERIPH	Timers, serial ports, and other peripherals	Each of the peripherals in this domain has an idle enable bit that determines whether the peripheral can be placed in its idle mode when the idle instruction is executed. If the PERIPH domain is configured to be idle and an idle enable bit is 1, the corresponding peripheral is placed in its idle mode. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.

Table 8–1. Idle Domains in the DSP (Continued)

Domain	Contents of the Domain	Configurability
CLKGEN	Clock generator, including the phase-lock loop (PLL) circuitry	When the idle instruction is executed, the clock generator remains active or becomes idle, depending on the chosen idle configuration. When the clock generator is in its idle mode, no clocking is available for the CPU or the DMA controller. If the clock generator is configured to be idle and the CPU, the DMA controller, or the cache is configured to be active, a bus error interrupt request is sent to the CPU. If properly enabled, the interrupt will force the CLKGEN domain to be reactivated. Peripherals that do not depend on the DSP clock signal are not affected by the state of the CLKGEN domain. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.
EMIF	External memory interface (EMIF)	When the idle instruction is executed, the EMIF is disabled or enabled, depending on the chosen idle configuration. Regardless of this domain's state before a DSP reset, it is active after a DSP reset.

8.2 Idle Configuration Process

The idle configuration indicates which idle domains will be idle, and which idle domains will be active, the next time the idle instruction is executed. The basic steps to the idle configuration process are:

1) Define a new idle configuration by writing to the bits in the idle configuration register (ICR). Make sure that you use a valid idle configuration (see section 8.3).

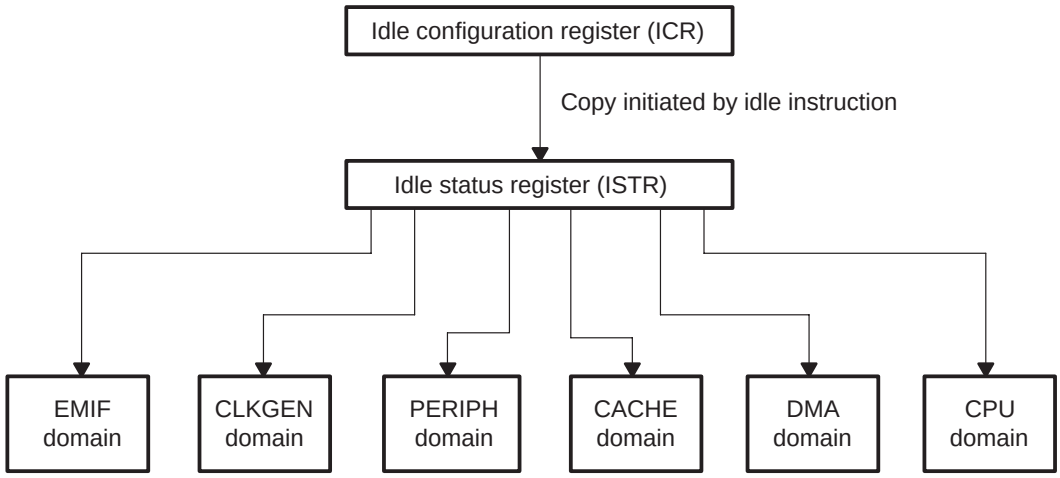
2) Apply the new idle configuration by executing the idle instruction. The effects are shown in the following figure. The content of ICR is copied to the idle status register (ISTR). The bits of ISTR are then propagated through the system to enable or disable each of the chosen domains.

The idle instruction cannot be executed in parallel with another instruction.

Notes:

If you intend to switch among multiple idle configurations, make sure that your system has the means to change from one idle configuration to the next. For important considerations, see section 8.4.

Figure 8–1. Idle Configuration Process



8.3 Valid Idle Configurations

Not all of the values that you can write to the idle configuration register (ICR) provide valid idle configurations. The valid configurations are limited by dependencies within the system. For example, when the CLKGEN domain is idle (the DSP clock generator is disabled), the DMA controller, the CPU, and any peripherals that do not have their own external clocks cannot operate.

8.4 To Change Idle Configurations (Key Conditions)

Before you use the idle instruction, make sure that there is a method for the DSP to change the idle configuration afterward. Table 8–2 summarizes the methods available under three key conditions. The table also describes the effects on the idle registers: the idle status register (ISTR) and the idle configuration register (ICR). For more details about these idle registers, see section 8.7 on page 8-9.

Table 8–2. Changing Idle Configurations

Condition	Available Methods For Changing Idle Configuration	ISTR After Change	ICR After Change
1. CLKGEN domain is active CPU domain is active (See section 8.4.1)	A. Write a new configuration to the idle configuration register (ICR), and then execute the idle instruction.	A. Modified by the idle instruction; contains a copy of the new ICR value	A. Contains the new value that was loaded by the program
	B. Initiate a DSP hardware reset.	B. Cleared (all 0s)	B. Cleared (all 0s)
2. CLKGEN domain is active CPU domain is idle (See section 8.4.2)	A. Use an unmasked hardware interrupt or the nonmaskable hardware interrupt called NMI_.	A. CLKGENIS and CPUIS bits are 0. No other bits were modified.	A. Not modified
	B. Initiate a DSP hardware reset.	B. Cleared (all 0s)	B. Cleared (all 0s)
3. CLKGEN domain is idle (See section 8.4.3)	A. Use an unmasked hardware interrupt or the nonmaskable hardware interrupt called NMI_.	A. CLKGENIS and CPUIS bits are 0. No other bits were modified.	A. Not modified
	B. Initiate a DSP hardware reset.	B. Cleared (all 0s)	B. Cleared (all 0s)

8.4.1 Condition 1: CLKGEN and CPU Domains Active

When the CLKGEN domain is active (the DSP clock generator is enabled) and the CPU domain is active (the DSP CPU is running), program flow continues. In this case, there are two methods of changing idle configurations:

- Write a new idle configuration to the idle configuration register (ICR), and then execute the idle instruction. The idle instruction copies the content of the ICR to the idle status register (ISTR), and the ISTR bit values are propagated to the idle domains. After the domains change states, the value in ISTR matches the value in ICR.
- Initiate a DSP hardware reset at the DSP reset pin. When the DSP resets, all domains are made active.

8.4.2 Condition 2: CLKGEN Domain Active, CPU Domain Idle

When the CPU domain is idle, program flow is halted. It is not possible to write a new value to the idle configuration register (ICR) or to execute the idle instruction. Two other methods are available for changing the idle configuration:

- ❏ Use an unmasked interrupt or the nonmaskable interrupt called NMI_. The interrupt clears the CLKGENIS and CPUIS bits of the idle status register (ISTR). The change to CPUIS reactivates the CPU domain, and the change to CLKGENIS ensures that the CLKGEN domain is also active. The content of the idle configuration register (ICR) is not modified. To learn how the CPU responds to the interrupt, see section 8.5.
- ❏ Initiate a DSP hardware reset at the DSP reset pin. When the DSP resets, all domains are made active.

Once program flow has begun again, you can reactivate or deactivate other domains by writing a new idle configuration to ICR and then executing the idle instruction.

8.4.3 Condition 3: CLKGEN Domain Idle

When the CLKGEN domain is idle (the DSP clock generator is disabled), no internal clocks are active, including the CPU clock. With the CPU halted, it is not possible to write a new value to the idle configuration register (ICR) or to execute the idle instruction. Two other methods are available for waking the DSP:

- ❏ Use an unmasked interrupt or the nonmaskable interrupt called NMI_. The interrupt clears the CLKGENIS and CPUIS bits of the idle status register (ISTR). The change to CPUIS reactivates the CPU domain, and the change to CLKGENIS ensures that the CLKGEN domain is also active. The content of the idle configuration register (ICR) is not modified. To learn how the CPU responds to the interrupt, see section 8.5.
- ❏ Initiate a DSP hardware reset at the DSP reset pin. When the DSP resets, all domains are made active.

Once program flow has begun again, you can reactivate or deactivate other domains by writing a new idle configuration to ICR and then executing the idle instruction.

8.5 Interrupt Handling When the CPU Is Reactivated

If the CPU has been halted by an idle configuration, it can be reactivated by a nonmaskable interrupt (NMI_ or RESET_) or by a maskable interrupt that is enabled in an interrupt enable register (IER0 or IER1). A maskable interrupt request will also set the corresponding interrupt flag bit in an interrupt flag register (IFR0 or IFR1). Table 8–3 summarizes how the CPU responds after being reactivated by maskable and nonmaskable interrupts. INTM is the global interrupt mask bit in status register ST1_55.

Table 8–3. CPU Response After Reactivation

Interrupt	CPU Response After Reactivation ...
A maskable interrupt	If INTM = 0: The CPU executes the interrupt service routine, executes the instruction that follows the idle instruction, and continues from there. If INTM = 1: The CPU executes the instruction that follows the idle instruction and then continues from there. The interrupt service routine cannot be executed until the interrupt has been enabled by INTM.
NMI_ (nonmaskable)	The CPU executes the interrupt service routine, executes the instruction that follows the idle instruction, and continues from there.
RESET_ (nonmaskable)	The DSP is reset. (During a DSP reset, all idle domains are made active.)

8.6 Effect of a DSP Reset on the Idle Domains

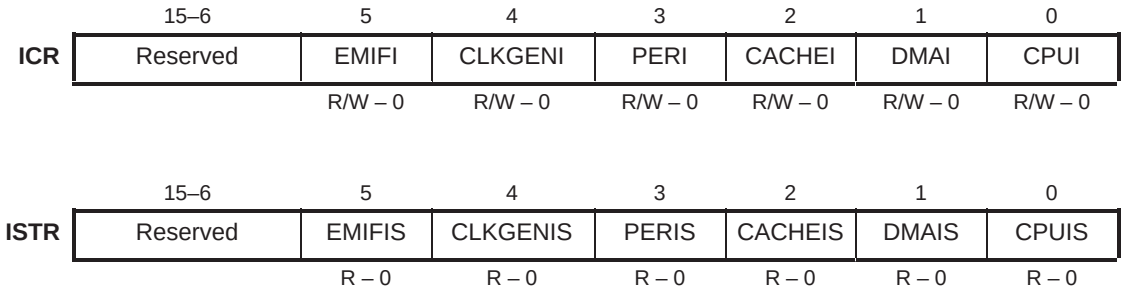
Driving the DSP reset signal low starts a DSP reset. During a DSP reset, all idle domains are made active.

8.7 Idle Registers

Two registers provide the means for you to individually configure and monitor each of the idle domains: the idle configuration register (ICR) and the idle status register (ISTR). These registers (see Figure 8–2) are part of the peripheral bus controller and are accessible to the CPU. Table 8–4 describes the read/write bits of ICR, and Table 8–5 describes the read-only bits of ISTR. For information about controlling the idle domains, see Chapter 8, *Idle Configurations*.

ICR lets you configure how each idle domain will respond the next time the idle instruction is executed. When you execute the idle instruction, the content of ICR is copied to ISTR. Then the ISTR values are propagated to the idle domains. Making the clock generator idle (CLKGENI = 1) provides the lowest level of power consumption in the DSP by stopping all the system clocks.

Figure 8–2. Idle Configuration Register (ICR) and Idle Status Register (ISTR)



Legend:

R	Read access
W	Write access
– 0	This bit is cleared by a DSP reset.
Reserved	A write to this bit field has no effect, and the bit or bits in this field always appear as 0s during read operations.

Table 8–4. ICR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–6	Reserved	These bits are not available for your use. They are read-only bits and return 0s when read.	–
5	EMIFI	EMIF-domain idle configuration bit. EMIFI determines whether the external memory interface (EMIF) will be idle after the next execution of the idle instruction: 0 EMIF will be active 1 EMIF will be idle	0
4	CLKGENI	CLKGEN-domain idle configuration bit. CLKGENI determines whether the DSP clock generator will be idle after the next execution of the idle instruction: 0 Clock generator will be active 1 Clock generator will be idle **Important:** For a proper power-down, when you set CLKGEN = 1, make sure you also set CPUI = 1, DMAI = 1, and CACHEI = 1. If you do not, a bus error interrupt (BERRINT) request is sent to the CPU.	0
3	PERI	PERIPH-domain idle configuration bit. Peripherals that are in the PERIPH domain each have an idle enable bit. PERI, in conjunction with the idle enable bits, determines which of the peripherals in the domain will be idle after the next execution of the idle instruction: 0 All peripherals in the domain will be active. 1 For each peripheral in the domain: If the idle enable bit is 1, the peripheral will be in its idle mode. If the idle enable bit is 0, the peripheral will be active.	0
2	CACHEI	CACHE-domain idle configuration bit. CACHEI determines whether the cache will be idle after the next execution of the idle instruction: 0 Cache will be active 1 Cache will be idle	0
1	DMAI	DMA-domain idle configuration bit. DMAI determines whether the DMA controller will be idle after the next execution of the idle instruction: 0 DMA controller will be active 1 DMA controller will be idle	0

Table 8–4. ICR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
0	CPUI	CPU-domain idle configuration bit. CPUI determines whether the DSP CPU will be idle after the next execution of the idle instruction: 0 CPU will be active 1 CPU will be idle	0

Table 8–5. ISTR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–6	Reserved	These bits are not available for your use. They are read-only bits and return 0s when read.	–
5	EMIFIS	EMIF-domain idle status bit. EMIFIS is a copy of EMIFI made during the execution of the idle instruction. EMIFIS reflects the current idle status of the external memory interface (EMIF): 0 EMIF is active 1 EMIF is idle	0
4	CLKGENIS	CLKGEN-domain idle status bit. CLKGENIS is a copy of CLKGENI made during the execution of the idle instruction. CLKGENIS reflects the current idle status of the DSP clock generator: 0 Clock generator is active 1 Clock generator is idle	0
3	PERIS	PERIPH-domain idle status bit. PERIS is a copy of PERI made during the execution of the idle instruction. PERIS reflects the current idle status of the peripherals in the PERIPH domain: 0 All peripherals in the domain are active 1 For each peripheral in the domain: If the idle enable bit is 1, the peripheral is idle. If the idle enable bit is 0, the peripheral is active.	0
2	CACHEIS	CACHE-domain idle status bit. CACHEIS is a copy of CACHEI made during the execution of the idle instruction. CACHEIS reflects the current idle status of the cache: 0 Cache is active 1 Cache is idle	0

Table 8–5. ISTR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
1	DMAIS	DMA-domain idle status bit. DMAIS is a copy of DMAI made during the execution of the idle instruction. DMAIS reflects the current idle status of the DMA controller: 0 DMA controller is active 1 DMA controller is idle	0
0	CPUIS	CPU-domain idle status bit. CPUIS is a copy of CPUI made during the execution of the idle instruction. CPUIS reflects the current idle status of the DSP CPU: 0 CPU is active 1 CPU is idle	0

Chapter 9

Multichannel Buffered Serial Port (McBSP)

This chapter describes the type of multichannel buffered serial port (McBSP) available on the TMS320C55x™ DSPs. The McBSPs provide a direct serial interface between a C55x™ DSP and other devices in a system. For the number of McBSPs available, see the data sheet for your C55x DSP.

Topic **Page**

9.1 Introduction to the McBSP

The TMS320C55x DSPs provide multiple high-speed, multichannel buffered serial ports (McBSPs) that allow direct interface to other C55x DSPs, codecs, and other devices in a system. For the number of McBSPs available, see the data sheet for your C55x DSP.

9.1.1 Key Features of the McBSP

The McBSP provides:

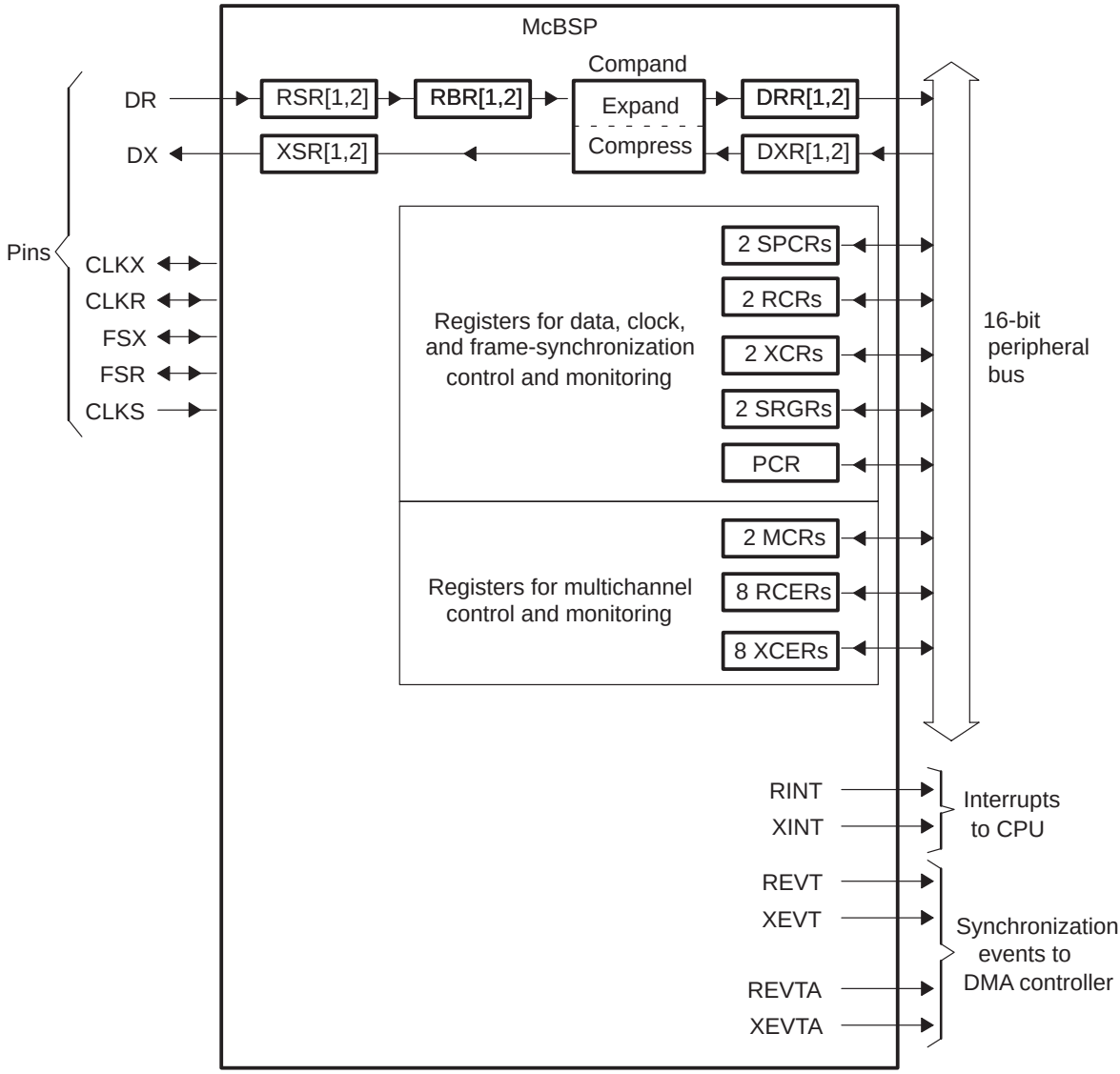
- ❏ Full-duplex communication
- ❏ Double-buffered transmission and triple-buffered reception, which allow a continuous data stream
- ❏ Independent clocking and framing for reception and for transmission
- ❏ The capability to send interrupts to the CPU and to send DMA events to the DMA controller
- ❏ 128 channels for transmission and for reception
- ❏ Multichannel selection modes that enable you to allow or block transfers in each of the channels
- ❏ Direct interface to industry-standard codecs, analog interface chips (AICs), and other serially connected A/D and D/A devices
- ❏ Support for external generation of clock signals and frame-synchronization (frame-sync) signals
- ❏ A programmable sample rate generator for internal generation and control of clock signals and frame-sync signals
- ❏ Programmable polarity for frame-sync pulses and for clock signals

- Direct interface to:
  - T1/E1 framers
  - MVIP switching compatible and ST-BUS compliant devices including:
    - MVIP framers
    - H.100 framers
    - SCSA framers
  - IOM-2 compliant devices
  - AC97 compliant devices (The necessary multiphase frame capability is provided.)
  - IIS compliant devices
  - SPI devices
- A wide selection of data sizes: 8, 12, 16, 20, 24, and 32 bits
  Note: A value of the chosen data size is referred to as a *serial word* or *word* throughout the McBSP documentation. Elsewhere, *word* is used to describe a 16-bit value.
- μ-law and A-law companding
- The option of transmitting/receiving 8-bit data with the LSB first
- Status bits for flagging exception/error conditions
- The capability to use the McBSP pins as general-purpose I/O pins

9.1.2 Block Diagram of the McBSP

The McBSP consists of a data-flow path and a control path connected to external devices by seven pins as shown in Figure 9–1.

Figure 9–1. Conceptual Block Diagram of the McBSP



Data is communicated to devices interfaced with the McBSP via the data transmit (DX) pin for transmission and the data receive (DR) pin for reception. Control information in the form of clocking and frame synchronization is communicated via the following pins: CLKX (transmit clock), CLKR (receive clock), FSX (transmit frame sync), and FSR (receive frame sync).

The CPU and the DMA controller communicate with the McBSP through 16-bit-wide registers accessible via the internal peripheral bus. The CPU or the DMA controller writes the data to be transmitted to the data transmit registers (DXR1, DXR2). Data written to the DXRs is shifted out to DX via the transmit shift registers (XSR1, XSR2). Similarly, receive data on the DR pin is shifted into the receive shift registers (RSR1, RSR2) and copied into the receive buffer registers (RBR1, RBR2). The contents of the RBRs is then copied to the DRRs, which can be read by the CPU or the DMA controller. This allows simultaneous movement of internal and external data communications.

DRR2, RBR2, RSR2, DXR2, and XSR2 are not used (written, read, or shifted) if the serial word length is 8 bits, 12 bits, or 16 bits. For larger word lengths, these registers are needed to hold the most significant bits.

The remaining registers in Figure 9–1 are registers for controlling McBSP operation. Details about these registers are available in section 9.13 (page 9-155).

9.1.3 McBSP Pins

The following table describes the McBSP interface pins. In the Type column, I = Input, O = Output, Z = High impedance. Information on using these pins for general purpose input/output is in section 9.10 on page 9-144.

Pin	Possible States	Possible Uses
CLKR	I/O/Z	Supplying or reflecting the receive clock; supplying the input clock of the sample rate generator; general-purpose I/O
CLKX	I/O/Z	Supplying or reflecting the transmit clock; supplying the input clock of the sample rate generator; general-purpose I/O
CLKS	I	Supplying the input clock of the sample rate generator; general-purpose input
DR	I	Receiving serial data; general-purpose input
DX	O/Z	Transmitting serial data; general-purpose output
FSR	I/O/Z	Supplying or reflecting the receive frame-sync signal; controlling sample rate generator synchronization for the case when GSYNC = 1 (see section 9.3.3 on page 9-29); general-purpose I/O
FSX	I/O/Z	Supplying or reflecting the transmit frame-sync signal; general-purpose I/O

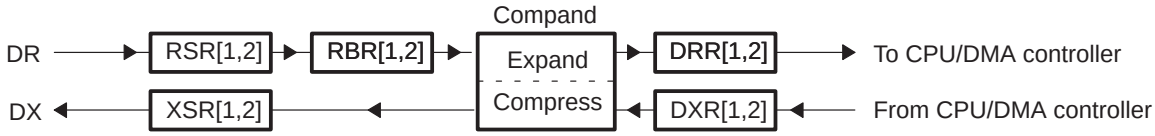
9.2 McBSP Operation

This section contains the following topics:

9.2.1 Data Transfer Process of a McBSP

Figure 9–2 shows a diagram of the McBSP data transfer paths. McBSP receive operation is triple buffered, and transmit operation is double buffered. The use of registers varies depending on whether the defined length of each serial word fits in 16 bits.

Figure 9–2. McBSP Data Transfer Paths



9.2.1.1 Data Transfer Process for Word Length of 8, 12, or 16 Bits

If the word length is 16 bits or smaller, only one 16-bit register is needed at each stage of the data transfer paths. The registers DRR2, RBR2, RSR2, DXR2, and XSR2 are not used (written, read, or shifted).

Receive data arrives on the DR pin and is shifted into receive shift register 1 (RSR1). Once a full word is received, the content of RSR1 is copied to receive buffer register 1 (RBR1), only if RBR1 is not full with previous data. RBR1 is then copied to data receive register 1 (DRR1), unless the previous content of DRR1 has not been read by the CPU or the DMA controller. If the companding feature of the McBSP is implemented, the required word length is 8 bits and receive data is expanded into the appropriate format before being passed from RBR1 to DRR1. For more details about reception, see section 9.2.5 on page 9-18.

Transmit data is written by the CPU or the DMA controller to data transmit register 1 (DXR1). If there is no previous data in transmit shift register (XSR1), the value in DXR1 is copied to XSR1; otherwise, DXR1 is copied to XSR1 when the last bit of the previous data is shifted out on the DX pin. If selected, the companding module compresses 16-bit data into the appropriate 8-bit format before passing it to XSR1. After transmit frame synchronization, the transmitter begins shifting bits from XSR1 to the DX pin. For more details about transmission, see section 9.2.6 on page 9-19.

9.2.1.2 Data Transfer Process for Word Length of 20, 24, or 32 Bits

If the word length is larger than 16 bits, two 16-bit registers are needed at each stage of the data transfer paths. The registers DRR2, RBR2, RSR2, DXR2, and XSR2 are needed to hold the most significant bits.

Receive data arrives on the DR pin and is shifted into RSR2 first and then into RSR1. Once the full word is received, the contents of RSR2 and RSR1 are copied to RBR2 and RBR1, respectively, only if RBR1 is not full. Then the contents of RBR2 and RBR1 are copied to DRR2 and DRR1, respectively, unless the previous content of DRR1 has not been read by the CPU or the DMA controller. The CPU or the DMA controller must read data from DRR2 first and then from DRR1. When DRR1 is read, the next RBR-to-DRR copy occurs. For more details about reception, see section 9.2.5 on page 9-18.

For transmission, the CPU or the DMA controller must write data to DXR2 first and then to DXR1. When new data arrives in DXR1, if there is no previous data in XSR1, the contents of DXR2 and DXR1 are copied to XSR2 and XSR1, respectively; otherwise, the contents of the DXRs are copied to the XSRs when the last bit of the previous data is shifted out on the DX pin. After transmit frame synchronization, the transmitter begins shifting bits from the XSRs to the DX pin. For more details about transmission, see section 9.2.6 on page 9-19.

9.2.2 Companding (Compressing and Expanding) Data

Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM are part of the CCITT G.711 recommendation.

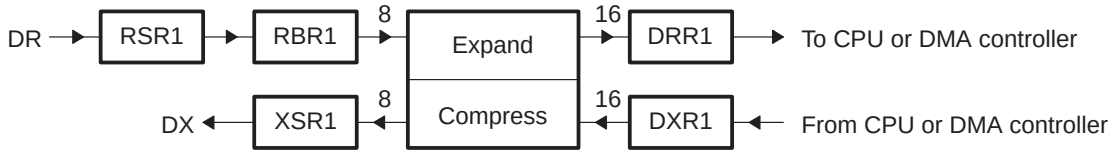
A-law and μ-law allow 13 bits and 14 bits of dynamic range, respectively. Any values outside this range are set to the most positive or most negative value. Thus, for companding to work best, the data transferred to and from the McBSP via the CPU or DMA controller must be at least 16 bits wide.

The μ-law and A-law formats both encode data into 8-bit code words. Companded data is always 8 bits wide; the appropriate word length bits (RWDLEN1,

RWDLEN2, XWDLEN1, XWDLEN2) must therefore be set to 0, indicating an 8-bit wide serial data stream. If companding is enabled and either of the frame phases does not have an 8-bit word length, companding continues as if the word length is 8 bits.

Figure 9–3 illustrates the companding processes. When companding is chosen for the transmitter, compression occurs during the process of copying data from DXR1 to XSR1. The transmit data is encoded according to the specified companding law (A-law or μ-law). When companding is chosen for the receiver, expansion occurs during the process of copying data from RBR1 to DRR1. The receive data is decoded to 2s-complement format.

Figure 9–3. Companding Processes

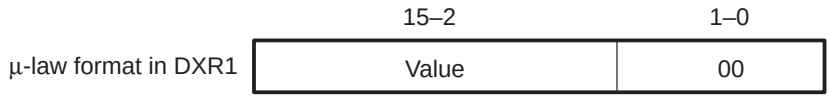


9.2.2.1 Companding Formats

For reception, the 8-bit compressed data in RBR1 is expanded to left-justified 16-bit data in DRR1. The receive sign-extension and justification mode specified in RJUST is ignored when companding is used.

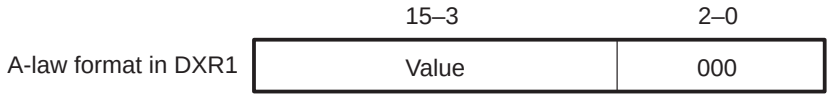
For transmission using μ-law compression, make sure the 14 data bits are left-justified in DXR1, with the remaining two low-order bits filled with 0s as shown in Figure 9–4.

Figure 9–4. μ-Law Transmit Data Companding Format



For transmission using A-law compression, make sure the 13 data bits are left-justified in DXR1, with the remaining three low-order bits filled with 0s as shown in Figure 9–5.

Figure 9–5. A-Law Transmit Data Companding Format



9.2.2.2 Capability to Compand Internal Data

If the McBSP is otherwise unused (the serial port transmit and receive sections are reset), the companding hardware can compand internal data. This can be used to:

- Convert linear to the appropriate μ-law or A-law format.
- Convert μ-law or A-law to the linear format.
- Observe the quantization effects in companding by transmitting linear data, and compressing and re-expanding this data. This is useful only if both XCOMPAND and RCOMPAND enable the same companding format.

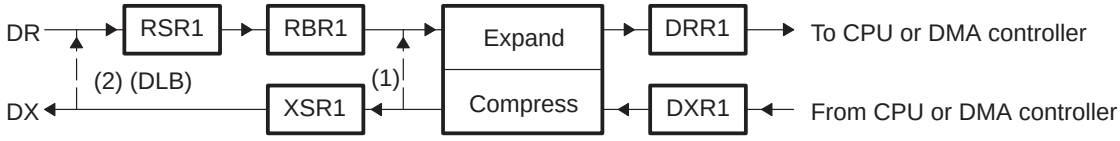
Figure 9–6 shows two methods by which the McBSP can compand internal data. Data paths for these two methods are used to indicate:

- When both the transmit and receive sections of the serial port are reset, DRR1 and DXR1 are connected internally through the companding logic. Values from DXR1 are compressed, as selected by XCOMPAND, and then expanded, as selected by RCOMPAND. Note that RRDY and XRDY bits are not set. However, data is available in DRR1 within four CPU clocks after being written to DXR1.

  The advantage of this method is its speed. The disadvantage is that there is no synchronization available to the CPU and DMA to control the flow. Note that DRR1 and DXR1 are internally connected if the (X/R)COMPAND bits are set to 10b or 11b (compand using μ-law or A-law).

- The McBSP is enabled in digital loopback mode with companding appropriately enabled by RCOMPAND and XCOMPAND. Receive and transmit interrupts (RINT when RINTM = 0 and XINT when XINTM = 0) or synchronization events (REVT and XEVT) allow synchronization of the CPU or DMA to these conversions, respectively. Here, the time for this companding depends on the serial bit rate selected.

Figure 9–6. Two Methods by Which the McBSP Can Compand Internal Data



9.2.2.3 Reversing Bit Order: Option to Transfer LSB First

Normally, the McBSP transmit or receives all data with the most significant bit (MSB) first. However, certain 8-bit data protocols (that do not use companded data) require the least significant bit (LSB) to be transferred first. If you set XCOMPAND = 01b in XCR2, the bit ordering of 8-bit words is reversed (LSB first) before being sent from the serial port. If you set RCOMPAND = 01b in RCR2, the bit ordering of 8-bit words is reversed during reception. Similar to companding, this feature is enabled only if the appropriate word length bits are set to 0, indicating that 8-bit words are to be transferred serially. If either phase of the frame does not have an 8-bit word length, the McBSP assumes the word length is eight bits, and LSB-first ordering is done.

9.2.3 Clocking and Framing Data

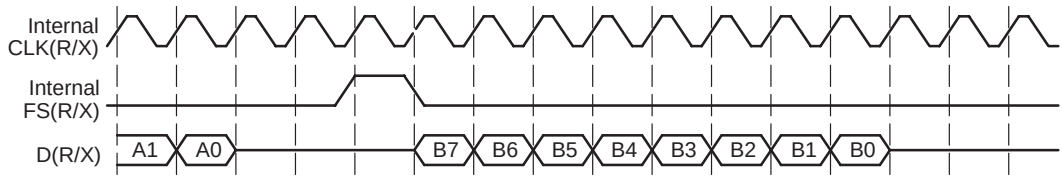
This section explains basic concepts and terminology important for understanding how McBSP data transfers are timed and delimited.

9.2.3.1 Clocking

Data is shifted one bit at a time from the DR pin to the RSR(s) or from the XSR(s) to the DX pin. The time for each bit transfer is controlled by the rising or falling edge of a clock signal.

The receive clock signal (CLKR) controls bit transfers from the DR pin to the RSR(s). The transmit clock signal (CLKX) controls bit transfers from the XSR(s) to the DX pin. CLKR or CLKX can be derived from a pin at the boundary of the McBSP or derived from inside the McBSP. The polarities of CLKR and CLKX are programmable.

In the following example, the clock signal controls the timing of each bit transfer on the pin.



Note:

The McBSP cannot operate at a frequency faster than 1/2 the CPU clock frequency. When driving CLKX or CLKR at the pin, choose an appropriate input clock frequency. When using the internal sample rate generator for CLKX and/or CLKR, choose an appropriate input clock frequency and divide down value (CLKDV).

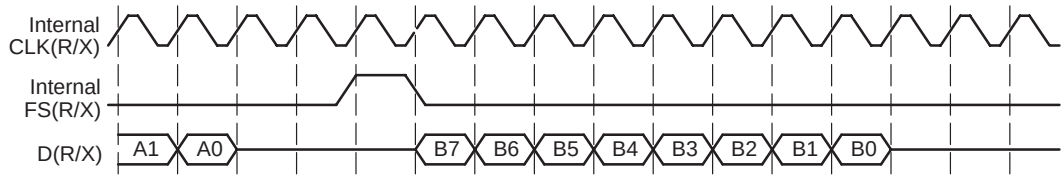
9.2.3.2 Serial Words

Bits traveling between a shift register (RSR or XSR) and a data pin (DR or DX) are transferred in a group called a **serial word**. You define how many bits are in a word.

Bits coming in on the DR pin are held in RSR until RSR holds a full serial word. Only then is the word passed to RBR (and ultimately to the DRR).

During transmission, XSR does not accept new data from DXR until a full serial word has been passed from XSR to the DX pin.

In the following example, an 8-bit word size was defined (see bits 7 through 0 of word B being transferred).



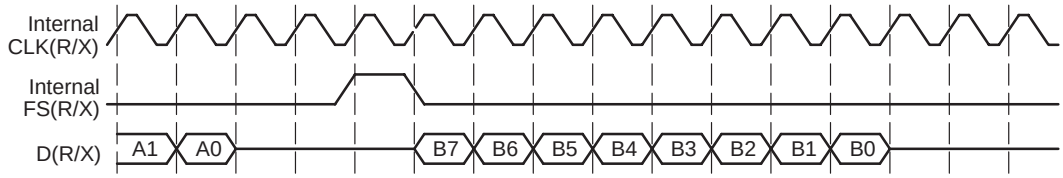
9.2.3.3 Frames and Frame Synchronization

One or more words are transferred in a group called a **frame**. You define how many words are in a frame.

All of the words in a frame are sent in a continuous stream. However, there can be pauses between frame transfers. The McBSP uses frame-synchronization (frame-sync) signals to determine when each frame is received/transmitted. When a pulse occurs on a frame-sync signal, the McBSP begins receiving/transmitting a frame of data. When the next pulse occurs, the McBSP receives/transmits the next frame, and so on.

Pulses on the receive frame-sync signal (FSR) initiate frame transfers on DR. Pulses on the transmit frame-sync signal (FSX) initiate frame transfers on DX. FSR or FSX can be derived from a pin at the boundary of the McBSP or derived from inside the McBSP.

In the following example, a 1-word frame is transferred when a frame-sync pulse occurs.



In McBSP operation, the inactive-to-active transition of the frame-synchronization signal indicates the start of the next frame. For this reason, the frame-sync signal may be high for an arbitrary number of clock cycles. Only after the signal is recognized to have gone inactive, and then active again, does the next frame synchronization occur.

9.2.3.4 Detecting Frame-Sync Pulses, Even in the Reset State

The McBSP can send receive and transmit interrupts to the CPU to indicate specific events in the McBSP. To facilitate detection of frame synchronization, these interrupts can be sent in response to frame-sync pulses. Set the appropriate interrupt mode bits to 10b (for reception, RINTM = 10b; for transmission, XINTM = 10b).

Unlike other serial port interrupt modes, this mode can operate while the associated portion of the serial port is in reset (such as activating RINT when the receiver is in reset). In this case, FSRM/FSXM and FSRP/FSXP still select the appropriate source and polarity of frame synchronization. Thus, even when the serial port is in the reset state, these signals are synchronized to the CPU clock and then sent to the CPU in the form of RINT and XINT at the point at which they feed the receiver and transmitter of the serial port. Consequently, a new frame-synchronization pulse can be detected, and after this occurs the CPU can take the serial port out of reset safely.

9.2.3.5 Ignoring Frame-Sync Pulses

The McBSP can be configured to ignore transmit and/or receive frame-synchronization pulses. To have the receiver or transmitter recognize frame-sync pulses, clear the appropriate frame-sync ignore bit (RFIG = 0 for the receiver, XFIG = 0 for the transmitter). To have the receiver or transmitter ignore frame-sync pulses until the desired frame length or number of words is reached, set the appropriate frame-sync ignore bit (RFIG = 1 for the receiver, XFIG = 1 for the transmitter). For more details on unexpected frame-sync pulses, see one of the following topics:

- *Unexpected Receive Frame-Sync Pulse* (page 9-38)
- *Unexpected Transmit Frame-Sync Pulse* (page 9-44)

You can also use the frame-sync ignore function for data packing (for more details, see section 9.12.2 on page 9-153).

9.2.3.6 Frame Frequency

The frame frequency is determined by the period between frame-synchronization pulses and is defined as shown by Equation 9–1.

Equation 9–1. Frame Frequency of a McBSP

$$\text{Frame Frequency} = \frac{\text{Clock Frequency}}{\text{Number of Clock Cycles Between Frame–Sync Pulses}}$$

The frame frequency may be increased by decreasing the time between frame-synchronization pulses (limited only by the number of bits per frame). As the frame transmit frequency increases, the inactivity period between the data packets for adjacent transfers decreases to zero.

9.2.3.7 Maximum Frame Frequency

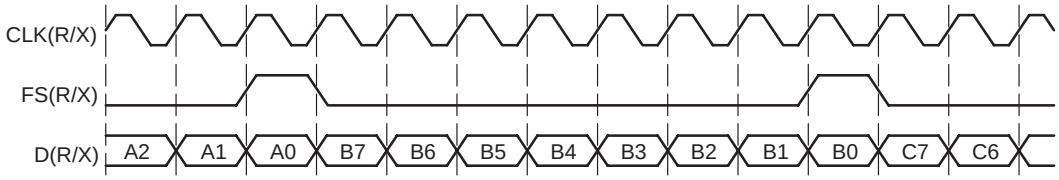
The minimum number of clock cycles between frame synchronization pulses is equal to the number of bits transferred per frame. The maximum frame frequency is defined as shown by Equation 9–2.

Equation 9–2. Maximum Frame Frequency of a McBSP

$$\text{Maximum Frame Frequency} = \frac{\text{Clock Frequency}}{\text{Number of Bits Per Frame}}$$

Figure 9–7 shows the McBSP operating at maximum packet frequency. At maximum packet frequency, the data bits in consecutive packets are transmitted contiguously with no inactivity between bits.

Figure 9–7. McBSP Operating at Maximum Packet Frequency



If there is a 1-bit data delay as shown in this figure, the frame-synchronization pulse overlaps the last bit transmitted in the previous frame. Effectively, this permits a continuous stream of data, making frame-synchronization pulses redundant. Theoretically, only an initial frame-synchronization pulse is required to initiate a multipacket transfer.

The McBSP supports operation of the serial port in this fashion by ignoring the successive frame-sync pulses. Data is clocked in to the receiver, or clocked out of the transmitter, during every clock cycle.

Note:

For XDATDLY = 0 (0-bit data delay), the first bit of data is transmitted asynchronously to the internal transmit clock signal (CLKX). For more details, see *Set the Transmit Data Delay* on page 9-126.

9.2.4 Frame Phases

The McBSP allows you to configure each frame to contain one or two phases. The number of words per frame, and the number of bits per word, can be specified differently for each of the two phases of a frame, allowing greater flexibility in structuring data transfers. For example, a user might define a frame as consisting of one phase containing two words of 16 bits each, followed by a second phase consisting of 10 words of 8 bits each. This configuration permits the user to compose frames for custom applications, or in general, to maximize the efficiency of data transfers.

9.2.4.1 Number of Phases, Words, and Bits Per Frame

Table 2–7 shows which bit fields in the receive control registers (RCR1 and RCR2) and in the transmit control registers (XCR1 and XCR2) determine the number of phases per frame, the number of words per frame, and number of bits per word for each phase, for the receiver and transmitter. The maximum number of words per frame is 128 for a single-phase frame and 256 for a dual-phase frame. The number of bits per word can be 8, 12, 16, 20, 24, or 32 bits.

Table 9–1. McBSP Register Bits That Determine the Number of Phases, Words, and Bits Per Frame

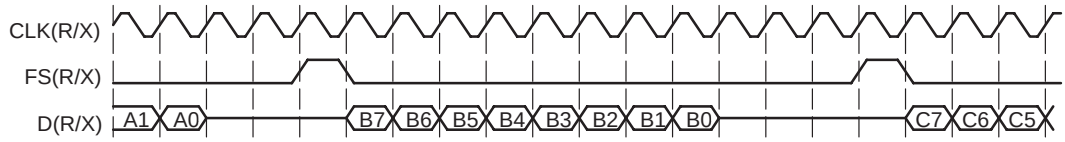
Operation	Number of Phases	Words Per Frame Set With ...	Bits Per Word Set With ...
Reception	1 (RPHASE = 0)	RFRLEN1	RWDLEN1
Reception	2 (RPHASE = 1)	RFRLEN1 and RFRLEN2	RWDLEN1 for phase 1 RWDLEN2 for phase 2
Transmission	1 (XPHASE = 0)	XFRLEN1	XWDLEN1
Transmission	2 (XPHASE = 1)	XFRLEN1 and XFRLEN2	XWDLEN1 for phase 1 XWDLEN2 for phase 2

9.2.4.2 Single-Phase Frame Example

Figure 9–8 shows an example of a single-phase data frame comprising one 8-bit word. Since the transfer is configured for one data bit delay, the data on the DX and DR pins are available one clock cycle after FS(R/X) goes active. The figure makes the following assumptions:

- (R/X)PHASE = 0: Single-phase frame
- (R/X)FRLEN1 = 0b: 1 word per frame
- (R/X)WDLEN1 = 000b: 8-bit word length
- (R/X)FRLEN2 and (R/X)WDLEN2 are ignored
- CLK(X/R)P = 0: Receive data clocked on falling edge; transmit data clocked on rising edge
- FS(R/X)P = 0: Active-high frame-sync signals
- (R/X)DATDLY = 01b: 1-bit data delay

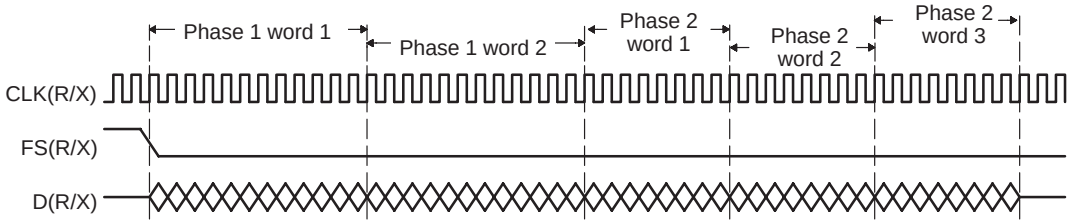
Figure 9–8. Single-Phase Frame for a McBSP Data Transfer



9.2.4.3 Dual-Phase Frame Example

Figure 9–9 shows an example of a frame where the first phase consists of 2 words of 12 bits each followed by a second phase of three words of 8 bits each. Note that the entire bit stream in the frame is contiguous. There are no gaps either between words or between phases.

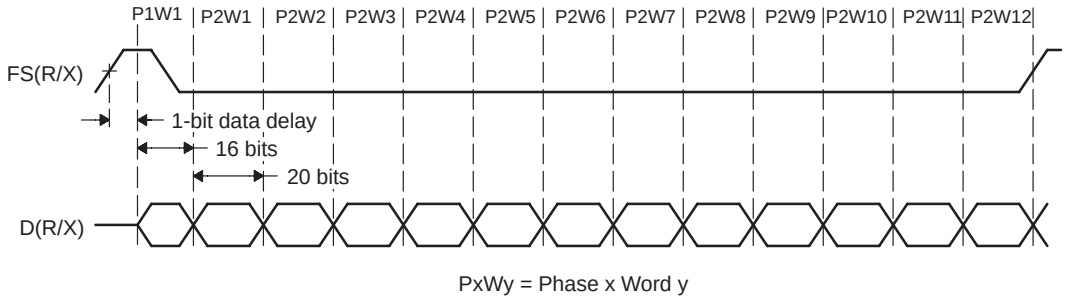
Figure 9–9. Dual-Phase Frame for a McBSP Data Transfer



9.2.4.4 Implementing the AC97 Standard with a Dual-Phase Frame

Figure 9–10 shows an example of the Audio Codec ‘97 (AC97) standard, which uses the dual-phase frame feature. Notice that words, not individual bits, are shown on the D(R/X) signal. The first phase (P1) consists of a single 16-bit word. The second phase (P2) consists of twelve 20-bit words. The phase configurations are listed after the figure.

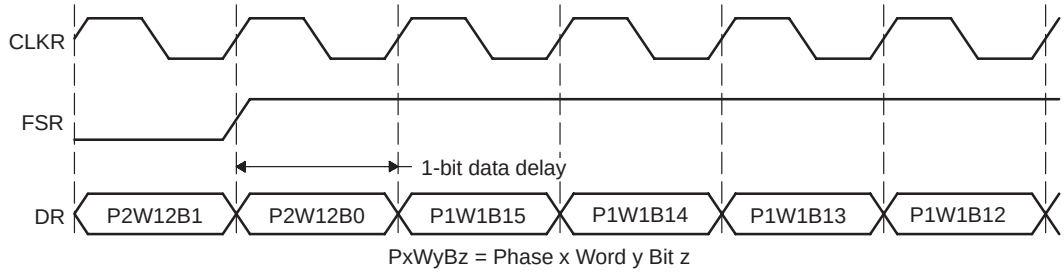
Figure 9–10. Implementing the AC97 Standard with a Dual-Phase Frame



- ❏ (R/X)PHASE = 1: Dual-phase frame
- ❏ (R/X)FRLEN1 = 0000000b: 1 word in phase 1
- ❏ (R/X)WDLEN1 = 010b: 16 bits per word in phase 1
- ❏ (R/X)FRLEN2 = 0001011b: 12 words in phase 2
- ❏ (R/X)WDLEN2 = 011b: 20 bits per word in phase 2
- ❏ CLKRP/CLKXP= 0: Receive data sampled on falling edge of internal CLKR / transmit data clocked on rising edge of internal CLKX
- ❏ FSRP/FSXP = 0: Active-high frame-sync signal
- ❏ (R/X)DATDLY = 01b: Data delay of 1 clock cycle (1-bit data delay)

Figure 9–11 shows the timing of an AC97-standard data transfer near frame synchronization. In this figure, individual bits are shown on D(R/X). Specifically, the figure shows the last two bits of phase 2 of one frame and the first four bits of phase 1 of the next frame. Regardless of the data delay, data transfers can occur without gaps. The first bit of the second frame (P1W1B15) immediately follows the last bit of the first frame (P2W12B0). Because a 1-bit data delay has been chosen, the transition on the frame-sync signal can occur when P2W12B0 is transferred.

Figure 9–11. Timing of an AC97-Standard Data Transfer Near Frame Synchronization

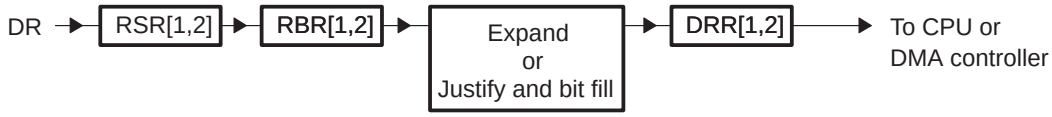


9.2.5 McBSP Reception

This section explains the fundamental process of reception in the McBSP. For details about how to program the McBSP receiver, see *Receiver Configuration* on page 9-73.

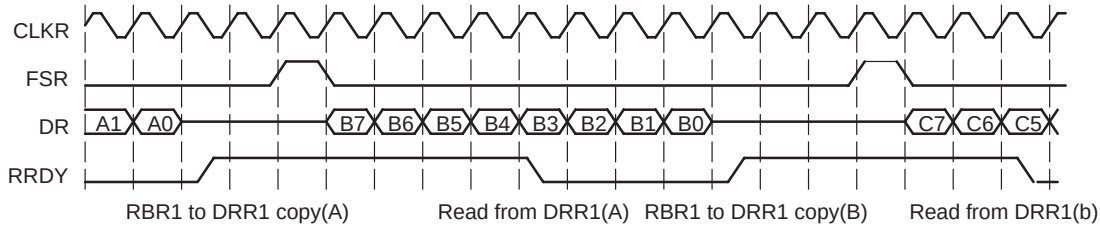
Figure 9–12 and Figure 9–13 show how reception occurs in the McBSP. Figure 9–12 shows the physical path for the data. Figure 9–13 is a timing diagram showing signal activity for one possible reception scenario. A description of the process follows the figures.

Figure 9–12. McBSP Reception Physical Data Path



RSR[1,2]: Receive shift registers 1 and 2
RBR[1,2]: Receive buffer registers 1 and 2
DRR[1,2]: Data receive registers 1 and 2

Figure 9–13. McBSP Reception Signal Activity



CLKR: Internal receive clock
FSR: Internal receive frame-sync signal
DR: Data on DR pin
RRDY: Status of receiver ready bit (high is 1)

The following process describes how data travels from the DR pin to the CPU or to the DMA controller:

1) The McBSP waits for a receive frame-sync pulse on internal FSR.

2) When the pulse arrives, the McBSP inserts the appropriate data delay that is selected with the RDATDLY bits of RCR2.

   In the preceding timing diagram (Figure 9–13), a 1-bit data delay is selected.

3) The McBSP accepts data bits on the DR pin and shifts them into the receive shift register(s).

   If the word length is 16 bits or smaller, only RSR1 is used. If the word length is larger than 16 bits, RSR2 and RSR1 are used, and RSR2 contains the

most significant bits. For details on choosing a word length, see *Set the Receive Word Length(s)* on page 9-82.

4) When a full word is received, the McBSP copies the contents of the receive shift register(s) to the receive buffer register(s), provided that RBR1 is not full with previous data.

   If the word length is 16 bits or smaller, only RBR1 is used. If the word length is larger than 16 bits, RBR2 and RBR1 are used, and RBR2 contains the most significant bits.

5) The McBSP copies the contents of the receive buffer register(s) into the data receive register(s), provided that DRR1 is not full with previous data. When DRR1 receives new data, the receiver ready bit (RRDY) is set in SPCR1. This indicates that receive data is ready to be read by the CPU or the DMA controller.

   If the word length is 16 bits or smaller, only DRR1 is used. If the word length is larger than 16 bits, DRR2 and DRR1 are used, and DRR2 contains the most significant bits.

   If companding is used during the copy (RCOMPAND = 10b or 11b in RCR2), the 8-bit compressed data in RBR1 is expanded to a left-justified 16-bit value in DRR1. If companding is disabled, the data copied from RBR[1,2] to DRR[1,2] is justified and bit filled according to the RJUST bits.

6) The CPU or the DMA controller reads the data from the data receive register(s). When DRR1 is read, RRDY is cleared and the next RBR-to-DRR copy is initiated.

Note:

If both DRRs are need (word length larger than 16 bits), the CPU or the DMA controller must read from DRR2 first and then from DRR1. As soon as DRR1 is read, the next RBR-to-DRR copy is initiated. If DRR2 is not read first, the data in DRR2 is lost.

When activity is not properly timed, errors can occur. See the following topics for more details:

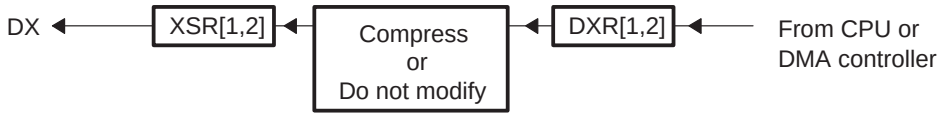
- *Overrun in the Receiver* (page 9-37)
- *Unexpected Receive Frame-Sync Pulse* (page 9-38)

9.2.6 McBSP Transmission

This section explains the fundamental process of transmission in the McBSP. For details about how to program the McBSP transmitter, see *Transmitter Configuration* on page 9-110.

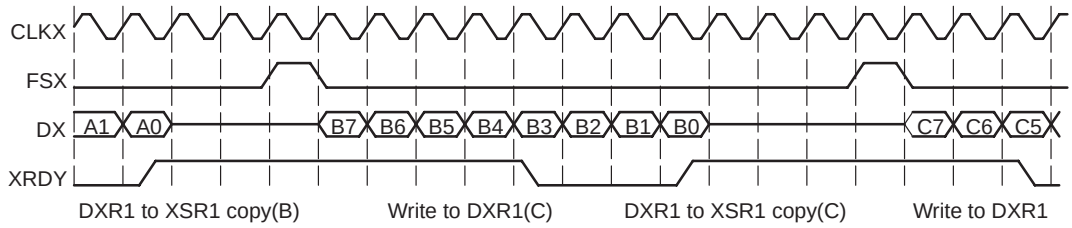
Figure 9–14 and Figure 9–15 show how transmission occurs in the McBSP. Figure 9–14 shows the physical path for the data. Figure 9–15 is a timing diagram showing signal activity for one possible transmission scenario. A description of the process follows the figures.

Figure 9–14. McBSP Transmission Physical Data Path



XSR[1,2]: Transmit shift registers 1 and 2 DXR[1,2]: Data transmit registers 1 and 2

Figure 9–15. McBSP Transmission Signal Activity



CLKX: Internal transmit clock DX: Data on DX pin
FSX: Internal transmit frame-sync signal XRDY: Status of transmitter ready bit (high is 1)

1) The CPU or the DMA controller writes data to the data transmit register(s). When DXR1 is loaded, the transmitter ready bit (XRDY) is cleared in SPCR2 to indicate that the transmitter is not ready for new data.

   If the word length is 16 bits or smaller, only DXR1 is used. If the word length is larger than 16 bits, DXR2 and DXR1 are used, and DXR2 contains the most significant bits. For details on choosing a word length, see *Set the Transmit Word Length(s)* on page 9-119.

Note:

If both DXRs are needed (word length larger than 16 bits), the CPU or the DMA controller must load DXR2 first and then load DXR1. As soon as DXR1 is loaded, the contents of both DXRs are copied to the transmit shift registers (XSRs), as described in the next step. If DXR2 is not loaded first, the previous content of DXR2 is passed to the XSR2.

2) When new data arrives in DXR1, the McBSP copies the content of the data transmit register(s) to the transmit shift register(s). In addition, the transmit ready bit (XRDY) is set. This indicates that the transmitter is ready to accept new data from the CPU or the DMA controller.

If the word length is 16 bits or smaller, only XSR1 is used. If the word length is larger than 16 bits, XSR2 and XSR1 are used, and XSR2 contains the most significant bits.

If companding is used during the transfer (XCOMPAND = 10b or 11b in XCR2), the McBSP compresses the 16-bit data in DXR1 to 8-bit data in the μ-law or A-law format in XSR1. If companding is disabled, the McBSP passes data from the DXR(s) to the XSR(s) without modification.

3) The McBSP waits for a transmit frame-sync pulse on internal FSX.

4) When the pulse arrives, the McBSP inserts the appropriate data delay that is selected with the XDATDLY bits of XCR2.

   In the preceding timing diagram (Figure 9–15), a 1-bit data delay is selected.

5) The McBSP shifts data bits from the transmit shift register(s) to the DX pin.

When activity is not properly timed, errors can occur. See the following topics for more details:

- *Overwrite in the Transmitter* (page 9-41)
- *Underflow in the Transmitter* (page 9-42)
- *Unexpected Transmit Frame-Sync Pulse* (page 9-44)

9.2.7 Interrupts and DMA Events Generated by a McBSP

The McBSP sends notification of important events to the CPU and DMA controller via the internal signals shown in Table 9–2.

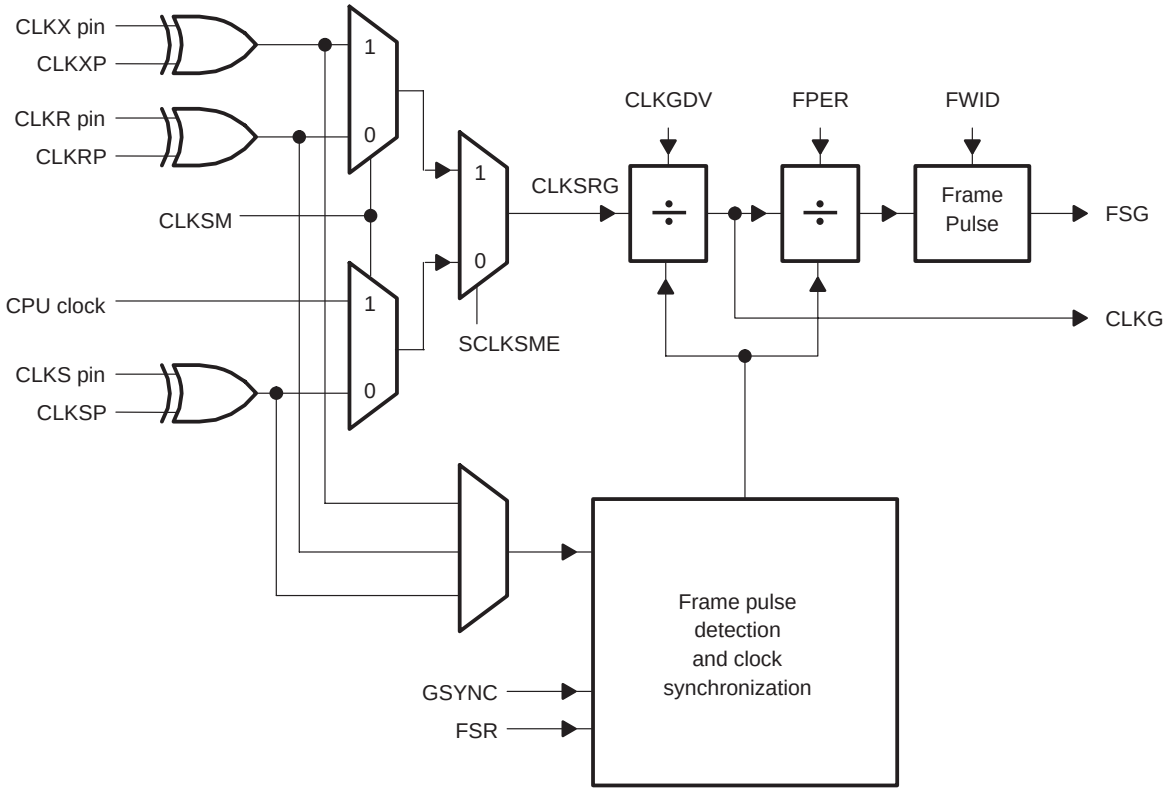
Table 9–2. Interrupts and DMA Events Generated by a McBSP

Internal Signal	Description
RINT	Receive interrupt The McBSP can send a receive interrupt request to CPU based upon a selected condition in the receiver of the McBSP (a condition selected by the RINTM bits of SPCR1).
XINT	Transmit interrupt The McBSP can send a transmit interrupt request to CPU based upon a selected condition in the transmitter of the McBSP (a condition selected by the XINTM bits of SPCR2).
REVT	Receive synchronization event An REVT signal is sent to the DMA controller when data has been received in the data receive registers (DRRs).
XEVT	Transmit synchronization event An XEVT signal is sent to the DMA controller when the data transmit registers (DXRs) are ready to accept the next serial word for transmission.
REVTA	A-bis mode receive synchronization event If ABIS = 1 (A-bis mode is enabled) an REVTA signal is sent to the DMA controller every 16 cycles.
XEVTA	A-bis mode transmit synchronization event If ABIS = 1 (A-bis mode is enabled) an XEVTA signal is sent to the DMA controller every 16 cycles.

9.3 Sample Rate Generator of the McBSP

Each McBSP contains a sample rate generator that can be used to generate an internal data clock (CLKG) and an internal frame-synchronization signal (FSG). CLKG can be used for bit shifting on the data receive (DR) pin and/or the data transmit (DX) pin. FSG can be used to initiate frame transfers on DR and/or DX. Figure 9–16 is a conceptual block diagram of the sample rate generator.

Figure 9–16. Conceptual Block Diagram of the Sample Rate Generator



The source clock for the sample rate generator (labeled CLKSRG in the diagram) can be supplied by the CPU clock or by an external pin (CLKS, CLKX, or CLKR). The source is selected with the SCLKME bit of PCR and the CLKSM bit of SRGR2. If a pin is used, the polarity of the incoming signal can be inverted with the appropriate polarity bit (CLKSP of SRGR2, CLKXP of PCR, or CLKRP of PCR).

The sample rate generator has a 3-stage clock divider that gives CLKG and FSG programmability. The three stages provide:

- Clock divide down. The source clock is divided according to the CLKGDV bits of SRGR1 to produce CLKG.
- Frame period divide down. CLKG is divided according to the FPER bits of SRGR2 to control the period from the start of a frame-sync pulse to the start of the next pulse.
- Frame-sync pulse width countdown. CLKG cycles are counted according to the FWID bits of SRGR1 to control the width of each frame-sync pulse.

Note:

The McBSP cannot operate at a frequency faster than 1/2 the CPU clock frequency. Choose an input clock frequency and a CLKDV value such that CLKG is less than or equal to 1/2 the CPU clock frequency.

In addition to the 3-stage clock divider, the sample rate generator has a frame-sync pulse detection and clock synchronization module that allows synchronization of the clock divide down with an incoming frame-sync pulse on the FSR pin. This feature is enabled or disabled with the GSYNC bit of SRGR2.

For details on getting the sample rate generator ready for operation, see the reset and initialization procedure on page 9-31.

9.3.1 Clock Generation in the Sample Rate Generator

The sample rate generator can produce a clock signal (CLKG) for use by the receiver, the transmitter, or both. Use of the sample rate generator to drive clocking is controlled by the clock mode bits (CLKRM and CLKXM) in the pin control register (PCR). When a clock mode bit is set to 1 (CLKRM = 1 for reception, CLKXM = 1 for transmission), the corresponding data clock (CLKR for reception, CLKX for transmission) is driven by the internal sample rate generator output clock (CLKG).

Note that the effects of CLKRM = 1 and CLKXM = 1 on the McBSP are partially affected by the use of the digital loopback mode and the clock stop (SPI) mode, respectively, as described in Table 2–7. The digital loopback mode (described in section 9.8.4 on page 9-77) is selected with the DLB bit of SPCR1. The clock stop mode (described in section 9.7, page 9-62) is selected with the CLKSTP bits of SPCR1.

When using the sample rate generator as a clock source, make sure the sample rate generator is enabled (GRST_ = 1).

Table 9–3. Effects of DLB and CLKSTP on Clock Modes

Mode Bit Settings		Effect
CLKRM = 1	DLB = 0 (Digital loopback mode disabled)	CLKR is an output pin driven by the sample rate generator output clock (CLKG).
	DLB = 1 (Digital loopback mode enabled)	CLKR is an output pin driven by internal CLKX. The source for CLKX depends on the CLKXM bit.
CLKXM = 1	CLKSTP = 00b or 01b (Clock stop (SPI) mode disabled)	CLKX is an output pin driven by the sample rate generator output clock (CLKG).
	CLKSTP = 10b or 11b (Clock stop (SPI) mode enabled)	The McBSP is a master in an SPI system. Internal CLKX drives internal CLKR and the shift clocks of any SPI-compliant slave devices in the system. CLKX is driven by the internal sample rate generator.

9.3.1.1 Choosing an Input Clock

The sample rate generator must be driven by an input clock signal from one of the four sources selectable with the SCLKME bit of PCR and the CLKSM bit of SRGR2 (see Table 9–4). When CLKSM = 1, the minimum divide down value in CLKGDV bits should be 1. CLKGDV is described in section 9.3.1.3.

Note:

The McBSP cannot operate at a frequency faster than 1/2 the CPU clock frequency. Choose an input clock frequency and a CLKDV value such that CLKG is less than or equal to 1/2 the CPU clock frequency.

Table 9–4. Choosing an Input Clock for the Sample Rate Generator with the SCLKME and CLKSM Bits

SCLKME	CLKSM	Input Clock For Sample Rate Generator
0	0	Signal on CLKS pin
0	1	CPU clock
1	0	Signal on CLKR pin
1	1	Signal on CLKX pin

9.3.1.2 Choosing a Polarity for the Input Clock

As shown in Figure 9–17, when the input clock is received from a pin, you can choose the polarity of the input clock. The rising edge of CLKSRG generates CLKG and FSG, but you can determine which edge of the input clock causes a rising edge on CLKSRG. The polarity options and their effects are described in Table 9–5.

Figure 9–17. Possible Inputs to the Sample Rate Generator and The Polarity Bits

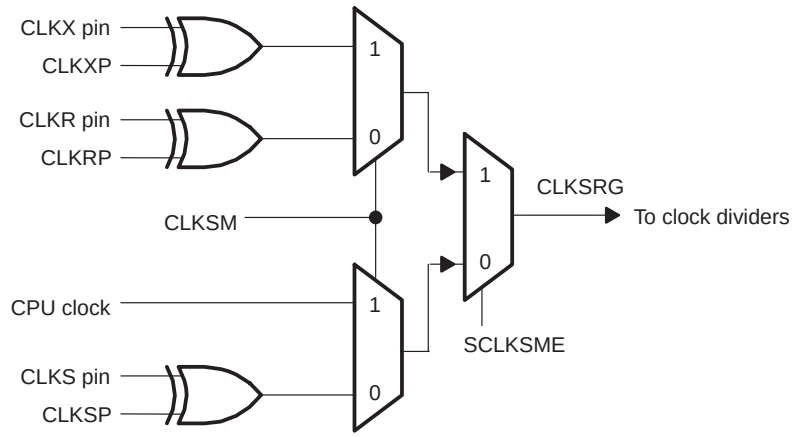


Table 9–5. Polarity Options for the Input to the Sample Rate Generator

Input Clock	Polarity Option	Effect
Signal on CLKS pin	CLKSP = 0 in SRGR2	Rising edge on CLKS pin generates transitions on CLKG and FSG.
	CLKSP = 1 in SRGR2	Falling edge on CLKS pin generates transitions on CLKG and FSG.
CPU clock	Always positive polarity	Rising edge of CPU clock generates transitions on CLKG and FSG.
Signal on CLKR pin	CLKRP = 0 in PCR	Falling edge on CLKR pin generates transitions on CLKG and FSG.
	CLKRP = 1 in PCR	Rising edge on CLKR pin generates transitions on CLKG and FSG.
Signal on CLKX pin	CLKXP = 0 in PCR	Rising edge on CLKX pin generates transitions on CLKG and FSG.
	CLKXP = 1 in PCR	Falling edge on CLKX pin generates transitions on CLKG and FSG.

9.3.1.3 Choosing a Frequency for the Output Clock (CLKG)

The input clock (CPU clock or external clock) can be divided down by a programmable value to drive CLKG. Regardless of the source to the sample rate generator, the rising edge of CLKSRG (see the sample rate generator diagram on page 9-23) generates CLKG and FSG.

The first divider stage of the sample rate generator creates the output clock from the input clock. This divider stage uses a counter that is preloaded with the divide down value in the CLKGDV bits of SRGR1. The output of this stage is the data clock (CLKG). CLKG has the frequency represented by the following equation.

$$\text{CLKG frequency} = \frac{\text{Input clock frequency}}{(\text{CLKGDV} + 1)}$$

Thus, the input clock frequency is divided by a value between 1 and 256. When CLKGDV is odd or equal to 0, the CLKG duty cycle is 50%. When CLKGDV is an even value, 2p, representing an odd divide down, the high-state duration is p+1 cycles and the low-state duration is p cycles.

Note:

The McBSP cannot operate at a frequency faster than 1/2 the CPU clock frequency. Choose an input clock frequency and a CLKDV value such that CLKG is less than or equal to 1/2 the CPU clock frequency.

9.3.1.4 Keeping CLKG Synchronized to an External Input Clock

When an external signal is selected to drive the sample rate generator (see section 9.3.1.1), the GSYNC bit in SRGR2 and the FSR pin can be used to configure the timing of the output clock (CLKG) relative to the input clock.

GSYNC = 1 ensures that the McBSP and an external device are dividing down the input clock with the same phase relationship. If GSYNC = 1, an inactive-to-active transition on the FSR pin triggers a resynchronization of CLKG and generation of FSG.

For more details about the synchronization, see section 9.3.3 on page 9-29.

9.3.2 Frame Sync Generation in the Sample Rate Generator

The sample rate generator can produce a frame-sync signal (FSG) for use by the receiver, the transmitter, or both.

If you want the **receiver** to use FSG for frame synchronization, make sure FSRM = 1. (When FSRM = 0, receive frame synchronization is supplied via the FSR pin.)

If you want the **transmitter** to use FSG for frame synchronization, you must set:

- ❏ FSXM = 1 in PCR: This indicates that transmit frame synchronization is supplied by the McBSP itself rather than from the FSX pin.
- ❏ FSGM = 1 in SRGR2: This indicates that when FSXM = 1, transmit frame synchronization is supplied by the sample rate generator. (When FSGM = 0 and FSXM = 1, the transmitter uses frame-sync pulses generated every time data is transferred from DXR[1,2] to XSR[1,2].)

In either case, the sample rate generator must be enabled (GRST_ = 1) and the frame-sync logic in the sample rate generator must be enabled (FRST_ = 0).

9.3.2.1 Choosing the Width of the Frame-Sync Pulse on FSG

Each pulse on FSG has a programmable width. You program the FWID bits of SRGR1, and the resulting pulse width is (FWID + 1) CLKG cycles, where CLKG is the output clock of the sample rate generator.

9.3.2.2 Controlling the Period Between the Starting Edges of Frame-Sync Pulses on FSG

You can control the amount of time from the starting edge of one FSG pulse to the starting edge of the next FSG pulse. This period is controlled in one of two ways, depending on the configuration of the sample rate generator:

- ❏ If the sample rate generator is using an external input clock and GSYNC = 1 in SRGR2, FSG pulses in response to an inactive-to-active transition on the FSR pin. Thus, the frame-sync period is controlled by an external device.
- ❏ Otherwise, you program the FPER bits of SRGR2, and the resulting frame-sync period is (FPER + 1) CLKG cycles, where CLKG is the output clock of the sample rate generator.

9.3.2.3 Keeping FSG Synchronized to an External Clock

When an external signal is selected to drive the sample rate generator (see section 9.3.1.1 on page 9-25), the GSYNC bit of SRGR2 and the FSR pin can be used to configure the timing of FSG pulses.

GSYNC = 1 ensures that the McBSP and an external device are dividing down the input clock with the same phase relationship. If GSYNC = 1, an inactive-to-active transition on the FSR pin triggers a resynchronization of CLKG and generation of FSG.

Section 9.3.3 has more details about the synchronization.

9.3.3 Synchronizing Sample Rate Generator Outputs to an External Clock

The sample rate generator can produce a clock signal (CLKG) and a frame-sync signal (FSG) based on an input clock signal that is either the CPU clock signal or a signal at the CLKS, CLKR, or CLKX pin. When an external clock is selected to drive the sample rate generator, the GSYNC bit of SRGR2 and the FSR pin can be used to control the timing of CLKG and the pulsing of FSG relative to the chosen input clock.

Make GSYNC = 1 when you want the McBSP and an external device to divide down the input clock with the same phase relationship. If GSYNC = 1:

- ❏ An inactive-to-active transition on the FSR pin triggers a resynchronization of CLKG and a pulsing of FSG.
- ❏ CLKG always begins with a high state after synchronization.
- ❏ FSR is always detected at the same edge of the input clock signal that generates CLKG, no matter how long the FSR pulse is.
- ❏ The FPER bits of SRGR2 are ignored because the frame-sync period on FSG is determined by the arrival of the next frame-sync pulse on the FSR pin.

If GSYNC = 0, CLKG runs freely and is not resynchronized, and the frame-sync period on FSG is determined by FPER.

9.3.3.1 Operating the Transmitter Synchronously with the Receiver

When GSYNC = 1, the transmitter can operate synchronously with the receiver, provided that:

- ❑ FSX is programmed to be driven by FSG (FSGM = 1 in SRGR2 and FSXM = 1 in PCR). If the input FSR has appropriate timing so that it can be sampled by the falling edge of CLKG, it can be used, instead, by setting FSXM = 0 and connecting FSR to FSX externally.
- ❑ The sample rate generator clock drives the transmit and receive clocking (CLKRM = CLKXM = 1 in PCR). Therefore, the CLK(R/X) pin should not be driven by any other driving source.

9.3.3.2 Synchronization Examples

Figure 9–18 and Figure 9–19 show the clock and frame-synchronization operation with various polarities of CLKS (the chosen input clock) and FSR. These figures assume FWID = 0 in SRGR1, for an FSG pulse that is 1 CLKG cycle wide. The FPER bits of SRGR2 are not programmed; the period from the start of a frame-sync pulse to the start of the next pulse is determined by the arrival of the next inactive-to-active transition on the FSR pin. Each of the figures shows what happens to CLKG when it is initially synchronized and GSYNC = 1, and when it is not initially synchronized and GSYNC = 1. The second figure has a slower CLKG frequency (it has a larger divide-down value in the CLKGDV bits of SRGR1).

Figure 9–18. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 1

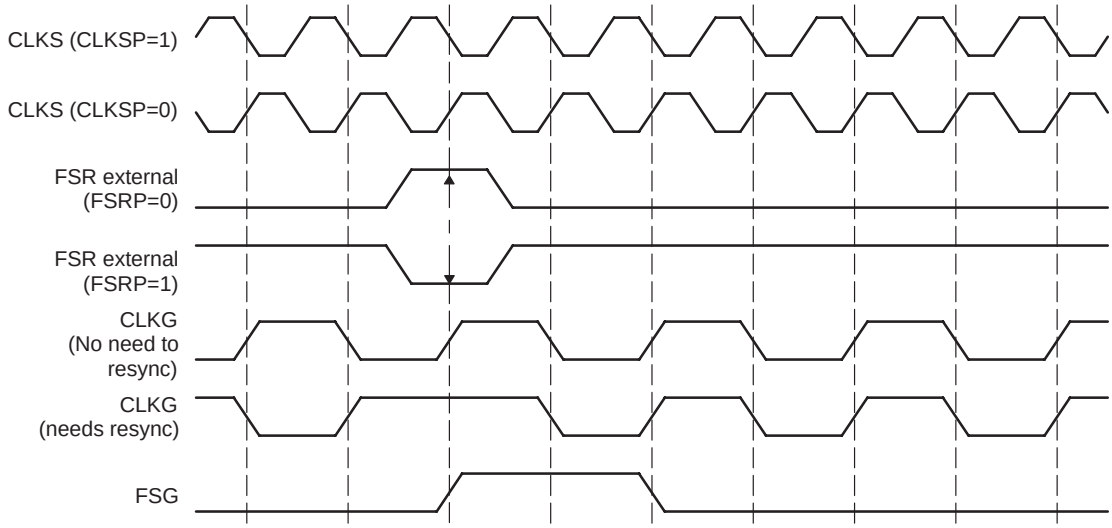
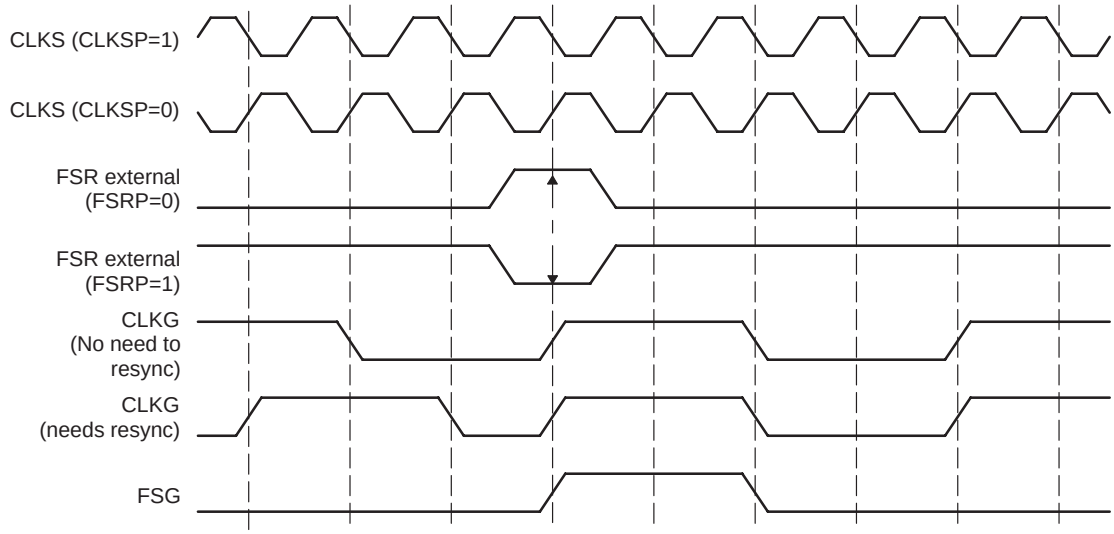


Figure 9–19. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 3



9.3.4 Reset and Initialization Procedure for the Sample Rate Generator

To reset and initialize the sample rate generator:

1) Place the McBSP/sample rate generator in reset.

During a DSP reset, the sample rate generator, the receiver, and the transmitter reset bits (GRST_, RRST_, and XRST_) are automatically forced to 0. Otherwise, during normal operation, the sample rate generator can be reset by making GRST_ = 0 in SPCR2, provided that CLKG and/or FSG is not used by any portion of the McBSP. Depending on your system you may also want to reset the receiver (RRST_ = 0 in SPCR1) and reset the transmitter (XRST_ = 0 in SPCR2).

If GRST_ = 0 due to a DSP reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven inactive-low. If GRST_ = 0 due to program code, CLKG and FSG are driven low (inactive).

2) Program registers that affect the sample rate generator.

Program the sample rate generator registers (SRGR1 and SRGR2) as required for your application. If necessary, other control registers can be loaded with desired values, provided the respective portion of the McBSP (the receiver or transmitter) is in reset.

After the sample rate generator registers are programmed, wait 2 CLKSRG cycles. This ensures proper synchronization internally.

3) Enable the sample rate generator (take it out of reset).

In SPCR2, make GRST_ = 1 to enable the sample rate generator.

After the sample rate generator is enabled, wait 2 CLKG cycles for the sample rate generator logic to stabilize.

On the next rising edge of CLKSRG, CLKG transitions to 1 and starts clocking with a frequency equal to

$$\text{CLKG frequency} = \frac{\text{Input clock frequency}}{(\text{CLKGDV} + 1)}$$

where the input clock is selected with the SCLKME bit of PCR and the CLKSM bit of SRGR2:

SCLKME	CLKSM	Input Clock For Sample Rate Generator
0	0	Signal on CLKS pin
0	1	CPU clock
1	0	Signal on CLKR pin
1	1	Signal on CLKX pin

4) If necessary, enable the receiver and/or the transmitter.

If necessary, remove the receiver and/or transmitter from reset by setting RRST_ and/or XRST_ = 1.

5) If necessary, enable the frame-sync logic of the sample rate generator.

After the required data acquisition setup is done (DXR[1/2] is loaded with data), set FRST_ = 1 in SPCR2 if an internally generated frame-sync pulse is required. FSG is generated with an active-high edge after the programmed number of CLKG clocks (FPER + 1) have elapsed.

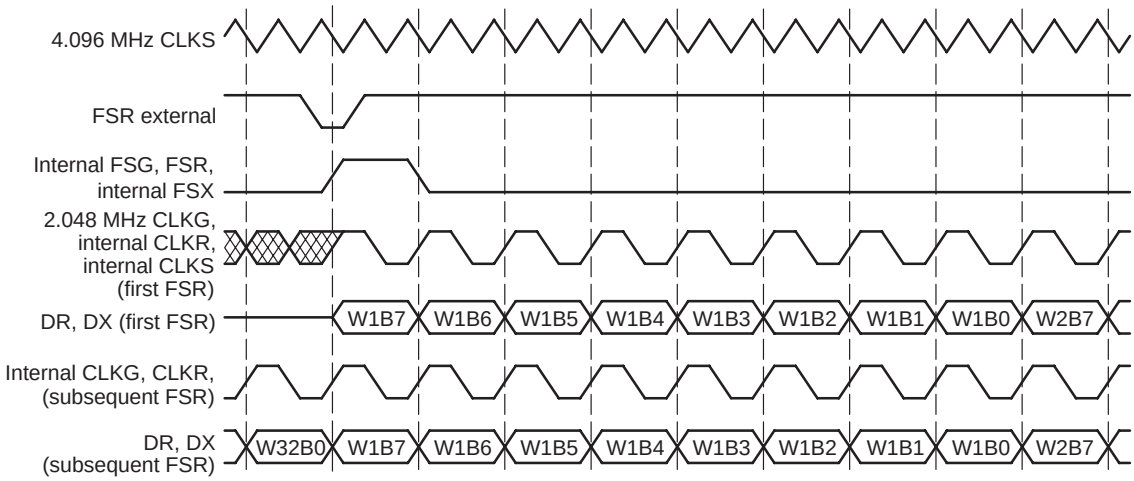
9.3.5 Sample Rate Generator Clocking Examples

This section shows three examples of using the sample rate generator to clock data during transmission and reception.

9.3.5.1 Double-Rate ST-Bus Clock

Figure 9–20 shows McBSP configuration to be compatible with the Mitel ST-Bus. Note that this operation is running at maximum frame frequency (described on page 9-14).

Figure 9–20. ST-BUS and MVIP Clocking Example



For this McBSP configuration:

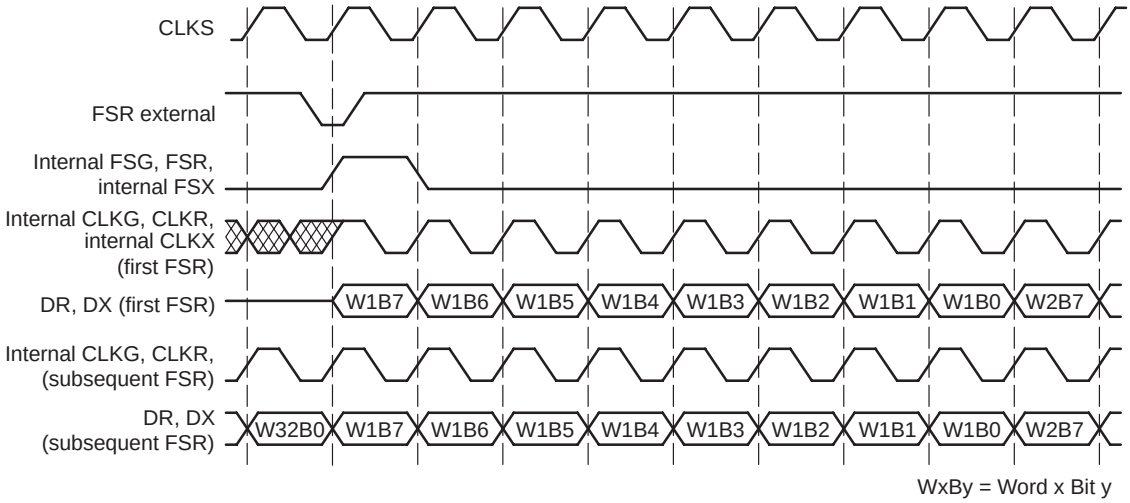
- DLB = 0: Digital loopback mode off, CLKSTP = 00b: Clock stop mode off, and CLKRM/CLKXM = 1: Internal CLKR/CLKX generated internally by sample rate generator
- GSYNC = 1: Synchronize CLKG with external frame-sync signal input on FSR pin. CLKG is not synchronized until the frame-sync signal is active. FSR is regenerated internally to form a minimum pulse width.
- SCLKME = 0 and CLKSM = 1: External clock signal at CLKS pin drives the sample rate generator
- CLKSP = 1: Falling edge of CLKS generates CLKG and thus internal CLK(R/X)
- CLKGDV = 1: Frequency of receive clock (shown as CLKR) is half CLKS frequency
- FSRP/FSXP = 1: Active-low frame-sync pulse
- RFRLEN1/XFRLEN1 = 11111b: 32 words per frame
- RWDLEN1/XWDLEN1 = 0: 8 bits per word
- RPHASE/XPHASE = 0: Single-phase frame and thus (R/X)FRLEN2 and (R/X)WDLEN2 are ignored
- RDATDLY/XDATDLY = 0: No data delay

9.3.5.2 Single-Rate ST-Bus Clock

The example in Figure 9–21 is the same as the double-rate ST-bus clock example in section 9.3.5.1 except that:

- CLKGDV = 0: CLKS drives internal CLK(R/X) without any divide down (single-rate clock).
- CLKSP = 0: Rising edge of CLKS generates CLKG and internal CLK(R/X)

Figure 9–21. Single-Rate Clock Example



The rising edge of CLKS is used to detect the external FSR pulse, which is used to resynchronize internal McBSP clocks and generate a frame-sync pulse for internal use. The internal frame-sync pulse is generated so that it is wide enough to be detected on the falling edge of internal clocks.

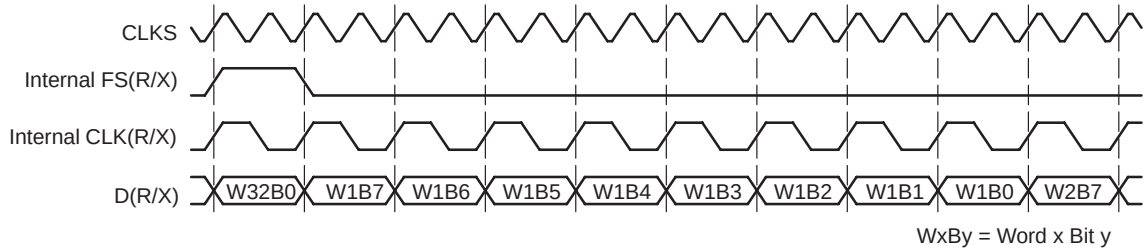
9.3.5.3 Other Double-Rate Clock

The example in Figure 9–22 is the same as the double-rate ST-bus clock example in section 9.3.5.1 except that:

- CLKSP = 0: Rising edge of CLKS generates CLKG and thus CLK(R/X)
- CLKGDV = 1: Frequency of CLKG (and thus internal CLKR and internal CLKX) is half CLKS frequency
- FSRM/FSXM = 0: Frame synchronization is externally generated. The frame-sync pulse is wide enough to be detected.
- GSYNC = 0: CLKS drives CLKG. CLKG runs freely; it is not resynchronized by a pulse on the FSR pin.

- FSRP/FSXP = 0: Active-high input frame-sync signal
- RDATDLY/XDATDLY = 1: Data delay of one bit

Figure 9–22. Double-Rate Clock Example



9.4 McBSP Exception/Error Conditions

There are five serial port events that may constitute a system error:

- **Receiver Overrun (RFULL = 1)**. This occurs when DRR1 has not been read since the last RBR-to-DRR copy. Consequently, the receiver does not copy a new word from the RBR(s) to the DRR(s), and the RSR(s) are now full with another new word shifted in from DR. Therefore, RFULL = 1 indicates an error condition wherein any new data that may arrive at this time on DR will replace the contents of the RSR(s), and thus, the previous word is lost. The RSR(s) continue to be overwritten as long as new data arrives on DR and DRR1 is not read. For more details about overrun in the receiver, see page 9-37.

- **Unexpected Receive Frame-Sync Pulse (RSYNCERR = 1)**. This occurs during reception when RFIG = 0 and an unexpected frame-sync pulse occurs. An unexpected frame-sync pulse is one that begins the next frame transfer before all the bits of the current frame have been received. Such a pulse causes data reception to abort and restart. If new data has been copied into the RBR(s) from the RSR(s) since the last RBR-to-DRR copy, this new data in the RBR(s) is lost. This is because no RBR-to-DRR copy occurs; the reception has been restarted. For more details about receive frame-sync errors, see page 9-38.

- **Transmitter Data Overwrite**. This occurs when the CPU or DMA controller overwrites data in the DXR(s) before the data is copied to the XSR(s). The overwritten data never reaches the DX pin. For more details about overwrite in the transmitter, see page 9-41.

- **Transmitter Underflow (XEMPTY_ = 0)**. If a new frame-sync signal arrives before new data is loaded into DXR1, the previous data in the DXR(s) is sent again. This will continue for every new frame-sync pulse that arrives until DXR1 is loaded with new data. For more details about underflow in the transmitter, see page 9-42.

- **Unexpected Transmit Frame-Synch Pulse (XSYNCERR = 1)**. This occurs during transmission when XFIG = 0 and an unexpected frame-sync pulse occurs. An unexpected frame-sync pulse is one that begins the next frame transfer before all the bits of the current frame have been transferred. Such a pulse causes the current data transmission to abort and restart. If new data has been written to the DXR(s) since the last DXR-to-XSR copy, the current value in the XSR(s) is lost. For more details about transmit frame-sync errors, see page 9-44.

9.4.1 Overrun in the Receiver

RFULL = 1 in SPCR1 indicates that the receiver has experienced overrun and is in an error condition. RFULL is set when all of the following conditions are met:

1) DRR1 has not been read since the last RBR-to-DRR copy (RRDY = 1).

2) RBR1 is full and an RBR-to-DRR copy has not occurred.

3) RSR1 is full and an RSR1-to-RBR copy has not occurred.

As described in the section on McBSP reception (page 9-18), data arriving on DR is continuously shifted into RSR1 (for word length of 16 bits or smaller) or RSR2 and RSR1 (for word length larger than 16 bits). Once a complete word is shifted into the RSR(s), an RSR-to-RBR copy can occur only if the previous data in RBR1 has been copied to DRR1. The RRDY bit is set when new data arrives in DRR1 and is cleared when that data is read from DRR1. Until RRDY = 0, the next RBR-to-DRR copy will not take place, and the data is held in the RSR(s). New data arriving on the DR pin is shifted into RSR(s), and the previous content of the RSR(s) is lost.

You can prevent the loss of data if DRR1 is read no later than 2.5 cycles before the end of the third word is shifted into the RSR1.

Important: If both DRRs are needed (word length larger than 16 bits), the CPU or the DMA controller must read from DRR2 first and then from DRR1. As soon as DRR1 is read, the next RBR-to-DRR copy is initiated. If DRR2 is not read first, the data in DRR2 is lost.

Note that after the receiver starts running from reset, a minimum of three words must be received before RFULL is set. Either of the following events clears the RFULL bit and allows subsequent transfers to be read properly:

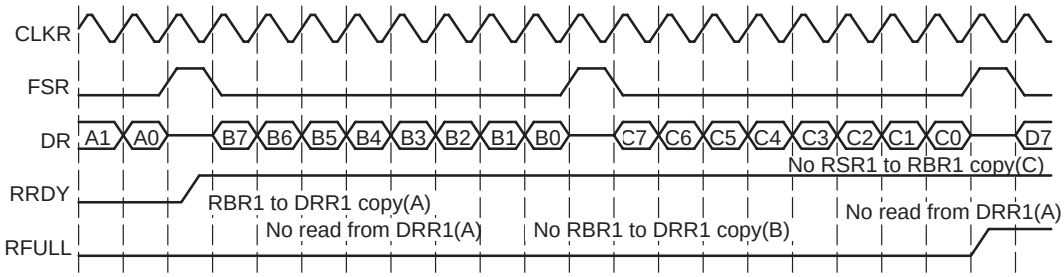
- ❏ The CPU or DMA controller reads DRR1.
- ❏ The receiver is reset individually (RRST_ = 0) or as part of a DSP reset.

Another frame-sync pulse is required to restart the receiver.

9.4.1.1 Example of the Overrun Condition

Figure 9–23 shows the receive overrun condition. Because serial word A is not read from DRR1 before serial word B arrives in RBR1, B is not transferred to DRR1 yet. Another new word (C) arrives and RSR1 is full with this data. DRR1 is finally read, but not earlier than 2.5 cycles before the end of word C. Therefore, new data (D) overwrites word C in RSR1. If DRR1 is not read in time, the next word can overwrite D.

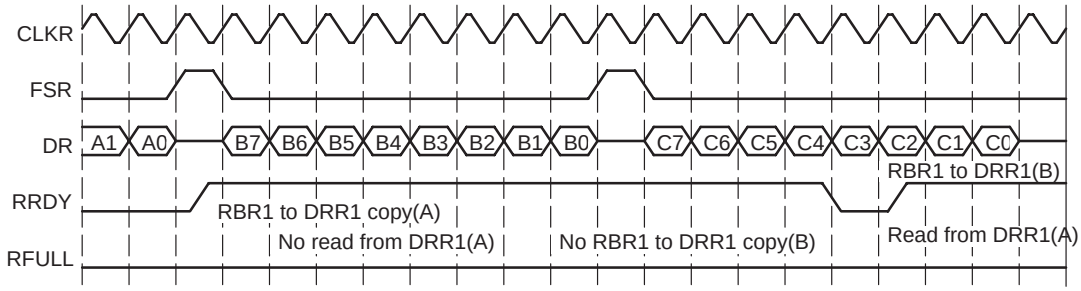
Figure 9–23. Overrun in the McBSP Receiver



9.4.1.2 Example of Preventing the Overrun Condition

Figure 9–24 shows the case where RFULL is set, but the overrun condition is prevented by a read from DRR1 at least 2.5 cycles before the next serial word (C) is completely shifted into RSR1. This ensures that an RBR1-to-DRR1 copy of word B occurs before receiver attempts to transfer word C from RSR1 to RBR1.

Figure 9–24. Overrun Prevented in the McBSP Receiver



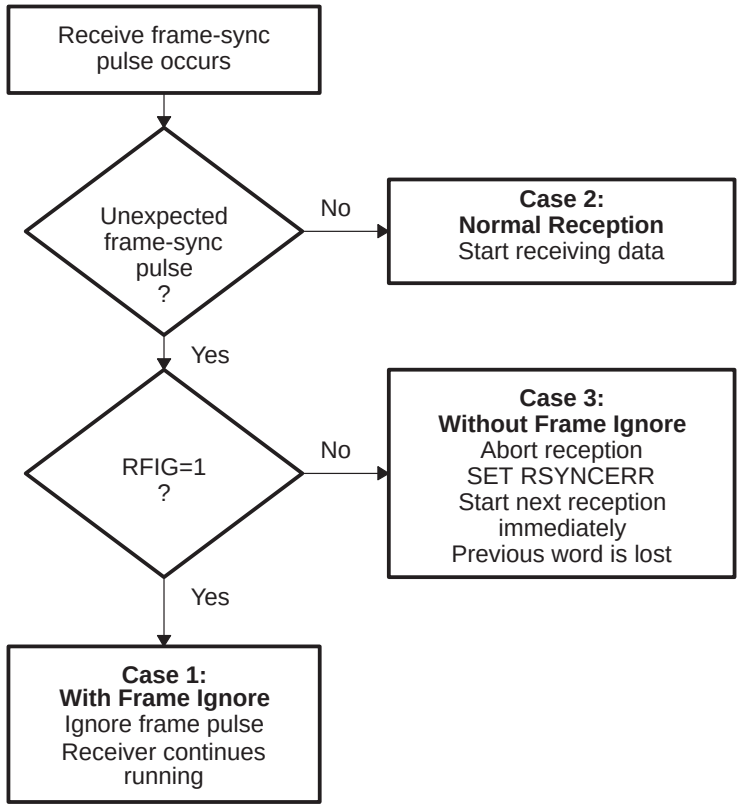
9.4.2 Unexpected Receive Frame-Sync Pulse

Section 9.4.2.1 shows how the McBSP responds to any receive frame-sync pulses, including an unexpected pulse. Sections 9.4.2.2 and 9.4.2.3 show an examples of a frame-sync error and an example of how to prevent such an error, respectively.

9.4.2.1 Possible Responses to Receive Frame-Sync Pulses

Figure 9–25 shows the decision tree that the receiver uses to handle all incoming frame-sync pulses. The figure assumes that the receiver has been started (RRST_ = 1 in SPCR1). Case 3 in the figure is the case in which an error occurs.

Figure 9–25. Possible Responses to Receive Frame-Sync Pulses



Any one of three cases can occur:

- **Case 1:** Unexpected internal FSR pulses with RFIG = 1 in RCR2. Receive frame-sync pulses are ignored, and the reception continues.

- **Case 2:** Normal serial port reception. Reception continues normally because the frame-sync pulse is not unexpected. There are three possible reasons why a receive operation might *not* be in progress when the pulse occurs:

  - The FSR pulse is the first after the receiver is enabled (RRST_ = 1 in SPCR1).

  - The FSR pulse is the first after DRR[1,2] is read, clearing a receiver full (RFULL = 1 in SPCR1) condition.

- The serial port is in the interpacket intervals. The programmed data delay for reception (programmed with the RDATDLY bits in RCR2) may start during these interpacket intervals for the first bit of the next word to be received. Thus, at maximum frame frequency, frame synchronization can still be received 0 to 2 clock cycles before the first bit of the synchronized frame.

❑ **Case 3:** Unexpected receive frame synchronization with RFIG = 0 (frame-sync pulses not ignored). Unexpected frame-sync pulses can originate from an external source or from the internal sample rate generator.

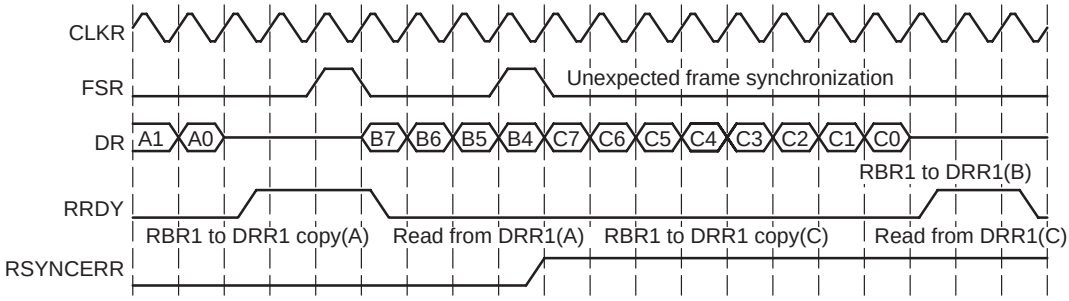
If a frame-sync pulse starts the transfer of a new frame before the current frame is fully received, this pulse is treated as an unexpected frame-sync pulse, and the receiver sets the receive frame-sync error bit (RSYNCERR) in SPCR1. RSYNCERR can be cleared only by a receiver reset or by a write of 0 to this bit.

If you want the McBSP to notify the CPU of receive frame-sync errors, you can set a special receive interrupt mode with the RINTM bits of SPCR1. When RINTM = 11b, the McBSP sends a receive interrupt (RINT) request to the CPU each time that RSYNCERR is set.

9.4.2.2 Example of an Unexpected Receive Frame-Sync Pulse

Figure 9–26 shows an unexpected receive frame-sync pulse during normal operation of the serial port, with time intervals between data packets. When the unexpected frame-sync pulse occurs, the RSYNCERR bit is set, the reception of data B is aborted, and the reception of data C begins. In addition, if RINTM = 11b, the McBSP sends a receive interrupt (RINT) request to the CPU.

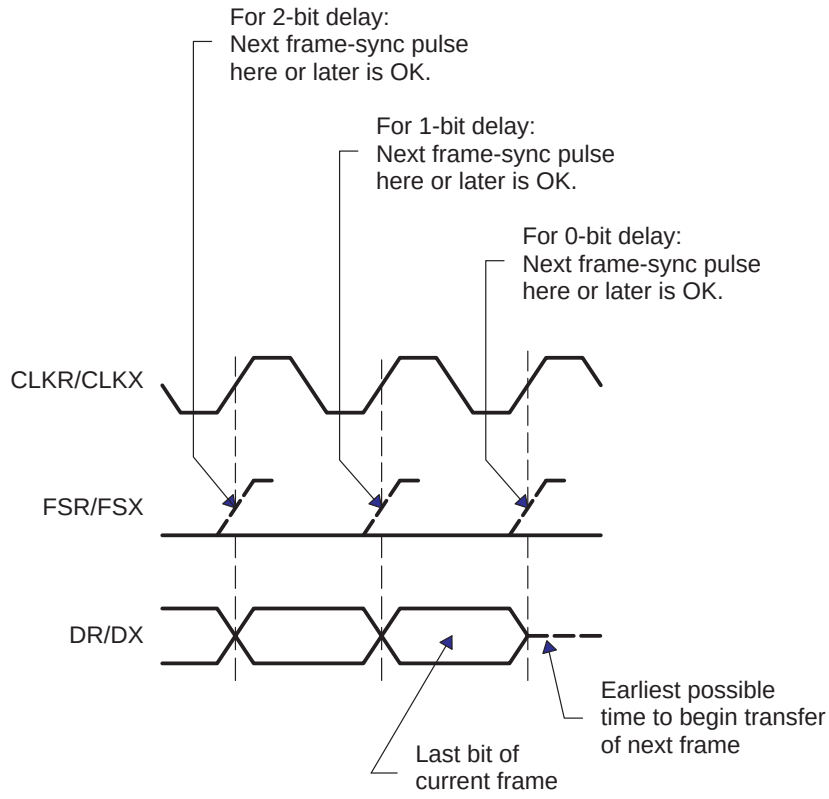
Figure 9–26. An Unexpected Frame-Sync Pulse During a McBSP Reception



9.4.2.3 Preventing Unexpected Receive Frame-Sync Pulses

Each frame transfer can be delayed by 0, 1, or 2 CLKR cycles, depending on the value in the RDATDLY bits of RCR2. For each possible data delay, Figure 9–27 shows when a new frame-sync pulse on FSR can safely occur relative to the last bit of the current frame.

Figure 9–27. Proper Positioning of Frame-Sync Pulses



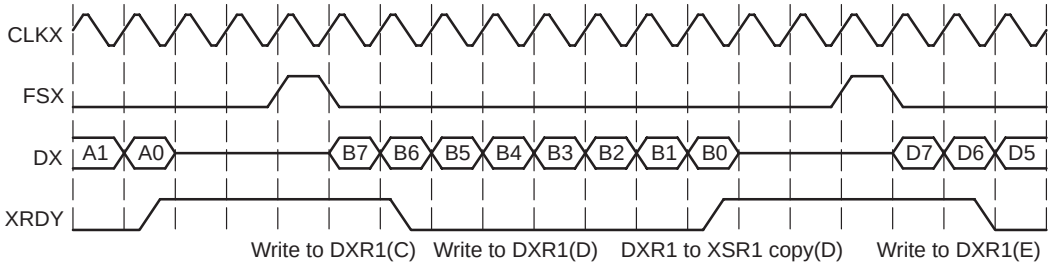
9.4.3 Overwrite in the Transmitter

As described in the section on McBSP transmission (page 9-19), after the CPU or DMA controller writes data to the DXR(s), the transmitter must then copy that data to the XSR(s) and then shift each bit from the XSR(s) to the DX pin. If new data is written to the DXR(s) before the previous data is copied to the XSR(s), the previous data in the DXR(s) is overwritten and thus lost.

9.4.3.1 Example of the Overwrite Condition

Figure 9–28 shows what happens if the data in DXR1 is overwritten before being transmitted. Initially, DXR1 is loaded with data C. A subsequent write to DXR1 overwrites C with D before C is copied to XSR1. Thus, C is never transmitted on DX.

Figure 9–28. Data in the McBSP Transmitter Overwritten and Thus Not Transmitted



9.4.3.2 Preventing Overwrites

You can prevent CPU overwrites by making the CPU:

- ❏ Poll for XRDY = 1 in SPCR2 before writing to the DXR(s). XRDY is set when data is copied from DXR1 to XSR1 and is cleared when new data is written to DXR1.
- ❏ Wait for a transmit interrupt (XINT) before writing to the DXR(s). When XINTM = 00b in SPCR2, the transmitter sends XINT to the CPU each time XRDY is set.

You can prevent DMA overwrites by synchronizing DMA transfers to the transmit synchronization event XEVT. The transmitter sends an XEVT signal each time XRDY is set.

9.4.4 Underflow in the Transmitter

The McBSP indicates a transmitter empty (or underflow) condition by clearing the XEMPTY_ bit in SPCR2. Either of the following events activates XEMPTY_ (XEMPTY_ = 0):

- ❏ DXR1 has not been loaded since the last DXR-to-XSR copy, and all bits of the data word in the XSR(s) have been shifted out on the DX pin.
- ❏ The transmitter is reset (by forcing XRST_ = 0 in SPCR2, or by a DSP reset) and is then restarted.

In the underflow condition, the transmitter continues to transmit the old data that is in the DXR(s) for every new transmit frame-sync signal until a new value is loaded into DXR1 by the CPU or the DMA controller.

Note:

If both DXRs are needed (word length larger than 16 bits), the CPU or the DMA controller must load DXR2 first and then load DXR1. As soon as DXR1 is loaded, the contents of both DXRs are copied to the transmit shift registers (XSRs). If DXR2 is not loaded first, the previous content of DXR2 is passed to the XSR2.

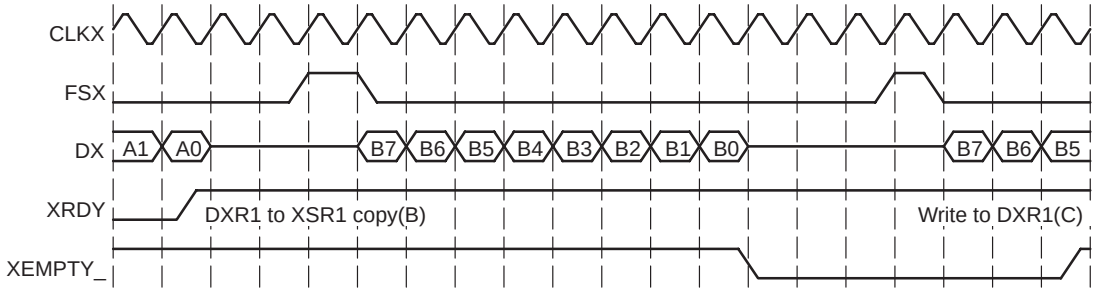
XEMPTY_ is deactivated (XEMPTY_ = 1) when a new word in DXR1 is transferred to XSR1. If FSXM = 1 in PCR and FSGM = 0 in SRGR2, the transmitter generates a single internal FSX pulse in response to a DXR-to-XSR copy. Otherwise, the transmitter waits for the next frame-sync pulse before sending out the next frame on DX.

When the transmitter is taken out of reset (XRST_ = 1), it is in a transmitter ready (XRDY = 1 in SPCR2) and transmitter empty (XEMPTY_ = 0) state. If DXR1 is loaded by the CPU or the DMA controller before internal FSX goes active high, a valid DXR-to-XSR transfer occurs. This allows for the first word of the first frame to be valid even before the transmit frame-sync pulse is generated or detected. Alternatively, if a transmit frame-sync pulse is detected before DXR1 is loaded, zeros will be output on DX.

9.4.4.1 Example of the Underflow Condition

Figure 9–29 shows an underflow condition. After B is transmitted, DXR1 is not reloaded before the subsequent frame-sync pulse. Thus, B is again transmitted on DX.

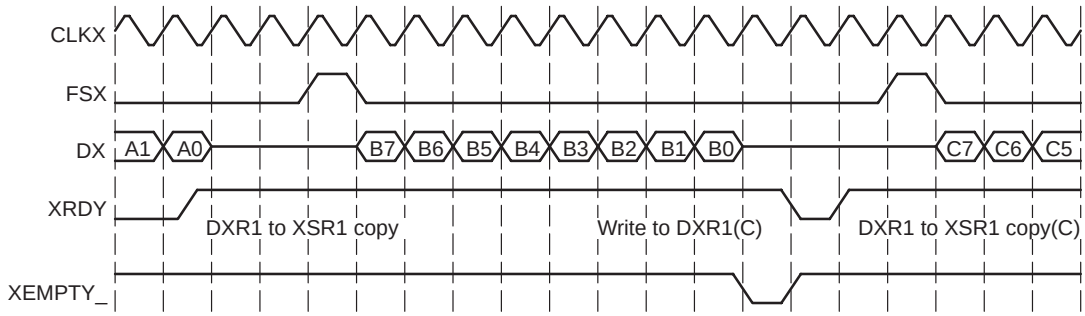
Figure 9–29. Underflow During McBSP Transmission



9.4.4.2 Example of Preventing the Underflow Condition

Figure 9–30 shows the case of writing to DXR1 just before an underflow condition would otherwise occur. After B is transmitted, C is written to DXR1 before the next frame-sync pulse. As a result, there is no underflow; B is not transmitted twice.

Figure 9–30. Underflow Prevented in the McBSP Transmitter



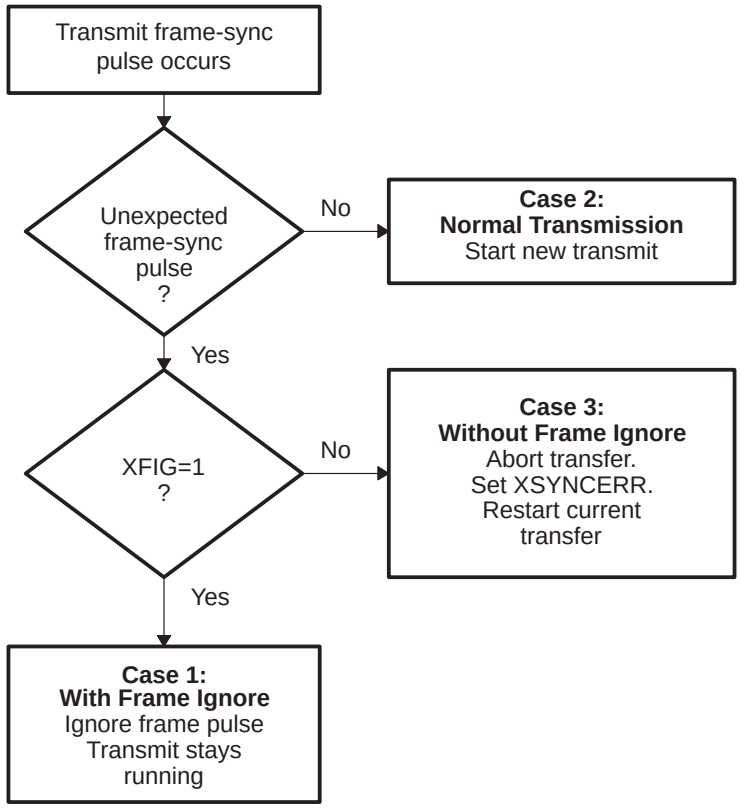
9.4.5 Unexpected Transmit Frame-Sync Pulse

Section 9.4.5.1 shows how the McBSP responds to any transmit frame-sync pulses, including an unexpected pulse. Sections 9.4.5.2 and 9.4.5.3 show an examples of a frame-sync error and an example of how to prevent such an error, respectively.

9.4.5.1 Possible Responses to Transmit Frame-Sync Pulses

Figure 9–31 shows the decision tree that the transmitter uses to handle all incoming frame-sync pulses. The figure assumes that the transmitter has been started (XRST_ = 1 in SPCR2). Case 3 in the figure is the case in which an error occurs.

Figure 9–31. Possible Responses to Transmit Frame-Sync Pulses



Any one of three cases can occur:

- **Case 1:** Unexpected internal FSX pulses with XFIG = 1 in XCR2. Transmit frame-sync pulses are ignored, and the transmission continues.

- **Case 2:** Normal serial port transmission. Transmission continues normally because the frame-sync pulse is not unexpected. There are two possible reasons why a transmit operations might *not* be in progress when the pulse occurs:

  This FSX pulse is the first after the transmitter is enabled (XRST_ = 1).

  The serial port is in the interpacket intervals. The programmed data delay for transmission (programmed with the XDATDLY bits of XCR2) may start during these interpacket intervals before the first bit of the previous word is transmitted. Thus, at maximum packet frequency, frame synchronization can still be received 0 to 2 clock cycles before the first bit of the synchronized frame.

- **Case 3:** Unexpected transmit frame synchronization with XFIG = 0 (frame-sync pulses not ignored). Unexpected frame-sync pulses can originate from an external source or from the internal sample rate generator.

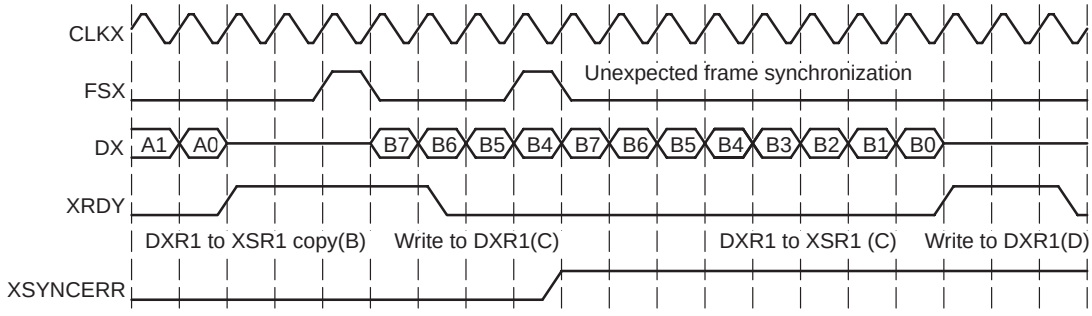
  If a frame-sync pulse starts the transfer of a new frame before the current frame is fully transmitted, this pulse is treated as an unexpected frame-sync pulse, and the transmitter sets the transmit frame-sync error bit (XSYNCERR) in SPCR2. XSYNCERR can be cleared only by a transmitter reset or by a write of 0 to this bit.

  If you want the McBSP to notify the CPU of frame-sync errors, you can set a special transmit interrupt mode with the XINTM bits of SPCR2. When XINTM = 11b, the McBSP sends a transmit interrupt (XINT) request to the CPU each time that XSYNCERR is set.

9.4.5.2 Example of an Unexpected Transmit Frame-Sync Pulse

Figure 9–32 shows an unexpected transmit frame-sync pulse during normal operation of the serial port, with intervals between the data packets. When the unexpected frame-sync pulse occurs, the XSYNCERR bit is set and because no new data has been passed to XSR1 yet, the transmission of data B is restarted. In addition, if XINTM = 11b, the McBSP sends a transmit interrupt (XINT) request to the CPU.

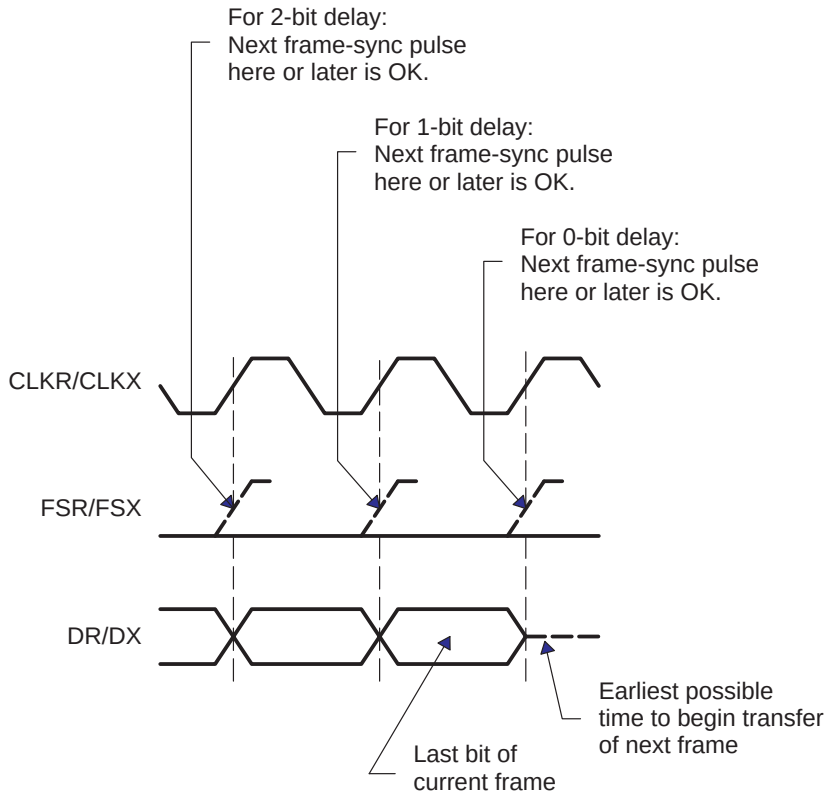
Figure 9–32. An Unexpected Frame-Sync Pulse During a McBSP Transmission



9.4.5.3 Preventing Unexpected Transmit Frame-Sync Pulses

Each frame transfer can be delayed by 0, 1, or 2 CLKX cycles, depending on the value in the XDATDLY bits of XCR2. For each possible data delay, Figure 9–33 shows when a new frame-sync pulse on FSX can safely occur relative to the last bit of the current frame.

Figure 9–33. Proper Positioning of Frame-Sync Pulses



9.5 Multichannel Selection Modes

9.5.1 Channels, Blocks, and Partitions

A McBSP **channel** is a time slot for shifting in/out the bits of one serial word. Each McBSP supports up to 128 channels for reception and 128 channels for transmission.

In the receiver and in the transmitter, the 128 available channels are divided into eight **blocks** that each contain 16 contiguous channels:

Block 0: Channels 0–15	Block 4: Channels 64–79
Block 1: Channels 16–31	Block 5: Channels 80–95
Block 2: Channels 32–47	Block 6: Channels 96–111
Block 3: Channels 48–63	Block 7: Channels 112–127

The blocks are assigned to **partitions** according to the selected partition mode. In the 2-partition mode (described in section 9.5.4), you assign one even-numbered block (0, 2, 4, or 6) to partition A and one odd-numbered block (1, 3, 5, or 7) to partition B. In the 8-partition mode (described in section 9.5.5), blocks 0 through 7 are automatically assigned to partitions, A through H, respectively.

The number of partitions for reception and the number of partitions for transmission are independent. For example, it is possible to use 2 receive partitions (A and B) and 8 transmit partitions (A–H).

9.5.2 Multichannel Selection

When a McBSP uses a time-division multiplexed (TDM) data stream while communicating with other McBSPs or serial devices, the McBSP may need to receive and/or transmit on only a few channels. To save memory and bus bandwidth, you can use a multichannel selection mode to prevent data flow in some of the channels.

Each channel partition has a dedicated channel enable register. If the appropriate multichannel selection mode is on, each bit in the register controls whether data flow is allowed or prevented in one of the channels that is assigned to that partition.

The McBSP has one receive multichannel selection mode (described in section 9.5.6) and three transmit multichannel selection modes (described in section 9.5.7).

9.5.3 Configuring a Frame for Multichannel Selection

Before you enable a multichannel selection mode, make sure you properly configure the data frame:

- ❏ Select a single-phase frame (RPHASE/XPHASE = 0). Each frame represents a TDM data stream.
- ❏ Set a frame length (in RFRLEN1/XFRLEN1) that includes the highest-numbered channel that will be used. For example, if you plan to use channels 0, 15, and 39 for reception, the receive frame length must be at least 40 (RFRLEN1 = 39). If XFRLEN1 = 39 in this case, the receiver creates 40 time slots per frame but only receives data during time slots 0, 15, and 39 of each frame.

9.5.4 Using Two Partitions

For multichannel selection operation in the receiver and/or the transmitter, you can use two partitions or eight partitions (described in section 9.5.5). If you choose the 2-partition mode (RMCME = 0 for reception, XMCME = 0 for transmission), McBSP channels are activated using an alternating scheme. In response to a frame-sync pulse, the receiver or transmitter begins with the channels in partition A and then alternates between partitions B and A until the complete frame has been transferred. When the next frame-sync pulse occurs, the next frame is transferred, beginning with the channels in partition A.

9.5.4.1 Assigning Blocks to Partitions A and B

For reception, any two of the eight receive-channel blocks can be assigned to receive partitions A and B, which means up to 32 receive channels can be enabled at any given point in time. Similarly, any two of the eight transmit-channel blocks (up 32 enabled transmit channels) can be assigned to transmit partitions A and B.

For reception:

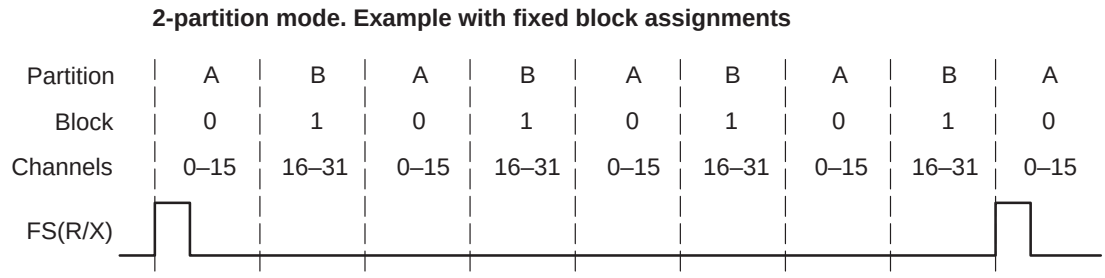
- ❏ Assign an even-numbered channel block (0, 2, 4, or 6) to receive partition A by writing to the RPABLK bits. In the receive multichannel selection mode (described in section 9.5.6), the channels in this partition are controlled by receive channel enable register A (RCERA).
- ❏ Assign an odd-numbered block (1, 3, 5, or 7) to receive partition B with the RPBBLK bits. In the receive multichannel selection mode, the channels in this partition are controlled by receive channel enable register B (RCERB).

For transmission:

- Assign an even-numbered channel block (0, 2, 4, or 6) to transmit partition A by writing to the XPABLK bits. In one of the transmit multichannel selection modes (described in section 9.5.7), the channels in this partition are controlled by transmit channel enable register A (XCERA).
- Assign an odd-numbered block (1, 3, 5, or 7) to transmit partition B with the XPBBLK bits. In one of the transmit multichannel selection modes, the channels in this partition are controlled by transmit channel enable register B (XCERB).

Figure 9–34 shows an example of alternating between the channels of partition A and the channels of partition B. Channels 0–15 have been assigned to partition A, and channels 16–31 have been assigned to partition B. In response to a frame-sync pulse, the McBSP begins a frame transfer with partition A and then alternates between partitions B and A until the complete frame is transferred.

Figure 9–34. Alternating Between the Channels of Partition A and the Channels of Partition B



As explained next, you can dynamically change which blocks of channels are assigned to the partitions.

9.5.4.2 Reassigning Blocks During Reception/Transmission

If you want to use more than 32 channels, you can change which channel blocks are assigned to partitions A and B during the course of a data transfer. However, these changes must be carefully timed. While a partition is being transferred, its the associated block assignment bits cannot be modified, and its associated channel enable register cannot be modified. For example, if block 3 is being transferred and block 3 is assigned to partition A, you cannot modify (R/X)PABLK to assign different channels to partition A, and you cannot modify (R/X)CERA to change the channel configuration for partition A. Several features of the McBSP help you time the reassignment:

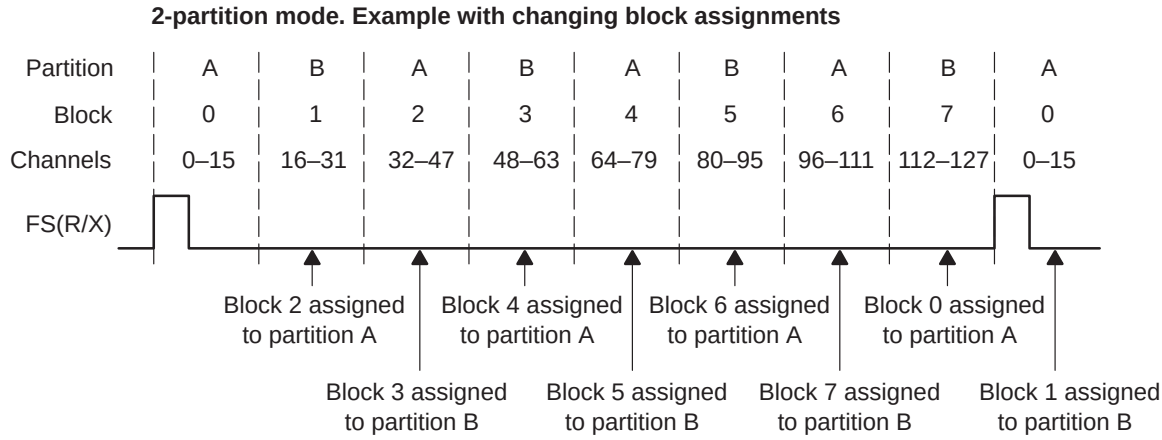
- The block of channels currently involved in reception/transmission (the current block) is reflected in the RCBLK/XCBLK bits. Your program can

poll these bits to determine which partition is active. When a partition is not active, it is safe to change its block assignment and channel configuration.

- ❑ At the end of every block (at the boundary of two partitions), an interrupt can be sent to the CPU. In response to the interrupt, the CPU can then check the RCBLK/XCBLK bits and update the inactive partition. See *Using Interrupts Between Block Transfers* on page 9-58.

Figure 9–35 shows an example of reassigning channels throughout a data transfer. In response to a frame-sync pulse, the McBSP alternates between partitions A and B. Whenever partition B is active, the CPU changes the block assignment for partition A. Whenever, partition A is active, the CPU changes the block assignment for partition B.

Figure 9–35. Reassigning Channel Blocks Throughout a McBSP Data Transfer



9.5.5 Using Eight Partitions

For multichannel selection operation in the receiver and/or the transmitter, you can use eight partitions or two partitions (described in section 9.5.4). If you choose the 8-partition mode (RMCME = 1 for reception, XMCME = 1 for transmission), McBSP channels are activated in the following order: A, B, C, D, E, F, G, H. In response to a frame-sync pulse, the receiver or transmitter begins with the channels in partition A and then continues with the other partitions in order until the complete frame has been transferred. When the next frame-sync pulse occurs, the next frame is transferred, beginning with the channels in partition A.

In the 8-partition mode, the (R/X)PABLK and (R/X)PBBLK bits are ignored and the 16-channel blocks are assigned to the partitions as shown in Table 9–6 and Table 9–7. These assignments cannot be changed. The tables also show the registers used to control the channels in the partitions.

Table 9–6. Receive Channel Assignment and Control When Eight Receive Partitions Are Used

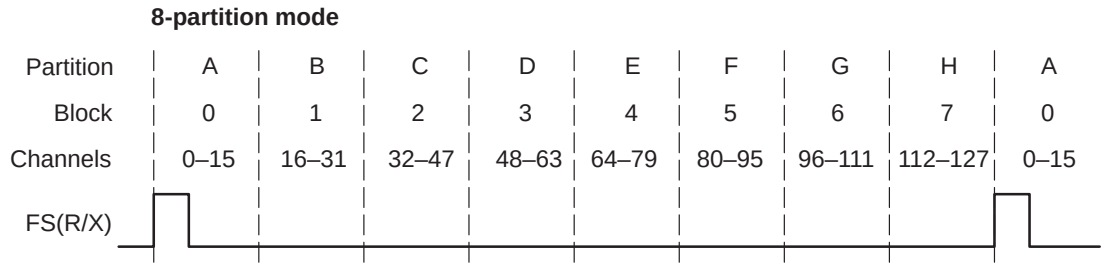
Receive Partition	Assigned Block of Receive Channels	Register Used For Channel Control
A	Block 0: channels 0 through 15	RCERA
B	Block 1: channels 16 through 31	RCERB
C	Block 2: channels 32 through 47	RCERC
D	Block 3: channels 48 through 63	RCERD
E	Block 4: channels 64 through 79	RCERE
F	Block 5: channels 80 through 95	RCERF
G	Block 6: channels 96 through 111	RCERG
H	Block 7: channels 112 through 127	RCERH

Table 9–7. Transmit Channel Assignment and Control When Eight Transmit Partitions Are Used

Transmit Partition	Assigned Block of Transmit Channels	Register Used For Channel Control
A	Block 0: channels 0 through 15	XCERA
B	Block 1: channels 16 through 31	XCERB
C	Block 2: channels 32 through 47	XCERC
D	Block 3: channels 48 through 63	XCERD
E	Block 4: channels 64 through 79	XCERE
F	Block 5: channels 80 through 95	XCERF
G	Block 6: channels 96 through 111	XCERG
H	Block 7: channels 112 through 127	XCERH

Figure 9–36 shows an example of the McBSP using the 8-partition mode. In response to a frame-sync pulse, the McBSP begins a frame transfer with partition A and then activates B, C, D, E, F, G, and H to complete a 128-word frame.

Figure 9–36. McBSP Data Transfer in the 8-Partition Mode



9.5.6 Receive Multichannel Selection Mode

The RMCM bit of MCR1 determines whether all channels or only selected channels are enabled for reception. When RMCM = 0, all 128 receive channels are enabled and cannot be disabled. When **RMCM = 1**, the receive multichannel selection mode is enabled. In this mode:

- ❏ Channels can be individually enabled or disabled. The only channels enabled are those selected in the appropriate receive channel enable registers (RCERs, described in section 9.13.9 on page 9-202). The way channels are assigned to the RCERs depends on the number of receive channel partitions (2 or 8), as defined by the RMCME bit of MCR1.

- ❏ If a receive channel is disabled, any bits received in that channel are passed only as far as the receive buffer register(s) (RBR(s)). The receiver does not copy the content of the RBR(s) to the DRR(s), and as a result, does not set the receiver ready bit (RRDY). Therefore, no DMA synchronization event (REVT) is generated, and if the receiver interrupt mode depends on RRDY (RINTM = 00b), no interrupt is generated.

As an example of how the McBSP behaves in the receive multichannel selection mode, suppose you enable only channels 0, 15, and 39 and that the frame length is 40. The McBSP:

1) Accepts bits shifted in from the DR pin in channel 0
2) Ignores bits received in channels 1–14
3) Accepts bits shifted in from the DR pin in channel 15
4) Ignores bits received in channels 16–38
5) Accepts bits shifted in from the DR pin in channel 39

9.5.7 Transmit Multichannel Selection Modes

The XMCM bits of XCR2 determine whether all channels or only selected channels are enabled and unmasked for transmission. More details on enabling and masking are in section 9.5.7.1. The McBSP has three transmit multichannel selection modes (XMCM = 01b, XMCM = 10b, and XMCM = 11b), which are described in the following table:

Table 9–8. Selecting a Transmit Multichannel Selection Mode with the XMCM Bits

XMCM	Transmit Multichannel Selection Mode
00b	No transmit multichannel selection mode is on. All channels are enabled and unmasked. No channels can be disabled or masked.
01b	All channels are disabled unless they are selected in the appropriate transmit channel enable registers (XCERs, described in section 9.13.10 on page 9-207). If enabled, a channel in this mode is also unmasked. The XMCME bit of MCR2 determines whether 32 channels or 128 channels are selectable in XCERs.
10b	All channels are enabled, but they are masked unless they are selected in the appropriate transmit channel enable registers (XCERs). The XMCME bit of MCR2 determines whether 32 channels or 128 channels are selectable in XCERs.
11b	This mode is used for symmetric transmission and reception. All channels are disabled for transmission unless they are enabled for reception in the appropriate receive channel enable registers (RCERs, described in section 9.13.9 on page 9-202). Once enabled, they are masked unless they are also selected in the appropriate transmit channel enable registers (XCERs). The XMCME bit of MCR2 determines whether 32 channels or 128 channels are selectable in RCERs and XCERs.

As an example of how the McBSP behaves in a transmit multichannel selection mode, suppose that XMCM = 01b (all channels disabled unless individually enabled) and that you have enabled only channels 0, 15, and 39. Suppose also that the frame length is 40. The McBSP …

1) Shifts data to the DX pin in channel 0
2) Places the DX pin in the high impedance state in channels 1–14
3) Shifts data to the DX pin in channel 15
4) Places the DX pin in the high impedance state in channels 16–38
5) Shifts data to the DX pin in channel 39

9.5.7.1 Disabling/Enabling Versus Masking/Unmasking

For transmission, a channel can be:

- Enabled and unmasked (transmission can begin and can be completed)
- Enabled but masked (transmission can begin but cannot be completed)
- Disabled (transmission cannot occur)

The following definitions explain the channel control options:

Enabled channel — A channel that can begin transmission by passing data from the data transmit register(s) (DXR(s)) to the transmit shift registers (XSR(s)).

Masked channel — A channel that cannot complete transmission. The DX pin is held in the high impedance state; data cannot be shifted out on the DX pin.

In systems where symmetric transmit and receive provides software benefits, this feature allows transmit channels to be disabled on a shared serial bus. A similar feature is not needed for reception because multiple receptions cannot cause serial bus contention.

Disabled channel — A channel that is not enabled. A disabled channel is also masked.

Because no DXR-to-XSR copy occurs, the XRDY bit of SPCR2 is not set. Therefore, no DMA synchronization event (XEVT) is generated, and if the transmit interrupt mode depends on XRDY (XINTM = 00b in SPCR2), no interrupt is generated.

The XEMPTY_ bit of SPCR2 is not affected.

Unmasked channel — A channel that is not masked. Data in the XSR(s) is shifted out on the DX pin.

9.5.7.2 Activity on McBSP Pins for Different Values of XMCM

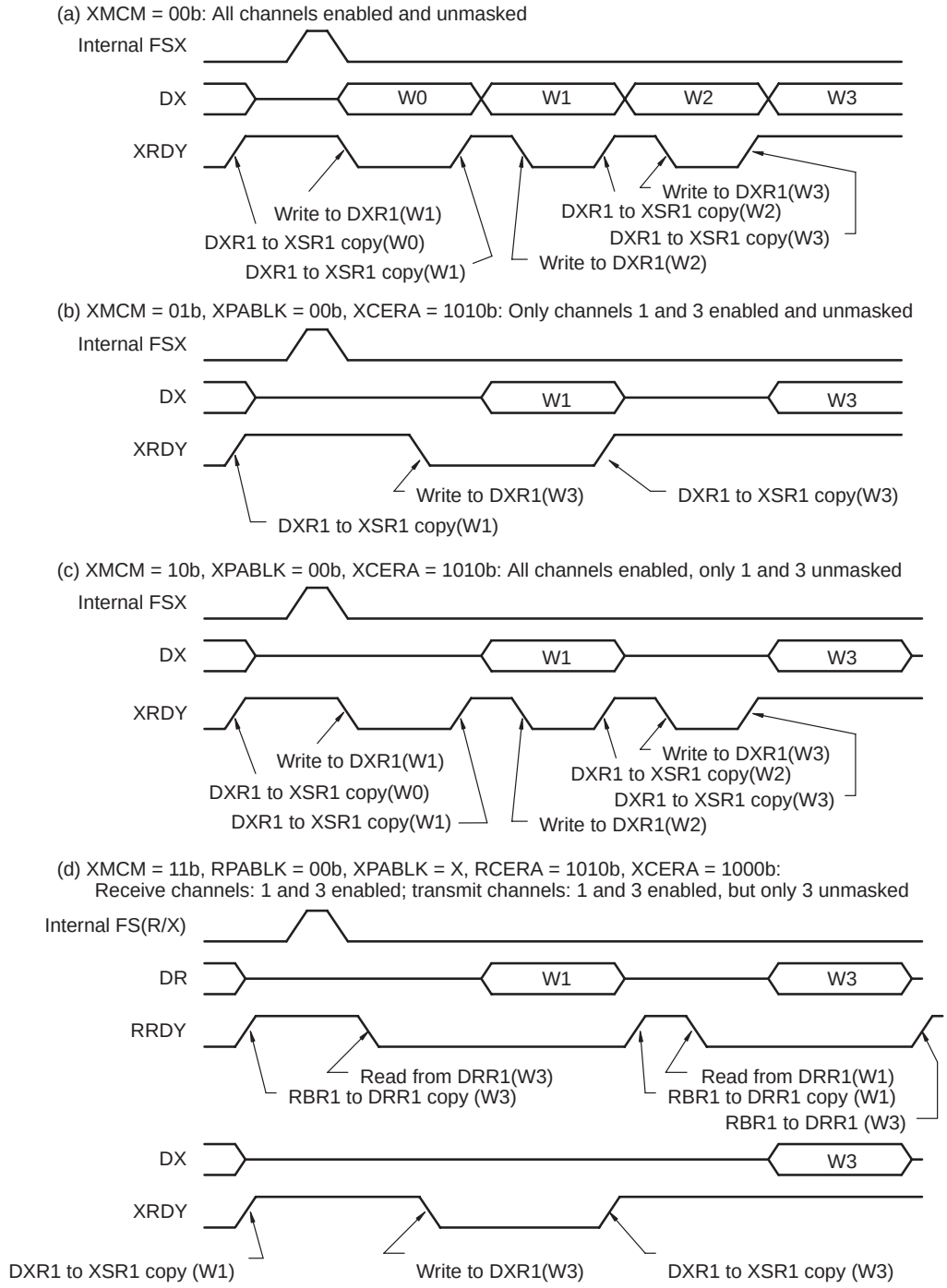
Figure 9–37 shows the activity on the McBSP pins for the various XMCM values. In all cases, the transmit frame is configured as follows:

- XPHASE = 0: Single-phase frame (required for multichannel selection modes)
- XFRLEN1 = 0000011b: 4 words per frame
- XWDLEN1 = 000b: 8 bits per word
- XMCME = 0: 2-partition mode (only partitions A and B used)

In the case where XMCM = 11b, transmission and reception are symmetric, which means the corresponding bits for the receiver (RPHASE, RFRLEN1, RWDLEN1, and RMCME) must have the same values as XPHASE, XFRLEN1, and XWDLEN1, respectively.

In the figure, the arrows showing where the various events occur are only sample indications. Wherever possible, there is a time window in which these events can occur.

Figure 9–37. Activity on McBSP Pins for the Possible Values of XMCM



9.5.8 Using Interrupts Between Block Transfers

When a multichannel selection mode is used, an interrupt request can be sent to the CPU at the end of every 16-channel block (at the boundary between partitions and at the end of the frame). In the receive multichannel selection mode, a receive interrupt (RINT) request is generated at the end of each block transfer if RINTM = 01b. In any of the transmit multichannel selection modes, a transmit interrupt (XINT) request is generated at the end of each block transfer if XINTM = 01b. When RINTM/XINTM = 01b, no interrupt is generated unless a multichannel selection mode is on.

These interrupt pulses are active high and last for 2 CPU clock cycles.

This type of interrupt is especially helpful if you are using the 2-partition mode (described in section 9.5.4) and you want to know when you can assign a different block of channels to partition A or B.

9.6 A-bis Mode

In the A-bis mode (ABIS = 1 in SPCR1), the McBSP can receive and transmit up to 1024 bits on a PCM link. The receive section can extract all 1024 bits from a 1024-bit PCM frame according to a given bit-enable pattern, and generate an interrupt to the CPU when 16 enabled bits have been compacted into a word in DRR1, or when a receive frame is complete. In addition, the transmit section can expand up to 1024 bits into a 1024-bit PCM frame at a specific position, according to a given bit-enable pattern, and generate an interrupt when 16 enabled bits have been transmitted or a transmit frame is complete.

The bit-enable patterns are specified with channel enable registers A and B (RCERA and RCERB for reception, XCERA and XCERB for transmission). These registers have a different function than in the multichannel selection modes (described in section 9.5). Instead of indicating which channels will be enabled, these registers indicate which bits in the data stream will be enabled. A 1 in a given position in the (R/X)CER(A/B) register enables a corresponding bit in the receive/transmit data stream.

The A-bis mode requires a word length of 16 bits (for reception: RWDLEN1 = 010b in RCR1, for transmission: XWDLEN1 = 010b in XCR1). Otherwise, operation in the A-bis mode is undetermined.

9.6.1 A-bis Mode Receive Operation

In the A-bis mode, bits that are not enabled in the RCERA and RCERB registers are ignored and are not compacted in the receiver. Bits that are enabled are received and compacted. When 16 enabled bits have been received, the received word is copied from RSR1 to DRR1 and the McBSP generates an interrupt to the CPU. RCERA and RCERB alternate specifying the receive masking pattern for each of the 16 receive clocks. Figure 9–38 shows an example bit sequence for the receiver (in the figure, – indicates that the bit on the DR pin is ignored and thus is not passed to DRR1).

Figure 9–38. A-bis Mode Receive Operation

RCERA	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1															
RCERB																	0	0	1	0	0	1	1	1	0	0	1	0	0	1	1	1
DR pin	1	0	1	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1
DRR1	–	0	–	1	–	1	–	1	–	0	–	1	0	–	–	1	–	–	1	–	–	1	1	0	–	–	0	–	–	0	1	1

75E3h

Note: – indicates that the bit on the DR pin is ignored and thus is not passed to DRR1.

9.6.2 A-bis Mode Transmit Operation

In the A-bis mode, only bits that are enabled in the XCERA and XCERB registers are transmitted out from the DX pin. Bits that are not enabled are not transmitted, and the DX pin is in the high-impedance state during that clock cycle. XCERA and XCERB alternate specifying the bit-enable pattern for each 16 clock cycles. When 16 enabled have been shifted out, the McBSP generates an interrupt to the CPU. Figure 9–39 shows an example bit sequence for the transmitter (in the figure, z indicates the high-impedance state).

Figure 9–39. A-bis Mode Transmit Operation

XCERA	0	1	1	1	1	1	0	0	0	1	1	0	0	1	1	1														
XCERB																	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0
DXR1	1	0	1	1	0	1	0	1	0	0	0	1	1	1	1	1	1	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0
DX pin	z	0	1	1	0	1	z	z	z	0	0	z	z	1	1	1	z	z	1	1	z	z	z	0	1	z	z	z	0	0	z	z

Note: z indicates the high-impedance state.

9.6.3 DMA Synchronization Events REVTA and XEVTA

In the A-bis mode (ABIS = 1), two DMA synchronization events, XEVTA (for A-bits mode transmission) and REVTA (for A-bis mode reception), can be used by the DMA controller to load patterns into the following channel enable registers: RCERA, RCERB, XCERA, and XCERB. This capability is used for bit sequences that are longer than the 32 bits covered by the two receive/transmit channel enable registers. An REVTA/XEVTA event is generated every 16 CLKR/CLKX cycles.

As an example, the following gives a description of A-bis mode operation on a 256-bit PCM link:

- ABIS = 1: A-bis mode enabled
- The initial pattern of bits that must be enabled is loaded into the channel enable registers.
- RPHASE = XPHASE = 0: Single-phase frame
- RFRLEN1 = XFRLEN1 = 1111b: 16 words in the frame
- RWDLEN1 = XWDLEN1 = 010b: 16 bits per word (required for the A-bis mode)

Two DMA channels (one for transmit, one for receive) are used to update the bit pattern selections in the channel enable registers as the operation proceeds. One 16-word block in memory contains the bit pattern selections for the receiver. Sixteen words of 16 bits each contain the entire receive selection pattern for the 256-bit PCM link.

On each REVTA event, the DMA controller copies new receive selection pattern data from memory to RCERA or RCERB and automatically toggles its destination pointer from RCERA to RCERB, or vice versa, as necessary.

The DMA channel is initially set to RCERA as a destination. After the first access to RCERA, the destination automatically toggles to RCERB. After the next RCERB access, the destination automatically toggles back to RCERA. Since the toggling between RCERA and RCERB is handled automatically, you do not need to configure the DMA controller to modify the destination address by other means. As the A-bis-mode transfer proceeds, the receiver alternates using RCERA and RCERB to specify the enable pattern for each group of 16 serial port clock cycles.

Transmitter operation is similar. Sixteen words of 16 bits each contain the entire transmit selection pattern for the 256-bit PCM link. On each XEVTA event, the DMA controller copies the new transmit selection pattern data from memory to XCERA or XCERB and automatically toggles its destination pointer from XCERA to XCERB, or vice versa, as necessary. As the A-bis-mode transfer proceeds, the transmitter alternates using XCERA and XCERB to specify the enable pattern for each group of 16 serial port clock cycles.

9.7 SPI Operation Using the Clock Stop Mode

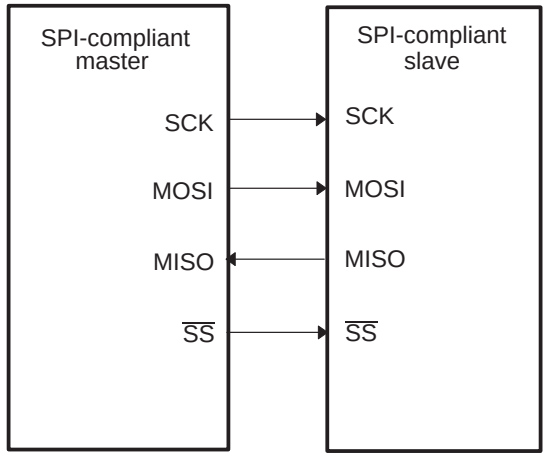
9.7.1 SPI Protocol

The SPI protocol is a master-slave configuration with one master device and one or more slave devices. The interface consists of the following four signals:

- ❏ Serial data input (also referred to as Master In – Slave Out, or MISO)
- ❏ Serial data output (also referred to as Master Out – Slave In, or MOSI)
- ❏ Shift-clock (also referred to as SCK)
- ❏ Slave-enable signal (also referred to as SS)

A typical SPI interface with a single slave device is shown in Figure 9–40.

Figure 9–40. Typical SPI Interface



The master device controls the flow of communication by providing shift-clock and slave-enable signals. The slave-enable signal is an optional active-low signal that enables the serial data input and output of the slave device (device not sending out the clock).

In the absence of a dedicated slave-enable signal, communication between the master and slave is determined by the presence or absence of an active shift-clock. In such a configuration, the slave device must remain enabled at all times, and multiple slaves cannot be used.

9.7.2 Clock Stop Mode

The clock stop mode of the McBSP provides compatibility with the SPI protocol. When the McBSP is configured in clock stop mode, the transmitter and receiver are internally synchronized, so that the McBSP functions as an SPI master or slave device. The transmit clock signal (CLKX) corresponds to the serial clock signal (SCK) of the SPI protocol, while the transmit frame-synchronization signal (FSX) is used as the slave-enable signal (SS_).

The receive clock signal (CLKR) and receive frame-synchronization signal (FSR) are not used in the clock stop mode because these signals are internally connected to their transmit counterparts, CLKX and FSX.

9.7.3 Bits Used to Enable and Configure the Clock Stop Mode

The bits required to configure the McBSP as an SPI device are introduced in Table 9–9. Table 9–10 shows how the various combinations of the CLKSTP bit and the polarity bits CLKXP and CLKRP create four possible clock stop mode configurations. The timing diagrams in section 9.7.4 show the effects of CLKSTP, CLKXP, and CLKRP.

Table 9–9. Bits Used to Enable and Configure the Clock Stop Mode

Bit Field	Description
CLKSTP bits of SPCR1	Use these bits to enable the clock stop mode and to select one of two timing variations. (See also Table 9–10.)
CLKXP bit of PCR	This bit determines the polarity of the CLKX signal. (See also Table 9–10.)
CLKRP bit of PCR	This bit determines the polarity of the CLKR signal. (See also Table 9–10.)
CLKXM bit of PCR	This bit determines whether CLKX is an input signal (McBSP as slave) or an output signal (McBSP as master).
XPHASE bit of XCR2	You must use a single-phase transmit frame (XPHASE = 0).
RPHASE bit of RCR2	You must use a single-phase receive frame (RPHASE = 0).
XFRLEN1 bits of XCR1	You must use a transmit frame length of 1 serial word (XFRLEN1 = 0).
RFRLEN1 bits of RCR1	You must use a receive frame length of 1 serial word (RFRLEN1 = 0).
XWDLEN1 bits of XCR1	The XWDLEN1 bits determine the transmit packet length. XWDLEN1 must be equal to RWDLEN1 because in the clock stop mode, the McBSP transmit and receive circuits are synchronized to a single clock.
RWDLEN1 bits of RCR1	The RWDLEN1 bits determine the receive packet length. RWDLEN1 must be equal to XWDLEN1 because in the clock stop mode, the McBSP transmit and receive circuits are synchronized to a single clock.

Table 9–10. Effects of CLKSTP, CLKXP, and CLKRP on the Clock Scheme

Bit Settings	Clock Scheme
CLKSTP = 00b or 01b CLKXP = 0 or 1 CLKRP = 0 or 1	Clock stop mode disabled. Clock enabled for non-SPI mode.
CLKSTP = 10b CLKXP = 0 CLKRP = 0	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of CLKR.
CLKSTP = 11b CLKXP = 0 CLKRP = 1	Low inactive state with delay: The McBSP transmits data one-half cycle ahead of the rising edge of CLKX and receives data on the rising edge of CLKR.
CLKSTP = 10b CLKXP = 1 CLKRP = 0	High inactive state without delay: The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of CLKR.
CLKSTP = 11b CLKXP = 1 CLKRP = 1	High inactive state with delay: The McBSP transmits data one-half cycle ahead of the falling edge of CLKX and receives data on the falling edge of CLKR.

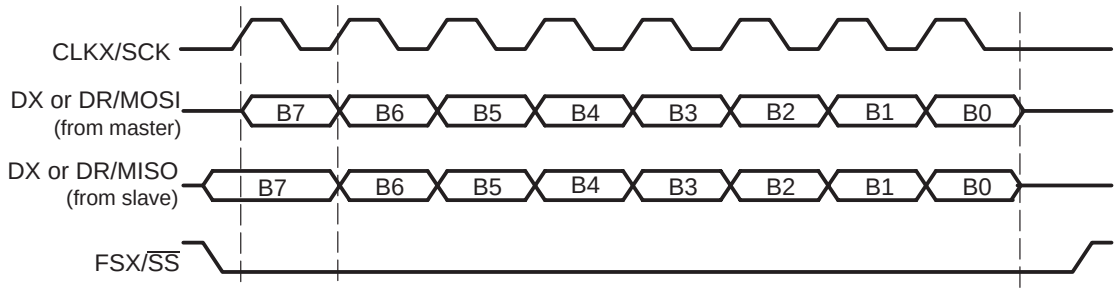
9.7.4 Clock Stop Mode Timing Diagrams

The timing diagrams for the four possible clock stop mode configurations are shown here. Notice that the frame-synchronization signal used in clock stop mode is active throughout the entire transmission as a slave-enable signal. Although the timing diagrams show 8-bit transfers, the packet length can be set to 8, 12, 16, 20, 24, or 32 bits per packet. The receive packet length is selected with the RWDLEN1 bits of RCR1, and the transmit packet length is selected with the XWDLEN1 bits of XCR1. For clock stop mode, the values of RWDLEN1 and XWDLEN1 must be the same because the McBSP transmit and receive circuits are synchronized to a single clock.

Note:

Even if multiple words are consecutively transferred, the CLKX signal is always stopped and the FSX signal returns to the inactive state after a packet transfer. When consecutive packet transfers are performed, this leads to a minimum idle time of two bit-periods between each packet transfer.

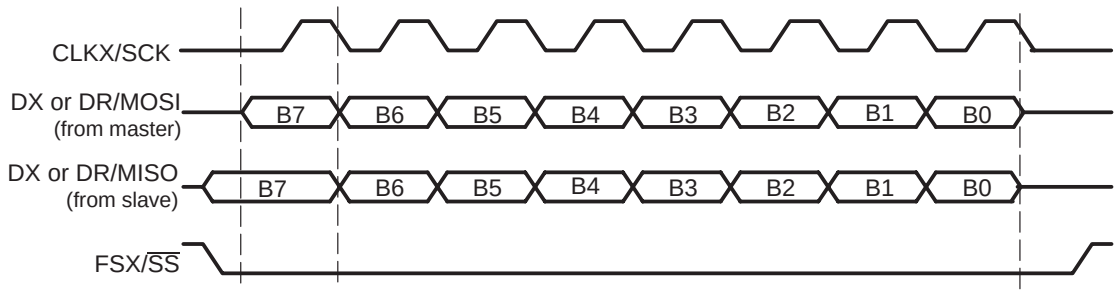
Figure 9–41. SPI Transfer With CLKSTP = 10b (no clock delay), CLKXP = 0, CLKRP = 0



Notes: 1) If the McBSP is the SPI master (CLKXM = 1), MOSI = DX. If the McBSP is the SPI slave (CLKXM = 0), MOSI = DR.

2) If the McBSP is the SPI master (CLKXM = 1), MISO = DR. If the McBSP is the SPI slave (CLKXM = 0), MISO = DX.

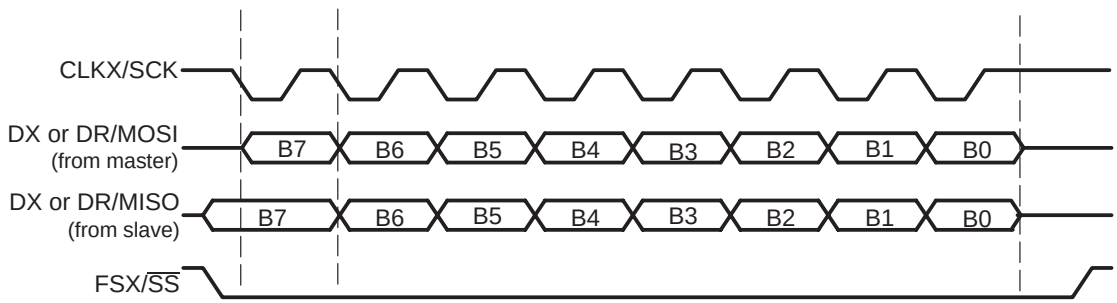
Figure 9–42. SPI Transfer With CLKSTP = 11b (clock delay), CLKXP = 0, CLKRP = 1



Notes: 1) If the McBSP is the SPI master (CLKXM = 1), MOSI = DX. If the McBSP is the SPI slave (CLKXM = 0), MOSI = DR.

2) If the McBSP is the SPI master (CLKXM = 1), MISO = DR. If the McBSP is the SPI slave (CLKXM = 0), MISO = DX.

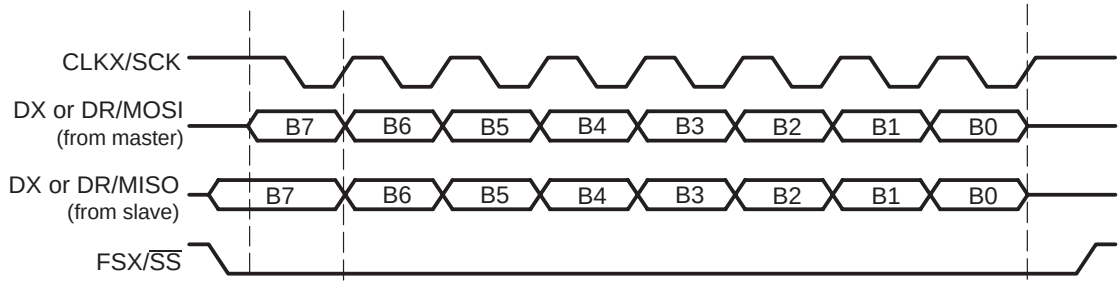
Figure 9–43. SPI Transfer With CLKSTP = 10b (no clock delay), CLKXP = 1, CLKRP = 0



Notes: 1) If the McBSP is the SPI master (CLKXM = 1), MOSI = DX. If the McBSP is the SPI slave (CLKXM = 0), MOSI = DR.

2) If the McBSP is the SPI master (CLKXM = 1), MISO = DR. If the McBSP is the SPI slave (CLKXM = 0), MISO = DX.

Figure 9–44. SPI Transfer With CLKSTP = 11b (clock delay), CLKXP = 1, CLKRP = 1



Notes: 1) If the McBSP is the SPI master (CLKXM = 1), MOSI=DX. If the McBSP is the SPI slave (CLKXM = 0), MOSI = DR.

2) If the McBSP is the SPI master (CLKXM = 1), MISO=DR. If the McBSP is the SPI slave (CLKXM = 0), MISO = DX.

9.7.5 Procedure for Configuring a McBSP for SPI Operation

To configure the McBSP for SPI master or slave operation:

1) Place the transmitter and receiver in reset.

Clear the transmitter reset bit (XRST_ = 0) in SPCR2, to reset the transmitter. Clear the receiver reset bit (RRST_ = 0) in SPCR1, to reset the receiver.

2) Place the sample rate generator in reset.

Clear the sample rate generator reset bit (GRST_ = 0) in SPCR2, to reset the sample rate generator.

3) Program registers that affect SPI operation.

Program the appropriate McBSP registers to configure the McBSP for proper operation as an SPI master or an SPI slave. For a list of important bits settings, see one of the following topics:

- *McBSP as the SPI Master* (page 9-68)
- *McBSP as an SPI Slave* (page 9-70)

4) Enable the sample rate generator.

To release the sample rate generator from reset, set the sample rate generator reset bit (GRST_ = 1) in SPCR2.

Make sure that during the write to SPCR2, you only modify GRST_. Otherwise, you will modify the McBSP configuration you selected in the previous step.

5) Enable the transmitter and receiver.

After the sample rate generator is released from reset, wait two sample rate generator clock periods for the McBSP logic to stabilize.

If the CPU services the McBSP transmit and receive buffers, then you can immediately enable the transmitter (XRST_ = 1 in SPCR2) and enable the receiver (RRST_ = 1 in SPCR1).

If the DMA controller services the McBSP transmit and receive buffers, then you must first configure the DMA controller (this includes enabling the channels that service the McBSP buffers). When the DMA controller is ready, make XRST_ = 1 and RRST_ = 1.

Note: In either case, make sure you only change XRST_ and RRST_ when you write to SPCR2 and SPCR1. Otherwise, you will modify the bit settings you selected earlier in this procedure.

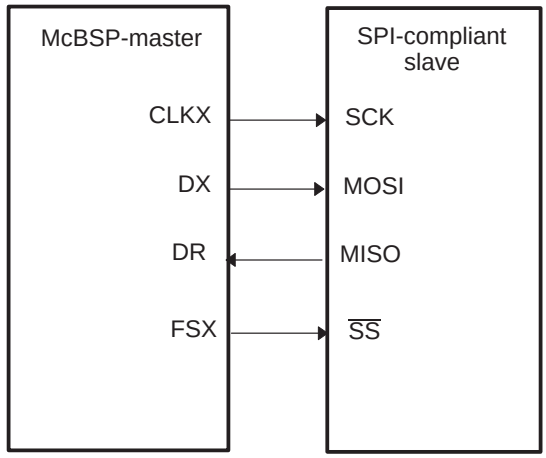
After the transmitter and receiver are released from reset, wait two sample rate generator clock periods for the McBSP logic to stabilize.

6) If necessary, enable the frame-sync logic of the sample rate generator.

After the required data acquisition setup is done (DXR[1/2] is loaded with data), set FRST_ = 1 if an internally generated frame-sync pulse is required (that is, if the McBSP is the SPI master).

9.7.6 McBSP as the SPI Master

An SPI interface with the McBSP used as the master is shown in the following figure. When the McBSP is configured as a master, the transmit output signal (DX) is used as the MOSI signal of the SPI protocol, and the receive input signal (DR) is used as the MISO signal.



The register bit values required to configure the McBSP as a master are listed in the following table. After the table are more details about the configuration requirements.

Table 9–11. Bit Values Required to Configure the McBSP as an SPI Master

Required Bit Setting	Description
CLKSTP = 10b or 11b	The clock stop mode (without or with a clock delay) is selected.
CLKXP = 0 or 1	The polarity of CLKX as seen on the CLKX pin is positive (CLKXP = 0) or negative (CLKXP = 1).
CLKRP = 0 or 1	The polarity of CLKR as seen on the CLKR pin is positive (CLKRP = 0) or negative (CLKRP = 1).
CLKXM = 1	The CLKX pin is an output pin driven by the internal sample rate generator. Because CLKSTP is equal to 10b or 11b, CLKR is driven internally by CLKX.
SCLKME = 0 CLKSM = 1	The clock generated by the sample rate generator (CLKG) is derived from the CPU clock.
CLKGDV is a value from 0 to 255	CLKGDV defines the divide down value for CLKG.
FSXM = 1	The FSX pin is an output pin driven according to the FSGM bit.
FSGM = 0	The transmitter drives a frame-sync pulse on the FSX pin every time data is transferred from DXR1 to XSR1.
FSXP = 1	The FSX pin is active low.
XDATDLY = 01b RDATDLY = 01b	This setting provides the correct setup time on the FSX signal.

When the McBSP functions as the SPI master, it controls the transmission of data by producing the serial clock signal. The clock signal on the CLKX pin is enabled only during packet transfers. When packets are not being transferred, the CLKX pin remains high or low depending on the polarity used.

For SPI master operation, the CLKX pin must be configured as an output. The sample rate generator is then used to derive the CLKX signal from the CPU clock. The clock stop mode internally connects the CLKX pin to the CLKR signal so that no external signal connection is required on the CLKR pin, and both the transmit and receive circuits are clocked by the master clock (CLKX).

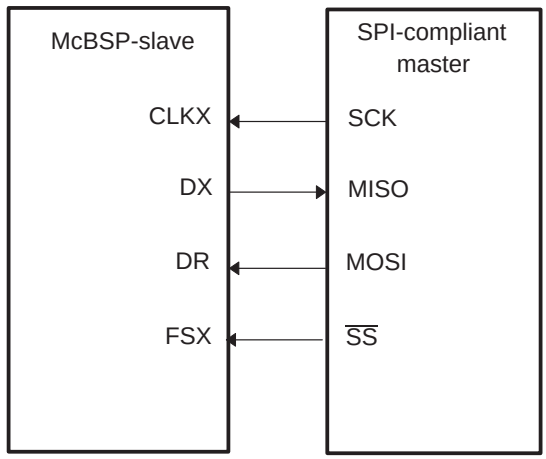
The data delay parameters of the McBSP (XDATDLY and RDATDLY) must be set to 1 for proper SPI master operation. A data delay value of 0 or 2 is undefined in the clock stop mode.

The McBSP can also provide a slave-enable signal (SS_) on the FSX pin. If a slave-enable signal is required, the FSX pin must be configured as an output, and the transmitter must be configured so that a frame-sync pulse is generated automatically each time a packet is transmitted (FSGM = 0). The polarity of the FSX pin is programmable high or low; however, in most cases the pin should be configured active-low.

When the McBSP is configured as described for SPI-master operation, the bit fields for frame-sync pulse width (FWID) and frame-sync period (FPER) are overridden, and custom frame-sync waveforms are not allowed. To see the resulting waveform produced on the FSX pin, see the timing diagrams in section 9.7.4. The signal becomes active before the first bit of a packet transfer, and remains active until the last bit of the packet is transferred. After the packet transfer is complete, the FSX signal returns to the inactive state.

9.7.7 McBSP as an SPI Slave

An SPI interface with the McBSP used as a slave is shown in the following figure. When the McBSP is configured as a slave, DX is used as the MISO signal, and DR is used as the MOSI signal.



The register bit values required to configure the McBSP as a slave are listed in the following table. After the table are more details about the configuration requirements.

Table 9–12. Bit Values Required to Configure the McBSP as an SPI Slave

Required Bit Setting	Description
CLKSTP = 10b or 11b	The clock stop mode (without or with a clock delay) is selected.
CLKXP = 0 or 1	The polarity of CLKX as seen on the CLKX pin is positive (CLKXP = 0) or negative (CLKXP = 1).
CLKRP = 0 or 1	The polarity of CLKR as seen on the CLKR pin is positive (CLKRP = 0) or negative (CLKRP = 1).
CLKXM = 0	The CLKX pin is an input pin, so that it can be driven by the SPI master. Because CLKSTP = 10b or 11b, CLKR is driven internally by CLKX.
SCLKME = 0 CLKSM = 1	The clock generated by the sample rate generator (CLKG) is derived from the CPU clock. (The sample rate generator is used to synchronize the McBSP logic with the externally-generated master clock.)
CLKGDV = 1	The sample rate generator divides the CPU clock by 2 before generating CLKG.
FSXM = 0	The FSX pin is an input pin, so that it can be driven by the SPI master.
FSXP = 1	The FSX pin is active low.
XDATDLY = 00b RDATDLY = 00b	These bits must be 0s for SPI slave operation.

When the McBSP is used as an SPI slave, the master clock and slave-enable signals are generated externally by a master device. Accordingly, the CLKX and FSX pins must be configured as inputs. The CLKX pin is internally connected to the CLKR signal, so that both the transmit and receive circuits of the McBSP are clocked by the external master clock. The FSX pin is also internally connected to the FSR signal, and no external signal connections are required on the CLKR and FSR pins.

Although the CLKX signal is generated externally by the master and is asynchronous to the McBSP, the sample rate generator of the McBSP must be enabled for proper SPI slave operation. The sample rate generator should be programmed to its maximum rate of half the CPU clock rate. The internal sample rate clock is then used to synchronize the McBSP logic to the external master clock and slave-enable signals.

The McBSP requires an active edge of the slave-enable signal on the FSX input for each transfer. This means that the master device must assert the

slave-enable signal at the beginning of each transfer, and deassert the signal after the completion of each packet transfer; the slave-enable signal cannot remain active between transfers.

The data delay parameters of the McBSP must be set to 0 for proper SPI slave operation. A value of 1 or 2 is undefined in the clock stop mode.

9.8 Receiver Configuration

To configure the McBSP receiver, perform the following procedure:

1) Place the McBSP/receiver in reset (see section 9.8.2).
2) Program the McBSP registers for the desired receiver operation (see section 9.8.1).
3) Take the receiver out of reset (see section 9.8.2).

9.8.1 Programming the McBSP Registers for the Desired Receiver Operation

The following is a list of important tasks to be performed when you are configuring the McBSP receiver. Each task corresponds to one or more McBSP register bit fields. Note that in the list, SRG is an abbreviation for sample rate generator.

It may be helpful to first photocopy the McBSP Register Worksheet (page 9-211) and to fill in the photocopy of the worksheet as you read the tasks.

- **Global behavior:**
  - Set the receiver pins to operate as McBSP pins
  - Enable/disable the digital loopback mode
  - Enable/disable the clock stop mode
  - Enable/disable the receive multichannel selection mode
  - Enable/disable the A-bis mode
- **Data behavior:**
  - Choose 1 or 2 phases for the receive frame
  - Set the receive word length(s)
  - Set the receive frame length
  - Enable/disable the receive frame-sync ignore function
  - Set the receive companding mode
  - Set the receive data delay
  - Set the receive sign-extension and justification mode
  - Set the receive interrupt mode
- **Frame-sync behavior:**
  - Set the receive frame-sync mode
  - Set the receive frame-sync polarity
  - Set the SRG frame-sync period and pulse width

- **Clock behavior:**
  - Set the receive clock mode
  - Set the receive clock polarity
  - Set the SRG clock divide-down value
  - Set the SRG clock synchronization mode
  - Set the SRG clock mode (choose an input clock)
  - Set the SRG input clock polarity

9.8.2 Resetting and Enabling the Receiver

The first step of the receiver configuration procedure is to reset the receiver, and the last step is to enable the receiver (to take it out of reset). Figure 9–45 and Table 9–13 describe the bits used for both of these steps.

Figure 9–45. Register Bits Used to Reset or Enable the McBSP Receiver

SPCR1

15–1	0
	RRST_
	R/W – 0

SPCR2

15–8	7	6	5–0
	FRST_	GRST_	
	R/W – 0	R/W – 0	

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–13. Register Bits Used to Reset or Enable the McBSP Receiver

Register	Bit	Name	Function	
SPCR1	0	RRST_	Receiver Reset	
			RRST_ = 0	The serial port receiver is disabled and in the reset state.
			RRST_ = 1	The serial port receiver is enabled.
SPCR2	6	GRST_	Sample Rate Generator Reset	
			GRST_ = 0	Sample rate generator is reset.
				If GRST_ = 0 due to a DSP reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven low (inactive). If GRST_ = 0 due to program code, CLKG and FSG are both driven low (inactive).
			GRST_ = 1	Sample rate generator is enabled. CLKG is driven according to the configuration programmed in the sample rate generator registers (SRGR[1,2]). If FRST_ = 1, the generator also generates the frame-sync signal FSG as programmed in the sample rate generator registers.
SPCR2	7	FRST_	Frame-Sync Logic Reset	
			FRST_ = 0	Frame-synchronization logic is reset. The sample rate generator does not generate frame-sync signal FSG, even if GRST_ = 1.
			FRST_ = 1	If GRST_ = 1, frame-sync signal FSG is generated after (FPER + 1) number of CLKG clock cycles; all frame counters are loaded with their programmed values.

9.8.2.1 Reset Considerations

The serial port can be reset in the following two ways:

1) A DSP reset (RESET_ signal driven low) places the receiver, transmitter, and sample rate generator in reset. When the device reset is removed (RESET_ signal released), GRST_ = FRST_ = RRST_ = XRST_ = 0, keeping the entire serial port in the reset state.

2) The serial port transmitter and receiver can be reset directly using the RRST_ and XRST_ bits in the serial port control registers. The sample rate generator can be reset directly using the GRST_ bit in SPCR2.

Table 9–14 shows the state of McBSP pins when the serial port is reset due to a DSP reset and a direct receiver/transmitter reset.

Table 9–14. Reset State of Each McBSP Pin

Pin	Possible State(s)	State Forced By DSP Reset	State Forced By Receiver/Transmitter Reset
			Receiver Reset (RRST_ = 0 and GRST_ = 1)
DR	I	Input	Input
CLKR	I/O/Z	Input	Known state if Input; CLKR running if output
FSR	I/O/Z	Input	Known state if Input; FSRP inactive state if output
CLKS	I/O/Z	Input	Input
			Transmitter Reset (XRST_ = 0 and GRST_ = 1)
DX	O/Z	High impedance	High impedance
CLKX	I/O/Z	Input	Known state if Input; CLKX running if output
FSX	I/O/Z	Input	Known state if Input; FSXP inactive state if output
CLKS	I	Input	Input

Note: In Possible State(s) column, I = Input, O = Output, Z = High impedance

For more details about McBSP reset conditions and effects, see *Resetting and Initializing a McBSP* on page 9-147.

9.8.3 Set the Receiver Pins to Operate as McBSP Pins

The RIOEN bit, shown in Figure 9–46 and described in Table 9–15, determines whether the receiver pins are McBSP pins or general-purpose I/O pins.

Figure 9–46. Register Bit Used to Set Receiver Pins to Operate as McBSP Pins

PCR

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–15. Register Bit Used to Set Receiver Pins to Operate as McBSP Pins

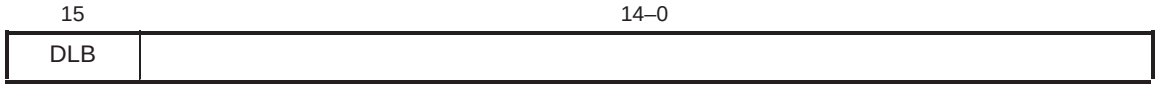
Register	Bit	Name	Function	
PCR	12	RIOEN	Receive I/O enable	
			This bit is only applicable when the receiver is in the reset state (RRST_ = 0 in SPCR1).	
			RIOEN = 0	The DR, FSR, CLKR, and CLKS pins are configured as serial port pins and do not function as general-purpose I/O pins.
			RIOEN = 1	The DR pin is a general-purpose input pin. The FSR and CLKR pins are general purpose I/O pins. These serial port pins do not perform serial port operation. The CLKS pin is a general-purpose input pin if RIOEN = XIOEN = 1 and RRST_ = XRST_ = 0. For more information on using these pins as general-purpose I/O pins, see page 9-144.

9.8.4 Enable/Disable the Digital Loopback Mode

The DLB bit determines whether the digital loopback mode is on. DLB is shown in Figure 9–47 and described in Table 9–16.

Figure 9–47. Register Bit Used to Enable/Disable the Digital Loopback Mode

SPCR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–16. Register Bit Used to Enable/Disable the Digital Loopback Mode

Register	Bit	Name	Function	
SPCR1	15	DLB	Digital Loopback Mode	
			DLB = 0	Digital loopback mode is disabled.
			DLB = 1	Digital loopback mode is enabled.

9.8.4.1 About the Digital Loopback Mode

In the digital loopback mode, the receive signals are connected internally through multiplexers to the corresponding transmit signals, as shown in Table 9–17. This mode allows testing of serial port code with a single DSP device; the McBSP receives the data it transmits.

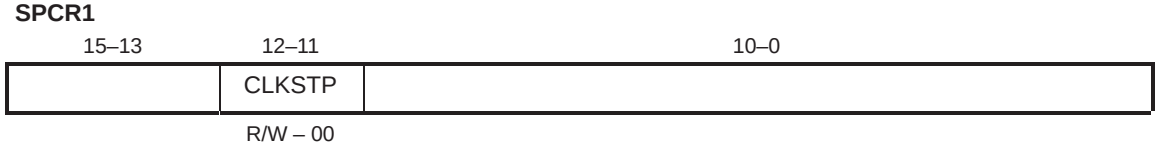
Table 9–17. Receive Signals Connected to Transmit Signals in Digital Loopback Mode

This Receive Signal …	**Is Fed Internally By This Transmit Signal …**
DR (receive data)	DX (transmit data)
FSR (receive frame synchronization)	FSX (transmit frame synchronization)
CLKR (receive clock)	CLKX (transmit clock)

9.8.5 Enable/Disable the Clock Stop Mode

The CLKSTP bits determine whether the clock stop mode is on. CLKSTP is shown in Figure 9–48 and described in Table 9–18.

Figure 9–48. Register Bits Used to Enable/Disable the Clock Stop Mode



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–18. Register Bits Used to Enable/Disable the Clock Stop Mode

Register	**Bit**	**Name**	**Function**	
SPCR1	12–11	CLKSTP	Clock Stop Mode	
			CLKSTP = 0Xb	Clock stop mode disabled; normal clocking for non-SPI mode.
			CLKSTP = 10b	Clock stop mode enabled, without clock delay
			CLKSTP = 11b	Clock stop mode enabled, with clock delay

9.8.5.1 About the Clock Stop Mode

The clock stop mode supports the SPI master-slave protocol. If you will not be using the SPI protocol, you can clear CLKSTP to disable the clock stop mode.

In the clock stop mode, the clock stops at the end of each data transfer. At the beginning of each data transfer, the clock starts immediately (CLKSTP = 10b) or after a half-cycle delay (CLKSTP = 11b). The CLKXP bit determines whether the starting edge of the clock on the CLKX pin is rising or falling. The CLKRP bit determines whether receive data is sampled on the rising or falling edge of the clock shown on the CLKR pin.

Table 9–19 summarizes the impact of CLKSTP, CLKXP, and CLKRP on serial port operation. Note that in the clock stop mode, the receive clock is tied internally to the transmit clock, and the receive frame-sync signal is tied internally to the transmit frame-sync signal.

Table 9–19. Effects of CLKSTP, CLKXP, and CLKRP on the Clock Scheme

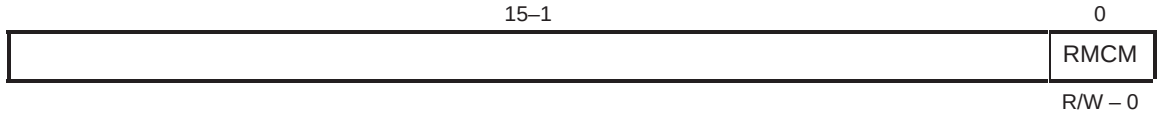
Bit Settings	Clock Scheme
CLKSTP = 00b or 01b CLKXP = 0 or 1 CLKRP = 0 or 1	Clock stop mode disabled. Clock enabled for non-SPI mode.
CLKSTP = 10b CLKXP = 0 CLKRP = 0	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of CLKR.
CLKSTP = 11b CLKXP = 0 CLKRP = 1	Low inactive state with delay: The McBSP transmits data one-half cycle ahead of the rising edge of CLKX and receives data on the rising edge of CLKR.
CLKSTP = 10b CLKXP = 1 CLKRP = 0	High inactive state without delay: The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of CLKR.
CLKSTP = 11b CLKXP = 1 CLKRP = 1	High inactive state with delay: The McBSP transmits data one-half cycle ahead of the falling edge of CLKX and receives data on the falling edge of CLKR.

9.8.6 Enable/Disable the Receive Multichannel Selection Mode

The RMCM bit determines whether the receive multichannel selection mode is on. RMCM is shown in Figure 9–49 and described in Table 9–20.

Figure 9–49. Register Bit Used to Enable/Disable the Receive Multichannel Selection Mode

MCR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–20. Register Bit Used to Enable/Disable the Receive Multichannel Selection Mode

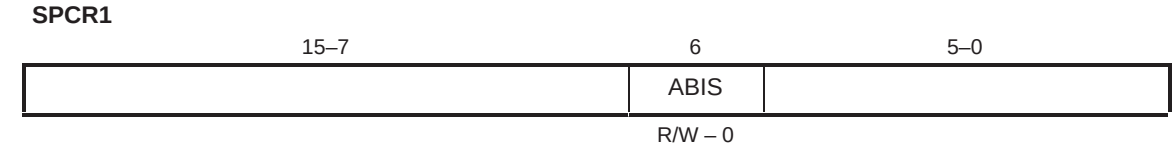
Register	Bit	Name	Function	
MCR1	0	RMCM	Receive Multichannel Selection Mode	
			RMCM = 0	The mode is disabled.
				All 128 channels are enabled.
			RMCM = 1	The mode is enabled.
				Channels can be individually enabled or disabled.
				The only channels enabled are those selected in the appropriate receive channel enable registers (RCERs). The way channels are assigned to the RCERs depends on the number of receive channel partitions (2 or 8), as defined by the RMCME bit.

For more details, see *Receive Multichannel Selection Mode* on page 9-53.

9.8.7 Enable/Disable the A-bis Mode

The ABIS bit determines whether the A-bis mode is on. ABIS is shown in Figure 9–50 and described in Table 9–21.

Figure 9–50. Register Bit Used to Enable/Disable the A-bis Mode



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–21. Register Bit Used to Enable/Disable the A-bis Mode

Register	Bit	Name	Function	
SPCR1	6	ABIS	A-bis Mode	
			ABIS = 0	The mode is disabled.
			ABIS = 1	The mode is enabled.
			Individual bits can be enabled or disabled during reception and transmission. For transmission, the bits are controlled by transmit channel enable registers A and B (XCERA and XCERB). For reception, the bits are controlled by receive channel enable registers A and B (RCERA and RCERB).	

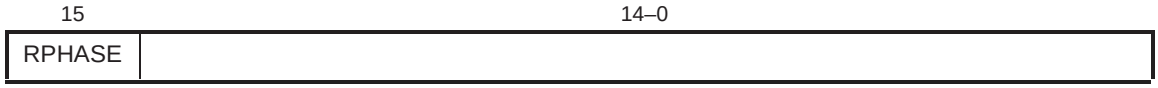
For more details, see *A-bis Mode* on page 9-59.

9.8.8 Choose 1 or 2 Phases for the Receive Frame

The RPHASE bit (see Figure 9–51 and Table 9–22) determines whether the receive data frame has one or two phases.

Figure 9–51. Register Bit Used to Choose 1 or 2 Phases for the Receive Frame

RCR2



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

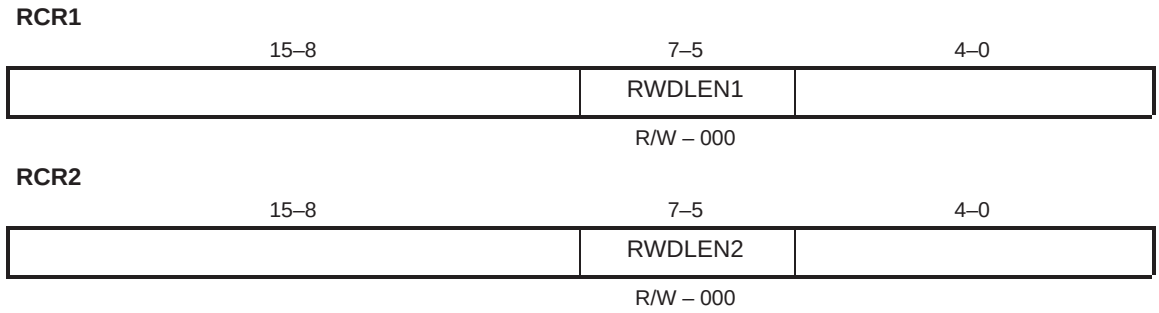
Table 9–22. Register Bit Used to Choose 1 or 2 Phases for the Receive Frame

Register	Bit	Name	Function	
RCR2	15	RPHASE	Receive phase number	
			Specifies whether the receive frame has 1 or 2 phases.	
			RPHASE = 0	Single-phase frame
			RPHASE = 1	Dual-phase frame

9.8.9 Set the Receive Word Length(s)

The RWDLEN1 and RWDLEN2 bit fields (see Figure 9–52 and Table 9–23) determine how many bits are in each serial word in phase 1 and in phase 2, respectively, of the receive data frame.

Figure 9–52. Register Bits Used to Set the Receive Word Length(s)



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–23. Register Bits Used to Set the Receive Word Length(s)

Register	Bit	Name	Function
RCR1	7–5	RWDLEN1	Receive word length 1 Specifies the length of every serial word in phase 1 of the receive frame. RWDLEN1 = 000 8 bits RWDLEN1 = 001 12 bits RWDLEN1 = 010 16 bits RWDLEN1 = 011 20 bits RWDLEN1 = 100 24 bits RWDLEN1 = 101 32 bits RWDLEN1 = 11X Reserved
RCR2	7–5	RWDLEN2	Receive word length 2 If a dual-phase frame is selected, RWDLEN2 specifies the length of every serial word in phase 2 of the frame. RWDLEN2 = 000 8 bits RWDLEN2 = 001 12 bits RWDLEN2 = 010 16 bits RWDLEN2 = 011 20 bits RWDLEN2 = 100 24 bits RWDLEN2 = 101 32 bits RWDLEN2 = 11X Reserved

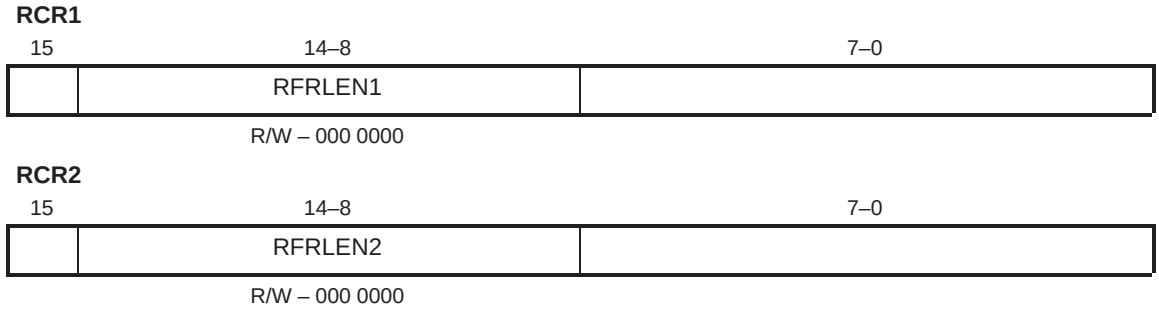
9.8.9.1 About the Word Length Bits

Each frame can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, RWDLEN1 selects the length for every serial word received in the frame. If a dual-phase frame is selected, RWDLEN1 determines the length of the serial words in phase 1 of the frame, and RWDLEN2 determines the word length in phase 2 of the frame.

9.8.10 Set the Receive Frame Length

The RFRLEN1 and RFRLEN2 bit fields (see Figure 9–53 and Table 9–24) determine how many serial words are in phase 1 and in phase 2, respectively, of the receive data frame.

Figure 9–53. Register Bits Used to Set the Receive Frame Length



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–24. Register Bits Used to Set the Receive Frame Length

Register	Bit	Name	Function
RCR1	14–8	RFRLEN1	Receive frame length 1
			(RFRLEN1 + 1) is the number of serial words in phase 1 of the receive frame.
			RFRLEN1 = 000 0000 1 word in phase 1
			RFRLEN1 = 000 0001 2 words in phase 1
			| |
			| |
			RFRLEN1 = 111 1111 128 words in phase 1
RCR2	14–8	RFRLEN2	Receive frame length 2
			If a dual-phase frame is selected, (RFRLEN2 + 1) is the number of serial words in phase 2 of the receive frame.
			RFRLEN2 = 000 0000 1 word in phase 2
			RFRLEN2 = 000 0001 2 words in phase 2
			| |
			| |
			RFRLEN2 = 111 1111 128 words in phase 2

9.8.10.1 About the Selected Frame Length

The receive frame length is the number of serial words in the receive frame. Each frame can have one or two phases, depending on value that you load into the RPHASE bit.

If a single-phase frame is selected (RPHASE = 0), the frame length is equal to the length of phase 1. If a dual-phase frame is selected (RPHASE = 1), the frame length is the length of phase 1 plus the length of phase 2:

The 7-bit RFRLEN fields allow up to 128 words per phase. See Table 9–25 for a summary of how to calculate the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization pulse.

Note: Program the RFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For the example, if you want a phase length of 128 words in phase 1, load 127 into RFRLEN1.

Table 9–25. How to Calculate the Length of the Receive Frame

RPHASE	RFRLEN1	RFRLEN2	Frame Length
0	0 ≤ RFRLEN1 ≤ 127	Don't care	(RFRLEN1 + 1) words
1	0 ≤ RFRLEN1 ≤ 127	0 ≤ RFRLEN2 ≤ 127	(RFRLEN1 + 1) + (RFRLEN2 + 1) words

9.8.11 Enable/Disable the Receive Frame-Sync Ignore Function

The RFIG bit (see Figure 9–54 and Table 9–26) controls the receive frame-sync ignore function.

Figure 9–54. Register Bit Used to Enable/Disable the Receive Frame-Sync Ignore Function

RCR2

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–26. Register Bit Used to Enable/Disable the Receive Frame-Sync Ignore Function

Register	Bit	Name	Function	
RCR2	2	RFIG	Receive Frame-Sync Ignore	
			RFIG = 0	An unexpected receive frame-sync pulse causes the McBSP to restart the frame transfer.
			RFIG = 1	The McBSP ignores unexpected receive frame-sync pulses.

9.8.11.1 About Unexpected Frame-Sync Pulses and the Frame-Sync Ignore Function

If a frame-synchronization (frame-sync) pulse starts the transfer of a new frame before the current frame is fully received, this pulse is treated as an unexpected frame-sync pulse.

When RFIG = 1, reception continues, ignoring the unexpected frame-sync pulses.

When RFIG = 0, an unexpected FSR pulse causes the McBSP to discard the contents of RSR[1,2] in favor of the new incoming data. Therefore, if RFIG = 0 and an unexpected frame-sync pulse occurs, the serial port:

1) Aborts the current data transfer

2) Sets RSYNCERR in SPCR1 to 1

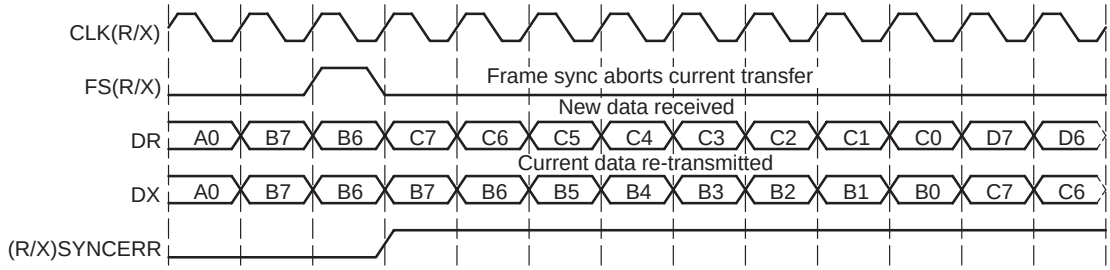
3) Begins the transfer of a new data word

For more details about the frame-sync error condition, see *Unexpected Receive Frame-Sync Pulse* on page 9-38.

9.8.11.2 Examples Showing the Effects of RFIG

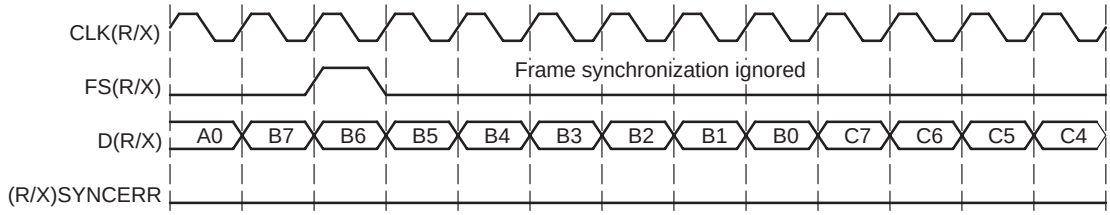
Figure 9–55 shows an example in which word B is interrupted by an unexpected frame-sync pulse when (R/X)FIG = 0. In the case of reception, the reception of B is aborted (B is lost), and a new data word (C in this example) is received after the appropriate data delay. This condition is a receive synchronization error, and thus sets the RSYNCERR bit.

Figure 9–55. Unexpected Frame-Sync Pulse With (R/X)FIG = 0



In contrast with Figure 9–55, Figure 9–56 shows McBSP operation when unexpected frame-sync signals are ignored (when (R/X)FIG = 1). Here, the transfer of word B is not affected by an unexpected pulse.

Figure 9–56. Unexpected Frame-Sync Pulse With (R/X)FIG = 1



9.8.12 Set the Receive Companding Mode

The RCOMPAND bits (see Figure 9–57 and Table 9–27) determine whether companding or another data transfer option is chosen for McBSP reception.

Figure 9–57. Register Bits Used to Set the Receive Companding Mode

RCR2

15–5	4–3	2–0
	RCOMPAND	
	R/W – 00	

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–27. Register Bits Used to Set the Receive Companding Mode

Register	Bit	Name	Function
RCR2	4–3	RCOMPAND	Receive companding mode Modes other than 00b are enabled only when the appropriate RWDLEN is 000b, indicating 8-bit data. RCOMPAND = 00 No companding, any size data, MSB received first RCOMPAND = 01 No companding, 8-bit data, LSB received first (for details, scroll down to Option to Receive LSB First) RCOMPAND = 10 μ-law companding, 8-bit data, MSB received first RCOMPAND = 11 A-law companding, 8-bit data, MSB received first

9.8.12.1 About Companding

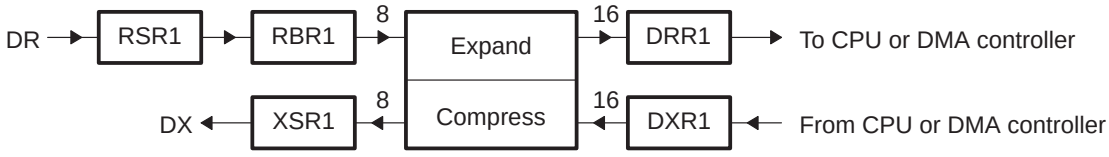
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM are part of the CCITT G.711 recommendation.

A-law and μ-law allow 13 bits and 14 bits of dynamic range, respectively. Any values outside this range are set to the most positive or most negative value. Thus, for companding to work best, the data transferred to and from the McBSP via the CPU or DMA controller must be at least 16 bits wide.

The μ-law and A-law formats both encode data into 8-bit code words. Companded data is always 8 bits wide; the appropriate word length bits (RWDLEN1, RWDLEN2, XWDLEN1, XWDLEN2) must therefore be set to 0, indicating an 8-bit wide serial data stream. If companding is enabled and either of the frame phases does not have an 8-bit word length, companding continues as if the word length is 8 bits.

Figure 9–58 illustrates the companding processes. When companding is chosen for the transmitter, compression occurs during the process of copying data from DXR1 to XSR1. The transmit data is encoded according to the specified companding law (A-law or μ-law). When companding is chosen for the receiver, expansion occurs during the process of copying data from RBR1 to DRR1. The receive data is decoded to 2s-complement format.

Figure 9–58. Companding Processes for Reception and for Transmission



9.8.12.2 Format of Expanded Data

For reception, the 8-bit compressed data in RBR1 is expanded to left-justified 16-bit data in DRR1. Note that the RJUST bit of SPCR1 is ignored when companding is used.

9.8.12.3 Companding Internal Data

If the McBSP is otherwise unused (the serial port transmit and receive sections are reset), the companding hardware can compand internal data. See *Capability to Compand Internal Data* on page 9-10.

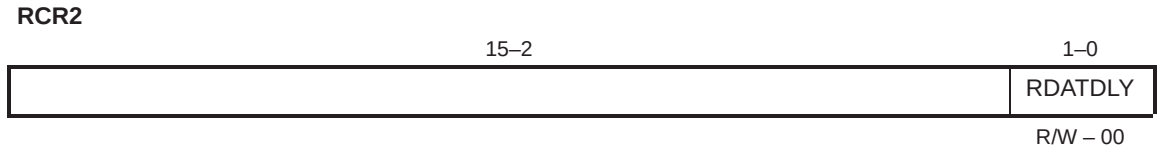
9.8.12.4 Option to Receive LSB First

Normally, the McBSP transmit or receives all data with the most significant bit (MSB) first. However, certain 8-bit data protocols (that do not use companded data) require the least significant bit (LSB) to be transferred first. If you set RCOMPAND = 01b in RCR2, the bit ordering of 8-bit words is reversed during reception. Similar to companding, this feature is enabled only if the appropriate word length bits are set to 0, indicating that 8-bit words are to be transferred serially. If either phase of the frame does not have an 8-bit word length, the McBSP assumes the word length is eight bits, and LSB-first ordering is done.

9.8.13 Set the Receive Data Delay

The RDATDLY bits (see Figure 9–59 and Table 9–28) determine the length of the data delay for the receive frame.

Figure 9–59. Register Bits Used to Set the Receive Data Delay



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–28. Register Bits Used to Set the Receive Data Delay

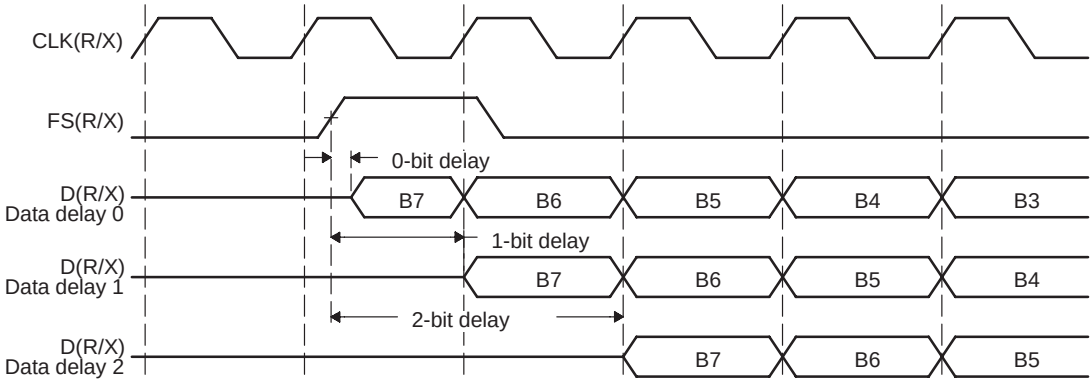
Register	Bit	Name	Function	
RCR2	1–0	RDATDLY	Receive data delay	
			RDATDLY = 00	0-bit data delay
			RDATDLY = 01	1-bit data delay
			RDATDLY = 10	2-bit data delay
			RDATDLY = 11	Reserved

9.8.13.1 About the Data Delay

The start of a frame is defined by the first clock cycle in which frame synchronization is found to be active. The beginning of actual data reception or transmission with respect to the start of the frame can be delayed if required. This delay is called data delay.

RDATDLY specifies the data delay for reception. The range of programmable data delay is zero to two bit-clocks (RDATDLY = 00b–10b), as described in Table 9–28 and shown in Figure 9–60. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on. Typically a 1-bit delay is selected, because data often follows a 1-cycle active frame-sync pulse.

Figure 9–60. Range of Programmable Data Delay



9.8.13.2 0-Bit Data Delay

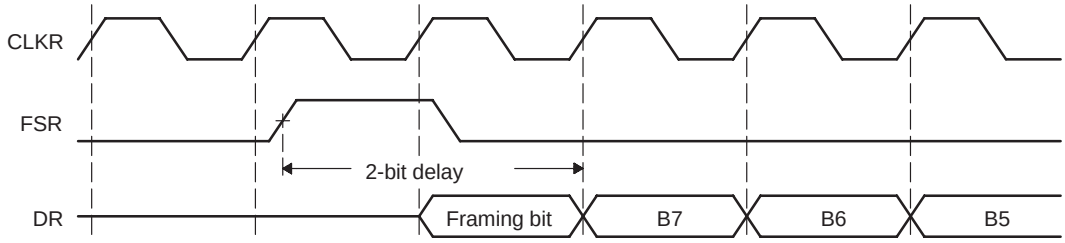
Normally, a frame-sync pulse is detected or sampled with respect to an edge of internal serial clock CLK(R/X). Thus, on the following cycle or later (depending on the data delay value), data may be received or transmitted. However, in the case of 0-bit data delay, the data must be ready for reception and/or transmission on the same serial clock cycle.

For reception this problem is solved because receive data is sampled on the first falling edge of CLKR where an active-high internal FSR is detected. However, data transmission must begin on the rising edge of the internal CLKX clock that generated the frame synchronization. Therefore, the first data bit is assumed to be present in XSR1, and thus on DX. The transmitter then asynchronously detects the frame-sync signal (FSX) going active high and immediately starts driving the first bit to be transmitted on the DX pin.

9.8.13.3 2-Bit Data Delay

A data delay of two bit periods allows the serial port to interface to different types of T1 framing devices where the data stream is preceded by a framing bit. During reception of such a stream with data delay of two bits (framing bit appears after a 1-bit delay and data appears after a 2-bit delay), the serial port essentially discards the framing bit from the data stream, as shown in Figure 9–61. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on.

Figure 9–61. 2-Bit Data Delay Used to Skip a Framing Bit



9.8.14 Set the Receive Sign-Extension and Justification Mode

The RJUST bits (see Figure 9–62 and Table 9–29) determine whether data received by the McBSP is sign extended and how it is justified.

Figure 9–62. Register Bits Used to Set the Receive Sign-Extension and Justification Mode

SPCR1

15	14–13	12–0
	RJUST	
	R/W – 00	

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–29. Register Bits Used to Set the Receive Sign-Extension and Justification Mode

Register	Bit	Name	Function	
SPCR1	14–13	RJUST	Receive Sign-Extension and Justification Mode	
			RJUST = 00	Right justify data and zero fill MSBs in DRR[1,2]
			RJUST = 01	Right justify data and sign extend it into the MSBs in DRR[1,2]
			RJUST = 10	Left justify data and zero fill LSBs in DRR[1,2]
			RJUST = 11	Reserved

9.8.14.1 About the Sign Extension and the Justification

RJUST in SPCR1 selects whether data in RBR[1,2] is right- or left-justified (with respect to the MSB) in DRR[1,2] and how unused bits in DRR[1,2] are filled—with zeros or with sign bits.

Table 9–30 and Table 9–31 show the effects of various RJUST values. The first table shows the effect on an example 12-bit receive-data value 0xABC. The second table shows the effect on an example 20-bit receive-data value 0xABCDE.

Table 9–30. Example: Use of RJUST Field With 12-Bit Data Value 0xABC

RJUST	Justification	Extension	Value in DRR2	Value in DRR1
00b	Right	Zero fill MSBs	0x0000	0x0ABC
01b	Right	Sign extend data into MSBs	0xFFFF	0xFABC
10b	Left	Zero fill LSBs	0x0000	0xABC0
11b	Reserved	Reserved	Reserved	Reserved

Table 9–31. Example: Use of RJUST Field With 20-Bit Data Value 0xABCDE

RJUST	Justification	Extension	Value in DRR2	Value in DRR1
00b	Right	Zero fill MSBs	0x000A	0xBCDE
01b	Right	Sign extend data into MSBs	0xFFFA	0xBCDE
10b	Left	Zero fill LSBs	0xABCD	0xE000
11b	Reserved	Reserved	Reserved	Reserved

9.8.15 Set the Receive Interrupt Mode

The RINTM bits (see Figure 9–63 and Table 9–32) determine which event generates a receive interrupt request to the CPU.

Figure 9–63. Register Bits Used to Set the Receive Interrupt Mode

SPCR1

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–32. Register Bits Used to Set the Receive Interrupt Mode

Register	Bit	Name	Function	
SPCR1	5–4	RINTM	Receive Interrupt Mode	
			RINTM = 00	RINT generated when RRDY changes from 0 to 1
			RINTM = 01	RINT generated by an end-of-block or end-of-frame condition in the receive multichannel selection mode
			RINTM = 10	RINT generated by a new receive frame-sync pulse
			RINTM = 11	RINT generated when RSYNCERR is set

9.8.15.1 About the Receive Interrupt and the Associated Modes

The receive interrupt (RINT) signals the CPU of changes to the serial port status. Four options exist for configuring this interrupt. The options are set by the receive interrupt mode bits, RINTM, in SPCR1.

- RINTM = 00b. Interrupt on every serial word by tracking the RRDY bit in SPCR1. Note that regardless of the value of RINTM, RRDY can be read to detect the RRDY = 1 condition.

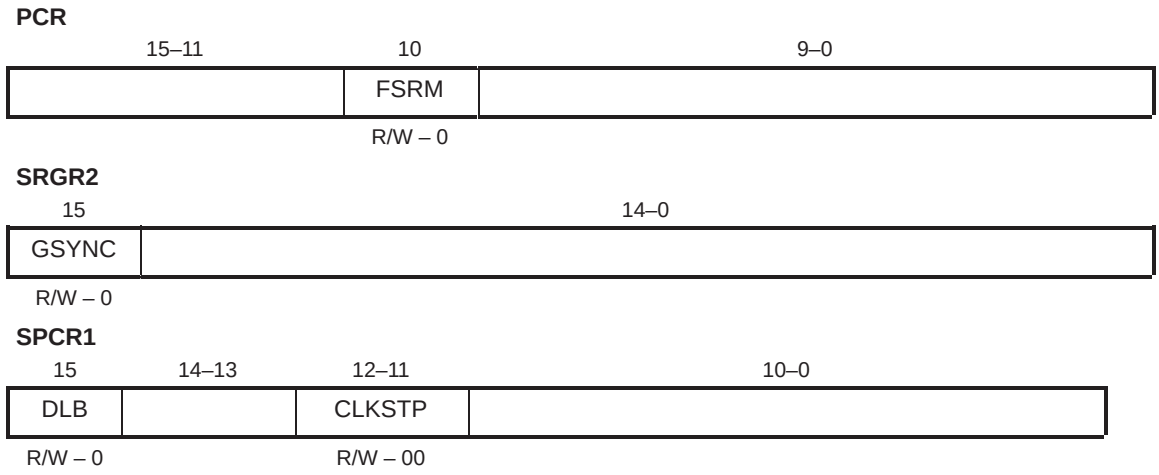
- RINTM = 01b. In the multichannel selection mode, interrupt after every 16-channel block boundary has been crossed within a frame and at the end of the frame. For details, see *Using Interrupts Between Block Transfers* on page 9-58. In any other serial transfer case, this setting is not applicable and, therefore, no interrupts are generated.

- RINTM = 10b. Interrupt on detection of receive frame-sync pulses. This generates an interrupt even when the receiver is in its reset state. This is done by synchronizing the incoming frame-sync pulse to the CPU clock and sending it to the CPU via RINT.
- RINTM = 11b. Interrupt on frame-synchronization error. Note that regardless of the value of RINTM, RSYNCERR can be read to detect this condition. For information on using RSYNCERR, see *Unexpected Receive Frame-Sync Pulse* on page 9-38.

9.8.16 Set the Receive Frame-Sync Mode

The bits shown in Figure 9–64 and described in Table 9–33 determine the source for receive frame synchronization and the function of the FSR pin.

Figure 9–64. Register Bits Used to Set the Receive Frame Sync Mode



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–33. Register Bits Used to Set the Receive Frame Sync Mode

Register	Bit	Name	Function	
PCR	10	FSRM	Receive Frame-Synchronization Mode	
			FSRM = 0	Receive frame synchronization is supplied by an external source via the FSR pin.
			FSRM = 1	Receive frame synchronization is supplied by the sample rate generator. FSR is an output pin reflecting internal FSR, except when GSYNC = 1 in SRGR2.

Table 9–33. Register Bits Used to Set the Receive Frame Sync Mode (Continued)

Register	Bit	Name	Function
SRGR2	15	GSYNC	Sample Rate Generator Clock Synchronization Mode If the sample rate generator creates a frame-sync signal (FSG) that is derived from an external input clock, the GSYNC bit determines whether FSG is kept synchronized with pulses on the FSR pin. GSYNC = 0: No clock synchronization is used: CLKG oscillates without adjustment, and FSG pulses every (FPER + 1) CLKG cycles. GSYNC = 1: Clock synchronization is used. When a pulse is detected on the FSR pin: ❑ CLKG is adjusted as necessary so that it is synchronized with the input clock on the CLKS, CLKR, or CLKX pin. ❑ FSG pulses. FSG **only** pulses in response to a pulse on the FSR pin. The frame-sync period defined in FPER is ignored. For more details, see *Synchronizing Sample Rate Generator Outputs to an External Clock* on page 9-29.
SPCR1	15	DLB	Digital Loopback Mode DLB = 0: Digital loopback mode is disabled. DLB = 1: Digital loopback mode is enabled. The receive signals, including the receive frame-sync signal, are connected internally through multiplexers to the corresponding transmit signals.
SPCR1	12–11	CLKSTP	Clock Stop Mode CLKSTP = 0Xb: Clock stop mode disabled; normal clocking for non-SPI mode. CLKSTP = 10b: Clock stop mode enabled, without clock delay. The internal receive clock signal (CLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX. CLKSTP = 11b: Clock stop mode enabled, with clock delay. The internal receive clock signal (CLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.

9.8.16.1 About the Receive Frame-Sync Modes

Table 9–34 shows how you may select various sources to provide the receive frame-synchronization signal and the effect on the FSR pin. The polarity of the signal on the FSR pin is determined by the FSRP bit.

Note that in the digital loop back mode (DLB = 1), the transmit frame-sync signal is used as the receive frame-sync signal.

Also, in the clock stop mode, the internal receive clock signal (CLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.

Table 9–34. Select Sources to Provide the Receive Frame-Synchronization Signal and the Effect on the FSR Pin

DLB	FSRM	GSYNC	Source of Receive Frame Synchronization	FSR Pin Status
0	0	0 or 1	An external frame-sync signal enters the McBSP through the FSR pin. The signal is then inverted as determined by FSRP before being used as internal FSR.	Input
0	1	0	Internal FSR is driven by the sample rate generator frame-sync signal (FSG).	Output. FSG is inverted as determined by FSRP before being driven out on the FSR pin.
0	1	1	Internal FSR is driven by the sample rate generator frame-sync signal (FSG).	Input. The external frame-sync input on the FSR pin is used to synchronize CLKG and generate FSG pulses.
1	0	0	Internal FSX drives internal FSR.	High impedance
1	0 or 1	1	Internal FSX drives internal FSR.	Input. If the sample rate generator is running, external FSR is used to synchronize CLKG and generate FSG pulses.
1	1	0	Internal FSX drives internal FSR.	Output. Receive (same as transmit) frame synchronization is inverted as determined by FSRP before being driven out on the FSR pin.

9.8.17 Set the Receive Frame-Sync Polarity

The FSRP bit (see Figure 9–65 and Table 9–35) determines whether frame-synchronization (frame-sync) pulses are active high or active low on the FSR pin.

Figure 9–65. Register Bit Used to Set Receive Frame-Sync Polarity

PCR

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–35. Register Bit Used to Set Receive Frame-Sync Polarity

Register	Bit	Name	Function	
PCR	2	FSRP	Receive Frame-Synchronization Polarity	
			FSRP = 0	Frame-synchronization pulse FSR is active high.
			FSRP = 1	Frame-synchronization pulse FSR is active low.

9.8.17.1 About Frame Sync Pulses, Clock Signals, and Their Polarities

Receive frame-sync pulses can be either generated internally by the sample rate generator (see section 9.3.2 on page 9-28) or driven by an external source. The source of frame sync is selected by programming the mode bit, FSRM, in PCR. FSR is also affected by the GSYNC bit in SRGR2. For information about the effects of FSRM and GSYNC, see *Set the Receive Frame-Sync Mode* on page 9-94. Similarly, receive clocks can be selected to be inputs or outputs by programming the mode bit, CLKRM, in the PCR (see *Set the Receive Clock Mode* on page 9-101).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-sync pulses), the McBSP detects them on the internal falling edge of clock, internal CLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal CLKR. Note that these internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

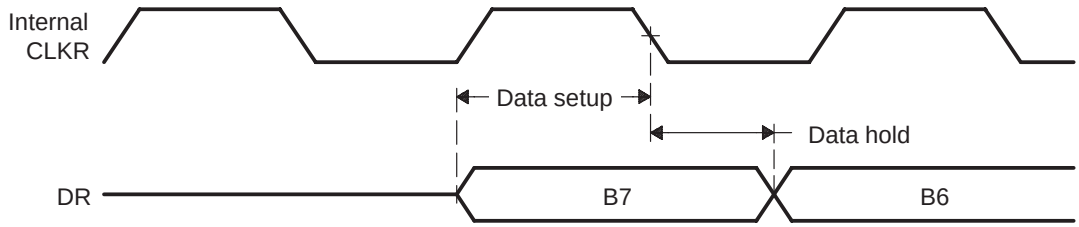
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, CLKR, and CLKX signals, respectively. All frame-sync signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP), and FSRP = FSXP = 1, the external active-low frame-sync signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active-high frame-sync signals are inverted, if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin. The figure in the topic Clock and Frame Generation shows this inversion using XOR gates.

On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Note that data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1, and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1, and internal clocking selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the CLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. Note that the receive data is always sampled on the falling edge of internal CLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and CLKR is an input pin), the external rising-edge triggered input clock on CLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1, and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the CLKR pin.

Note that CLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold of data around this edge. Figure 9–66 shows how data clocked by an external serial device using a rising edge, may be sampled by the McBSP receiver on the falling edge of the same clock.

Figure 9–66. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge

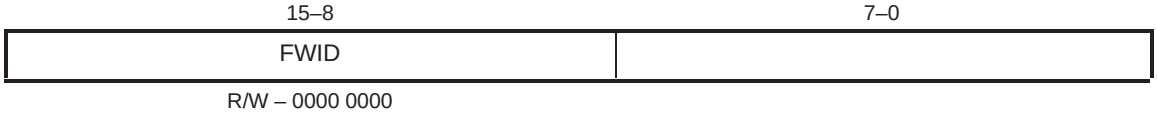


9.8.18 Set the SRG Frame-Sync Period and Pulse Width

Figure 9–67. Register Bits Used to Set the SRG Frame-Sync Period and Pulse Width

SRGR2

SRGR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–36. Register Bits Used to Set the SRG Frame-Sync Period and Pulse Width

Register	Bit	Name	Function
SRGR2	11–0	FPER	Sample Rate Generator Frame-Sync Period For the frame-sync signal FSG, (FPER + 1) determines the period from the start of a frame-sync pulse to the start of the next frame-sync pulse. Range for (FPER + 1): 1 to 4096 CLKG cycles.
SRGR1	15–8	FWID	Sample Rate Generator Frame-Sync Pulse Width This field plus 1 determines the width of each frame-sync pulse on FSG. Range for (FWID + 1): 1 to 256 CLKG cycles.

9.8.18.1 About the Frame-Sync Period and the Frame-Sync Pulse Width

The sample rate generator can produce a clock signal, CLKG, and a frame-sync signal, FSG. If the sample rate generator is supplying receive or transmit frame synchronization, you must program the bit fields FPER and FWID.

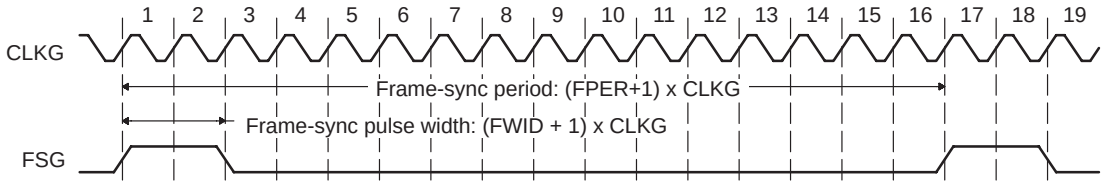
On FSG, the period from the start of a frame-sync pulse to the start of the next pulse is (FPER + 1) CLKG cycles. The 12 bits of FPER allow a frame-sync period of 1 to 4096 CLKG cycles, which allows up to 4096 data bits per frame. When GSYNC = 1, FPER is a don't care value.

Each pulse on FSG has a width of (FWID + 1) CLKG cycles. The eight bits of FWID allow a pulse width of 1 to 256 CLKG cycles. It is recommended that FWID be programmed to a value less than the programmed word length.

The values in FPER and FWID are loaded into separate down-counters. The 12-bit FPER counter counts down the generated clock cycles from the programmed value (4095 maximum) to 0. The 8-bit FWID counter counts down from the programmed value (255 maximum) to 0.

Figure 9–68 shows a frame-sync period of 16 CLKG periods (FPER = 15 or 00001111b) and a frame-sync pulse with an active width of 2 CLKG periods (FWID = 1).

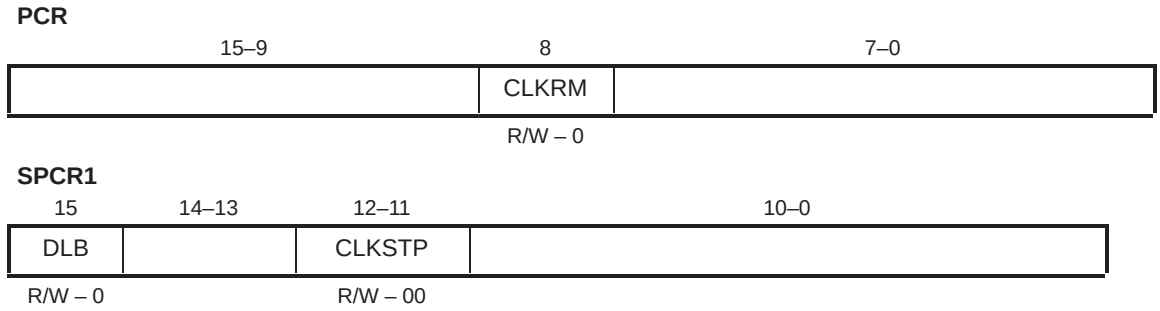
Figure 9–68. Frame of Period 16 CLKG Periods and Active Width of 2 CLKG Periods



When the sample rate generator comes out of reset, FSG is in its inactive state. Then, when FRST_ = 1 and FSGM = 1, a frame-sync pulse is generated. The frame width value (FWID + 1) is counted down on every CLKG cycle until it reaches 0, at which time FSG goes low. At the same time, the frame period value (FPER + 1) is also counting down. When this value reaches 0, FSG goes high, indicating a new frame.

9.8.19 Set the Receive Clock Mode

Figure 9–69. Register Bits Used to Set the Receive Clock Mode



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–37. Register Bits Used to Set the Receive Clock Mode

Register	Bit	Name	Function	
PCR	8	CLKRM	Receive Clock Mode	
			Case 1: Digital loopback mode not set (DLB = 0) in SPCR1.	
			CLKRM = 0	The CLKR pin is an input pin that supplies the internal receive clock (CLKR).
			CLKRM = 1	Internal CLKR is driven by the sample rate generator of the McBSP. The CLKR pin is an output pin that reflects internal CLKR.
			Case 2: Digital loopback mode set (DLB = 1) in SPCR1.	
			CLKRM = 0	The CLKR pin is in the high impedance state. The internal receive clock (CLKR) is driven by the internal transmit clock (CLKX). Internal CLKX is derived according to the CLKXM bit of PCR.
			CLKRM = 1	Internal CLKR is driven by internal CLKX. The CLKR pin is an output pin that reflects internal CLKR. Internal CLKX is derived according to the CLKXM bit of PCR.
SPCR1	15	DLB	Digital Loopback Mode	
			DLB = 0	Digital loopback mode is disabled.
			DLB = 1	Digital loopback mode is enabled. The receive signals, including the receive frame-sync signal, are connected internally through multiplexers to the corresponding transmit signals.

Table 9–37. Register Bits Used to Set the Receive Clock Mode (Continued)

Register	Bit	Name	Function	
SPCR1	12–11	CLKSTP	Clock Stop Mode	
			CLKSTP = 0Xb	Clock stop mode disabled; normal clocking for non-SPI mode.
			CLKSTP = 10b	Clock stop mode enabled, without clock delay. The internal receive clock signal (CLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.
			CLKSTP = 11b	Clock stop mode enabled, with clock delay. The internal receive clock signal (CLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.

9.8.19.1 Selecting a Source for the Receive Clock and a Data Direction for the CLKR Pin

Table 9–38 shows how you may select various sources to provide the receive clock signal and the effect on the CLKR pin. The polarity of the signal on the CLKR pin is determined by the CLKRP bit.

Note that in the digital loop back mode (DLB = 1), the transmit clock signal is used as the receive clock signal.

Also, in the clock stop mode, the internal receive clock signal (CLKR) and the internal receive frame-synchronization signal (FSR) are internally connected to their transmit counterparts, CLKX and FSX.

Table 9–38. Select Sources to Provide the Receive Clock Signal and the Effect on the CLKR Pin

DLB in SPCR1	CLKRM in PCR	Source of Receive Clock	CLKR Pin Status
0	0	The CLKR pin is an input driven by an external clock. The external clock signal is inverted as determined by CLKRP before being used.	Input
0	1	The sample rate generator clock (CLKG) drives internal CLKR.	Output. CLKG, inverted as determined by CLKRP, is driven out on the CLKR pin.

Table 9–38. Select Sources to Provide the Receive Clock Signal and the Effect on the CLKR Pin (Continued)

DLB in SPCR1	CLKRM in PCR	Source of Receive Clock	CLKR Pin Status
1	0	Internal CLKX drives internal CLKR. To configure CLKX, see *Set the Transmit Clock Mode* on page 9-136.	High impedance
1	1	Internal CLKX drives internal CLKR. To configure CLKX, see *Set the Transmit Clock Mode* on page 9-136.	Output. Internal CLKR (same as internal CLKX) is inverted as determined by CLKRP before being driven out on the CLKR pin.

9.8.20 Set the Receive Clock Polarity

Figure 9–70. Register Bit Used to Set Receive Clock Polarity

PCR

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–39. Register Bit Used to Set Receive Clock Polarity

Register	Bit	Name	Function	
PCR	0	CLKRP	Receive Clock Polarity	
			CLKRP = 0	Receive data sampled on falling edge of CLKR.
			CLKRP = 1	Receive data sampled on rising edge of CLKR.

9.8.20.1 About Frame Sync Pulses, Clock Signals, and Their Polarities

Receive frame-sync pulses can be either generated internally by the sample rate generator (see section 9.3.2 on page 9-28) or driven by an external source. The source of frame sync is selected by programming the mode bit, FSRM, in PCR. FSR is also affected by the GSYNC bit in SRGR2. For information about the effects of FSRM and GSYNC, see *Set the Receive Frame-Sync Mode* on page 9-94. Similarly, receive clocks can be selected to be inputs or outputs by programming the mode bit, CLKRM, in the PCR (see *Set the Receive Clock Mode* on page 9-101).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-sync pulses), the McBSP detects them on the internal falling edge of clock, internal CLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal CLKR. Note that these internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, CLKR, and CLKX signals, respectively. All frame-sync signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP), and FSRP = FSXP = 1, the external active-low frame-sync signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active-high frame-sync signals are inverted, if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin. The figure in the topic Clock and Frame Generation shows this inversion using XOR gates.

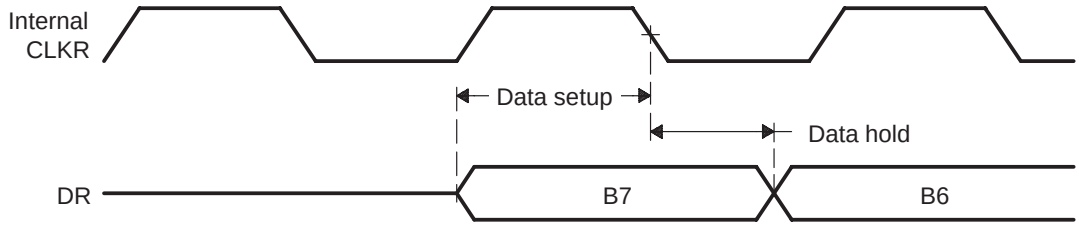
On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Note that data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1, and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1, and internal clocking selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the CLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. Note that the receive data is always sampled on the falling edge of internal CLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and CLKR is an input pin), the external rising-edge triggered input clock on CLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1, and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the CLKR pin.

Note that CLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the

opposite edge as the transmitter to ensure valid setup and hold of data around this edge. Figure 9–71 shows how data clocked by an external serial device using a rising edge, may be sampled by the McBSP receiver on the falling edge of the same clock.

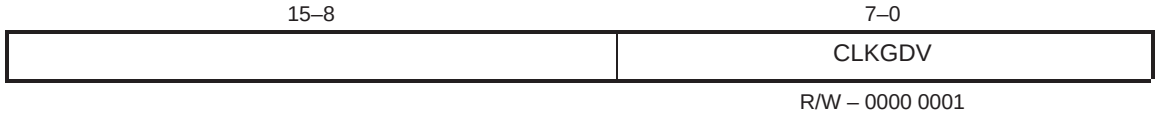
Figure 9–71. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



9.8.21 Set the SRG Clock Divide-Down Value

Figure 9–72. Register Bits Used to Set the Sample Rate Generator (SRG) Clock Divide-Down Value

SRGR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–40. Register Bits Used to Set the Sample Rate Generator (SRG) Clock Divide-Down Value

Register	Bit	Name	Function
SRGR1	7–0	CLKGDV	Sample Rate Generator Clock Divide-Down Value The input clock of the sample rate generator is divided by (CLKGDV + 1) to generate the required sample rate generator clock frequency. The default value of CLKGDV is 1 (divide input clock by 2).

9.8.21.1 About the Sample Rate Generator Clock Divider

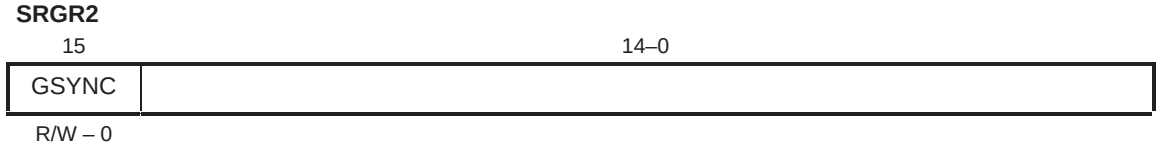
The first divider stage generates the serial data bit clock from the input clock. This divider stage utilizes a counter, preloaded by CLKGDV, that contains the divide ratio value.

The output of the first divider stage is the data bit clock, which is output as CLKG and which serves as the input for the second and third stages of the divider.

CLKG has a frequency equal to 1/(CLKGDV + 1) of sample rate generator input clock. Thus, the sample generator input clock frequency is divided by a value between 1 and 256. When CLKGDV is odd or equal to 0, the CLKG duty cycle is 50%. When CLKGDV is an even value, 2p, representing an odd divide-down, the high-state duration is p+1 cycles and the low-state duration is p cycles.

9.8.22 Set the SRG Clock Synchronization Mode

Figure 9–73. Register Bit Used to Set the SRG Clock Synchronization Mode



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

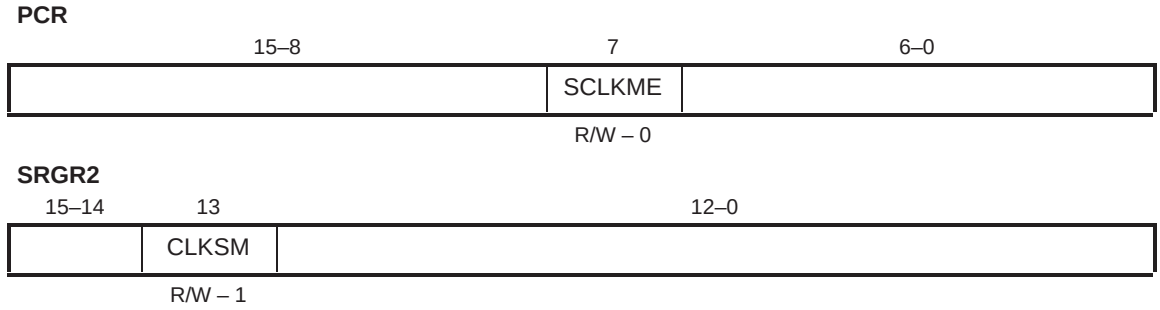
Table 9–41. Register Bit Used to Set the SRG Clock Synchronization Mode

Register	Bit	Name	Function
SRGR2	15	GSYNC	Sample Rate Generator Clock Synchronization GSYNC is used only when the input clock source for the sample rate generator is external—on the CLKS, CLKR, or CLKX pin. GSYNC = 0 The sample rate generator clock (CLKG) is free running. CLKG oscillates without adjustment, and FSG pulses every (FPER + 1) CLKG cycles. GSYNC = 1 Clock synchronization is performed. When a pulse is detected on the FSR pin: ❑ CLKG is adjusted as necessary so that it is synchronized with the input clock on the CLKS, CLKR, or CLKX pin. ❑ FSG pulses. FSG **only** pulses in response to a pulse on the FSR pin. The frame-sync period defined in FPER is ignored.

For more details on using the clock synchronization feature, see *Synchronizing Sample Rate Generator Outputs to an External Clock* on page 9-29.

9.8.23 Set the SRG Clock Mode (Choose an Input Clock)

Figure 9–74. Register Bits Used to Set the SRG Clock Mode (Choose an Input Clock)



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–42. Register Bits Used to Set the SRG Clock Mode (Choose an Input Clock)

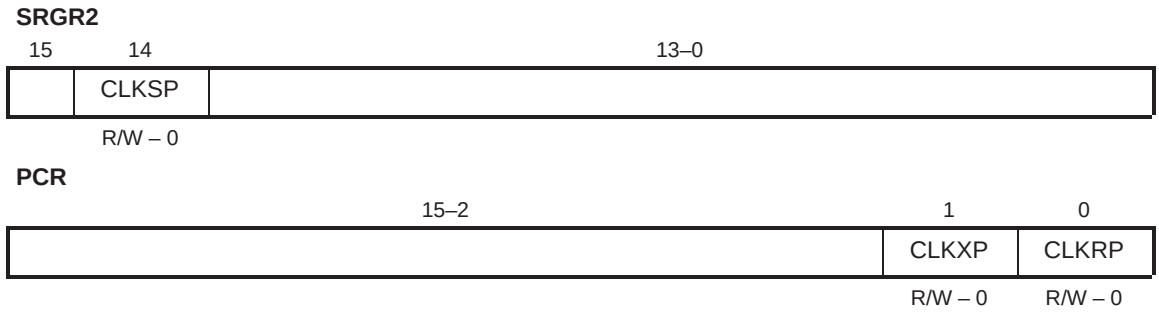
Register	Bit	Name	Function	
PCR SRGR2	7 13	SCLKME CLKSM	Sample Rate Generator Clock Mode	
			SCLKME = 0 CLKSM = 0	Sample rate generator clock derived from CLKS pin
			SCLKME = 0 CLKSM = 1	Sample rate generator clock derived from CPU clock (This is the condition forced by a DSP reset.)
			SCLKME = 1 CLKSM = 0	Sample rate generator clock derived from CLKR pin
			SCLKME = 1 CLKSM = 1	Sample rate generator clock derived from CLKX pin

9.8.23.1 About the SRG Clock Mode

The sample rate generator can produce a clock signal (CLKG) for use by the receiver, the transmitter, or both, but CLKG is derived from an input clock. The preceding table shows the four possible sources of the input clock. For more details on generating CLKG, see *Clock Generation in the Sample Rate Generator* on page 9-24.

9.8.24 Set the SRG Input Clock Polarity

Figure 9–75. Register Bits Used to Set the SRG Input Clock Polarity



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–43. Register Bits Used to Set the SRG Input Clock Polarity

Register	Bit	Name	Function
SRGR2	14	CLKSP	CLKS Pin Polarity
			CLKSP determines the input clock polarity when the CLKS pin supplies the input clock (SCLKME = 0 and CLKSM = 0).
			CLKSP = 0 Rising edge on CLKS pin generates CLKG and FSG.
			CLKSP = 1 Falling edge on CLKS pin generates CLKG and FSG.
PCR	1	CLKXP	CLKX Pin Polarity
			CLKXP determines the input clock polarity when the CLKX pin supplies the input clock (SCLKME = 1 and CLKSM = 1).
			CLKXP = 0 Rising edge on CLKX pin generates transitions on CLKG and FSG.
			CLKXP = 1 Falling edge on CLKX pin generates transitions on CLKG and FSG.
PCR	0	CLKRP	CLKR Pin Polarity
			CLKRP determines the input clock polarity when the CLKR pin supplies the input clock (SCLKME = 1 and CLKSM = 0).
			CLKRP = 0 Falling edge on CLKR pin generates transitions on CLKG and FSG.
			CLKRP = 1 Rising edge on CLKR pin generates transitions on CLKG and FSG.

9.8.24.1 Using CLKSP/CLKXP/CLKRP to Choose an Input Clock Polarity

The sample rate generator can produce a clock signal (CLKG) and a frame-sync signal (FSG) for use by the receiver, the transmitter, or both. To produce CLKG and FSG, the sample rate generator must be driven by an input clock signal derived from the CPU clock or from an external clock on the CLKS, CLKX, or CLKR pin. If you use a pin, choose a polarity for that pin by using the appropriate polarity bit (CLKSP for the CLKS pin, CLKXP for the CLKX pin, CLKRP for the CLKR pin). The polarity determines whether the rising or falling edge of the input clock generates transitions on CLKG and FSG.

9.9 Transmitter Configuration

To configure the McBSP transmitter, perform the following procedure:

1) Place the McBSP/transmitter in reset (see section 9.9.2).
2) Program the McBSP registers for the desired transmitter operation (see 9.9.1).
3) Take the transmitter out of reset (see section 9.9.2).

9.9.1 Programming the McBSP Registers for the Desired Transmitter Operation

The following is a list of important tasks to be performed when you are configuring the McBSP transmitter. Each task corresponds to one or more McBSP register bit fields. Note that in the list, SRG is an abbreviation for sample rate generator.

It may be helpful to print the McBSP Register Worksheet first and to fill in the worksheet as you read the tasks.

- **Global behavior:**
  - Set the transmitter pins to operate as McBSP pins
  - Enable/disable the digital loopback mode
  - Enable/disable the clock stop mode
  - Enable/disable transmit multichannel selection
  - Enable/disable the A-bis mode
- **Data behavior:**
  - Choose 1 or 2 phases for the transmit frame
  - Set the transmit word length(s)
  - Set the transmit frame length
  - Enable/disable the transmit frame-sync ignore function
  - Set the transmit companding mode
  - Set the transmit data delay
  - Set the transmit DXENA mode
  - Set the transmit interrupt mode
- **Frame-sync behavior:**
  - Set the transmit frame-sync mode
  - Set the transmit frame-sync polarity
  - Set the SRG frame-sync period and pulse width

- **Clock behavior:**
  - Set the transmit clock mode
  - Set the transmit clock polarity
  - Set the SRG clock divide-down value
  - Set the SRG clock synchronization mode
  - Set the SRG clock mode (choose an input clock)
  - Set the SRG input clock polarity

9.9.2 Resetting and Enabling the Transmitter

The first step of the transmitter configuration procedure is to reset the transmitter, and the last step is to enable the transmitter (to take it out of reset). Figure 9–76 and Table 9–44 describe the bits used for both of these steps.

Figure 9–76. Register Bits Used to Place Transmitter in Reset

SPCR2

15–8	7	6	5–1	0
	FRST_	GRST_		XRST_
	R/W – 0	R/W – 0		R/W – 0

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–44. Register Bits Used to Place Transmitter in Reset

Register	Bit	Name	Function	
SPCR2	0	XRST_	Transmitter Reset	
			XRST_ = 0	The serial port transmitter is disabled and in the reset state.
			XRST_ = 1	The serial port transmitter is enabled.
SPCR2	6	GRST_	Sample Rate Generator Reset	
			GRST_ = 0	Sample rate generator is reset.
				If GRST_ = 0 due to a DSP reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven low (inactive). If GRST_ = 0 due to program code, CLKG and FSG are both driven low (inactive).
			GRST_ = 1	Sample rate generator is enabled. CLKG is driven according to the configuration programmed in the sample rate generator registers (SRGR[1,2]). If FRST_ = 1, the generator also generates the frame-sync signal FSG as programmed in the sample rate generator registers.
SPCR2	7	FRST_	Frame-Sync Logic Reset	
			FRST_ = 0	Frame-synchronization logic is reset. The sample rate generator does not generate frame-sync signal FSG, even if GRST_ = 1.
			FRST_ = 1	If GRST_ = 1, frame-sync signal FSG is generated after (FPER + 1) number of CLKG clock cycles; all frame counters are loaded with their programmed values.

9.9.2.1 Reset Considerations

The serial port can be reset in the following two ways:

1) A DSP reset (RESET_ signal driven low) places the receiver, transmitter, and sample rate generator in reset. When the device reset is removed (RESET_ signal released), GRST_ = FRST_ = RRST_ = XRST_ = 0, keeping the entire serial port in the reset state.

2) The serial port transmitter and receiver can be reset directly using the RRST_ and XRST_ bits in the serial port control registers. The sample rate generator can be reset directly using the GRST_ bit in SPCR2.

Table 9–45 shows the state of McBSP pins when the serial port is reset due to a DSP reset and a direct receiver/transmitter reset.

Table 9–45. Reset State of Each McBSP Pin

Pin	Possible State(s)	State Forced By DSP Reset	State Forced By Receiver/Transmitter Reset
			Receiver Reset (RRST_ = 0 and GRST_ = 1)
DR	I	Input	Input
CLKR	I/O/Z	Input	Known state if Input; CLKR running if output
FSR	I/O/Z	Input	Known state if Input; FSRP inactive state if output
CLKS	I/O/Z	Input	Input
			Transmitter Reset (XRST_ = 0 and GRST_ = 1)
DX	O/Z	High impedance	High impedance
CLKX	I/O/Z	Input	Known state if Input; CLKX running if output
FSX	I/O/Z	Input	Known state if Input; FSXP inactive state if output
CLKS	I	Input	Input

For more details about McBSP reset conditions and effects, see *Resetting and Initializing a McBSP* on page 9-147.

9.9.3 Set the Transmitter Pins to Operate as McBSP Pins

Figure 9–77. Register Bit Used to Set Transmitter Pins to Operate as McBSP Pins

PST

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–46. Register Bit Used to Set Transmitter Pins to Operate as McBSP Pins

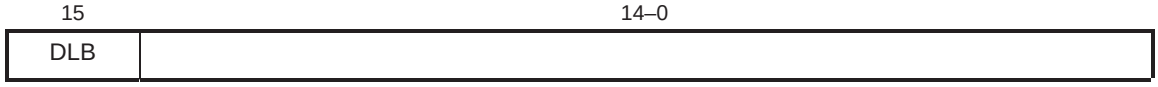
Register	Bit	Name	Function	
PCR	13	XIOEN	Transmit I/O enable	
			This bit is only applicable when the transmitter is in the reset state (XRST_ = 0 in SPCR2).	
			XIOEN = 0	The DX, FSX, CLKX, and CLKS pins are configured as serial port pins and do not function as general-purpose I/Os.
			XIOEN = 1	The DX pin is a general-purpose output pin. The FSX and CLKX pins are general-purpose I/O pins. These serial port pins do not perform serial port operation. The CLKS pin is a general-purpose input pin if RIOEN = XIOEN = 1 and RRST_ = XRST_ = 0. For more information on using these pins as general-purpose I/O pins, see page 9-144.

9.9.4 Enable/Disable the Digital Loopback Mode

The DLB bit determines whether the digital loopback mode is on. DLB is shown in Figure 9–78 and described in Table 9–47.

Figure 9–78. Register Bit Used to Enable/Disable the Digital Loopback Mode

SPCR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–47. Register Bit Used to Enable/Disable the Digital Loopback Mode

Register	Bit	Name	Function	
SPCR1	15	DLB	Digital Loopback Mode	
			DLB = 0	Digital loopback mode is disabled.
			DLB = 1	Digital loopback mode is enabled.

9.9.4.1 About the Digital Loopback Mode

In the digital loopback mode, the receive signals are connected internally through multiplexers to the corresponding transmit signals, as shown in Table 9–48. This mode allows testing of serial port code with a single DSP device; the McBSP receives the data it transmits.

Table 9–48. Receive Signals Connected to Transmit Signals in Digital Loopback Mode

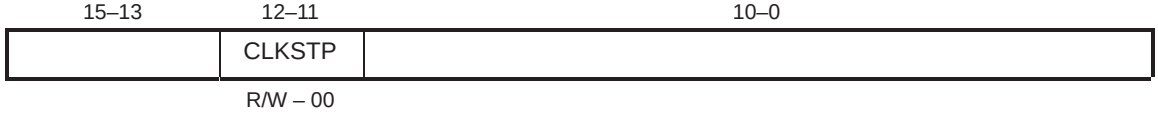
This Receive Signal …	Is Fed Internally By This Transmit Signal …
DR (receive data)	DX (transmit data)
FSR (receive frame synchronization)	FSX (transmit frame synchronization)
CLKR (receive clock)	CLKX (transmit clock)

9.9.5 Enable/Disable the Clock Stop Mode

The CLKSTP bits determine whether the clock stop mode is on. CLKSTP is shown in Figure 9–79 and described in Table 9–49.

Figure 9–79. Register Bits Used to Enable/Disable the Clock Stop Mode

SPCR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–49. Register Bits Used to Enable/Disable the Clock Stop Mode

Register	Bit	Name	Function	
SPCR1	12–11	CLKSTP	Clock Stop Mode	
			CLKSTP = 0Xb	Clock stop mode disabled; normal clocking for non-SPI mode.
			CLKSTP = 10b	Clock stop mode enabled, without clock delay
			CLKSTP = 11b	Clock stop mode enabled, with clock delay

9.9.5.1 About the Clock Stop Mode

The clock stop mode supports the SPI master-slave protocol. If you will not be using the SPI protocol, you can clear CLKSTP to disable the clock stop mode.

In the clock stop mode, the clock stops at the end of each data transfer. At the beginning of each data transfer, the clock starts immediately (CLKSTP = 10b) or after a half-cycle delay (CLKSTP = 11b). The CLKXP bit determines whether the starting edge of the clock on the CLKX pin is rising or falling. The CLKRP bit determines whether receive data is sampled on the rising or falling edge of the clock shown on the CLKR pin.

Table 9–50 summarizes the impact of CLKSTP, CLKXP, and CLKRP on serial port operation. Note that in the clock stop mode, the receive clock is tied internally to the transmit clock, and the receive frame-sync signal is tied internally to the transmit frame-sync signal.

Table 9–50. Effects of CLKSTP, CLKXP, and CLKRP on the Clock Scheme

Bit Settings	Clock Scheme
CLKSTP = 00b or 01b CLKXP = 0 or 1 CLKRP = 0 or 1	Clock stop mode disabled. Clock enabled for non-SPI mode.
CLKSTP = 10b CLKXP = 0 CLKRP = 0	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of CLKR.
CLKSTP = 11b CLKXP = 0 CLKRP = 1	Low inactive state with delay: The McBSP transmits data one-half cycle ahead of the rising edge of CLKX and receives data on the rising edge of CLKR.
CLKSTP = 10b CLKXP = 1 CLKRP = 0	High inactive state without delay: The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of CLKR.
CLKSTP = 11b CLKXP = 1 CLKRP = 1	High inactive state with delay: The McBSP transmits data one-half cycle ahead of the falling edge of CLKX and receives data on the falling edge of CLKR.

9.9.6 Enable/Disable Transmit Multichannel Selection

Figure 9–80. Register Bits Used to Enable/Disable Transmit Multichannel Selection

MCR2

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–51. Register Bits Used to Enable/Disable Transmit Multichannel Selection

Register	Bit	Name	Function	
MCR2	1–0	XMCM	Transmit Multichannel Selection	
			XMCM = 00b	No transmit multichannel selection mode is on. All channels are enabled and unmasked. No channels can be disabled or masked.
			XMCM = 01b	All channels are disabled unless they are selected in the appropriate transmit channel enable registers (XCERs). If enabled, a channel in this mode is also unmasked.
				The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.
			XMCM = 10b	All channels are enabled, but they are masked unless they are selected in the appropriate transmit channel enable registers (XCERs).
				The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.
			XMCM = 11b	This mode is used for symmetric transmission and reception.
				All channels are disabled for transmission unless they are enabled for reception in the appropriate receive channel enable registers (RCERs). Once enabled, they are masked unless they are also selected in the appropriate transmit channel enable registers (XCERs).
				The XMCME bit determines whether 32 channels or 128 channels are selectable in RCERs and XCERs.

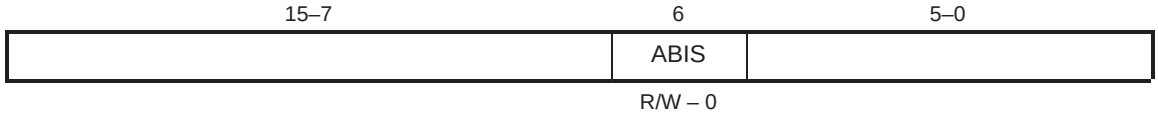
For more details, see *Transmit Multichannel Selection Modes* on page 9-54.

9.9.7 Enable/Disable the A-bis Mode

The ABIS bit determines whether the A-bis mode is on. ABIS is shown in Figure 9–81 and described in Table 9–52.

Figure 9–81. Register Bit Used to Enable/Disable the A-bis Mode

SPCR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–52. Register Bit Used to Enable/Disable the A-bis Mode

Register	Bit	Name	Function	
SPCR1	6	ABIS	A-bis Mode	
			ABIS = 0	The mode is disabled.
			ABIS = 1	The mode is enabled. Individual bits can be enabled or disabled during reception and transmission. For transmission, the bits are controlled by transmit channel enable registers A and B (XCERA and XCERB). For reception, the bits are controlled by receive channel enable registers A and B (RCERA and RCERB).

For more details, see *A-bis Mode* on page 9-59.

9.9.8 Choose 1 or 2 Phases for the Transmit Frame

Figure 9–82. Register Bit Used to Choose 1 or 2 Phases for the Transmit Frame

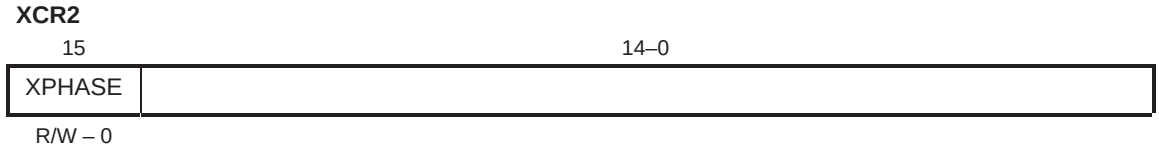


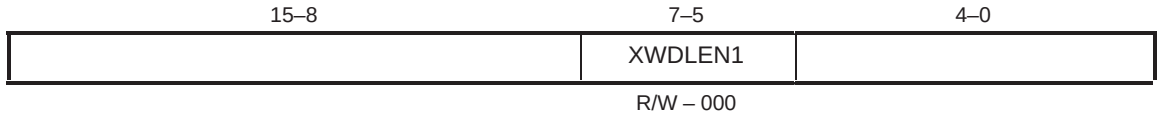
Table 9–53. Register Bit Used to Choose 1 or 2 Phases for the Transmit Frame

Register	Bit	Name	Function	
XCR2	15	XPHASE	Transmit phase number	
			Specifies whether the transmit frame has 1 or 2 phases.	
			XPHASE = 0	Single-phase frame
			XPHASE = 1	Dual-phase frame

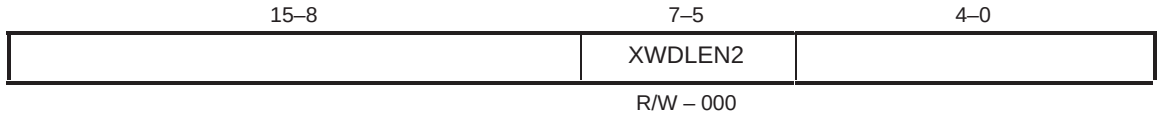
9.9.9 Set the Transmit Word Length(s)

Figure 9–83. Register Bits Used to Set the Transmit Word Length(s)

XCR1



XCR2



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–54. Register Bits Used to Set the Transmit Word Length(s)

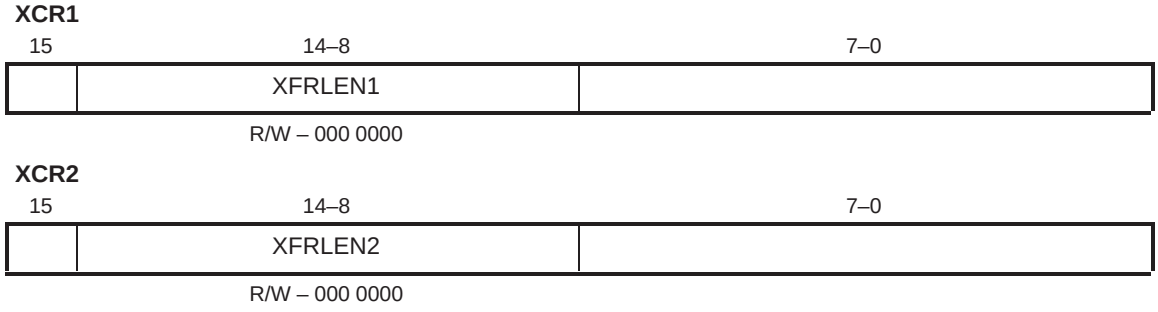
Register	Bit	Name	Function
XCR1	7–5	XWDLEN1	Transmit Word Length of Frame Phase 1
			XWDLEN1 = 000b 8 bits
			XWDLEN1 = 001b 12 bits
			XWDLEN1 = 010b 16 bits
			XWDLEN1 = 011b 20 bits
			XWDLEN1 = 100b 24 bits
			XWDLEN1 = 101b 32 bits
			XWDLEN1 = 11Xb Reserved
XCR2	7–5	XWDLEN2	Transmit Word Length of Frame Phase 2
			XWDLEN2 = 000b 8 bits
			XWDLEN2 = 001b 12 bits
			XWDLEN2 = 010b 16 bits
			XWDLEN2 = 011b 20 bits
			XWDLEN2 = 100b 24 bits
			XWDLEN2 = 101b 32 bits
			XWDLEN2 = 11Xb Reserved

9.9.9.1 About the Word Length Bits

Each frame can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, XWDLEN1 selects the length for every serial word transmitted in the frame. If a dual-phase frame is selected, XWDLEN1 determines the length of the serial words in phase 1 of the frame, and XWDLEN2 determines the word length in phase 2 of the frame.

9.9.10 Set the Transmit Frame Length

Figure 9–84. Register Bits Used to Set the Transmit Frame Length



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–55. Register Bits Used to Set the Transmit Frame Length

Register	Bit	Name	Function
XCR1	14–8	XFRLEN1	Transmit frame length 1
			(XFRLEN1 + 1) is the number of serial words in phase 1 of the transmit frame.
			XFRLEN1 = 000 0000 1 word in phase 1
			XFRLEN1 = 000 0001 2 words in phase 1
			| |
			| |
			XFRLEN1 = 111 1111 128 words in phase 1
XCR2	14–8	XFRLEN2	Transmit frame length 2
			If a dual-phase frame is selected, (XFRLEN2 + 1) is the number of serial words in phase 2 of the transmit frame.
			XFRLEN2 = 000 0000 1 word in phase 2
			XFRLEN2 = 000 0001 2 words in phase 2
			| |
			| |
			XFRLEN2 = 111 1111 128 words in phase 2

9.9.10.1 About the Selected Frame Length

The transmit frame length is the number of serial words in the transmit frame. Each frame can have one or two phases, depending on value that you load into the XPHASE bit.

If a single-phase frame is selected (XPHASE = 0), the frame length is equal to the length of phase 1. If a dual-phase frame is selected (XPHASE = 1), the frame length is the length of phase 1 plus the length of phase 2.

The 7-bit XFRLEN fields allow up to 128 words per phase. See Table 9–56 for a summary of how to calculate the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization pulse.

Note: Program the XFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For the example, if you want a phase length of 128 words in phase 1, load 127 into XFRLEN1.

Table 9–56. How to Calculate Frame Length

XPHASE	XFRLEN1	XFRLEN2	Frame Length
0	0 ≤ XFRLEN1 ≤ 127	Don't care	(XFRLEN1 + 1) words
1	0 ≤ XFRLEN1 ≤ 127	0 ≤ XFRLEN2 ≤ 127	(XFRLEN1 + 1) + (XFRLEN2 + 1) words

9.9.11 Enable/Disable the Transmit Frame-Sync Ignore Function

Figure 9–85. Register Bit Used to Enable/Disable the Transmit Frame-Sync Ignore Function

XCR2

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–57. Register Bit Used to Enable/Disable the Transmit Frame-Sync Ignore Function

Register	Bit	Name	Function	
XCR2	2	XFIG	Transmit Frame-Sync Ignore	
			XFIG = 0	An unexpected transmit frame-sync pulse causes the McBSP to restart the frame transfer.
			XFIG = 1	The McBSP ignores unexpected transmit frame-sync pulses.

9.9.11.1 About Unexpected Frame-Sync Pulses and the Frame-Sync Ignore Function

If a frame-synchronization (frame-sync) pulse starts the transfer of a new frame before the current frame is fully transmitted, this pulse is treated as an unexpected frame-sync pulse.

When XFIG = 1, normal transmission continues with unexpected frame-sync signals ignored.

When XFIG = 0 and an unexpected frame-sync pulse occurs, the serial port:

1) Aborts the present transmission

2) Sets XSYNCERR to 1 in SPCR2

3) Re-initiates transmission of the current word that was aborted

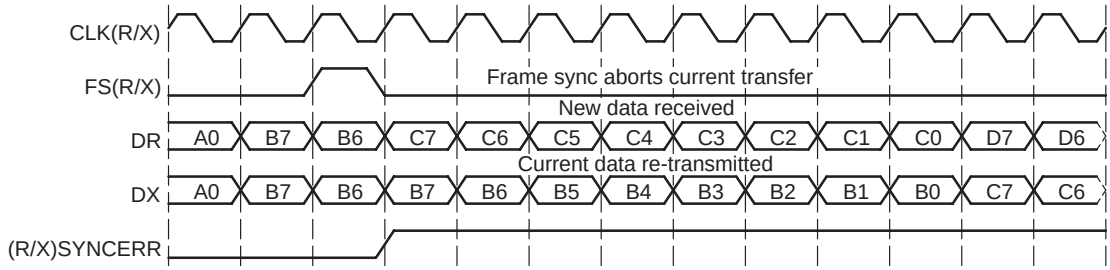
For more details about the frame-sync error condition, see *Unexpected Transmit Frame-Sync Pulse* on page 9-44.

9.9.11.2 Examples Showing the Effects of XFIG

Figure 9–86 shows an example in which word B is interrupted by an unexpected frame-sync pulse when (R/X)FIG = 0. In the case of transmission, the

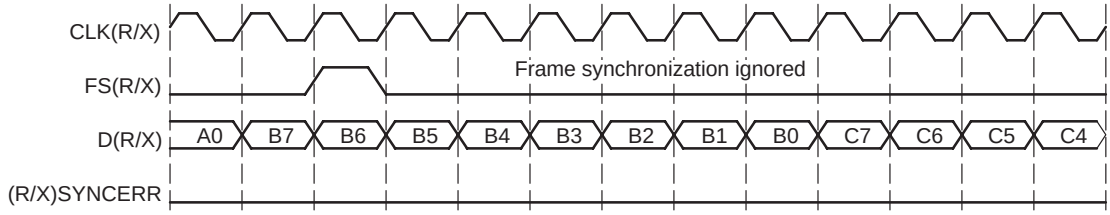
transmission of B is aborted (B is lost). This condition is a transmit synchronization error, and thus sets the XSYNCERR bit. No new data has been written to DXR[1,2], and therefore, the McBSP transmits B again.

Figure 9–86. Unexpected Frame-Sync Pulse With (R/X)FIG = 0



In contrast with Figure 9–86, Figure 9–87 shows McBSP operation when unexpected frame-sync signals are ignored (when (R/X)FIG = 1). Here, the transfer of word B is not affected by an unexpected frame-sync pulse.

Figure 9–87. Unexpected Frame-Sync Pulse With (R/X)FIG = 1



9.9.12 Set the Transmit Companding Mode

Figure 9–88. Register Bits Used to Set the Transmit Companding Mode

XCR2

15–5	4–3	2–0
	XCOMPAND	
	R/W – 00	

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–58. Register Bits Used to Set the Transmit Companding Mode

Register	Bit	Name	Function	
XCR2	4–3	XCOMPAND	Transmit Companding Mode	
			Modes other than 00b are enabled only when the appropriate XWDLEN is 000b, indicating 8-bit data.	
			XCOMPAND = 00b	No companding, any size data, MSB transmitted first
			XCOMPAND = 01b	No companding, 8-bit data, LSB transmitted first (for details, scroll down to Option to Transmit LSB First)
			XCOMPAND = 10b	μ-law companding, 8-bit data, MSB transmitted first
			XCOMPAND = 11b	A-law companding, 8-bit data, MSB transmitted first

9.9.12.1 About Companding

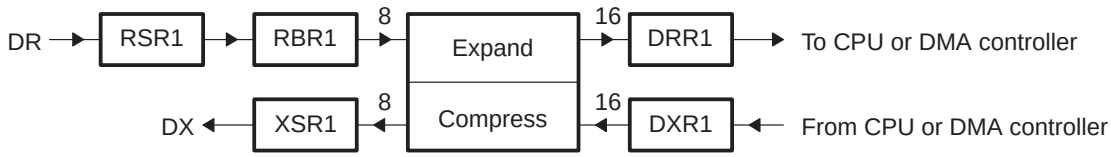
Companding (COMpressing and exPANDing) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred to as A-law. The specifications for μ-law and A-law log PCM are part of the CCITT G.711 recommendation.

A-law and μ-law allow 13 bits and 14 bits of dynamic range, respectively. Any values outside this range are set to the most positive or most negative value. Thus, for companding to work best, the data transferred to and from the McBSP via the CPU or DMA controller must be at least 16 bits wide.

The μ-law and A-law formats both encode data into 8-bit code words. Companded data is always 8 bits wide; the appropriate word length bits (RWDLEN1, RWDLEN2, XWDLEN1, XWDLEN2) must therefore be set to 0, indicating an 8-bit wide serial data stream. If companding is enabled and either of the frame phases does not have an 8-bit word length, companding continues as if the word length is 8 bits.

Figure 9–89 illustrates the companding processes. When companding is chosen for the transmitter, compression occurs during the process of copying data from DXR1 to XSR1. The transmit data is encoded according to the specified companding law (A-law or μ-law). When companding is chosen for the receiver, expansion occurs during the process of copying data from RBR1 to DRR1. The receive data is decoded to 2s-complement format.

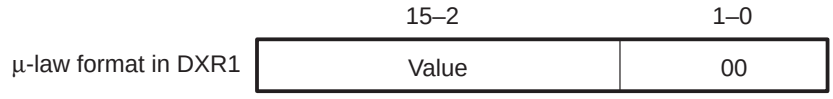
Figure 9–89. Companding Processes for Reception and for Transmission



9.9.12.2 Format for Data To Be Compressed

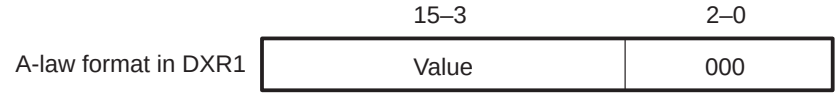
For transmission using μ-law compression, make sure the 14 data bits are left-justified in DXR1, with the remaining two low-order bits filled with 0s as shown in Figure 9–90.

Figure 9–90. μ-Law Transmit Data Companding Format



For transmission using A-law compression, make sure the 13 data bits are left-justified in DXR1, with the remaining three low-order bits filled with 0s as shown in Figure 9–91.

Figure 9–91. A-Law Transmit Data Companding Format



9.9.12.3 Capability to Compand Internal Data

If the McBSP is otherwise unused (the serial port transmit and receive sections are reset), the companding hardware can compand internal data. See section 9.2.2.2 on page 9-10.

9.9.12.4 Option to Transmit LSB First

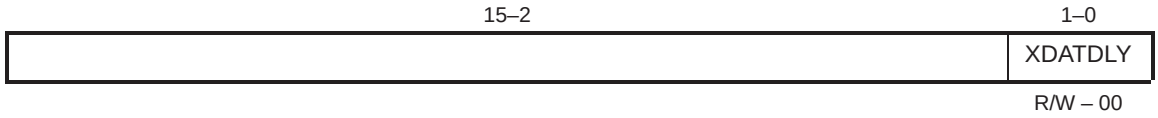
Normally, the McBSP transmit or receives all data with the most significant bit (MSB) first. However, certain 8-bit data protocols (that do not use companded data) require the least significant bit (LSB) to be transferred first. If you set XCOMPAND = 01b in XCR2, the bit ordering of 8-bit words is reversed (LSB first) before being sent from the serial port. Similar to companding, this feature is enabled only if the appropriate word length bits are set to 0, indicating that

8-bit words are to be transferred serially. If either phase of the frame does not have an 8-bit word length, the McBSP assumes the word length is eight bits, and LSB-first ordering is done.

9.9.13 Set the Transmit Data Delay

Figure 9–92. Register Bits Used to Set the Transmit Data Delay

XCR2



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–59. Register Bits Used to Set the Transmit Data Delay

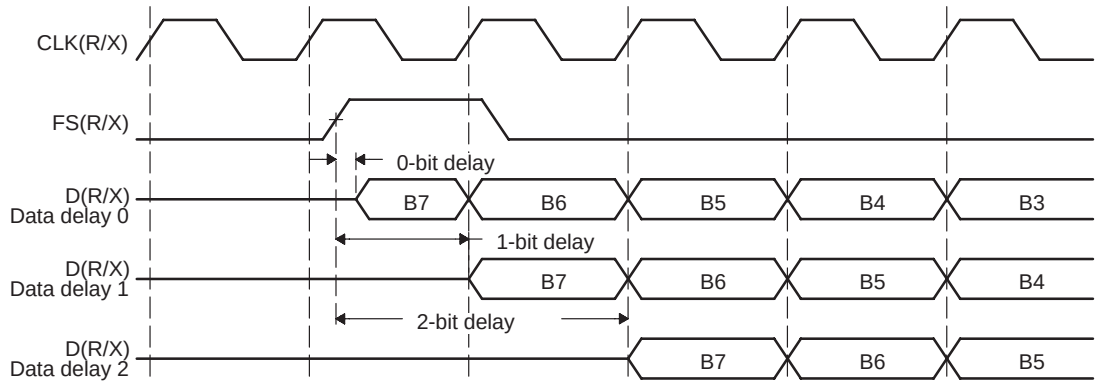
Register	Bit	Name	Function	
XCR2	1–0	XDATDLY	Transmitter data delay	
			XDATDLY = 00	0-bit data delay
			XDATDLY = 01	1-bit data delay
			XDATDLY = 10	2-bit data delay
			XDATDLY = 11	Reserved

9.9.13.1 About the Data Delay

The start of a frame is defined by the first clock cycle in which frame synchronization is found to be active. The beginning of actual data reception or transmission with respect to the start of the frame can be delayed if required. This delay is called data delay.

XDATDLY specifies the data delay for transmission. The range of programmable data delay is zero to two bit-clocks (XDATDLY = 00b–10b), as described in Table 9–59 and Figure 9–93. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on. Typically a 1-bit delay is selected, because data often follows a 1-cycle active frame-sync pulse.

Figure 9–93. Range of Programmable Data Delay



9.9.13.2 0-Bit Data Delay

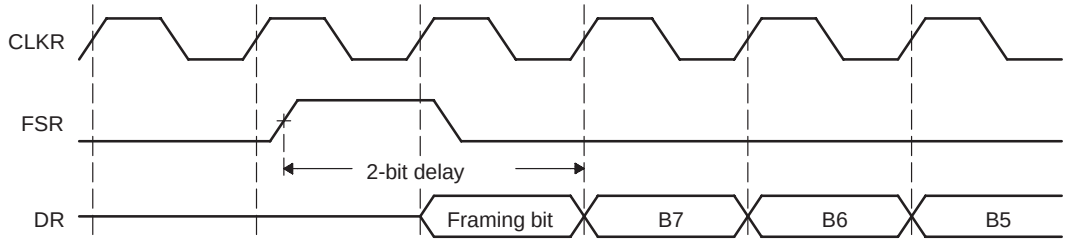
Normally, a frame-sync pulse is detected or sampled with respect to an edge of serial clock internal CLK(R/X). Thus, on the following cycle or later (depending on the data delay value), data may be received or transmitted. However, in the case of 0-bit data delay, the data must be ready for reception and/or transmission on the same serial clock cycle.

For reception this problem is solved because receive data is sampled on the first falling edge of CLKR where an active-high internal FSR is detected. However, data transmission must begin on the rising edge of the internal CLKX clock that generated the frame synchronization. Therefore, the first data bit is assumed to be present in XSR1, and thus DX. The transmitter then asynchronously detects the frame synchronization, FSX, going active high, and immediately starts driving the first bit to be transmitted on the DX pin.

9.9.13.3 2-Bit Data Delay

A data delay of two bit periods allows the serial port to interface to different types of T1 framing devices where the data stream is preceded by a framing bit. During reception of such a stream with data delay of two bits (framing bit appears after a 1-bit delay and data appears after a 2-bit delay), the serial port essentially discards the framing bit from the data stream, as shown in the following figure. In this figure, the data transferred is an 8-bit value with bits labeled B7, B6, B5, and so on.

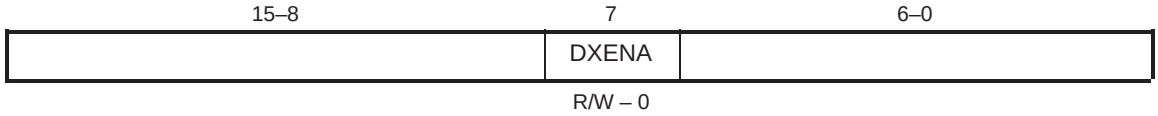
Figure 9–94. 2-Bit Data Delay Used to Skip a Framing Bit



9.9.14 Set the Transmit DXENA Mode

Figure 9–95. Register Bit Used to Set the Transmit DXENA (DX Delay Enabler) Mode

SPCR1



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–60. Register Bit Used to Set the Transmit DXENA (DX Delay Enabler) Mode

Register	Bit	Name	Function	
SPCR1	7	DXENA	DX Delay Enabler Mode	
			DXENA = 0	DX delay enabler is off.
			DXENA = 1	DX delay enabler is on.

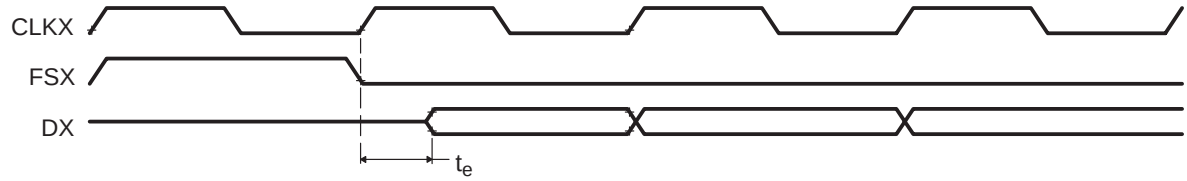
9.9.14.1 About the DXENA Mode

The DXENA bit controls the delay enabler on the DX pin. Set DXENA to enable an extra delay for turn-on time (for the length of the delay, see the data sheet for your TMS320C55x DSP). Note that this bit does not control the data itself, so only the first bit is delayed, unless the A-bis mode is on. In the A-bis mode, any bit can be delayed because any bit can go from the high-impedance state to the valid state.

If you tie together the DX pins of multiple McBSPs, make sure DXENA = 1 to avoid having more than one McBSP transmit on the data line at one time.

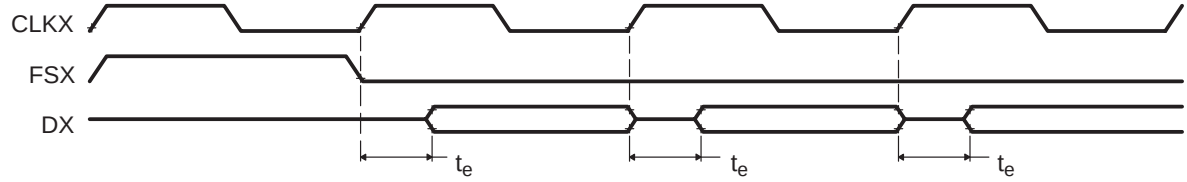
The following two figures show the timing of the DX pin for DXENA = 1. The first figure shows the effect of the DX delay enabler when the A-bis mode is off. The second figure shows the effect of the DX delay enabler when the A-bis mode is on.

Figure 9–96. DX Delay When A-bis Mode is Off



Note: te = extra delay for turn on time with DXENA = 1

Figure 9–97. DX Delays When A-bis Mode is On



Note: te = extra delay for turn on time with DXENA = 1

9.9.15 Set the Transmit Interrupt Mode

Figure 9–98. Register Bits Used to Set the Transmit Interrupt Mode

SPCR2

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–61. Register Bits Used to Set the Transmit Interrupt Mode

Register	Bit	Name	Function	
SPCR2	5–4	XINTM	Transmit Interrupt Mode	
			XINTM = 00	XINT generated when XRDY changes from 0 to 1
			XINTM = 01	XINT generated by an end-of-block or end-of-frame condition in a transmit multichannel selection mode
			XINTM = 10	XINT generated by a new transmit frame-sync pulse
			XINTM = 11	XINT generated when XSYNCERR is set

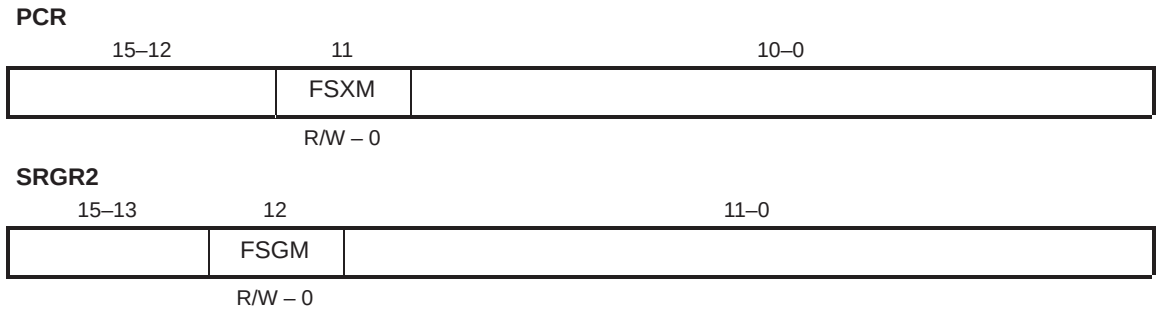
9.9.15.1 About the Transmitter Interrupt and the Associated Modes

The transmitter interrupt (XINT) signals the CPU of changes to the serial port status. Four options exist for configuring this interrupt. The options are set by the transmit interrupt mode bits, XINTM, in SPCR2.

- ❏ XINTM = 00b. Interrupt on every serial word by tracking the XRDY bit in SPCR2. Note that regardless of the value of XINTM, XRDY can be read to detect the XRDY = 1 condition.
- ❏ XINTM = 01b. In any of the transmit multichannel selection modes, interrupt after every 16-channel block boundary has been crossed within a frame and at the end of the frame. For details, see *Using Interrupts Between Block Transfers* on page 9-58. In any other serial transfer case, this setting is not applicable and, therefore, no interrupts are generated.
- ❏ XINTM = 10b. Interrupt on detection of each transmit frame-sync pulse. This generates an interrupt even when the transmitter is in its reset state. This is done by synchronizing the incoming frame-sync pulse to the CPU clock and sending it to the CPU via XINT.
- ❏ XINTM = 11b. Interrupt on frame-synchronization error. Note that regardless of the value of XINTM, XSYNCERR can be read to detect this condition. For more information on using XSYNCERR, see *Unexpected Transmit Frame-Sync Pulse* on page 9-44.

9.9.16 Set the Transmit Frame-Sync Mode

Figure 9–99. Register Bits Used to Set the Transmit Frame-Sync Mode



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–62. Register Bits Used to Set the Transmit Frame-Sync Mode

Register	Bit	Name	Function	
PCR	11	FSXM	Transmit Frame-Synchronization Mode	
			FSXM = 0	Transmit frame synchronization is supplied by an external source via the FSX pin.
			FSXM = 1	Transmit frame synchronization is supplied by the McBSP, as determined by the FSGM bit of SRGR2.
SRGR2	12	FSGM	Sample Rate Generator Transmit Frame-Synchronization Mode	
			Used when FSXM = 1 in PCR.	
			FSGM = 0	The McBSP generates a transmit frame-sync pulse when the content of DXR[1,2] is copied to XSR[1,2].
			FSGM = 1	The transmitter uses frame-sync pulses generated by the sample rate generator. Program the FWID bits to set the width of each pulse. Program the FPER bits to set the frame-sync period.

9.9.16.1 About the Transmit Frame-Sync Modes

Table 9–63 shows how FSXM and FSGM select the source of transmit frame-sync pulses. The three choices are:

- External frame-sync input
- Sample rate generator frame-sync signal (FSG).
- Internal signal that indicates a DXR-to-XSR copy has been made

Table 9–63 also shows the effect of each bit setting on the FSX pin. The polarity of the signal on the FSX pin is determined by the FSXP bit.

Table 9–63. How FSXM and FSGM Select the Source of Transmit Frame-Sync Pulses

FSXM	FSGM	Source of Transmit Frame Synchronization	FSX Pin Status
0	0 or 1	An external frame-sync signal enters the McBSP through the FSX pin. The signal is then inverted by FSXP before being used as internal FSX.	Input
1	1	Internal FSX is driven by the sample rate generator frame-sync signal (FSG).	Output. FSG is inverted by FSXP before being driven out on FSX pin.
1	0	A DXR-to-XSR copy causes the McBSP to generate a transmit frame-sync pulse that is 1 cycle wide.	Output. The generated frame-sync pulse is inverted as determined by FSXP before being driven out on FSX pin.

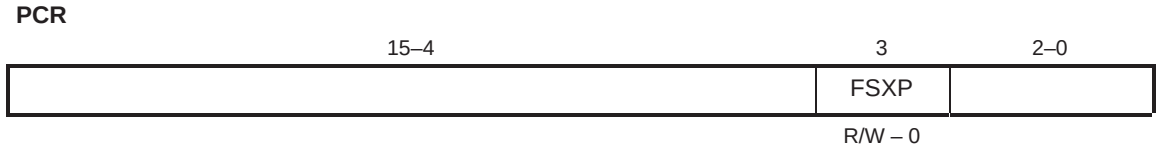
9.9.16.2 Other Considerations

If the sample rate generator creates a frame-sync signal (FSG) that is derived from an external input clock, the GSYNC bit determines whether FSG is kept synchronized with pulses on the FSR pin. For more details, see *Synchronizing Sample Rate Generator Outputs to an External Clock* on page 9-29.

In the clock stop mode (CLKSTP = 10b or 11b), the McBSP can act as a master or as a slave in the SPI protocol. If the McBSP is a master and must provide a slave-enable signal (SS_) on the FSX pin, make sure that FSXM = 1 and FSGM = 0, so that FSX is an output and is driven active for the duration of each transmission. If the McBSP is a slave, make sure that FSXM = 0, so that the McBSP can receive the slave-enable signal on the FSX pin.

9.9.17 Set the Transmit Frame-Sync Polarity

Figure 9–100. Register Bit Used to Set Transmit Frame-Sync Polarity



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–64. Register Bit Used to Set Transmit Frame-Sync Polarity

Register	Bit	Name	Function	
PCR	3	FSXP	Transmit Frame-Synchronization Polarity	
			FSXP = 0	Frame-synchronization pulse FSX is active high.
			FSXP = 1	Frame-synchronization pulse FSX is active low.

9.9.17.1 About Frame Sync Pulses, Clock Signals, and Their Polarities

Transmit frame-sync pulses can be either generated internally by the sample rate generator (see section 9.3.2 on page 9-28) or driven by an external source. The source of frame sync is selected by programming the mode bit, FSXM, in PCR. FSX is also affected by the FSGM bit in SRGR2. For information about the effects of FSXM and FSGM, see *Set the Transmit Frame-Sync Mode* on page 9-130). Similarly, transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLKXM, in the PCR (see *Set the Transmit Clock Mode* on page 9-136).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-sync pulses), the McBSP detects them on the internal falling edge of clock, internal CLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal CLKR. Note that these internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

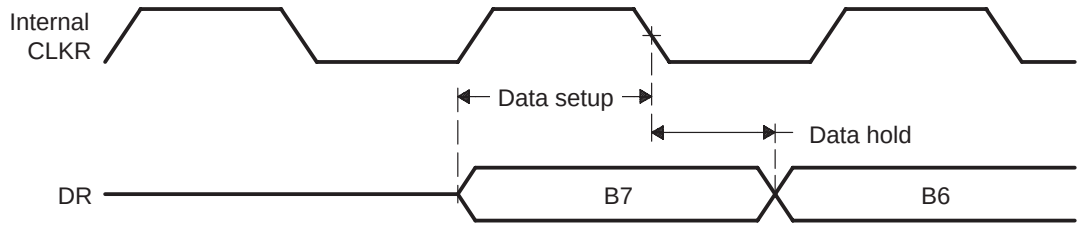
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, CLKR, and CLKX signals, respectively. All frame-sync signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP), and FSRP = FSXP = 1, the external active-low frame-sync signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active-high frame-sync signals are inverted, if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin. The figure in the topic Clock and Frame Generation shows this inversion using XOR gates.

On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Note that data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1, and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1, and internal clocking selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the CLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. Note that the receive data is always sampled on the falling edge of internal CLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and CLKR is an input pin), the external rising-edge triggered input clock on CLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1, and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the CLKR pin.

Note that CLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold of data around this edge. Figure 9–101 shows how data clocked by an external serial device using a rising edge, may be sampled by the McBSP receiver on the falling edge of the same clock.

Figure 9–101. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



9.9.18 Set the SRG Frame-Sync Period and Pulse Width

Figure 9–102. Register Bits Used to Set the SRG Frame-Sync Period and Pulse Width

SRGR2

15–12	11–0
	FPER
	R/W – 0000 0000 0000

SRGR1

15–8	7–0
FWID	
R/W – 0000 0000	

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–65. Register Bits Used to Set the SRG Frame-Sync Period and Pulse Width

Register	Bit	Name	Function
SRGR2	11–0	FPER	Sample Rate Generator Frame-Sync Period For the frame-sync signal FSG, (FPER + 1) determines the period from the start of a frame-sync pulse to the start of the next frame-sync pulse. Range for (FPER + 1): 1 to 4096 CLKG cycles.
SRGR1	15–8	FWID	Sample Rate Generator Frame-Sync Pulse Width This field plus 1 determines the width of each frame-sync pulse on FSG. Range for (FWID + 1): 1 to 256 CLKG cycles.

9.9.18.1 About the Frame-Sync Period and the Frame-Sync Pulse Width

The sample rate generator can produce a clock signal, CLKG, and a frame-sync signal, FSG. If the sample rate generator is supplying receive or transmit frame synchronization, you must program the bit fields FPER and FWID.

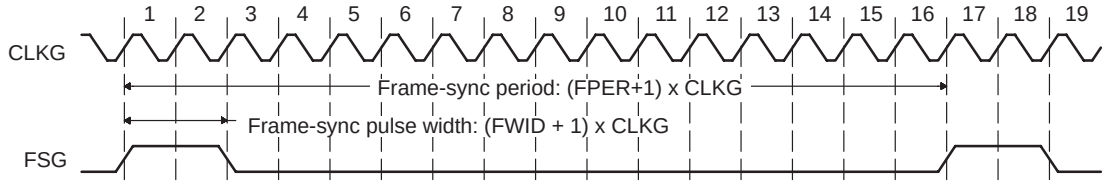
On FSG, the period from the start of a frame-sync pulse to the start of the next pulse is (FPER + 1) CLKG cycles. The 12 bits of FPER allow a frame-sync period of 1 to 4096 CLKG cycles, which allows up to 4096 data bits per frame. When GSYNC = 1, FPER is a don't care value.

Each pulse on FSG has a width of (FWID + 1) CLKG cycles. The eight bits of FWID allow a pulse width of 1 to 256 CLKG cycles. It is recommended that FWID be programmed to a value less than the programmed word length.

The values in FPER and FWID are loaded into separate down-counters. The 12-bit FPER counter counts down the generated clock cycles from the programmed value (4095 maximum) to 0. The 8-bit FWID counter counts down from the programmed value (255 maximum) to 0.

Figure 9–103 shows a frame-sync period of 16 CLKG periods (FPER = 15 or 00001111b) and a frame-sync pulse with an active width of 2 CLKG periods (FWID = 1).

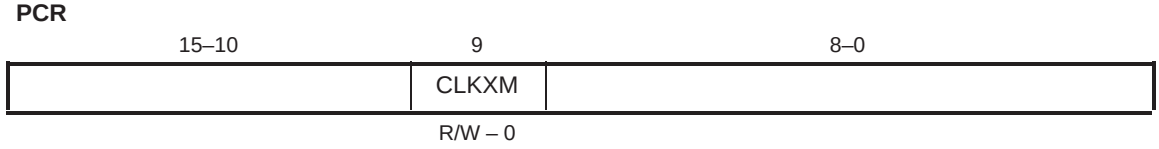
Figure 9–103. Frame of Period 16 CLKG Periods and Active Width of 2 CLKG Periods



When the sample rate generator comes out of reset, FSG is in its inactive state. Then, when FRST_ = 1 and FSGM = 1, a frame-sync pulse is generated. The frame width value (FWID + 1) is counted down on every CLKG cycle until it reaches 0, at which time FSG goes low. At the same time, the frame period value (FPER + 1) is also counting down. When this value reaches 0, FSG goes high, indicating a new frame.

9.9.19 Set the Transmit Clock Mode

Figure 9–104. Register Bit Used to Set the Transmit Clock Mode



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–66. Register Bit Used to Set the Transmit Clock Mode

Register	Bit	Name	Function	
PCR	9	CLKXM	Transmit Clock Mode	
			CLKXM = 0	The transmitter gets its clock signal from an external source via the CLKX pin.
			CLKXM = 1	The CLKX pin is an output pin driven by the sample rate generator of the McBSP.

9.9.19.1 Selecting a Source for the Transmit Clock and a Data Direction for the CLKX Pin

Table 9–67 shows how the CLKXM bit selects the transmit clock and the corresponding status of the CLKX pin. The polarity of the signal on the CLKX pin is determined by the CLKXP bit.

Table 9–67. How the CLKXM Bit Selects the Transmit Clock and the Corresponding Status of the CLKX Pin

CLKXM in PCR	Source of Transmit Clock	CLKX Pin Status
0	Internal CLKX is driven by an external clock on the CLKX pin. CLKX is inverted as determined by CLKXP before being used.	Input
1	Internal CLKX is driven by the sample rate generator clock, CLKG.	Output. CLKG, inverted as determined by CLKXP, is driven out on CLKX.

9.9.19.2 Other Considerations

If the sample rate generator creates a clock signal (CLKG) that is derived from an external input clock, the GSYNC bit determines whether CLKG is kept synchronized with pulses on the FSR pin. For more details, see *Synchronizing Sample Rate Generator Outputs to an External Clock* on page 9-29.

In the clock stop mode (CLKSTP = 10b or 11b), the McBSP can act as a master or as a slave in the SPI protocol. If the McBSP is a master, make sure that CLKXM = 1, so that CLKX is an output to supply the master clock to any slave devices. If the McBSP is a slave, make sure that CLKXM = 0, so that CLKX is an input to accept the master clock signal.

9.9.20 Set the Transmit Clock Polarity

Figure 9–105. Register Bit Used to Set Transmit Clock Polarity

PCR

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–68. Register Bit Used to Set Transmit Clock Polarity

Register	Bit	Name	Function	
PCR	1	CLKXP	Transmit Clock Polarity	
			CLKXP = 0	Transmit data sampled on rising edge of CLKX.
			CLKXP = 1	Transmit data sampled on falling edge of CLKX.

9.9.20.1 About Frame Sync Pulses, Clock Signals, and Their Polarities

Transmit frame-sync pulses can be either generated internally by the sample rate generator (see section 9.3.2 on page 9-28) or driven by an external source. The source of frame sync is selected by programming the mode bit, FSXM, in PCR. FSX is also affected by the FSGM bit in SRGR2. For information about the effects of FSXM and FSGM, see *Set the Transmit Frame-Sync Mode* on page 9-130). Similarly, transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLKXM, in the PCR (see *Set the Transmit Clock Mode* on page 9-136).

When FSR and FSX are inputs (FSXM = FSRM= 0, external frame-sync pulses), the McBSP detects them on the internal falling edge of clock, internal CLKR, and internal CLKX, respectively. The receive data arriving at the DR pin is also sampled on the falling edge of internal CLKR. Note that these internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs, implying that they are driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X). Similarly, data on the DX pin is output on the rising edge of internal CLKX.

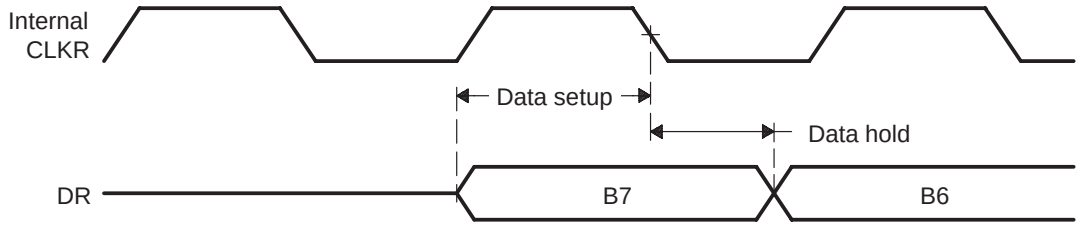
FSRP, FSXP, CLKRP, and CLKXP in the pin control register (PCR) configure the polarities of the FSR, FSX, CLKR, and CLKX signals, respectively. All frame-sync signals (internal FSR, internal FSX) that are internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP), and FSRP = FSXP = 1, the external active-low frame-sync signals are inverted before being sent to the receiver (internal FSR) and transmitter (internal FSX). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active-high frame-sync signals are inverted, if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin. The figure in the topic Clock and Frame Generation shows this inversion using XOR gates.

On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Note that data is always transmitted on the rising edge of internal CLKX. If CLKXP = 1, and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1, and internal clocking selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, internal CLKX, is inverted before being sent out on the CLKX pin.

Similarly, the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. Note that the receive data is always sampled on the falling edge of internal CLKR. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and CLKR is an input pin), the external rising-edge triggered input clock on CLKR is inverted to a falling-edge triggered clock before being sent to the receiver. If CLKRP = 1, and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge triggered clock before being sent out on the CLKR pin.

Note that CLKRP = CLKXP in a system where the same clock (internal or external) is used to clock the receiver and transmitter. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold of data around this edge. Figure 9–106 shows how data clocked by an external serial device using a rising edge, may be sampled by the McBSP receiver on the falling edge of the same clock.

Figure 9–106. Data Clocked Externally Using a Rising Edge and Sampled by the McBSP Receiver on a Falling Edge



9.9.21 Set the SRG Clock Divide-Down Value

Figure 9–107. Register Bits Used to Set the Sample Rate Generator (SRG) Clock Divide-Down Value

SRGR1

15–8	7–0
	CLKGDV
	R/W – 0000 0001

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–69. Register Bits Used to Set the Sample Rate Generator (SRG) Clock Divide-Down Value

Register	Bit	Name	Function
SRGR1	7–0	CLKGDV	Sample Rate Generator Clock Divide-Down Value The input clock of the sample rate generator is divided by (CLKGDV + 1) to generate the required sample rate generator clock frequency. The default value of CLKGDV is 1 (divide input clock by 2).

9.9.21.1 About the Sample Rate Generator Clock Divider

The first divider stage generates the serial data bit clock from the input clock. This divider stage utilizes a counter, preloaded by CLKGDV, that contains the divide ratio value.

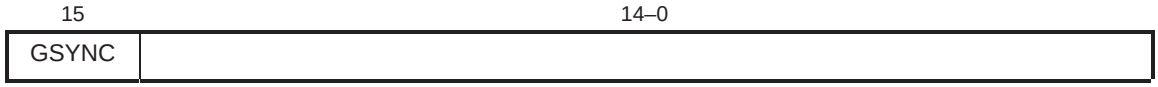
The output of the first divider stage is the data bit clock, which is output as CLKG and which serves as the input for the second and third stages of the divider.

CLKG has a frequency equal to 1/(CLKGDV + 1) of sample rate generator input clock. Thus, the sample generator input clock frequency is divided by a value between 1 and 256. When CLKGDV is odd or equal to 0, the CLKG duty cycle is 50%. When CLKGDV is an even value, 2p, representing an odd divide-down, the high-state duration is p+1 cycles and the low-state duration is p cycles.

9.9.22 Set the SRG Clock Synchronization Mode

Figure 9–108. Register Bit Used to Set the SRG Clock Synchronization Mode

SRGR2



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–70. Register Bit Used to Set the SRG Clock Synchronization Mode

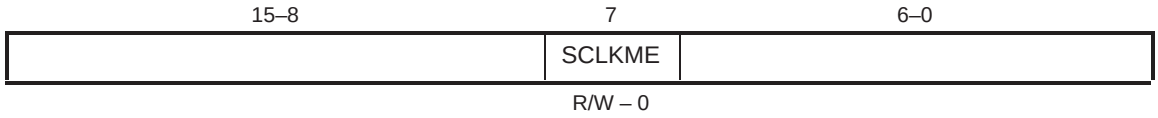
Register	Bit	Name	Function
SRGR2	15	GSYNC	Sample Rate Generator Clock Synchronization GSYNC is used only when the input clock source for the sample rate generator is external—on the CLKS, CLKR, or CLKX pin. GSYNC = 0 The sample rate generator clock (CLKG) is free running. CLKG oscillates without adjustment, and FSG pulses every (FPER + 1) CLKG cycles. GSYNC = 1 Clock synchronization is performed. When a pulse is detected on the FSR pin: ❑ CLKG is adjusted as necessary so that it is synchronized with the input clock on the CLKS, CLKR, or CLKX pin. ❑ FSG pulses. FSG **only** pulses in response to a pulse on the FSR pin. The frame-sync period defined in FPER is ignored.

For more details on using the clock synchronization feature, see *Synchronizing Sample Rate Generator Outputs to an External Clock* on page 9-29.

9.9.23 Set the SRG Clock Mode (Choose an Input Clock)

Figure 9–109. Register Bits Used to Set the SRG Clock Mode (Choose an Input Clock)

PCR



SRGR2

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–71. Register Bits Used to Set the SRG Clock Mode (Choose an Input Clock)

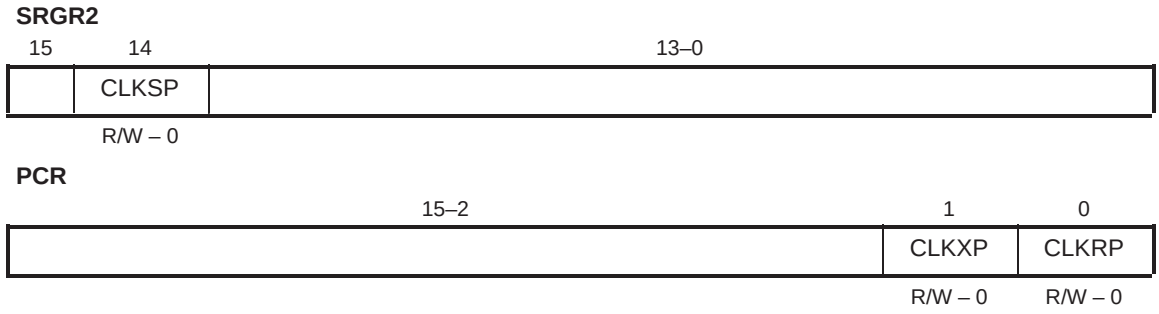
Register	Bit	Name	Function	
PCR SRGR2	7 13	SCLKME CLKSM	Sample Rate Generator Clock Mode	
			SCLKME = 0 CLKSM = 0	Sample rate generator clock derived from CLKS pin
			SCLKME = 0 CLKSM = 1	Sample rate generator clock derived from CPU clock (This is the condition forced by a DSP reset.)
			SCLKME = 1 CLKSM = 0	Sample rate generator clock derived from CLKR pin
			SCLKME = 1 CLKSM = 1	Sample rate generator clock derived from CLKX pin

9.9.23.1 About the SRG Clock Mode

The sample rate generator can produce a clock signal (CLKG) for use by the receiver, the transmitter, or both, but CLKG is derived from an input clock. The preceding table shows the four possible sources of the input clock. For more details on generating CLKG, see *Clock Generation in the Sample Rate Generator* on page 9-24.

9.9.24 Set the SRG Input Clock Polarity

Figure 9–110. Register Bits Used to Set the SRG Input Clock Polarity



Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–72. Register Bits Used to Set the SRG Input Clock Polarity

Register	Bit	Name	Function
SRGR2	14	CLKSP	CLKS Pin Polarity CLKSP determines the input clock polarity when the CLKS pin supplies the input clock (SCLKME = 0 and CLKSM = 0). CLKSP = 0 Rising edge on CLKS pin generates CLKG and FSG. CLKSP = 1 Falling edge on CLKS pin generates CLKG and FSG.
PCR	1	CLKXP	CLKX Pin Polarity CLKXP determines the input clock polarity when the CLKX pin supplies the input clock (SCLKME = 1 and CLKSM = 1). CLKXP = 0 Rising edge on CLKX pin generates transitions on CLKG and FSG. CLKXP = 1 Falling edge on CLKX pin generates transitions on CLKG and FSG.
PCR	0	CLKRP	CLKR Pin Polarity CLKRP determines the input clock polarity when the CLKR pin supplies the input clock (SCLKME = 1 and CLKSM = 0). CLKRP = 0 Falling edge on CLKR pin generates transitions on CLKG and FSG. CLKRP = 1 Rising edge on CLKR pin generates transitions on CLKG and FSG.

9.9.24.1 Using CLKSP/CLKXP/CLKRP to Choose an Input Clock Polarity

The sample rate generator can produce a clock signal (CLKG) and a frame-sync signal (FSG) for use by the receiver, the transmitter, or both. To produce CLKG and FSG, the sample rate generator must be driven by an input clock signal derived from the CPU clock or from an external clock on the CLKS, CLKX, or CLKR pin. If you use a pin, choose a polarity for that pin by using the appropriate polarity bit (CLKSP for the CLKS pin, CLKXP for the CLKX pin, CLKRP for the CLKR pin). The polarity determines whether the rising or falling edge of the input clock generates transitions on CLKG and FSG.

9.10 General-Purpose I/O on the McBSP Pins

Table 9–73 summarizes how to use the McBSP pins as general-purpose I/O pins. All of the bits mentioned in the table except XRST_ and RRST_ are in the pin control register (PCR, described in section 9.13.8 on page 9-195). XRST_ and RRST_ are in the serial port control registers (SPCRs, described in section 9.13.3 on page 9-158).

To use **receiver pins CLKR, FSR, and DR** as general purpose I/O pins rather than as serial port pins, you must set two conditions:

- ❏ The receiver of the serial port is in reset (RRST_ = 0 in SPCR1).
- ❏ General-purpose I/O is enabled for the serial port receiver (RIOEN = 1 in PCR).

The CLKR and FSR pins can be individually configured as either input or output pins with the CLKRM and FSRM bits, respectively. The DR pin can only be an input pin. The following table shows which bits in PCR are used to read from/write to these pins.

For the **transmitter pins CLKX, FSX, and DX**, you must meet two similar conditions:

- ❏ The transmitter of the serial port is in reset (XRST_ = 0 in SPCR2).
- ❏ General-purpose I/O is enabled for the serial port transmitter (XIOEN = 1 in PCR).

The CLKX and FSX pins can be individually configured as input or output pins with the CLKXM and FSXM bits, respectively. The DX pin can only be an output pin. The following table shows which bits in PCR are used to read from/write to these pins.

For the **CLKS pin**, all of the reset and I/O enable conditions must be met:

- ❏ Both the receiver and transmitter of the serial port are in reset (RRST_ = 0 and XRST_ = 0).
- ❏ General-purpose I/O is enabled for both the receiver and the transmitter (RIOEN = 1 and XIOEN = 1).

The CLKS pin can only be an input pin. To read the status of the signal on the CLKS pin, read the CLKS_STAT bit in PCR.

Table 9–73. How Use McBSP Pins for General-Purpose Input/Output

Pin	General Purpose Use Enabled by This Bit Combination	Selected as Output When …	Output Value Driven From This Bit	Selected As Input When …	Input Value Read From This Bit
CLKX	XRST_ = 0 XIOEN = 1	CLKXM = 1	CLKXP	CLKXM = 0	CLKXP
FSX	XRST_ = 0 XIOEN = 1	FSXM = 1	FSXP	FSXM = 0	FSXP
DX	XRST_ = 0 XIOEN = 1	Always	DX_STAT	Never	Does not apply
CLKR	RRST_ = 0 RIOEN = 1	CLKRM = 1	CLKRP	CLKRM = 0	CLKRP
FSR	RRST_ = 0 RIOEN = 1	FSRM = 1	FSRP	FSRM = 0	FSRP
DR	RRST_ = 0 RIOEN = 1	Never	Does not apply	Always	DR_STAT
CLKS	RRST_ = XRST_ = 0 RIOEN = XIOEN = 1	Never	Does not apply	Always	CLKS_STAT

9.11 Emulation, Power, and Reset Considerations

This section covers the following topics:

- How to program the McBSP's response to a breakpoint in the high-level language debugger (section 9.11.1)
- How to conserve power in the DSP by placing the McBSP into its idle mode (section 9.11.2)
- How to reset and initialize the various parts of the McBSP (section 9.11.3)

9.11.1 McBSP Emulation Mode

FREE and SOFT are special emulation bits in SPCR2 that determine the state of the McBSP when a breakpoint is encountered in the high-level language debugger. If FREE = 1, upon a software breakpoint the clock continues to run and data is still shifted out. When FREE = 1, the SOFT bit is a *don't care.*

If FREE = 0, the SOFT bit takes effect: If SOFT = 0 when breakpoint occurs, the clock stops immediately, thus aborting a transmission. If SOFT = 1 and a breakpoint occurs while transmission is in progress, the transmission continues until completion of the transfer, and then the clock halts. These options are listed in the following table.

The McBSP receiver functions in a similar fashion. Note that if a mode other than the immediate stop mode (SOFT = FREE = 0) is chosen, the receiver continues running and an overrun error is possible.

Table 9–74. McBSP Emulation Modes Selectable with the FREE and SOFT Bits of SPCR2

FREE	SOFT	McBSP Emulation Mode
0	0	Immediate stop mode (reset condition) The transmitter or receiver stops immediately in response to a breakpoint.
0	1	Soft stop mode When a breakpoint occurs, the transmitter stops after completion of the current word. The receiver is not affected.
1	0 or 1	Free run mode The transmitter and receiver continue to run when a breakpoint occurs.

9.11.2 Reducing Power Consumed by a McBSP

The McBSP is placed into its idle mode with reduced power consumption when the PERIPH idle domain is idle (PERIS = 1 in ISTR) and the McBSP idle enable bit is set (IDLE_EN = 1 in PCR). PERIPH and the other idle domains are described in section 8.1 on page 8-2.

In the McBSP idle mode:

- If the McBSP is configured to operate with internally generated clocking and frame synchronization, it will be completely stopped.
- If the McBSP is configured to operate with externally generated clocking and frame synchronization (either directly or through the sample rate generator), the external interface portion of the McBSP continues to function during external clock activity periods. The McBSP sends a request to activate the PERIPH and DMA idle domains when it needs to be serviced. If the domains were idle, they are made idle again after the McBSP has been serviced.

When IDLE_EN = 0 in PCR, the McBSP keeps running, regardless of whether the PERIPH domain is idle.

9.11.3 Resetting and Initializing a McBSP

9.11.3.1 McBSP Pin States: DSP Reset Versus Receiver/Transmitter Reset

Table 9–75 shows the state of McBSP pins when the serial port is reset due to a DSP reset and due to a direct receiver or transmitter reset.

Table 9–75. Reset State of Each McBSP Pin

Pin	Possible State(s)	State Forced By DSP Reset	State Forced By Receiver/Transmitter Reset
			Receiver Reset (RRST_ = 0 and GRST_ = 1)
DR	I	Input	Input
CLKR	I/O/Z	Input	Known state if Input; CLKR running if output
FSR	I/O/Z	Input	Known state if Input; FSRP inactive state if output
CLKS	I/O/Z	Input	Input
			Transmitter Reset (XRST_ = 0 and GRST_ = 1)
DX	O/Z	High impedance	High impedance
CLKX	I/O/Z	Input	Known state if Input; CLKX running if output
FSX	I/O/Z	Input	Known state if Input; FSXP inactive state if output
CLKS	I	Input	Input

Note: In Possible State(s) column, I = Input, O = Output, Z = High impedance

9.11.3.2 DSP Reset, McBSP Reset, and Sample Rate Generator Reset

When the McBSP is reset in either of the above two ways, the machine is reset to its initial state, including reset of all counters and status bits. The receive status bits include RFULL, RRDY, and RSYNCERR. The transmit status bits include XEMPTY_, XRDY, and XSYNCERR.

- **DSP reset.** When the whole DSP is reset (RESET_ signal is driven low), the entire serial port, including the transmitter, receiver, and the sample rate generator, is reset. All input-only pins and three-state pins should be in a known state. The output-only pin DX is in the high-impedance state.

  The DSP reset forces the sample rate generator clock, CLKG, to have half the frequency of the CPU clock. No pulses are generated on the sample rate generator's frame-sync signal, FSG.

  When the device is pulled out of reset, the serial port remains in the reset state. In this state the DR and DX pins may be used as general-purpose I/O pins as described in section 9.10 on page 9-144.

- **McBSP reset.** When the receiver and transmitter reset bits, RRST_ and XRST_, are loaded with 0s, the respective portions of the McBSP are reset, and activity in the corresponding section of the serial port stops. All input-only pins, such as DR and CLKS, and all other pins that are configured as inputs, are in a known state. The FSR and FSX pins are driven to their inactive state if they are not outputs. If the CLKR and CLKX pins are programmed as outputs, they will be driven by CLKG, provided that GRST_ = 1. Lastly, the DX pin will be in the high-impedance state when the transmitter and/or the device is reset.

  During normal operation, the sample rate generator is reset if the GRST_ bit is cleared. GRST_ should be 0 only when neither the transmitter nor the receiver is using the sample rate generator. In this case, the internal sample rate generator clock (CLKG) and its frame-sync signal (FSG) are driven inactive low.

  When the sample rate generator is not in the reset state (GRST_ = 1), pins FSR and FSX are in an inactive state when RRST_ = 0 and XRST_ = 0, respectively, even if they are outputs driven by FSG. This ensures that when only one portion of the McBSP is in reset, the other portion can continue operation when FRST_ = 1 and its frame synchronization is driven by FSG.

- **Sample rate generator reset.** The sample rate generator is reset when the DSP is reset or when GRST_ is loaded with 0. In the case of a DSP reset, the sample rate generator clock, CLKG, is driven by the CPU clock divided by 2, and the frame-sync signal, FSG, is driven inactive low.

  When neither the transmitter nor the receiver is fed by CLKG and FSG, you can reset the sample rate generator by clearing GRST_. In this case, CLKG and FSG are driven inactive low. If you then set GRST_, CLKG starts and runs as programmed. Later, if FRST_ = 1, FSG pulses active high after the programmed number of CLKG cycles has elapsed.

9.11.3.3 McBSP Initialization Procedure

The serial port initialization procedure is as follows:

1) Make XRST_ = RRST_ = FRST_ = 0 in SPCR[1,2]. If coming out of a DSP reset, this step is not required.

2) While the serial port is in the reset state, program only the McBSP configuration registers (not the data registers) as required.

3) Wait for two clock cycles. This ensures proper internal synchronization.

4) Set up data acquisition as required (such as writing to DXR[1,2]).

5) Make XRST_ = RRST_ = 1 to enable the serial port. Make sure that as you set these reset bits, you do not modify any of the other bits in SPCR1 and SPCR2. Otherwise, you will change the configuration you selected in step 2.

6) Set FRST_ = 1, if internally generated frame synchronization is required.

7) Wait two clock cycles for the receiver and transmitter to become active.

Alternatively, on either write (step 1 or 5), the transmitter and receiver may be placed in or taken out of reset individually by modifying the desired bit.

The above procedure for reset/initialization can be applied in general when the receiver or transmitter has to be reset during its normal operation, and also when the sample rate generator is not used for either operation.

Notes:

1) The necessary duration of the active-low period of XRST_ or RRST_ is at least two CLKR/CLKX cycles.

2) The appropriate bits in serial port configuration registers SPCR[1,2], PCR, RCR[1,2], XCR[1,2], and SRGR[1,2] should only be modified when the affected portion of the serial port is in its reset state.

3) In most cases, the data transmit registers (DXR[1,2]) should be loaded by the CPU or by the DMA controller only when the transmitter is enabled (XRST_ = 1). An exception to this rule is when these registers are used for companding internal data (see section 9.2.2.2 on page 9-10).

4) The bits of the channel control registers—MCR[1,2], RCER[A–H], XCER[A–H]—can be modified at any time as long as they are not being used by the current reception/transmission in a multichannel selection mode.

9.11.3.4 Example: Resetting the Transmitter While the Receiver is Running

The following example shows values in the control registers that reset and configure the transmitter while the receiver is running.

Example 9–1. Resetting and Configuring the McBSP Transmitter While the McBSP Receiver is Running

```
SPCR1 = 0001h      ; The receiver is running with the receive
SPCR2 = 0030h      ; interrupt (RINT) triggered by the
                   ; receiver ready bit (RRDY). The
                   ; transmitter is in its reset state. The
                   ; transmit interrupt (XINT) will be
                   ; triggered by the transmit frame–sync
                   ; error bit (XSYNCERR).

PCR = 0900h        ; Transmit frame synchronization is
                   ; generated internally according to the
                   ; FSGM bit of SRGR2. The transmit clock
                   ; is driven by an external source. The
                   ; receive clock continues to be driven by
                   ; sample rate generator. The input clock
                   ; of the sample rate generator is supplied
                   ; by the CLKS pin or by the CPU clock
                   ; depending on the CLKSM bit of SRGR2.

SRGR1 = 0001h      ; The CPU clock is the input clock for
SRGR2 = 2000h      ; the sample rate generator. The sample
                   ; rate generator divides the CPU clock by
                   ; 2 to generate its output clock (CLKG).
                   ; Transmit frame synchronization is tied
                   ; to the automatic copying of data from
                   ; the DXR(s) to the XSR(s).

XCR1 = 0740h       ; The transmit frame has two phases.
XCR2 = 8321h       ; Phase 1 has eight 16–bit words. Phase 2
                   ; has four 12–bit words. There is 1–bit
                   ; data delay between the start of a
                   ; frame–sync pulse and the first data bit
                   ; transmitted.

SPCR2 = 0x0031     ; The transmitter is taken out of reset.
```

9.12 Data Packing Examples

This section shows two ways you can implement data packing in the McBSP.

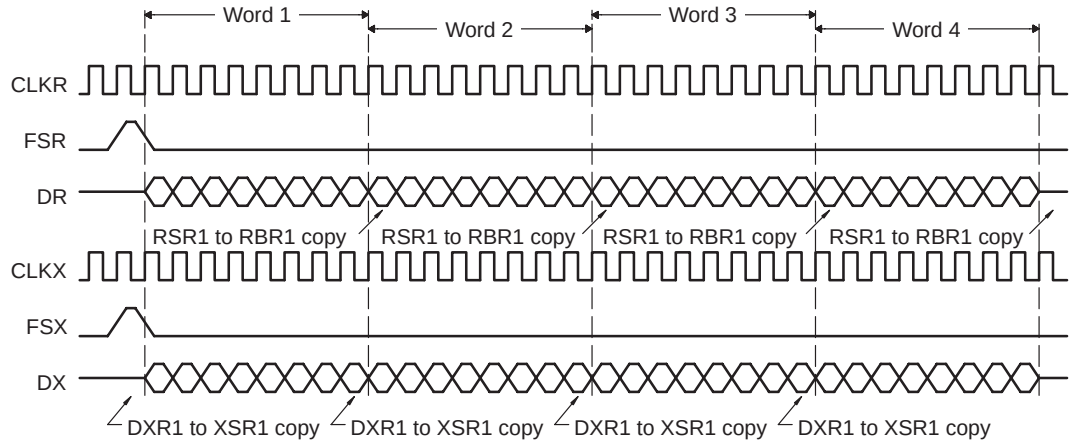
9.12.1 Data Packing Using Frame Length and Word Length

The frame length and word length can be manipulated to effectively pack data. For example, consider a situation where four 8-bit words are transferred in a single-phase frame as shown in Figure 9–111. In this case:

- (R/X)PHASE = 0: Single-phase frame
- (R/X)FRLEN1 = 0000011b: 4-word frame
- (R/X)WDLEN1 = 000b: 8-bit words

Four 8-bit data words are transferred to and from the McBSP by the CPU or by the DMA controller. Thus, four reads from DRR1 and four writes to DXR1 are necessary for each frame.

Figure 9–111. Four 8-Bit Data Words Transferred To/From the McBSP



This data can also be treated as a single-phase frame consisting of one 32-bit data word, as shown in Figure 9–112. In this case:

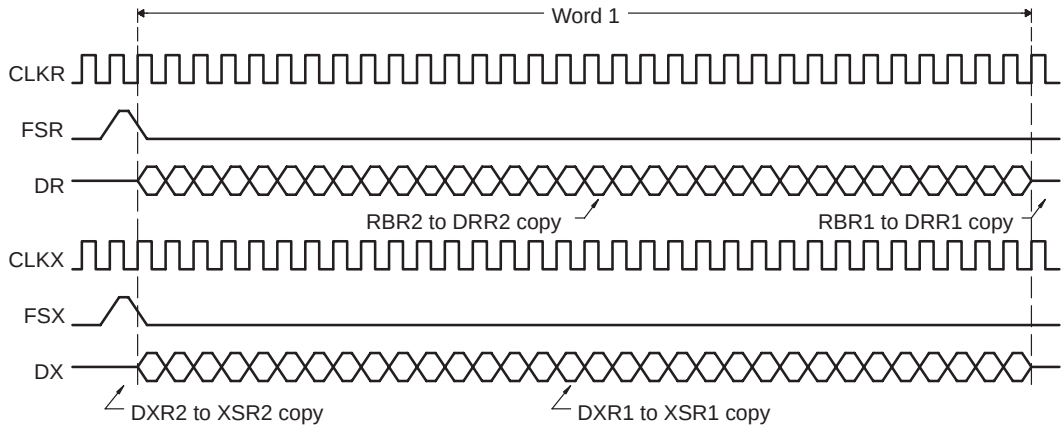
- (R/X)PHASE = 0: Single-phase frame
- (R/X)FRLEN1 = 0000000b: 1-word frame
- (R/X)WDLEN1 = 101b: 32-bit word

Two 16-bit data words are transferred to and from the McBSP by the CPU or DMA controller. Thus, two reads, from DRR2 and DRR1, and two writes, to DXR2 and DXR1, are necessary for each frame. This results in only half the number of transfers compared to the previous case. This manipulation reduces the percentage of bus time required for serial port data movement.

Note:

When the word length is larger than 16 bits, make sure you access DRR2/DXR2 before you access DRR1/DXR1. McBSP activity is tied to accesses of DRR1/DXR1. During the reception of 24-bit or 32-bit words, read DRR2 and then read DRR1. Otherwise, the next RBR[1,2]-to-DRR[1,2] copy occurs before DRR2 is read. Similarly, during the transmission of 24-bit or 32-bit words, write to DXR2 and then write to DXR1. Otherwise, the next DXR[1,2]-to-XSR[1,2] copy occurs before DXR2 is loaded with new data.

Figure 9–112. One 32-Bit Data Word Transferred To/From the McBSP



9.12.2 Data Packing Using Word Length and the Frame-Sync Ignore Function

When there are multiple words per frame, you can implement data packing by increasing the word length (defining a serial word with more bits) and by ignoring frame-sync pulses. First, consider Figure 9–113, which shows the McBSP operating at the maximum packet frequency. Here, each frame only has a single 8-bit word. Note the frame-sync pulse that initiates each frame transfer for reception and for transmission. For reception, this configuration requires one read operation for each word. For transmission, this configuration requires one write operation for each word.

Figure 9–113. 8-Bit Data Words Transferred at Maximum Packet Frequency

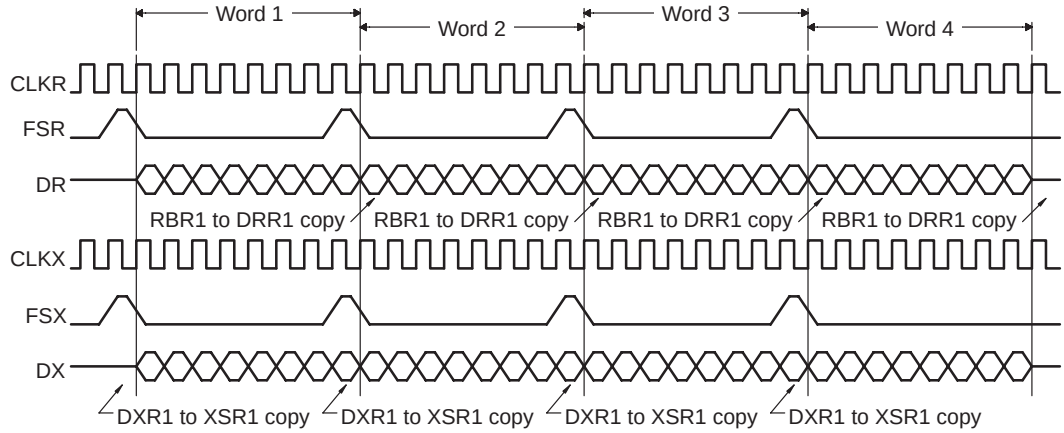
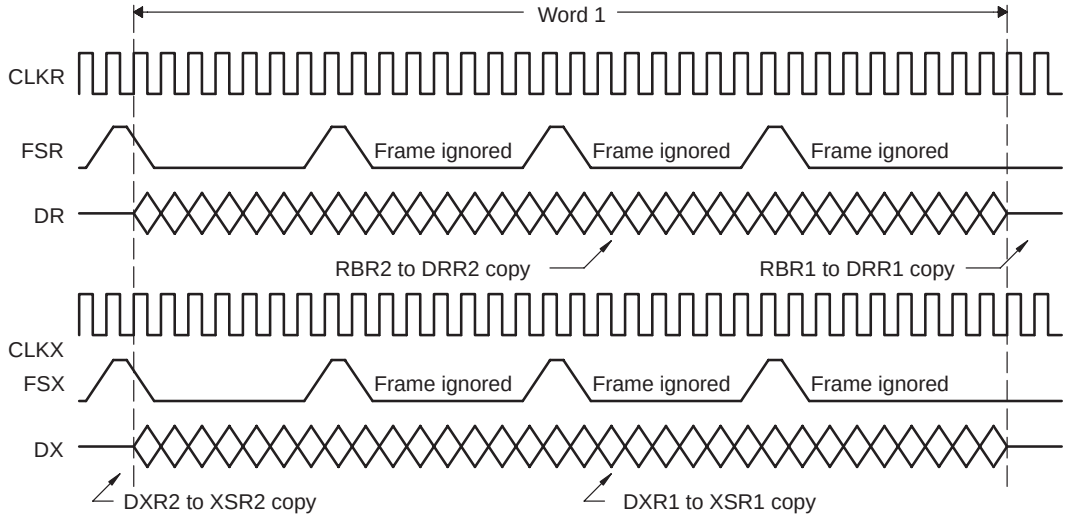


Figure 9–114 shows the McBSP configured to treat this data stream as a continuous 32-bit word. In this example, the McBSP responds to an initial frame-sync pulse. However, (R/X)FIG = 1 so that the McBSP ignores subsequent pulses. Only two read transfers or two write transfers are needed every 32 bits. This configuration effectively reduces the required bus bandwidth to half the bandwidth needed to transfer four 8-bit words.

Figure 9–114. Configuring the Data Stream of Figure 9–113 as a Continuous 32-Bit Word



9.13 McBSP Registers

For each McBSP, the DSP contains the following registers. For the I/O address of each register, see the data sheet for your TMS320C55x DSP.

Table 9–76. McBSP Registers

Register	Description	For Details, See ...
DRR1 DRR2	Data receive register 1 Data receive register 2 (one pair for each McBSP)	Page 9-156
DXR1 DXR2	Data transmit register 1 Data transmit register 2 (one pair for each McBSP)	Page 9-157
SPCR1 SPCR2	Serial port control register 1 Serial port control register 2 (one pair for each McBSP)	Page 9-158
RCR1 RCR2	Receive control register 1 Receive control register 2 (one pair for each McBSP)	Page 9-169
XCR1 XCR2	Transmit control register 1 Transmit control register 2 (one pair for each McBSP)	Page 9-175
SRGR1 SRGR2	Sample rate generator register 1 Sample rate generator register 2 (one pair for each McBSP)	Page 9-181
MCR1 MCR2	Multichannel control register 1 Multichannel control register 2 (one pair for each McBSP)	Page 9-185
PCR	Pin control register (one PCR for each McBSP)	Page 9-195

Table 9–76. McBSP Registers (Continued)

Register	Description	For Details, See ...
RCERA RCERB RCERC RCERD RCERE RCERF RCERG RCERH	Receive channel enable register for partition A Receive channel enable register for partition B Receive channel enable register for partition C Receive channel enable register for partition D Receive channel enable register for partition E Receive channel enable register for partition F Receive channel enable register for partition G Receive channel enable register for partition H (one set for each McBSP)	Page 9-202
XCERA XCERB XCERC XCERD XCERE XCERF XCERG XCERH	Transmit channel enable register for partition A Transmit channel enable register for partition B Transmit channel enable register for partition C Transmit channel enable register for partition D Transmit channel enable register for partition E Transmit channel enable register for partition F Transmit channel enable register for partition G Transmit channel enable register for partition H (one set for each McBSP)	Page 9-207

9.13.1 Data Receive Registers (DRR2 and DRR1)

The CPU or the DMA controller reads received data from one or both of the data receive registers. If the serial word length is 16 bits or smaller only DRR1 is used. If the serial length is larger than 16 bits, both DRR1 and DRR2 are used, and DRR2 holds the most significant bits. Each frame of receive data in the McBSP can have one phase or two phases, each with its own serial word length.

DRR1 and DRR2 are I/O mapped registers; they are accessible at addresses in I/O space.

Figure 9–115. Data Receive Registers (DRR2 and DRR1)

	15–0
DRR2	High part of receive data (for 20-, 24- or 32-bit data)
DRR1	Receive data (for 8-, 12-, or 16-bit data) or Low part of receive data (for 20-, 24- or 32-bit data)

9.13.1.1 How Data Travels From the Data Receive (DR) Pin to the DRRs

If the serial word length is 16 bits or smaller, receive data on the DR pin is shifted into receive shift register 1 (RSR1) and then copied into receive buffer register 1 (RBR1). The content of RBR1 is then copied to DRR1, which can be read by the CPU or by the DMA controller.

If the serial word length is larger than 16 bits, receive data on the DR pin is shifted into both of the receive shift registers (RSR2, RSR1) and then copied into both of the receive buffer registers (RBR2, RBR1). The content of the RBRs is then copied into both of the DRRs, which can be read by the CPU or by the DMA controller.

If companding is used during the copy from RBR1 to DRR1 (RCOMPAND = 10b or 11b), the 8-bit compressed data in RBR1 is expanded to a left-justified 16-bit value in DRR1. If companding is disabled, the data copied from RBR[1,2] to DRR[1,2] is justified and bit filled according to the RJUST bits.

The RSRs and RBRs are not accessible. They are not mapped to I/O space like the DRRs.

9.13.2 Data Transmit Registers (DXR2 and DXR1)

For transmission, the CPU or the DMA controller writes data to one or both of the data transmit registers. If the serial word length is 16 bits or smaller, only DXR1 is used. If the word length is larger than 16 bits, both DXR1 and DXR2 are used, and DXR2 holds the most significant bits. Each frame of transmit data in the McBSP can have one phase or two phases, each with its own serial word length.

DXR1 and DXR2 are I/O mapped registers; they are accessible at addresses in I/O space.

Figure 9–116. Data Transmit Registers (DXR2 and DXR1)

	15–0
DXR2	High part of transmit data (for 20-, 24- or 32-bit data)
DXR1	Transmit data (for 8-, 12-, or 16-bit data) or Low part of transmit data (for 20-, 24- or 32-bit data)

9.13.2.1 How Data Travels From the DXRs to the Data Transmit (DX) Pin

If the serial word length is 16 bits or fewer, data written to DXR1 is copied to transmit shift register 1 (XSR1). From XSR1, the data is shifted onto the DX pin one bit at a time.

If the serial word length is more than 16 bits, data written to DXR1 and DXR2 is copied to both transmit shift registers (XSR2, XSR1). From the XSRs, the data is shifted onto the DX pin one bit at a time.

If companding is used during the transfer from DXR1 to XSR1 (XCOMPAND = 10b or 11b), the McBSP compresses the 16-bit data in DXR1 to 8-bit data in the μ-law or A-law format in XSR1. If companding is disabled, the McBSP passes data from the DXR(s) to the XSR(s) without modification.

The XSRs are not accessible. They are not mapped to I/O space like the DXRs.

9.13.3 Serial Port Control Registers (SPCR2 and SPCR1)

Each McBSP has two serial port control registers of the form shown in Figure 9–117. and describe the bits in SPCR1 and SPCR2, respectively. These I/O-mapped registers enable you to:

- ❏ Control various McBSP modes: digital loopback mode (DLB), sign-extension and justification mode for reception (RJUST), clock stop mode (CLKSTP), A-bis mode (ABIS), interrupt modes (RINTM and XINTM), emulation mode (FREE and SOFT)
- ❏ Turn on and off the DX-pin delay enabler (DXENA)
- ❏ Check the status of receive and transmit operations (RSYNCERR, XSYNCERR, RFULL, XEMPTY_, RRDY, XRDY)
- ❏ Reset portions of the McBSP (RRST_, XRST_, FRST_, GRST_)

Figure 9–117. Serial Port Control Registers (SPCR2 and SPCR1)

SPCR1

15	14–13	12–11	10–8
DLB	RJUST	CLKSTP	Rsvd
R/W – 0	R/W – 00	R/W – 00	

7	6	5–4	3	2	1	0
DXENA	ABIS	RINTM	RSYNCERR	RFULL	RRDY	RRST_
R/W – 0	R/W – 0	R/W – 00	R/W – 0	R – 0	R – 0	R/W – 0

SPCR2

15–10	9	8
Rsvd	FREE	SOFT
	R/W – 0	R/W – 0

7	6	5–4	3	2	1	0
FRST_	GRST_	XINTM	XSYNCERR	XEMPTY_	XRDY	XRST_
R/W – 0	R/W – 0	R/W – 00	R/W – 0	R – 0	R – 0	R/W – 0

Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset.

Table 9–77. SPCR1 Bit Descriptions

Bits	Name	Description	Reset Value
15	DLB	Digital loopback mode bit	0
		DLB disables or enables the digital loopback mode of the McBSP:	
		0 Disabled Internal DR is supplied by the DR pin. Internal FSR and internal CLKR can be supplied by their respective pins or by the sample rate generator, depending on the mode bits FSRM and CLKRM.	
		1 Enabled Internal receive signals are supplied by internal transmit signals: DR connected to DX FSR connected to FSX CLKR connected to CLKX Internal DX is supplied by the DX pin. Internal FSX and internal CLKX are supplied by their respective pins or are generated internally, depending on the mode bits FSXM and CLKXM. This mode allows you to test serial port code with a single DSP. The McBSP transmitter directly supplies data, frame synchronization, and clocking to the McBSP receiver.	
14–13	RJUST	Receive sign-extension and justification mode bits	00b
		During reception, RJUST determines how data is justified and bit filled before being passed to the data receive registers (DRR1, DRR2):	
		00b Right justify the data and zero fill the MSBs.	
		01b Right justify the data and sign-extend the data into the MSBs.	
		10b Left justify the data and zero fill the LSBs.	
		11b Reserved (do not use)	
		Note: RJUST is ignored if you enable a companding mode with the RCOMPAND bits. In a companding mode, the 8-bit compressed data in RBR1 is expanded to left-justified 16-bit data in DRR1.	
		For more details about the effects of RJUST, see *Set the Receive Sign-Extension and Justification Mode* on page 9-91.	

Table 9–77. SPCR1 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
12–11	CLKSTP	Clock stop mode bits CLKSTP allows you to use the clock stop mode to support the SPI master-slave protocol. If you will not be using the SPI protocol, you can clear CLKSTP to disable the clock stop mode. 00b/01b Clock stop mode is disabled. 10b Clock stop mode, without clock delay 11b Clock stop mode, with half-cycle clock delay In the clock stop mode, the clock stops at the end of each data transfer. At the beginning of each data transfer, the clock starts immediately (CLKSTP = 10b) or after a half-cycle delay (CLKSTP = 11b). For more details, see *Enable/Disable the Clock Stop Mode* on page 9-78.	00b
10–8	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.	
7	DXENA	DX delay enabler mode bit DXENA controls the delay enabler for the DX pin. The enabler creates an extra delay for turn-on time (for the length of the delay, see the data sheet for your TMS320C55x DSP). For more details about the effects of DXENA, see *Set the Transmit DXENA Mode* on page 9-128. 0 DX delay enabler off 1 DX delay enabler on	0
6	ABIS	A-bis mode bit ABIS enables or disables the A-bis mode of the McBSP: 0 Disabled 1 Enabled	0

Table 9–77. SPCR1 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
5–4	RINTM	Receive interrupt mode bits RINTM determines which event in the McBSP receiver generates a receive interrupt (RINT) request. If RINT is properly enabled inside the CPU, the CPU will service the interrupt request; otherwise, the CPU will ignore the request. 00b **If ABIS = 0 (A-bis mode disabled):** The McBSP sends a receive interrupt (RINT) request to the CPU when the RRDY bit changes from 0 to 1, indicating that receive data is ready to be read (the content of RBR[1,2] has been copied to DRR[1,2]): **Note**: Regardless of the value of RINTM, you can check RRDY to determine whether a word transfer is complete. **If ABIS = 1 (A-bis mode enabled):** The McBSP sends a RINT request to the CPU when 16 enabled bits have been received on the DR pin. 01b In the multichannel selection mode, the McBSP sends a RINT request to the CPU after every 16-channel block is received in a frame. Outside of the multichannel selection mode, no interrupt request is sent. 10b The McBSP sends a RINT request to the CPU when each receive frame-sync pulse is detected. The interrupt request is sent even if the receiver is in its reset state. 11b The McBSP sends a RINT request to the CPU when the RSYNCERR bit is set, indicating a receive frame-sync error. Note: Regardless of the value of RINTM, you can check RSYNCERR to determine whether a receive frame-sync error occurred.	00b

Table 9–77. SPCR1 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
3	RSYNCERR	Receive frame-sync error bit RSYNCERR is set when a receive frame-sync error is detected by the McBSP. If RINTM = 11b, the McBSP sends a receive interrupt (RINT) request to the CPU when RSYNCERR is set. The flag remains set until you write a 0 to it or reset the receiver. **Caution:** If RINTM = 11b, writing a 1 to RSYNCERR triggers a receive interrupt just as if a receive frame-sync error occurred. 0 No error 1 Receive frame-sync error For more details about this error, see *Unexpected Receive Frame-Sync Pulse* on page 9-38.	0
2	RFULL	Receiver full bit RFULL is set when the receiver is full with new data and the previously received data has not been read (receiver-full condition). For more details about this condition, see *Overrun in the Receiver* on page 9-37. 0 No receiver-full condition 1 Receiver-full condition: RSR[1,2] and RBR[1,2] are full with new data, but the previous data in DRR[1,2] has not been read.	0
1	RRDY	Receiver ready bit RRDY is set when data is ready to be read from DRR[1,2]. Specifically, RRDY is set in response to a copy from RBR1 to DRR1. 0 Receiver not ready When the content of DRR1 is read, RRDY is automatically cleared. 1 Receiver ready: New data can be read from DRR[1,2]. Important: If both DRRs are need (word length larger than 16 bits), the CPU or the DMA controller must read from DRR2 first and then from DRR1. As soon as DRR1 is read, the next RBR-to-DRR copy is initiated. If DRR2 is not read first, the data in DRR2 is lost. If the receive interrupt mode is RINTM = 00b, the McBSP sends a receive interrupt request to the CPU when RRDY changes from 0 to 1. Also, when RRDY changes from 0 to 1, the McBSP sends a receive synchronization event (REVT) signal to the DMA controller.	0

Table 9–77. SPCR1 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
0	RRST_	Receiver reset bit You can use RRST_ to take the McBSP receiver into and out of its reset state. The underscore (_) at the end of the bit name is a reminder of the negative polarity of the bit; RRST_ = 0 indicates the reset state. 0 If you read a 0, the receiver is in its reset state. If you write a 0, you reset the receiver. 1 If you read a 1, the receiver is enabled. If you write a 1, you enable the receiver by taking it out of its reset state. To read about the effects of a receiver reset, see *Resetting and Initializing a McBSP* on page 9-147.	0

Table 9–78. SPCR2 Bit Descriptions

Bits	Name	Description	Reset Value
15–10	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.	–
9	FREE	McBSP emulation mode bit	0
8	SOFT	McBSP emulation mode bit FREE and SOFT determine the response (if any) of the McBSP transmit and receive clocks when a breakpoint is encountered in the high-level language debugger. If FREE = 1, neither the transmit nor receive clock stops in response to a software breakpoint, regardless of the value of SOFT. If FREE = 0, SOFT determines how the clocks respond. When one of the clocks stops, the corresponding data transfer (transmission or reception) stops. **FREE** **SOFT** **When A Breakpoint Is Encountered** … 0 0 The McBSP transmit and receive clocks are stopped immediately. (Reset condition) 0 1 The McBSP transmit clock stops after completion of the current serial word transfer. The McBSP receive clock is not affected. 1 0 or 1 The McBSP transmit and receive clocks continue to run	0

Table 9–78. SPCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
7	FRST_	Frame-sync logic reset bit The sample rate generator of the McBSP includes frame-sync logic to generate an internal frame-sync signal. You can use FRST_ to take the frame-sync logic into and out of its reset state. The underscore (_) at the end of the bit name is a reminder of the negative polarity of the bit; FRST_ = 0 indicates the reset state. 0 If you read a 0, the frame-sync logic is in its reset state. If you write a 0, you reset the frame-sync logic. In the reset state, the frame-sync logic does not generate a frame-sync signal (FSG). 1 If you read a 1, the frame-sync logic is enabled. If you write a 1, you enable the frame-sync logic by taking it out of its reset state. When the frame-sync logic is enabled (FRST_ = 1) and the sample rate generator as a whole is enabled (GRST_ = 1), the frame-sync logic generates the frame-sync signal FSG as programmed.	0
6	GRST_	Sample rate generator reset bit You can use GRST_ to take the McBSP sample rate generator into and out of its reset state. The underscore (_) at the end of the bit name is a reminder of the negative polarity of the bit; GRST_ = 0 indicates the reset state. 0 If you read a 0, the sample rate generator is in its reset state. If you write a 0, you reset the sample rate generator. If GRST_ = 0 due to a DSP reset, CLKG is driven by the CPU clock divided by 2, and FSG is driven low (inactive). If GRST_ = 0 due to program code, CLKG and FSG are both driven low (inactive). 1 If you read a 1, the sample rate generator is enabled. If you write a 1, you enable the sample rate generator by taking it out of its reset state. When enabled, the sample rate generator generates the clock signal CLKG as programmed in the sample rate generator registers. If FRST_ = 1, the generator also generates the frame-sync signal FSG as programmed in the sample rate generator registers. To read about the effects of a sample rate generator reset, see *Resetting and Initializing a McBSP* on page 9-147.	0

Table 9–78. SPCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
5–4	XINTM	Transmit interrupt mode bits XINTM determines which event in the McBSP transmitter generates a transmit interrupt (XINT) request. If XINT is properly enabled, the CPU will service the interrupt request; otherwise, the CPU will ignore the request. 00b **If ABIS = 0 (A-bis mode disabled):** The McBSP sends a transmit interrupt (XINT) request to the CPU when the XRDY bit changes from 0 to 1, indicating that transmitter is ready to accept new data (the content of DXR[1,2] has been copied to XSR[1,2]): **Note:** Regardless of the value of XINTM, you can check XRDY to determine whether a word transfer is complete. **If ABIS = 1 (A-bis mode enabled):** The McBSP sends an XINT request to the CPU when 16 enabled bits have been transmitted on the DX pin. 01b In the multichannel selection mode, the McBSP sends an XINT request to the CPU after every 16-channel block is transmitted in a frame. Outside of the multichannel selection mode, no interrupt request is sent. 10b The McBSP sends an XINT request to the CPU when each transmit frame-sync pulse is detected. The interrupt request is sent even if the transmitter is in its reset state. 11b The McBSP sends an XINT request to the CPU when the XSYNCERR bit is set, indicating a transmit frame-sync error. Note: Regardless of the value of XINTM, you can check XSYNCERR to determine whether a transmit frame-sync error occurred.	00b

Table 9–78. SPCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
3	XSYNCERR	Transmit frame-sync error bit XSYNCERR is set when a transmit frame-sync error is detected by the McBSP. If XINTM = 11b, the McBSP sends a transmit interrupt (XINT) request to the CPU when XSYNCERR is set. The flag remains set until you write a 0 to it or reset the transmitter. **Caution:** if XINTM = 11b, writing a 1 to XSYNCERR triggers a transmit interrupt just as if a transmit frame-sync error occurred. 0 No error 1 Transmit frame-sync error For details about this error see *Unexpected Transmit Frame-Sync Pulse* on page 9-44.	0
2	XEMPTY_	Transmitter empty bit XEMPTY_ is cleared when the transmitter is ready to send new data but no new data is available (transmitter-empty condition). The underscore (_) at the end of the bit name is a reminder of the negative polarity of the bit. A transmitter-empty condition is indicated by XEMPTY_ = 0. 0 Transmitter-empty condition Typically this indicates that all the bits of the current word have been transmitted but there is no new data in DXR1. XEMPTY_ is also cleared if the transmitter is reset and then restarted. For more details about this error condition, see *Underflow in the Transmitter* on page 9.4.4. 1 No transmitter-empty condition	0

Table 9–78. SPCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
1	XRDY	Transmitter ready bit XRDY is set when the transmitter is ready to accept new data in DXR[1,2]. Specifically, XRDY is set in response to a copy from DXR1 to XSR1. 0 Transmitter not ready When DXR1 is loaded, XRDY is automatically cleared. 1 Transmitter ready: DXR[1,2] is ready to accept new data. **Important:** If both DXRs are needed (word length larger than 16 bits), the CPU or the DMA controller must load DXR2 first and then load DXR1. As soon as DXR1 is loaded, the contents of both DXRs are copied to the transmit shift registers (XSRs), as described in the next step. If DXR2 is not loaded first, the previous content of DXR2 is passed to the XSR2. If the transmit interrupt mode is XINTM = 00b, the McBSP sends a transmit interrupt (XINT) request to the CPU when XRDY changes from 0 to 1. Also, when XRDY changes from 0 to 1, the McBSP sends a transmit synchronization event (XEVT) signal to the DMA controller.	0
0	XRST_	Transmitter reset bit You can use XRST_ to take the McBSP transmitter into and out of its reset state. The underscore (_) at the end of the bit name is a reminder of the negative polarity of the bit; XRST_ = 0 indicates the reset state. 0 If you read a 0, the transmitter is in its reset state. If you write a 0, you reset the transmitter. 1 If you read a 1, the transmitter is enabled. If you write a 1, you enable the transmitter by taking it out of its reset state. To read about the effects of a transmitter reset, see *Resetting and Initializing a McBSP* on page 9-147.	0

9.13.4 Receive Control Registers (RCR2 and RCR1)

Each McBSP has two receive control registers of the form shown in Figure 9–118. Table 9–79 and Table 9–80 describe the bis of RCR1 and RCR2, respectively. These I/O-mapped registers enable you to:

- ❑ Specify one or two phases for each frame of receive data (RPHASE)
- ❑ Define two parameters for phase 1 and (if necessary) phase 2: the serial word length (RWDLEN1, RWDLEN2) and the number of words (RFRLEN1, RFRLEN2)
- ❑ Choose a receive companding mode, if any (RCOMPAND)
- ❑ Enable or disable the receive frame-sync ignore function (RFIG)
- ❑ Choose a receive data delay (RDATDLY)

Figure 9–118. Receive Control Registers (RCR2 and RCR1)

RCR1

15	14–8	7–5	4–0
Rsvd	RFRLEN1	RWDLEN1	Rsvd
	R/W – 000 0000	R/W – 000	

RCR2

15	14–8	7–5	4–3	2	1–0
RPHASE	RFRLEN2	RWDLEN2	RCOMPAND	RFIG	RDATDLY
R/W – 0	R/W – 000 0000	R/W – 000	R/W – 00	R/W – 0	R/W – 00

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–79. RCR1 BIt Descriptions

Bits	Name	Description	Reset Value
15	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.	
14–8	RFRLEN1	Receive frame length 1 Each frame of receive data can have one or two phases, depending on value that you load into the RPHASE bit. If a single-phase frame is selected, RFRLEN1 in RCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, RFRLEN1 determines the number of serial words in phase 1 of the frame, and RFRLEN2 in RCR2 determines the number of words in phase 2 of the frame. The 7-bit RFRLEN fields allow up to 128 words per phase. See the following table for a summary of how you determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period. **Note:** Program the RFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 1, load 127 into RFRLEN1.	000 0000b

RPHASE	RFRLEN1	RFRLEN2	Frame Length
0	0 ≤ RFRLEN1 ≤ 127	Not used	(RFRLEN1 + 1) words
1	0 ≤ RFRLEN1 ≤ 127	0 ≤ RFRLEN2 ≤ 127	(RFRLEN1 + 1) + (RFRLEN2 + 1) words

Table 9–79. RCR1 BIt Descriptions (Continued)

Bits	Name	Description	Reset Value
7–5	RWDLEN1	Receive word length 1 Each frame of receive data can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, RWDLEN1 in RCR1 selects the length for every serial word received in the frame. If a dual-phase frame is selected, RWDLEN1 determines the length of the serial words in phase 1 of the frame, and RWDLEN2 in RCR2 determines the word length in phase 2 of the frame. 000b 8 bits 001b 12 bits 010b 16 bits 011b 20 bits 100b 24 bits 101b 32 bits 110b Reserved (do not use) 111b Reserved (do not use)	000b
4–0	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.	

Table 9–80. RCR2 Bit Descriptions

Bits	Name	Description	Reset Value
15	RPHASE	Receive phase number bit RPHASE determines whether the receive frame has one phase or two phases. For each phase you can define the serial word length and the number of serial words in the phase. To set up phase 1, program RWDLEN1 (word length) and RFRLEN1 (number of words). To set up phase 2 (if there are two phases), program RWDLEN2 and RFRLEN2. 0 Single-phase frame The receive frame has only one phase, phase 1. 1 Dual-phase frame The receive frame has two phases, phase 1 and phase 2.	0
14–8	RFRLEN2	Receive frame length 2 Each frame of receive data can have one or two phases, depending on value that you load into the RPHASE bit. If a single-phase frame is selected, RFRLEN1 in RCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, RFRLEN1 determines the number of serial words in phase 1 of the frame, and RFRLEN2 in RCR2 determines the number of words in phase 2 of the frame. The 7-bit RFRLEN fields allow up to 128 words per phase. See the following table for a summary of how to determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period. **Note:** Program the RFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 2, load 127 into RFRLEN2.	000 0000b

RPHASE	RFRLEN1	RFRLEN2	Frame Length
0	0 ≤ RFRLEN1 ≤ 127	Not used	(RFRLEN1 + 1) words
1	0 ≤ RFRLEN1 ≤ 127	0 ≤ RFRLEN2 ≤ 127	(RFRLEN1 + 1) + (RFRLEN2 + 1) words

Table 9–80. RCR2 BIt Descriptions (Continued)

Bits	Name	Description		Reset Value
7–5	RWDLEN2	Receive word length 2		000b
		Each frame of receive data can have one or two phases, depending on the value that you load into the RPHASE bit. If a single-phase frame is selected, RWDLEN1 in RCR1 selects the length for every serial word received in the frame. If a dual-phase frame is selected, RWDLEN1 determines the length of the serial words in phase 1 of the frame, and RWDLEN2 in RCR2 determines the word length in phase 2 of the frame.		
		000b	8 bits	
		001b	12 bits	
		010b	16 bits	
		011b	20 bits	
		100b	24 bits	
		101b	32 bits	
		110b	Reserved (do not use)	
		111b	Reserved (do not use)	
4–3	RCOMPAND	Receive companding mode bits		00b
		Companding (COMpress and exPAND) hardware allows compression and expansion of data in either μ-law or A-law format.		
		RCOMPAND allows you to choose one of the following companding modes for the McBSP receiver:		
		00b	No companding, any size data, MSB received first	
		01b	No companding, 8-bit data, LSB received first	
		10b	μ-law companding, 8-bit data, MSB received first	
		11b	A-law companding, 8-bit data, MSB received first	
		For more details about these companding modes, see *Companding (Compressing and Expanding) Data* on page 9-8.		

Table 9–80. RCR2 BIt Descriptions (Continued)

Bits	Name	Description	Reset Value
2	RFIG	Receive frame-sync ignore bit If a frame-sync pulse starts the transfer of a new frame before the current frame is fully received, this pulse is treated as an unexpected frame-sync pulse. For more details about the frame-sync error condition, see *Unexpected Receive Frame-Sync Pulse* on page 9-38. Setting RFIG causes the serial port to ignore unexpected frame-sync signals during reception. For more details on the effects of RFIG, see *Enable/Disable the Receive Frame-Sync Ignore Function* on page 9-85. 0 Disabled An unexpected FSR pulse causes the receiver to discard the contents of RSR[1,2] in favor of the new incoming data. The receiver: 1) Aborts the current data transfer 2) Sets RSYNCERR in SPCR1 3) Begins the transfer of a new data word 1 Enabled An unexpected FSR pulse is ignored. Reception continues uninterrupted.	0
1–0	RDATDLY	Receive data delay bits RDATDLY specifies a data delay of 0, 1, or 2 receive clock cycles after frame-synchronization and before the reception of the first bit of the frame. For more details, see *Set the Receive Data Delay* on page 9.8.13. 00b 0-bit data delay 01b 1-bit data delay 10b 2-bit data delay 11b Reserved (do not use)	00b

9.13.5 Transmit Control Registers (XCR2 and XCR1)

Each McBSP has two transmit control registers of the form shown in Figure 9–119. Table 9–81 and Table 9–82 describe the bits of XCR1 and XCR2, respectively. These I/O-mapped registers enable you to:

- Specify one or two phases for each frame of transmit data (XPHASE)
- Define two parameters for phase 1 and (if necessary) phase 2: the serial word length (XWDLEN1, XWDLEN2) and the number of words (XFRLEN1, XFRLEN2)
- Choose a transmit companding mode, if any (XCOMPAND)
- Enable or disable the transmit frame-sync ignore function (XFIG)
- Choose a transmit data delay (XDATDLY)

Figure 9–119. Transmit Control Registers (XCR2 and XCR1)

XCR1

15	14–8	7–5	4–0
Rsvd	XFRLEN1	XWDLEN1	Rsvd
	R/W – 000 0000	R/W – 000	

XCR2

15	14–8	7–5	4–3	2	1–0
XPHASE	XFRLEN2	XWDLEN2	XCOMPAND	XFIG	XDATDLY
R/W – 0	R/W – 000 0000	R/W – 000	R/W – 00	R/W – 0	R/W – 00

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–81. XCR1 Bit Descriptions

Bits	Name	Description	Reset Value
15	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.	
14–8	XFRLEN1	Transmit frame length 1 Each frame of transmit data can have one or two phases, depending on value that you load into the XPHASE bit. If a single-phase frame is selected, XFRLEN1 in XCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, XFRLEN1 determines the number of serial words in phase 1 of the frame, and XFRLEN2 in XCR2 determines the number of words in phase 2 of the frame. The 7-bit XFRLEN fields allow up to 128 words per phase. See the following table for a summary of how you determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period. **Note:** Program the XFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 1, load 127 into XFRLEN1.	000 0000b

XPHASE	XFRLEN1	XFRLEN2	Frame Length
0	0 ≤ XFRLEN1 ≤ 127	Not used	(XFRLEN1 + 1) words
1	0 ≤ XFRLEN1 ≤ 127	0 ≤ XFRLEN2 ≤ 127	(XFRLEN1 + 1) + (XFRLEN2 + 1) words

Table 9–81. XCR1 Bit Descriptions (Continued)

Bits	Name	Description		Reset Value
7–5	XWDLEN1	Transmit word length 1 Each frame of transmit data can have one or two phases, depending on the value that you load into the XPHASE bit. If a single-phase frame is selected, XWDLEN1 in XCR1 selects the length for every serial word transmitted in the frame. If a dual-phase frame is selected, XWDLEN1 determines the length of the serial words in phase 1 of the frame, and XWDLEN2 in XCR2 determines the word length in phase 2 of the frame.		000b
		000b	8 bits	
		001b	12 bits	
		010b	16 bits	
		011b	20 bits	
		100b	24 bits	
		101b	32 bits	
		110b	Reserved (do not use)	
		111b	Reserved (do not use)	
4–0	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.		

Table 9–82. XCR2 Bit Descriptions

Bits	Name	Description	Reset Value
15	XPHASE	Transmit phase number bit XPHASE determines whether the transmit frame has one phase or two phases. For each phase you can define the serial word length and the number of serial words in the phase. To set up phase 1, program XWDLEN1 (word length) and XFRLEN1 (number of words). To set up phase 2 (if there are two phases), program XWDLEN2 and XFRLEN2. 0 Single-phase frame The transmit frame has only one phase, phase 1. 1 Dual-phase frame The transmit frame has two phases, phase 1 and phase 2.	0
14–8	XFRLEN2	Transmit frame length 2 Each frame of transmit data can have one or two phases, depending on value that you load into the XPHASE bit. If a single-phase frame is selected, XFRLEN1 in XCR1 selects the number of serial words (8, 12, 16, 20, 24, or 32 bits per word) in the frame. If a dual-phase frame is selected, XFRLEN1 determines the number of serial words in phase 1 of the frame, and XFRLEN2 in XCR2 determines the number of words in phase 2 of the frame. The 7-bit XFRLEN fields allow up to 128 words per phase. See the following table for a summary of how to determine the frame length. This length corresponds to the number of words or logical time slots or channels per frame-synchronization period. **Note:** Program the XFRLEN fields with [*w minus 1*], where *w* represents the number of words per phase. For example, if you want a phase length of 128 words in phase 1, load 127 into XFRLEN1.	000 0000b

XPHASE	XFRLEN1	XFRLEN2	Frame Length
0	0 ≤ XFRLEN1 ≤ 127	Not used	(XFRLEN1 + 1) words
1	0 ≤ XFRLEN1 ≤ 127	0 ≤ XFRLEN2 ≤ 127	(XFRLEN1 + 1) + (XFRLEN2 + 1) words

Table 9–82. XCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
7–5	XWDLEN2	Transmit word length 2 Each frame of transmit data can have one or two phases, depending on the value that you load into the XPHASE bit. If a single-phase frame is selected, XWDLEN1 in XCR1 selects the length for every serial word transmitted in the frame. If a dual-phase frame is selected, XWDLEN1 determines the length of the serial words in phase 1 of the frame, and XWDLEN2 in XCR2 determines the word length in phase 2 of the frame. 000b 8 bits 001b 12 bits 010b 16 bits 011b 20 bits 100b 24 bits 101b 32 bits 110b Reserved (do not use) 111b Reserved (do not use)	000b
4–3	XCOMPAND	Transmit companding mode bits Companding (COMpress and exPAND) hardware allows compression and expansion of data in either μ-law or A-law format. For more details, see *Companding Data* on page 9-8. XCOMPAND allows you to choose one of the following companding modes for the McBSP transmitter: 00b No companding, any size data, MSB transmitted first 01b No companding, 8-bit data, LSB transmitted first 10b μ-law companding, 8-bit data, MSB transmitted first 11b A-law companding, 8-bit data, MSB transmitted first For more details about these companding modes, see *Companding (Compressing and Expanding) Data* on page 9-8.	00b

Table 9–82. XCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
2	XFIG	Transmit frame-sync ignore bit If a frame-sync pulse starts the transfer of a new frame before the current frame is fully transmitted, this pulse is treated as an unexpected frame-sync pulse. For more details about the frame-sync error condition, see *Unexpected Transmit Frame-Sync Pulse* on page 9-44. Setting XFIG causes the serial port to ignore unexpected frame-sync pulses during transmission. For more details on the effects of XFIG, see *Enable/Disable the Transmit Frame-Sync Ignore Function* on page 9-122. 0 Disabled An unexpected FSX pulse causes the transmitter to discard the content of XSR[1,2]. The transmitter: 1) Aborts the present transmission 2) Sets XSYNCERR in SPCR2 3) Begins a new transmission from DXR[1,2]. If new data was written to DXR[1,2] since the last DXR[1,2]-to-XSR[1,2] copy, the current value in XSR[1,2] is lost. Otherwise, the same data is transmitted. 1 Enabled An unexpected FSX pulse is ignored. Transmission continues uninterrupted.	0
1–0	XDATDLY	Transmit data delay bits XDATDLY specifies a data delay of 0, 1, or 2 transmit clock cycles after frame synchronization and before the transmission of the first bit of the frame. For more details, see *Set the Transmit Data Delay* on page 9-126. 00b 0-bit data delay 01b 1-bit data delay 10b 2-bit data delay 11b Reserved (do not use)	00b

9.13.6 Sample Rate Generator Registers (SRGR2 and SRGR1)

Each McBSP has two sample rate generator registers of the form shown in Figure 9–120. Table 9–83 and Table 9–84 describe the bits of SRGR1 and SRGR2, respectively. The sample rate generator can generate a clock signal (CLKG) and a frame-sync signal (FSG). The I/O-mapped registers SRGR1 and SRGR2 enable you to:

- ❑ Select the input clock source for the sample rate generator (CLKSM, in conjunction with the SCLKME bit of PCR)
- ❑ Divide down the frequency of CLKG (CLKGDV)
- ❑ Select whether internally-generated transmit frame-sync pulse are driven by FSG or by activity in the transmitter (FSGM).
- ❑ Specify the width of frame-sync pulses on FSG (FWID) and specify the period between those pulses (FPER)

When an external source (via the CLKS, CLKR, or CLKX pin) provides the input clock source for the sample rate generator:

- ❑ If the CLKS pin provides the input clock, the CLKSP bit in SRGR2 allows you to select whether the rising edge or the falling edge of CLKS triggers CLKG and FSG. If the CLKX/CLKR pin is used instead of the CLKS pin, the polarity of the input clock is selected with CLKXP/CLKRP of PCR.
- ❑ The GSYNC bit of SRGR2 allows you to make CLKG synchronized to an external frame-sync signal on the FSR pin, so that CLKG is kept in phase with the input clock.

Figure 9–120. Sample Rate Generator Registers (SRGR2 and SRGR1)

SRGR1

15–8	7–0
FWID	CLKGDV
R/W – 0000 0000	R/W – 0000 0001

SRGR2

15	14	13	12	11–0
GSYNC	CLKSP	CLKSM	FSGM	FPER
R/W – 0	R/W – 0	R/W – 1	R/W – 0	R/W – 0000 0000 0000

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–83. SRGR1 Bit Descriptions

Bits	Name	Description	Reset Value
15–8	FWID	Frame-sync pulse width bits for FSG The sample rate generator can produce a clock signal, CLKG, and a frame-sync signal, FSG. For frame-sync pulses on FSG, (FWID + 1) is the pulse width in CLKG cycles. The eight bits of FWID allow a pulse width of 1 to 256 CLKG cycles: $0 \leq FWID \leq 255$ $1 \leq (FWID + 1) \leq 256$ CLKG cycles The period between the frame-sync pulses on FSG is defined by the FPER bits.	0000 0000b
7–0	CLKGDV	Divide-down value for CLKG The sample rate generator can accept an input clock signal and divide it down according to CLKGDV to produce an output clock signal, CLKG. The frequency of CLKG is: CLKG frequency = (Input clock frequency) / (CLKGDV + 1) The input clock is selected by the SCLKME and CLKSM bits: **SCLKME CLKSM Input Clock For Sample Rate Generator** 0 0 Signal on CLKS pin 0 1 CPU clock 1 0 Signal on CLKR pin 1 1 Signal on CLKX pin A DSP reset forces the CLKG frequency to 1/2 the input clock frequency (CLKGDV = 1), and the reset selects the CPU clock as the input clock.	0000 0001b

Table 9–84. SRGR2 Bit Descriptions

Bits	Name	Description	Reset Value
15	GSYNC	Clock synchronization mode bit for CLKG GSYNC is used only when the input clock source for the sample rate generator is external—on the CLKS, CLKR, or CLKX pin. When GSYNC = 1, the clock signal (CLKG) and the frame-sync signal (FSG) generated by the sample rate generator are made dependent on pulses on the FSR pin. 0 No clock synchronization CLKG oscillates without adjustment, and FSG pulses every (FPER + 1) CLKG cycles. 1 Clock synchronization ❑ CLKG is adjusted as necessary so that it is synchronized with the input clock on the CLKS, CLKR, or CLKX pin. ❑ FSG pulses. FSG **only** pulses in response to a pulse on the FSR pin. The frame-sync period defined in FPER is ignored. For more details, see *Synchronizing Sample Rate Generator Outputs to an External Clock* on page 9-29.	0
14	CLKSP	CLKS pin polarity bit CLKSP is used only when the CLKS pin is the input clock source for the sample rate generator. The bit determines which edge of CLKS drives the clock signal (CLKG) and the frame-sync signal (FSG) that are generated by the sample rate generator: 0 A rising edge on the CLKS pin 1 A falling edge on the CLKS pin	0

Table 9–84. SRGR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
13	CLKSM	Sample rate generator input clock mode bit The sample rate generator can accept an input clock signal and divide it down according to CLKGDV to produce an output clock signal, CLKG. The frequency of CLKG is: CLKG frequency = (Input clock frequency) / (CLKGDV + 1) CLKSM is used in conjunction with the SCLKME bit to determine the source for the input clock: **SCLKME** **CLKSM** **Input Clock For Sample Rate Generator** 0 0 Signal on CLKS pin 0 1 CPU clock 1 0 Signal on CLKR pin 1 1 Signal on CLKX pin A DSP reset selects the CPU clock as the input clock and forces the CLKG frequency to 1/2 the CPU clock frequency.	1
12	FSGM	Sample rate generator transmit frame-sync mode bit The transmitter can get frame synchronization from the FSX pin (FSXM = 0) or from inside the McBSP (FSXM = 1). When FXSM = 1, the FSGM bit determines how the McBSP supplies frame-sync pulses: **FSXM** **FSGM** **Transmit Frame-Sync Mode** 1 0 The McBSP generates a transmit frame-sync pulse when the content of DXR[1,2] is copied to XSR[1,2]. 1 1 The transmitter uses frame-sync pulses generated by the sample rate generator. Program the FWID bits to set the width of each pulse. Program the FPER bits to set the period between pulses.	0

Table 9–84. SRGR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
11–0	FPER	Frame-sync period bits for FSG The sample rate generator can produce a clock signal, CLKG, and a frame-sync signal, FSG. The period between frame-sync pulses on FSG is (FPER + 1) CLKG cycles. The 12 bits of FPER allow a frame-sync period of 1 to 4096 CLKG cycles: $0 \leq FPER \leq 4095$ $1 \leq (FPER + 1) \leq 4096$ CLKG cycles The width of each frame-sync pulse on FSG is defined by the FWID bits.	0000 0000 0000b

9.13.7 Multichannel Control Registers (MCR2 and MCR1)

Each McBSP has two multichannel control registers of the form shown in Figure 9–121. MCR1 has control and status bits (with an R prefix) for multichannel selection operation in the receiver. MCR2 contains the same type of bits (bit with an X prefix) for the transmitter. The bits of MCR1 and MCR2 are described in Table 9–85 and Table 9–86, respectively. These I/O-mapped registers enable you to:

- ❏ Enable all channels or only selected channels for reception (RMCM)
- ❏ Choose which channels are enabled/disabled and masked/unmasked for transmission (XMCM)
- ❏ Specify whether two partitions (32 channels at a time) or eight partitions (128 channels at a time) can be used (RMCME for reception, XMCME for transmission)
- ❏ Assign blocks of 16 channels to partitions A and B when the 2-partition mode is selected (RPABLK and RPBBLK for reception, XPABLK and XPBBLK for transmission)
- ❏ Determine which block of 16 channels is currently involved in a data transfer (RCBLK for reception, XCBLK for transmission)

Figure 9–121. Multichannel Control Registers (MCR2 and MCR1)

MCR1

15–10	9	8–7	6–5	4–2	1	0
Rsvd	RMCME	RPBBLK	RPABLK	RCBLK	Rsvd	RMCM
	R/W – 0	R/W – 00	R/W – 00	R – 000		R/W – 0

MCR2

15–10	9	8–7	6–5	4–2	1–0
Rsvd	XMCME	XPBBLK	XPABLK	XCBLK	XMCM
	R/W – 0	R/W – 00	R/W – 00	R – 000	R/W – 00

Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset.

Table 9–85. MCR1 Bit Descriptions

Bits	Name	Description	Reset Value
15–10	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.	
9	RMCME	Receive multichannel partition mode bit RMCME is only applicable if channels can be individually enabled or disabled for reception (RMCM = 1). RMCME determines whether only 32 channels or all 128 channels are to be individually selectable. 0 2-partition mode Only partitions A and B are used. You can control up to 32 channels in the receive multichannel selection mode (RMCM = 1). Assign 16 channels to partition A with the RPABLK bits. Assign 16 channels to partition B with the RPBBLK bits. You control the channels with the appropriate receive channel enable registers: RCERA: Channels in partition A RCERB: Channels in partition B 1 8-partition mode All partitions (A through H) are used. You can control up to 128 channels in the receive multichannel selection mode. You control the channels with the appropriate receive channel enable registers: RCERA: Channels 0 through 15 RCERB: Channels 16 through 31 RCERC: Channels 32 through 47 RCERD: Channels 48 through 63 RCERE: Channels 64 through 79 RCERF: Channels 80 through 95 RCERG: Channels 96 through 111 RCERH: Channels 112 through 127	0

Table 9–85. MCR1 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
8–7	RPBBLK	Receive partition B block bits RPBBLK is only applicable if channels can be individually enabled or disabled (RMCM = 1) and the 2-partition mode is selected (RMCME = 0). Under these conditions, the McBSP receiver can accept or ignore data in any of the 32 channels that are assigned to partitions A and B of the receiver. The 128 receive channels of the McBSP are divided equally among 8 blocks (0 through 7). When RPBBLK is applicable, use RPBBLK to assign one of the odd-numbered blocks (1, 3, 5, or 7) to partition B, as shown in the following table. Use the RPABLK bit to assign one of the even-numbered blocks (0, 2, 4, or 6) to partition A. If you want to use more than 32 channels, you can change block assignments dynamically. You can assign a new block to one partition while the receiver is handling activity in the other partition. For example, while the block in partition A is active, you can change which block is assigned to partition B. The RCBLK bits are regularly updated to indicate which block is active. **Note:** When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive block bits (RPABLK and RPBBLK) rather than the transmit block bits (XPABLK and XPBBLK). **If RMCM = 1 and RMCME = 0:** 00b Block 1: channels 16 through 31 01b Block 3: channels 48 through 63 10b Block 5: channels 80 through 95 11b Block 7: channels 112 through 127	00b

Table 9–85. MCR1 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
6–5	RPABLK	Receive partition A block bits RPABLK is only applicable if channels can be individually enabled or disabled (RMCM = 1) and the 2-partition mode is selected (RMCME = 0). Under these conditions, the McBSP receiver can accept or ignore data in any of the 32 channels that are assigned to partitions A and B of the receiver. The 128 receive channels of the McBSP are divided equally among 8 blocks (0 through 7). When RPABLK is applicable, use RPABLK to assign one of the even-numbered blocks (0, 2, 4, or 6) to partition A, as shown in the following table. Use the RPBBLK bit to assign one of the odd-numbered blocks (1, 3, 5, or 7) to partition B. If you want to use more than 32 channels, you can change block assignments dynamically. You can assign a new block to one partition while the receiver is handling activity in the other partition. For example, while the block in partition B is active, you can change which block is assigned to partition A. The RCBLK bits are regularly updated to indicate which block is active. **Note:** When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive block bits (RPABLK and RPBBLK) rather than the transmit block bits (XPABLK and XPBBLK). **If RMCM = 1 and RMCME = 0:** 00b Block 0: channels 0 through 15 01b Block 2: channels 32 through 47 10b Block 4: channels 64 through 79 11b Block 6: channels 96 through 111	00b

Table 9–85. MCR1 Bit Descriptions (Continued)

Bits	Name	Description		Reset Value
4–2	RCBLK	Receive current block indicator		000b
		RCBLK indicates which block of 16 channels is involved in the current McBSP reception:		
		000b	Block 0: channels 0 through 15	
		001b	Block 1: channels 16 through 31	
		010b	Block 2: channels 32 through 47	
		011b	Block 3: channels 48 through 63	
		100b	Block 4: channels 64 through 79	
		101b	Block 5: channels 80 through 95	
		110b	Block 6: channels 96 through 111	
		111b	Block 7: channels 112 through 127	
1	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.		
0	RMCM	Receive multichannel selection mode bit		0
		RMCM determines whether all channels or only selected channels are enabled for reception:		
		0	Disabled All 128 channels are enabled.	
		1	Enabled Channels can be individually enabled or disabled. The only channels enabled are those selected in the appropriate receive channel enable registers (RCERs). The way channels are assigned to the RCERs depends on the number of receive channel partitions (2 or 8), as defined by the RMCME bit.	

Table 9–86. MCR2 Bit Descriptions

Bits	Name	Description	Reset Value
15–10	Reserved	Reserved bits (not available for your use). They are read-only bits and return 0s when read.	–
9	XMCME	Transmit multichannel partition mode bit XMCME is only applicable if channels can be individually disabled/enabled or masked/unmasked for transmission (XMCM is nonzero). XMCME determines whether only 32 channels or all 128 channels are to be individually selectable. 0 2-partition mode Only partitions A and B are used. You can control up to 32 channels in the transmit multichannel selection mode selected with the XMCM bits. **If XMCM = 01b or 10b**, assign 16 channels to partition A with the XPABLK bits. Assign 16 channels to partition B with the XPBBLK bits. **If XMCM = 11b** (for symmetric transmission and reception), assign 16 channels to receive partition A with the RPABLK bits. Assign 16 channels to receive partition B with the RPBBLK bits. You control the channels with the appropriate transmit channel enable registers: XCERA: Channels in partition A XCERB: Channels in partition B 1 8-partition mode All partitions (A through H) are used. You can control up to 128 channels in the transmit multichannel selection mode selected with the XMCM bits. You control the channels with the appropriate transmit channel enable registers: XCERA: Channels 0 through 15 XCERB: Channels 16 through 31 XCERC: Channels 32 through 47 XCERD: Channels 48 through 63 XCERE: Channels 64 through 79 XCERF: Channels 80 through 95 XCERG: Channels 96 through 111 XCERH: Channels 112 through 127	0

Table 9–86. MCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
8–7	XPBBLK	Transmit partition B block bits XPBBLK is only applicable if channels can be individually disabled/enabled and masked/unmasked (XMCM is nonzero) and the 2-partition mode is selected (XMCME = 0). Under these conditions, the McBSP transmitter can transmit or withhold data in any of the 32 channels that are assigned to partitions A and B of the transmitter. The 128 transmit channels of the McBSP are divided equally among 8 blocks (0 through 7). When XPBBLK is applicable, use XPBBLK to assign one of the odd-numbered blocks (1, 3, 5, or 7) to partition B, as shown in the following table. Use the XPABLK bit to assign one of the even-numbered blocks (0, 2, 4, or 6) to partition A. If you want to use more than 32 channels, you can change block assignments dynamically. You can assign a new block to one partition while the transmitter is handling activity in the other partition. For example, while the block in partition A is active, you can change which block is assigned to partition B. The XCBLK bits are regularly updated to indicate which block is active. **Note:** When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive block bits (RPABLK and RPBBLK) rather than the transmit block bits (XPABLK and XPBBLK). **If XMCM = 01b or 10b, and XMCME = 0:** 00b Block 1: channels 16 through 31 01b Block 3: channels 48 through 63 10b Block 5: channels 80 through 95 11b Block 7: channels 112 through 127	00b

Table 9–86. MCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
6–5	XPABLK	Transmit partition A block bits XPABLK is only applicable if channels can be individually disabled/enabled and masked/unmasked (XMCM is nonzero) and the 2-partition mode is selected (XMCME = 0). Under these conditions, the McBSP transmitter can transmit or withhold data in any of the 32 channels that are assigned to partitions A and B of the transmitter. The 128 transmit channels of the McBSP are divided equally among 8 blocks (0 through 7). When XPABLK is applicable, use XPABLK to assign one of the even-numbered blocks (0, 2, 4, or 6) to partition A, as shown in the following table. Use the XPBBLK bit to assign one of the odd-numbered blocks (1, 3, 5, or 7) to partition B. If you want to use more than 32 channels, you can change block assignments dynamically. You can assign a new block to one partition while the transmitter is handling activity in the other partition. For example, while the block in partition B is active, you can change which block is assigned to partition A. The XCBLK bits are regularly updated to indicate which block is active. **Note:** When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive block bits (RPABLK and RPBBLK) rather than the transmit block bits (XPABLK and XPBBLK). **If XMCM = 01b or 10b, and XMCME = 0:** 00b Block 0: channels 0 through 15 01b Block 2: channels 32 through 47 10b Block 4: channels 64 through 79 11b Block 6: channels 96 through 111	00b

Table 9–86. MCR2 Bit Descriptions (Continued)

Bits	Name	Description		Reset Value
4–2	XCBLK	Transmit current block indicator		000b
		XCBLK indicates which block of 16 channels is involved in the current McBSP transmission:		
		000b	Block 0: channels 0 through 15	
		001b	Block 1: channels 16 through 31	
		010b	Block 2: channels 32 through 47	
		011b	Block 3: channels 48 through 63	
		100b	Block 4: channels 64 through 79	
		101b	Block 5: channels 80 through 95	
		110b	Block 6: channels 96 through 111	
		111b	Block 7: channels 112 through 127	

Table 9–86. MCR2 Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
1–0	XMCM	Transmit multichannel selection mode bits	00b
		XMCM determines whether all channels or only selected channels are enabled and unmasked for transmission. For more details on how the channels are affected, see *Transmit Multichannel Selection Modes* on page 9-54.	
		00b — No transmit multichannel selection mode is on. All channels are enabled and unmasked. No channels can be disabled or masked.	
		01b — All channels are disabled unless they are selected in the appropriate transmit channel enable registers (XCERs). If enabled, a channel in this mode is also unmasked. The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.	
		10b — All channels are enabled, but they are masked unless they are selected in the appropriate transmit channel enable registers (XCERs). The XMCME bit determines whether 32 channels or 128 channels are selectable in XCERs.	
		11b — This mode is used for symmetric transmission and reception. All channels are disabled for transmission unless they are enabled for reception in the appropriate receive channel enable registers (RCERs). Once enabled, they are masked unless they are also selected in the appropriate transmit channel enable registers (XCERs). The XMCME bit determines whether 32 channels or 128 channels are selectable in RCERs and XCERs.	

9.13.8 Pin Control Register (PCR)

Each McBSP has one pin control register of the form shown in Figure 9–122. Table 9–87 describes the bits of PCR. This I/O-mapped register enables you to:

- ❑ Allow the McBSP to enter a low-power mode when the idle instruction is executed (IDLE_EN, in conjunction with the PERI bit of ICR)
- ❑ Specify whether McBSP pins can be used as general-purpose I/O pins when the transmitter and/or receiver is in its reset state (XIOEN and RIOEN)

- ❑ Choose a frame-sync mode for the transmitter (FSXM) and for the receiver (FSRM)
- ❑ Choose a clock mode for transmitter (CLKXM) and for the receiver (CLKRM)
- ❑ Select the input clock source for the sample rate generator (SCLKME, in conjunction with the CLKSM bit of SRGR2)
- ❑ Read or write data when the CLKS, DX, and DR pins are configured as general-purpose I/O pins (CLKS_STAT, DX_STAT, and DX_STAT)
- ❑ Choose whether frame-sync signals are active low or active high (FSXP for transmission, FSRP for reception)
- ❑ Specify whether data is sampled on the falling edge or the rising edge of the clock signals (CLKXP for transmission, CLKRP for reception)

Figure 9–122. Pin Control Register (PCR)

PCR

15	14	13	12	11	10	9	8
Rsvd	IDLE_EN	XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM
	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
SCLKME	CLKS_STAT	DX_STAT	DR_STAT	FSXP	FSRP	CLKXP	CLKRP
R/W – 0	R – 0	R/W – 0	R – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0

Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset.

Table 9–87. PCR Bit Descriptions

Bits	Name	Description	Reset Value
15	Reserved	Reserved bit (not available for your use). It is a read-only bit and returns a 0 when read.	
14	IDLE_EN	Idle enable bit for the McBSP	0
		If the PERIPH idle domain is configured to be idle and IDLE_EN = 1, the McBSP stops and enters a low-power state.	
		0 The McBSP is running.	
		1 If the PERIPH domain is idle (PERIS = 1 in the idle status register), the McBSP is stopped in a low-power state.	

Table 9–87. PCR Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
13	XIOEN	Transmit I/O enable bit	0
12	RIOEN	Receive I/O enable bit XIOEN is only applicable when the transmitter is in reset (XRST_ = 0). RIOEN is only applicable when the receiver is in reset (RRST_ = 0). As shown in the following table, these bits enable or disable the use of McBSP pins as general-purpose I/O pins. The transmit pins CLKX, FSX, and DX can only be general-purpose pins when the transmitter is in reset and XIOEN = 1. The receive pins CLKR, FSR, and DR can only be general-purpose pins when the receiver is in reset and RIOEN = 1. The sample rate generator pin CLKS can only be a general-purpose pin when both the transmitter and the receiver are in reset and both I/O enable bits are set. XRST_ and RRST_ are in the serial port control registers. All of the other bits mentioned in the table are in the pin control register.	0

Pin	General Purpose Use Enabled by This Bit Combination	Selected as Output When …	Output Value Driven From This Bit	Selected As Input When …	Input Value Read From This Bit
CLKX	XRST_ = 0 XIOEN = 1	CLKXM = 1	CLKXP	CLKXM = 0	CLKXP
FSX	XRST_ = 0 XIOEN = 1	FSXM = 1	FSXP	FSXM = 0	FSXP
DX	XRST_ = 0 XIOEN = 1	Always	DX_STAT	Never	Does not apply
CLKR	RRST_ = 0 RIOEN = 1	CLKRM = 1	CLKRP	CLKRM = 0	CLKRP
FSR	RRST_ = 0 RIOEN = 1	FSRM = 1	FSRP	FSRM = 0	FSRP
DR	RRST_ = 0 RIOEN = 1	Never	Does not apply	Always	DR_STAT
CLKS	RRST_ = XRST_ = 0 RIOEN = XIOEN = 1	Never	Does not apply	Always	CLKS_STAT

Table 9–87. PCR Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
11	FSXM	Transmit frame-sync mode bit As shown in the following table, FSXM determines whether transmit frame-sync pulses are supplied externally or internally. **Note:** The polarity of the signal on the FSX pin is determined by the FSXP bit. 0 Transmit frame synchronization is supplied by an external source via the FSX pin. 1 Transmit frame synchronization is supplied by the McBSP, as determined by the FSGM bit of SRGR2.	0
10	FSRM	Receive frame-sync mode bit As shown in the following table, FSRM determines whether receive frame-sync pulses are supplied externally or internally. **Note:** The polarity of the signal on the FSR pin is determined by the FSRP bit. 0 Receive frame synchronization is supplied by an external source via the FSR pin. 1 Receive frame synchronization is supplied by the sample rate generator. FSR is an output pin reflecting internal FSR, except when GSYNC = 1 in SRGR2.	0
9	CLKXM	Transmit clock mode bit CLKXM determines whether the source for the transmit clock is external or internal. The polarity of the signal on the CLKX pin is determined by the CLKXP bit. 0 The transmitter gets its clock signal from an external source via the CLKX pin. 1 Internal CLKX is driven by the sample rate generator of the McBSP. The CLKX pin is an output pin that reflects internal CLKX.	0

Table 9–87. PCR Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
9	CLKXM (continued)	In the clock stop mode (CLKSTP = 10b or 11b), the McBSP can act as a master or as a slave in the SPI protocol. If the McBSP is a master, make sure that CLKXM = 1, so that CLKX is an output to supply the master clock to any slave devices. If the McBSP is a slave, make sure that CLKXM = 0, so that CLKX is an input to accept the master clock signal. The following table summarizes the transmit clocking options available in the clock stop mode:	

CLKSTP	CLKXM	Transmit Clocking Option
10b/11b Clock stop mode	0	The McBSP is a slave in the SPI protocol. The internal transmit clock (CLKX) is driven by the SPI master via the CLKX pin. The internal receive clock (CLKR) is driven internally by CLKX, so that both the transmitter and the receiver are controlled by the external master clock.
	1	The McBSP is a master in the SPI protocol. The sample rate generator drives the internal transmit clock (CLKX). Internal CLKX is reflected on the CLKX pin to drive the shift clock of the SPI-compliant slaves in the system. Internal CLKX also drives the internal receive clock (CLKR), so that both the transmitter and the receiver are controlled by the internal master clock.

Table 9–87. PCR Bit Descriptions (Continued)

Bits	Name	Description			Reset Value
8	CLKRM	Receive clock mode bit			0
		As shown in the following table, the role of CLKRM depends on whether the McBSP is in the digital loopback mode (DLB = 1).			
		Note: The polarity of the signal on the CLKR pin is determined by the CLKRP bit.			
		DLB	**CLKRM**	**Receive Clock Mode**	
		0	0	The CLKR pin is an input pin that supplies the internal receive clock (CLKR).	
			1	Internal CLKR is driven by the sample rate generator of the McBSP. The CLKR pin is an output pin that reflects internal CLKR.	
		1 Digital loopback mode	0	The CLKR pin is in the high impedance state. The internal receive clock (CLKR) is driven by the internal transmit clock (CLKX). CLKX is derived according to the CLKXM bit.	
			1	Internal CLKR is driven by internal CLKX. The CLKR pin is an output pin that reflects internal CLKR. CLKX is derived according to the CLKXM bit.	
7	SCLKME	Sample rate generator input clock mode bit			0
		The sample rate generator can produce a clock signal, CLKG. The frequency of CLKG is:			
		CLKG frequency = (Input clock frequency) / (CLKGDV + 1)			
		SCLKME is used in conjunction with the CLKSM bit to select the input clock:			
		SCLKME	**CLKSM**	**Input Clock For Sample Rate Generator**	
		0	0	Signal on CLKS pin	
		0	1	CPU clock	
		1	0	Signal on CLKR pin	
		1	1	Signal on CLKX pin	

Table 9–87. PCR Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
6	CLKS_STAT	CLKS pin status bit CLKS_STAT is only applicable when the transmitter and receiver are both in reset (XRST_ = RRST_ = 0) and CLKS is configured for use as a general-purpose input pin (XIOEN = RIOEN = 1). When CLKS_STAT is applicable, it reflects the level on the CLKS pin: **If XRST_ = RRST_ = 0 and XIOEN = RIOEN = 1:** 0 The signal on the CLKS pin is low. 1 The signal on the CLKS pin is high.	0
5	DX_STAT	DX pin status bit DX_STAT is only applicable when the transmitter is in reset (XRST_ = 0) and DX is configured for use as a general-purpose output pin (XIOEN = 1). When DX_STAT is applicable, you can toggle the signal on DX by writing to DX_STAT: **If XRST_ = 0 and XIOEN = 1:** 0 Drive the signal on the DX pin low. 1 Drive the signal on the DX pin high.	0
4	DR_STAT	DR pin status bit DR_STAT is only applicable when the receiver is in reset (RRST_ = 0) and DR is configured for use as a general-purpose input pin (RIOEN = 1). When DR_STAT is applicable, it reflects the level on the DR pin: **If RRST_ = 0 and RIOEN = 1:** 0 The signal on DR pin is low. 1 The signal on DR pin is high.	0
3	FSXP	Transmit frame-sync polarity bit FSXP determines the polarity of FSX as seen on the FSX pin: 0 Transmit frame-sync pulses are active high. 1 Transmit frame-sync pulses are active low.	0

Table 9–87. PCR Bit Descriptions (Continued)

Bits	Name	Description	Reset Value
2	FSRP	Receive frame-sync polarity bit	0
		FSRP determines the polarity of FSR as seen on the FSR pin:	
		0 Receive frame-sync pulses are active high.	
		1 Receive frame-sync pulses are active low.	
1	CLKXP	Transmit clock polarity bit	0
		CLKXP determines the polarity of CLKX as seen on the CLKX pin:	
		0 Transmit data is sampled on the rising edge of CLKX.	
		1 Transmit data is sampled on the falling edge of CLKX.	
0	CLKRP	Receive clock polarity bit	0
		CLKRP determines the polarity of CLKR as seen on the CLKR pin:	
		0 Receive data is sampled on the falling edge of CLKR.	
		1 Receive data is sampled on the rising edge of CLKR.	

9.13.9 Receive Channel Enable Registers (RCERA, RCERB, RCERC, RCERD, RCERE, RCERF, RCERG, RCERH)

Each McBSP has eight receive channel enable registers of the form shown in Figure 9–123. There is one for each of the receive partitions: A, B, C, D, E, F, G, and H. Table 9–88 provides a general bit description that applies to each of the receive channel enable registers.

These I/O-mapped registers are only used when the receiver is configured to allow individual enabling and disabling of the channels (RMCM = 1) or when the receiver needs bit-enable patterns for the A-bis mode (ABIS = 1). For more details about the way these registers are used be sure to read the topics at the end of this section:

- *RCERs Used in the Receive Multichannel Selection Mode* (page 9-205)
- *RCERs Used in the A-bis Mode* (page 9-206)

Figure 9–123. Receive Channel Enable Registers(RCERA–RCERH)

RCERp (p = partition = A, B, C, D, E, F, G, or H)

15	14	13	12	11	10	9	8
RCEp15	RCEp14	RCEp13	RCEp12	RCEp11	RCEp10	RCEp9	RCEp8
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
RCEp7	RCEp6	RCEp5	RCEp4	RCEp3	RCEp2	RCEp1	RCEp0
R/W – 0	R/W – 0	RW – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Table 9–88. RCEp0–RCEp15 Bit Descriptions

Bits	Name	Description	Reset Value
15–0	RCEp15 through RCEp0	Receive channel enable bits for partition p (p = A, B, C, D, E, F, G, or H) The following summary describes the functions of the receive channel enable bits for two scenarios: the receive multichannel selection mode and the A-bis mode. In the first column of the table, p is one of the partition letters. For example, partition G uses RCERG, which contains bits RCEG0 through RCEG15. In the second column, n is the first of 16 contiguous channels that are assigned to partition p. In the receive multichannel selection mode (RMCM = 1), either two receive channel enable registers are used (RCERA and RCERB) or eight receive channel enable registers are used (RCERA through RCERH). In the A-bis mode (ABIS = 1), RCERA and RCERB (only) are used.	0

Bit Setting	Description For Multichannel Selection Mode	Description For A-bis Mode
RCEp15 = 0	Channel (n + 15) disabled	1st bit to arrive is ignored
RCEp15 = 1	Channel (n + 15) enabled	1st bit to arrive is stored
RCEp14 = 0	Channel (n + 14) disabled	2nd bit to arrive is ignored
RCEp14 = 1	Channel (n + 14) enabled	2nd bit to arrive is stored
RCEp13 = 0	Channel (n + 13) disabled	3rd bit to arrive is ignored
RCEp13 = 1	Channel (n + 13) enabled	3rd bit to arrive is stored

Table 9–88. RCEp0–RCEp15 Bit Descriptions (Continued)

Bits	Name	Description			Reset Value
		RCEp12 = 0	Channel (n + 12) disabled	4th bit to arrive is ignored	
		RCEp12 = 1	Channel (n + 12) enabled	4th bit to arrive is stored	
15–0	(cont.)	RCEp11 = 0	Channel (n + 11) disabled	5th bit to arrive is ignored	
		RCEp11 = 1	Channel (n + 11) enabled	5th bit to arrive is stored	
		RCEp10 = 0	Channel (n + 10) disabled	6th bit to arrive is ignored	
		RCEp10 = 1	Channel (n + 10) enabled	6th bit to arrive is stored	
		RCEp9 = 0	Channel (n + 9) disabled	7th bit to arrive is ignored	
		RCEp9 = 1	Channel (n + 9) enabled	7th bit to arrive is stored	
		RCEp8 = 0	Channel (n + 8) disabled	8th bit to arrive is ignored	
		RCEp8 = 1	Channel (n + 8) enabled	8th bit to arrive is stored	
		RCEp7 = 0	Channel (n + 7) disabled	9th bit to arrive is ignored	
		RCEp7 = 1	Channel (n + 7) enabled	9th bit to arrive is stored	
		RCEp6 = 0	Channel (n + 6) disabled	10th bit to arrive is ignored	
		RCEp6 = 1	Channel (n + 6) enabled	10th bit to arrive is stored	
		RCEp5 = 0	Channel (n + 5) disabled	11th bit to arrive is ignored	
		RCEp5 = 1	Channel (n + 5) enabled	11th bit to arrive is stored	
		RCEp4 = 0	Channel (n + 4) disabled	12th bit to arrive is ignored	
		RCEp4 = 1	Channel (n + 4) enabled	12th bit to arrive is stored	
		RCEp3 = 0	Channel (n + 3) disabled	13th bit to arrive is ignored	
		RCEp3 = 1	Channel (n + 3) enabled	13th bit to arrive is stored	
		RCEp2 = 0	Channel (n + 2) disabled	14th bit to arrive is ignored	
		RCEp2 = 1	Channel (n + 2) enabled	14th bit to arrive is stored	
		RCEp1 = 0	Channel (n + 1) disabled	15th bit to arrive is ignored	
		RCEp1 = 1	Channel (n + 1) enabled	15th bit to arrive is stored	
		RCEp0 = 0	Channel n disabled	16th bit to arrive is ignored	
		RCEp0 = 1	Channel n enabled	16th bit to arrive is stored	

9.13.9.1 RCERs Used in the Receive Multichannel Selection Mode

For multichannel selection operation, the assignment of channels to the RCERs depends on whether 32 or 128 channels are individually selectable, as defined by the RMCME bit (see Table 9–89).

Table 9–89. Use of the Receive Channel Enable Registers in the Receive Multichannel Selection Mode

Selectable Channels	Block Assignments	Channel Assignments
32 (RMCME = 0)	RCERA: Channels n–(n + 15) (channels assigned with the RPABLK bits)	RCEA0: Channel n RCEA1: Channel (n + 1) RCEA2: Channel (n + 2) : RCEA15: Channel (n + 15)
	RCERB: Channels m–(m + 15) (channels assigned with the RPBBLK bits)	RCEB0: Channel m RCEB1: Channel (m + 1) RCEB2: Channel (m + 2) : RCEB15: Channel (m + 15)
	Other RCERs not used	–
128 (RMCME = 1)	RCERA: Block 0	RCEA0: Channel 0 RCEA1: Channel 1 RCEA2: Channel 2 : RCEA15: Channel 15
	RCERB: Block 1	RCEB0: Channel 16 RCEB1: Channel 17 RCEB2: Channel 18 : RCEB15: Channel 31
	RCERC: Block 2	RCEC0: Channel 32 RCEC1: Channel 33 RCEC2: Channel 34 : RCEC15: Channel 47
	RCERD: Block 3	RCED0: Channel 48 RCED1: Channel 49 RCED2: Channel 50 : RCED15: Channel 63

Table 9–89. Use of the Receive Channel Enable Registers in the Receive Multichannel Selection Mode (Continued)

Selectable Channels	Block Assignments	Channel Assignments
128 (continued)	RCERE: Block 4	RCEE0: Channel 64 RCEE1: Channel 65 RCEE2: Channel 66 : RCEE15: Channel 79
	RCERF: Block 5	RCEF0: Channel 80 RCEF1: Channel 81 RCEF2: Channel 82 : RCEF15: Channel 95
	RCERG: Block 6	RCEG0: Channel 96 RCEG1: Channel 97 RCEG2: Channel 98 : RCEG15: Channel 111
	RCERH: Block 7	RCEH0: Channel 112 RCEH1: Channel 113 RCEH2: Channel 114 : RCEH15: Channel 127

9.13.9.2 RCERs Used in the A-bis Mode

In A-bis mode operation, only RCERA and RCERB are used. Sixteen bits at a time are passed to the receiver. As each of the 16 bits arrives on the DR pin (the MSB, bit 15, arrives first), the bit is stored or ignored, depending on the bit-enable pattern in RCERA or RCERB. The first 16 bits that arrive are handled according to the bit-enable pattern in RCERA. For example, if RCEA6 = 1 and all the other bits of RCERA are 0s, only bit 6 is stored. Each of the next 16 bits is stored or ignored according to the bit-enable pattern in RCERB. The receiver alternately uses RCERA and RCERB for consecutive 16-bit words.

Table 9–90 shows how bits of RCERA correspond to the bits of the incoming word.

Table 9–90. Use of Receive Channel Enable Registers A and B in the A-bis Mode

Register	Bits
RCERA	RCEA15: Enables or masks the first bit of the incoming word RCEA14: Enables or masks bit 14 of the incoming word RCEA13: Enables or masks bit 13 of the incoming word : RCEA0: Enables or masks the last bit of the incoming word
RCERB	RCEB15: Enables or masks the first bit of the incoming word RCEB14: Enables or masks bit 14 of the incoming word RCEB13: Enables or masks bit 13 of the incoming word : RCEB0: Enables or masks the last bit of the incoming word

9.13.10 Transmit Channel Enable Registers (XCERA, XCERB, XCERC, XCERD, XCERE, XCERF, XCERG, XCERH)

Each McBSP has eight transmit channel enable registers of the form shown in the following figure. There is one for each of the transmit partitions: A, B, C, D, E, F, G, and H. provides a general bit description that applies to each of the transmit channel enable registers.

These I/O-mapped registers are only used when transmitter is configured to allow individual disabling/enabling and masking/unmasking of the channels (XMCM is nonzero) or when the transmitter needs bit-enable patterns for the A-bis mode (ABIS = 1). For more details about the way these registers are used, read the topics at the end of this section:

- *XCERs Used in a Transmit Multichannel Selection Mode* (page 9-208)
- *XCERs Used in the A-bis Mode* (page 9-209)

Figure 9–124. Transmit Channel Enable Registers (XCERA–XCERH

XCERp (p = partition = A, B, C, D, E, F, G, or H)

15	14	13	12	11	10	9	8
XCEp15	XCEp14	XCEp13	XCEp12	XCEp11	XCEp10	XCEp9	XCEp8
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
XCEp7	XCEp6	XCEp5	XCEp4	XCEp3	XCEp2	XCEp1	XCEp0
R/W – 0	R/W – 0	RW – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

9.13.10.1 XCERs Used in a Transmit Multichannel Selection Mode

For multichannel selection operation, the assignment of channels to the XCERs, depends on whether 32 or 128 channels are individually selectable, as defined by the XMCME bit, as shown in the following table.

Note:

When XMCM = 11b (for symmetric transmission and reception), the transmitter uses the receive channel enable registers (RCERs) to enable channels and uses the XCERs to unmask channels for transmission.

Table 9–91. Use of the Transmit Channel Enable Registers in a Transmit Multichannel Selection Mode

Selectable Channels	Block Assignments	Channel Assignments
32 (XMCME = 0)	XCERA: Channels n–(n + 15) (channels assigned with the XPABLK bits for XMCM = 01b or 10b; assigned with the RPABLK bits for XMCM = 11b)	XCEA0: Channel n XCEA1: Channel (n + 1) XCEA2: Channel (n + 2) : XCEA15: Channel (n + 15)
	XCERB: Channels m–(m + 15) (channels assigned with the XPBBLK bits for XMCM = 01b or 10b; assigned with the RPBBLK bits for XMCM = 11b)	XCEB0: Channel m XCEB1: Channel (m + 1) XCEB2: Channel (m + 2) : XCEB15: Channel (m + 15)
	Other XCERs not used	–
128 (XMCME = 1)	XCERA: Block 0	XCEA0: Channel 0 XCEA1: Channel 1 XCEA2: Channel 2 : XCEA15: Channel 15
	XCERB: Block 1	XCEB0: Channel 16 XCEB1: Channel 17 XCEB2: Channel 18 : XCEB15: Channel 31
	XCERC: Block 2	XCEC0: Channel 32 XCEC1: Channel 33 XCEC2: Channel 34 : XCEC15: Channel 47

Table 9–91. Use of the Transmit Channel Enable Registers in a Transmit Multichannel Selection Mode (Continued)

Selectable Channels	Block Assignments	Channel Assignments
128 (continued)	XCERD: Block 3	XCED0: Channel 48 XCED1: Channel 49 XCED2: Channel 50 : XCED15: Channel 63
	XCERE: Block 4	XCEE0: Channel 64 XCEE1: Channel 65 XCEE2: Channel 66 : XCEE15: Channel 79
	XCERF: Block 5	XCEF0: Channel 80 XCEF1: Channel 81 XCEF2: Channel 82 : XCEF15: Channel 95
	XCERG: Block 6	XCEG0: Channel 96 XCEG1: Channel 97 XCEG2: Channel 98 : XCEG15: Channel 111
	XCERH: Block 7	XCEH0: Channel 112 XCEH1: Channel 113 XCEH2: Channel 114 : XCEH15: Channel 127

9.13.10.2 XCERs Used in the A-bis Mode

In A-bis mode operation, only XCERA and XCERB are used. Sixteen bits at a time are passed to data transmit register 1 (DXR1) by the CPU or by the DMA controller. Each of the 16 bits is transmitted on the DX pin or is ignored, depending on the bit-enable pattern in XCERA or XCERB. The first 16 bits that enter the transmitter are handled according to the bit-enable pattern in XCERA. For example, if XCEA13 = 1 and all the other bits of XCERA are 0s, only bit 13 is transmitted. Each of the next 16 bits is transmitted or ignored according to the bit-enable pattern in XCERB. The transmitter alternately uses XCERA and XCERB for consecutive 16-bit words.

Table 9–92 shows how bits of XCERA and XCERB correspond to the bits of a word written to DXR1 (MSB = most significant bit, LSB = least significant bit).

Table 9–92. Use of Transmit Channel Enable Registers A and B in the A-bis Mode

Register	Bits
XCERA	XCEA15: Enables or masks the MSB of the word written to DXR1 XCEA14: Enables or masks bit 14 of the word written to DXR1 XCEA13: Enables or masks bit 13 of the word written to DXR1 : XCEA0: Enables or masks the LSB of the word written to DXR1
XCERB	XCEB15: Enables or masks the MSB of the word written to DXR1 XCEB14: Enables or masks bit 14 of the word written to DXR1 XCEB13: Enables or masks bit 13 of the word written to DXR1 : XCEB0: Enables or masks the LSB of the word written to DXR1

9.14 McBSP Register Worksheet

This register worksheet is meant to be printed and used as a guide for configuring the McBSP registers. Each figure on the worksheet provides space in every register field for entering the binary value that will need to be loaded into that field. For read-only fields, you can use 0s or 1s. When all of the fields have been filled in, you can use the line above the register figure to record the corresponding hexadecimal value to load into the register during initialization.

9.14.1 General Control Registers

SPCR1 – Initialization Value: _____

15	14–13	12–11	10–8
DLB	RJUST	CLKSTP	Rsvd
			Read-only

7	6	5–4	3	2	1	0
DXENA	ABIS	RINTM	RSYNCERR	RFULL	RRDY	RRST_
				Read-only	Read-only	

SPCR2 – Initialization Value: _____

15–10	9	8
Rsvd	FREE	SOFT
Read-only		

7	6	5–4	3	2	1	0
FRST_	GRST_	XINTM	XSYNCERR	XEMPTY_	XRDY	XRST_
				Read-only	Read-only	

PCR – Initialization Value: _____

15	14	13	12	11	10	9	8
Rsvd	IDLE_EN	XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM

Read-only

7	6	5	4	3	2	1	0
SCLKME	CLKS_STAT	DX_STAT	DR_STAT	FSXP	FSRP	CLKXP	CLKRP

Read-only Read-only

RCR1 – Initialization Value: _____

15	14–8	7–5	4–0
Rsvd	RFRLEN1	RWDLEN1	Rsvd

Read-only Read-only

RCR2 – Initialization Value: _____

15	14–8	7–5	4–3	2	1–0
RPHASE	RFRLEN2	RWDLEN2	RCOMPAND	RFIG	RDATDLY

XCR1 – Initialization Value: _____

15	14–8	7–5	4–0
Rsvd	XFRLEN1	XWDLEN1	Rsvd
Read-only			Read-only

XCR2 – Initialization Value: _____

15	14–8	7–5	4–3	2	1–0
XPHASE	XFRLEN2	XWDLEN2	XCOMPAND	XFIG	XDATDLY

SRGR1 – Initialization Value: _____

15–8	7–0
FWID	CLKGDV

SRGR2 – Initialization Value: _____

15	14	13	12	11–0
GSYNC	CLKSP	CLKSM	FSGM	FPER

9.14.2 Multichannel Selection Control Registers

MCR1 – Initialization Value: _____

15–10	9	8–7	6–5	4–2	1	0
Rsvd	RMCME	RPBBLK	RPABLK	RCBLK	Rsvd	RMCM
Read-only				Read-only	Read-only	

MCR2 – Initialization Value: _____

15–10	9	8–7	6–5	4–2	1–0
Rsvd	XMCME	XPBBLK	XPABLK	XCBLK	XMCM
Read-only				Read-only	

RCERA – Initialization Value: _____

15	14	13	12	11	10	9	8
RCEA15	RCEA14	RCEA13	RCEA12	RCEA11	RCEA10	RCEA9	RCEA8
Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____

7	6	5	4	3	2	1	0
RCEA7	RCEA6	RCEA5	RCEA4	RCEA3	RCEA2	RCEA1	RCEA0
Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____

RCERB – Initialization Value: _____

15	14	13	12	11	10	9	8
RCEB15	RCEB14	RCEB13	RCEB12	RCEB11	RCEB10	RCEB9	RCEB8
Channel	Channel	Channel	Channel	Channel	Channel	Channel	Channel
_____	_____	_____	_____	_____	_____	_____	_____

7	6	5	4	3	2	1	0
RCEB7	RCEB6	RCEB5	RCEB4	RCEB3	RCEB2	RCEB1	RCEB0
Channel	Channel	Channel	Channel	Channel	Channel	Channel	Channel
_____	_____	_____	_____	_____	_____	_____	_____

RCERC – Initialization Value: _____

15	14	13	12	11	10	9	8
RCEC15	RCEC14	RCEC13	RCEC12	RCEC11	RCEC10	RCEC9	RCEC8
Channel 47	Channel 46	Channel 45	Channel 44	Channel 43	Channel 42	Channel 41	Channel 40

7	6	5	4	3	2	1	0
RCEC7	RCEC6	RCEC5	RCEC4	RCEC3	RCEC2	RCEC1	RCEC0
Channel 39	Channel 38	Channel 37	Channel 36	Channel 35	Channel 34	Channel 33	Channel 32

RCERD – Initialization Value: _____

15	14	13	12	11	10	9	8
RCED15	RCED14	RCED13	RCED12	RCED11	RCED10	RCED9	RCED8
Channel 63	Channel 62	Channel 61	Channel 60	Channel 59	Channel 58	Channel 57	Channel 56

7	6	5	4	3	2	1	0
RCED7	RCED6	RCED5	RCED4	RCED3	RCED2	RCED1	RCED0
Channel 55	Channel 54	Channel 53	Channel 52	Channel 51	Channel 50	Channel 49	Channel 48

RCERE – Initialization Value: _____

15	14	13	12	11	10	9	8
RCEE15	RCEE14	RCEE13	RCEE12	RCEE11	RCEE10	RCEE9	RCEE8
Channel 79	Channel 78	Channel 77	Channel 76	Channel 75	Channel 74	Channel 73	Channel 72

7	6	5	4	3	2	1	0
RCEE7	RCEE6	RCEE5	RCEE4	RCEE3	RCEE2	RCEE1	RCEE0
Channel 71	Channel 70	Channel 69	Channel 68	Channel 67	Channel 66	Channel 65	Channel 64

RCERF – Initialization Value: _____

15	14	13	12	11	10	9	8
RCEF15	RCEF14	RCEF13	RCEF12	RCEF11	RCEF10	RCEF9	RCEF8
Channel 95	Channel 94	Channel 93	Channel 92	Channel 91	Channel 90	Channel 89	Channel 88

7	6	5	4	3	2	1	0
RCEF7	RCEF6	RCEF5	RCEF4	RCEF3	RCEF2	RCEF1	RCEF0
Channel 87	Channel 86	Channel 85	Channel 84	Channel 83	Channel 82	Channel 81	Channel 80

RCERG – Initialization Value: _____

15	14	13	12	11	10	9	8
RCEG15	RCEG14	RCEG13	RCEG12	RCEG11	RCEG10	RCEG9	RCEG8
Channel 111	Channel 110	Channel 109	Channel 108	Channel 107	Channel 106	Channel 105	Channel 104

7	6	5	4	3	2	1	0
RCEG7	RCEG6	RCEG5	RCEG4	RCEG3	RCEG2	RCEG1	RCEG0
Channel 103	Channel 102	Channel 101	Channel 100	Channel 99	Channel 98	Channel 97	Channel 96

RCERH – Initialization Value: _____

15	14	13	12	11	10	9	8
RCEH15	RCEH14	RCEH13	RCEH12	RCEH11	RCEH10	RCEH9	RCEH8
Channel 127	Channel 126	Channel 125	Channel 124	Channel 123	Channel 122	Channel 121	Channel 120

7	6	5	4	3	2	1	0
RCEH7	RCEH6	RCEH5	RCEH4	RCEH3	RCEH2	RCEH1	RCEH0
Channel 119	Channel 118	Channel 117	Channel 116	Channel 115	Channel 114	Channel 113	Channel 112

XCERA – Initialization Value: _____

15	14	13	12	11	10	9	8
XCEA15	XCEA14	XCEA13	XCEA12	XCEA11	XCEA10	XCEA9	XCEA8
Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____

7	6	5	4	3	2	1	0
XCEA7	XCEA6	XCEA5	XCEA4	XCEA3	XCEA2	XCEA1	XCEA0
Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____	Channel _____

XCERB – Initialization Value: _____

15	14	13	12	11	10	9	8
XCEB15	XCEB14	XCEB13	XCEB12	XCEB11	XCEB10	XCEB9	XCEB8
Channel	Channel	Channel	Channel	Channel	Channel	Channel	Channel
_____	_____	_____	_____	_____	_____	_____	_____

7	6	5	4	3	2	1	0
XCEB7	XCEB6	XCEB5	XCEB4	XCEB3	XCEB2	XCEB1	XCEB0
Channel	Channel	Channel	Channel	Channel	Channel	Channel	Channel
_____	_____	_____	_____	_____	_____	_____	_____

XCERC – Initialization Value: _____

15	14	13	12	11	10	9	8
XCEC15	XCEC14	XCEC13	XCEC12	XCEC11	XCEC10	XCEC9	XCEC8
Channel 47	Channel 46	Channel 45	Channel 44	Channel 43	Channel 42	Channel 41	Channel 40

7	6	5	4	3	2	1	0
XCEC7	XCEC6	XCEC5	XCEC4	XCEC3	XCEC2	XCEC1	XCEC0
Channel 39	Channel 38	Channel 37	Channel 36	Channel 35	Channel 34	Channel 33	Channel 32

XCERD – Initialization Value: _____

15	14	13	12	11	10	9	8
XCED15	XCED14	XCED13	XCED12	XCED11	XCED10	XCED9	XCED8
Channel 63	Channel 62	Channel 61	Channel 60	Channel 59	Channel 58	Channel 57	Channel 56

7	6	5	4	3	2	1	0
XCED7	XCED6	XCED5	XCED4	XCED3	XCED2	XCED1	XCED0
Channel 55	Channel 54	Channel 53	Channel 52	Channel 51	Channel 50	Channel 49	Channel 48

XCERE – Initialization Value: _____

15	14	13	12	11	10	9	8
XCEE15	XCEE14	XCEE13	XCEE12	XCEE11	XCEE10	XCEE9	XCEE8
Channel 79	Channel 78	Channel 77	Channel 76	Channel 75	Channel 74	Channel 73	Channel 72

7	6	5	4	3	2	1	0
XCEE7	XCEE6	XCEE5	XCEE4	XCEE3	XCEE2	XCEE1	XCEE0
Channel 71	Channel 70	Channel 69	Channel 68	Channel 67	Channel 66	Channel 65	Channel 64

XCERF – Initialization Value: _____

15	14	13	12	11	10	9	8
XCEF15	XCEF14	XCEF13	XCEF12	XCEF11	XCEF10	XCEF9	XCEF8
Channel 95	Channel 94	Channel 93	Channel 92	Channel 91	Channel 90	Channel 89	Channel 88

7	6	5	4	3	2	1	0
XCEF7	XCEF6	XCEF5	XCEF4	XCEF3	XCEF2	XCEF1	XCEF0
Channel 87	Channel 86	Channel 85	Channel 84	Channel 83	Channel 82	Channel 81	Channel 80

XCERG – Initialization Value: _____

15	14	13	12	11	10	9	8
XCEG15	XCEG14	XCEG13	XCEG12	XCEG11	XCEG10	XCEG9	XCEG8
Channel 111	Channel 110	Channel 109	Channel 108	Channel 107	Channel 106	Channel 105	Channel 104

7	6	5	4	3	2	1	0
XCEG7	XCEG6	XCEG5	XCEG4	XCEG3	XCEG2	XCEG1	XCEG0
Channel 103	Channel 102	Channel 101	Channel 100	Channel 99	Channel 98	Channel 97	Channel 96

XCERH – Initialization Value: _____

15	14	13	12	11	10	9	8
XCEH15	XCEH14	XCEH13	XCEH12	XCEH11	XCEH10	XCEH9	XCEH8
Channel 127	Channel 126	Channel 125	Channel 124	Channel 123	Channel 122	Channel 121	Channel 120

7	6	5	4	3	2	1	0
XCEH7	XCEH6	XCEH5	XCEH4	XCEH3	XCEH2	XCEH1	XCEH0
Channel 119	Channel 118	Channel 117	Channel 116	Channel 115	Channel 114	Channel 113	Channel 112

Chapter 10

MMC Controller

If your TMS320C55x™ DSP supports MMC/SD communication, it contains two MMC controllers. With each MMC controller, the DSP can read from or write to the memory on a MultiMediaCard (an MMC) or a Secure Digital Memory Card (an SD card).

Topic **Page**

10.1 Introduction to the MMC Controller

Each of the two MMC controllers includes:

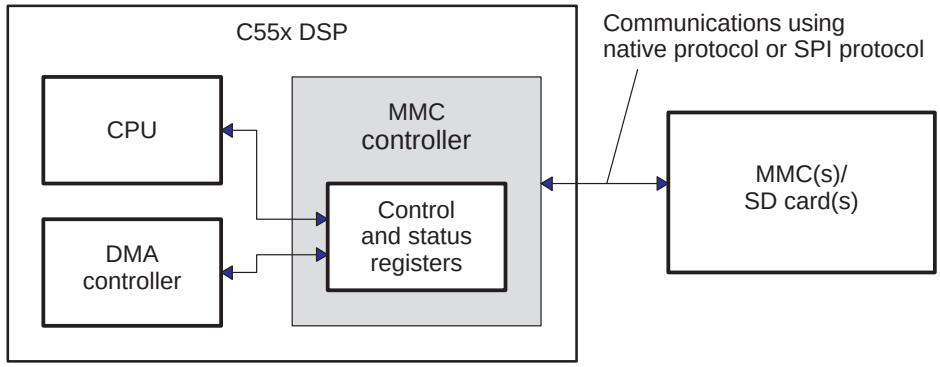
- ❏ Support for a MultiMediaCard (an MMC) or a Secure Digital Memory Card (an SD card)
- ❏ The capability to use either the MMC/SD protocol or the SPI protocol
- ❏ Software-oriented implementation for future extensions
- ❏ A programmable frequency for the operation of the MMC controller.
- ❏ A programmable frequency for the clock that controls the timing of transfers between the MMC controller and the memory card.

10.1.1 Role of the MMC Controller

As shown in Figure 10–1, the MMC passes data between the CPU or the DMA controller on one side and a memory card or cards on the other side. The CPU or the DMA controller can read from or write to the control and status registers in the MMC controller. As necessary, the CPU and/or the DMA controller can store or retrieve data in the DSP memory or in the registers of other peripherals. The MMC controller can notify the DMA controller of data activity with the two events described in section 10.1.5 on page 10-7.

The MMC controller initiates transfers between itself and the memory card(s). The communication follows either the controller's native protocol or a protocol that is based on the SPI standard. The native protocol can use one bidirectional data line (for MMCs) or four parallel data lines (for SD cards). The SPI protocol uses two serial data lines: one for storing data to the card and one for retrieving data from the card.

Figure 10–1. Role of the MMC Controller



10.1.2 MMC Controller Pins

Table 10–1 describes which of the seven pins of the MMC controller are used for each of the three types of communications.

The native protocol provides a clock signal (to time transfers on the other pins), a command line (for two-way communication with a controller on the memory card), and one or four data lines. MMC communications use one data line, for serial data. SD communications use four data lines, for 4-bit parallel data.

SPI communications use four pins: one for a clock, one for card selection, and two for data. The card select line is used to select one card to transmit or receive data. Additional cards may be selected with any of the DSP's general-purpose I/O (GPIO) pins. One of the data pins acts as the DataIn line, carrying serial data in to the selected card. The other data pin, the DataOut line, carries serial data out of the selected card.

Table 10–1. Summary of the MMC Controller Pins

		Function		
Pin	**Type†**	**MMC Communications**	**SD Communications**	**SPI Communications**
CLK	O	Clock line	Clock line	Clock line
CMD	I/O/Z	Command line	Command line	DataIn line (data to the card)
DAT0	I/O/Z	Data line 0	Data line 0	DataOut line (data from the card)
DAT1	I/O/Z	(Not used)	Data line 1	(Not used)
DAT2	I/O/Z	(Not used)	Data line 2	(Not used)
DAT3	I/O/Z	(Not used)	Data line 3	Card select line

† I = Input to the MMC controller; O = Output from the MMC controller; Z = High-impedance

10.1.3 Function Clock and Memory Clock

You must set the desired frequency for two clock signals in the MMC module: the function clock and the memory clock.

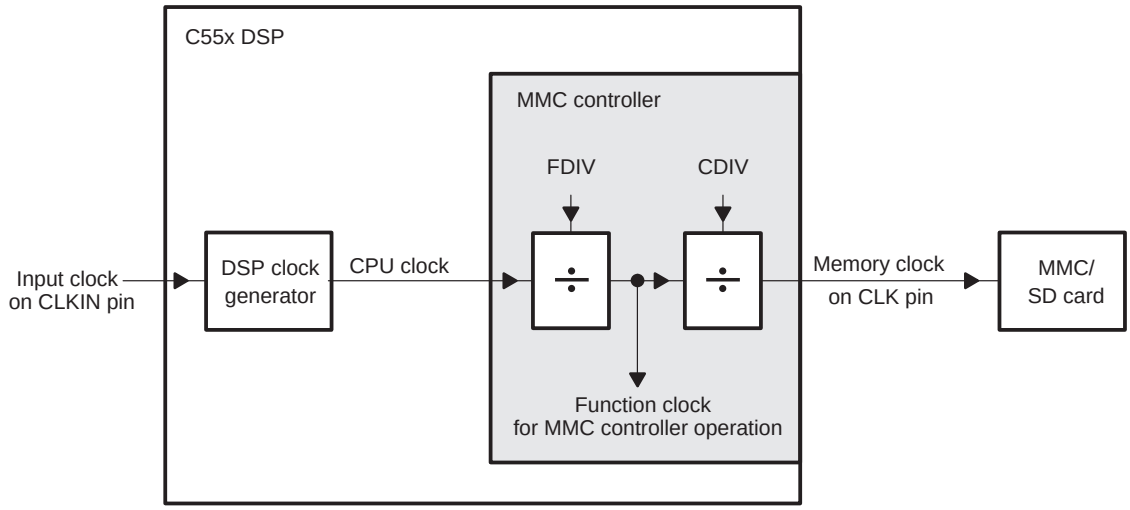
The **function clock** determines the frequency at which the MMC controller operates. Figure 10–2 shows the source of this clock. The DSP clock generator (described in Chapter 2) receives a signal from an external clock source and produces a CPU clock with a programmed frequency. A programmable clock divider in the MMC controller divides down the CPU clock to produce the function clock. To specify the divide-down value, initialize the FDIV field of the function clock control register, MMCFCLKn (see page 10-54). The resulting frequency is:

$$\text{function clock frequency} = \frac{\text{CPU clock frequency}}{(\text{FDIV} + 1)}$$

The **memory clock** appears on the CLK pin of the MMC controller interface. This clock controls the timing of communication between the MMC controller and the attached memory card(s). As shown in Figure 10–2, a second clock divider in the MMC controller divides down the function clock to produce the memory clock. Load the divide-down value into the CDIV field of the clock control register, MMCCLK (see page 10-54). The resulting frequency is:

$$\text{memory clock frequency} = \frac{\text{function clock frequency}}{(\text{CDIV} + 1)}$$

Figure 10–2. Clocking Diagram for the MMC Controller



10.1.4 Interrupt Activity in the MMC Controller

Each MMC module can generate the interrupt requests described in Table 10–2 and shown in Figure 10–3. When an interrupt event occurs, its flag bit is set in MMC status register 0 (MMCST0). If the corresponding enable bit is set in the MMC interrupt enable register (MMCIE), an interrupt request is generated. All such requests are multiplexed to a single MMC interrupt request for the CPU.

The MMC 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 (ISR). The ISR for the MMC interrupt can determine the event that caused the interrupt by checking the bits in MMC status register 0 (MMCST0). When the CPU reads MMCST0, all of the register's bits are automatically cleared except for DRRDY and DXRDY. DRRDY and DXRDY will remain set until your code explicitly clears them.

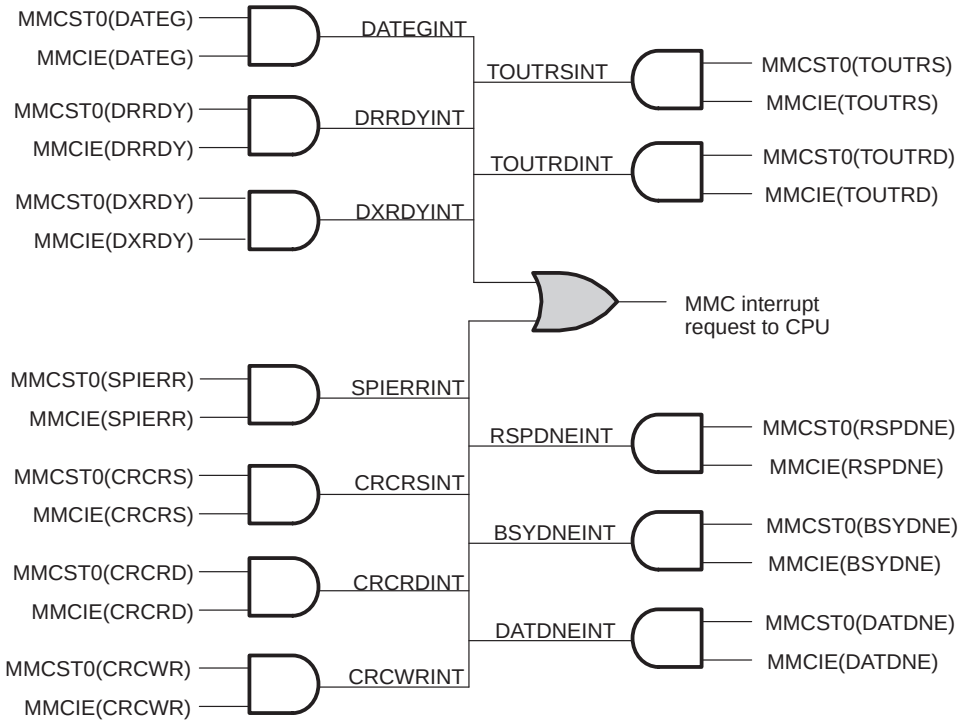
Table 10–2. Descriptions of the MMC Interrupt Requests

Interrupt Request	Interrupt Event
DATEGINT	An edge was detected on the DAT3 pin.
DRRDYINT	MMCDRR is ready to be read (data received).
DXRDYINT	MMCDXR is ready for new data (data transmitted).
SPIERRINT	A data error token was received during a transfer in the SPI mode.
CRCRSINT	A CRC error was detected in a response from the memory card.
CRCRDINT	A CRC error was detected while data was being read from the memory card.
CRCWRINT	A CRC error was detected while data was being written to the memory card.
TOUTRSINT	A time-out condition occurred while the MMC controller was waiting for a response to a command.
TOUTRDINT	A time-out condition occurred while the MMC controller was waiting for data from the memory card.

Table 10–2. Descriptions of the MMC Interrupt Requests (Continued)

Interrupt Request	Interrupt Source
RSPDNEINT	**For a command that requires a response:** The MMC controller has received the response without a CRC error. **For a command that does not require a response:** The MMC controller has finished sending the command.
BSYDNEINT	The memory card is no longer sending a busy signal.
DATDNEINT	**For read operations:** The MMC controller has received data without a CRC error. **For write operations:** The MMC controller has finished sending data.

Figure 10–3. Enable Paths of the MMC Interrupt Requests



10.1.5 DMA Events Generated by the MMC Controller

If the DMA event enable bit is set (DMAEN = 1 MMCCTL), each of the MMC controllers can generate the two DMA events described in Table 10–3. These events are sent to the DSP DMA controller, which is described in Chapter 3. Activity each DMA channel can be synchronized to respond to one of the two DMA events from the MMC controllers. For more details on channel synchronization, see section 3.8 on page 3-17.

Table 10–3. DMA Events Generated by the MMC Controller

DMA Event	Description
MMC receive event	New data is available to be read from the MMC data receive register (MMCDRR).
MMC transmit event	The MMC data transmit register (MMCDXR) is ready to accept new data for transmission.

10.1.6 Data Flow in the Data Registers (MMCDRR and MMCDXR)

The DSP (via the CPU or the DMA controller) reads 16 bits at a time from the data receive register (MMCDRR) and writes 16 bits at a time to the data transmit register (MMCDXR). However, the memory cards are 8-bit devices; they receive or transmit one byte at a time. Figure 10–4 and Figure 10–5 show how this data-size difference is handled via the data registers.

In most cases, once DRR is filled with two bytes, the MMC controller generates a data receive ready (DRRDY) event. If an odd number of bytes is received, the last byte is loaded into the low half of DRR and a DRRDY event is generated.

During transmission, the DSP typically loads 2 bytes to DXR. When the second byte leaves DXR, a data transmit ready (DXRDY) event is generated. If an odd number of bytes is transmitted, the DSP writes the last byte right aligned (see Figure 10–5). The transmission of the last byte causes a DXRDY event.

Figure 10–4. Data Flow in the Data Receive Register (MMCDRR)

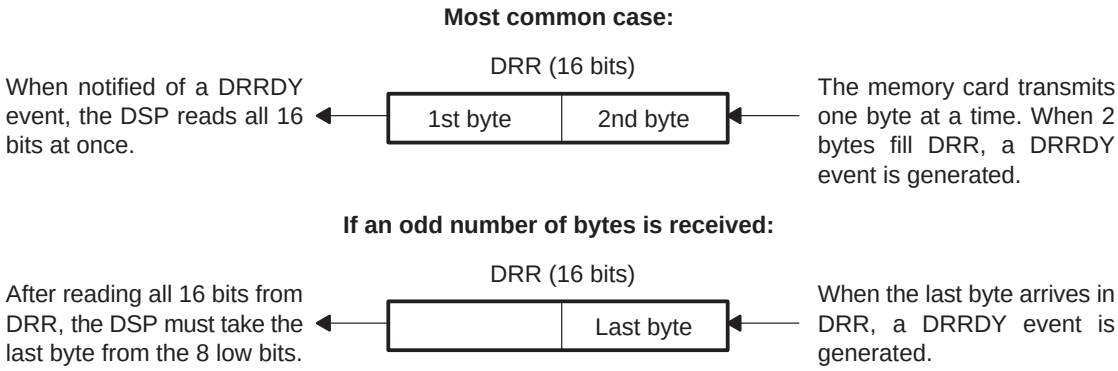
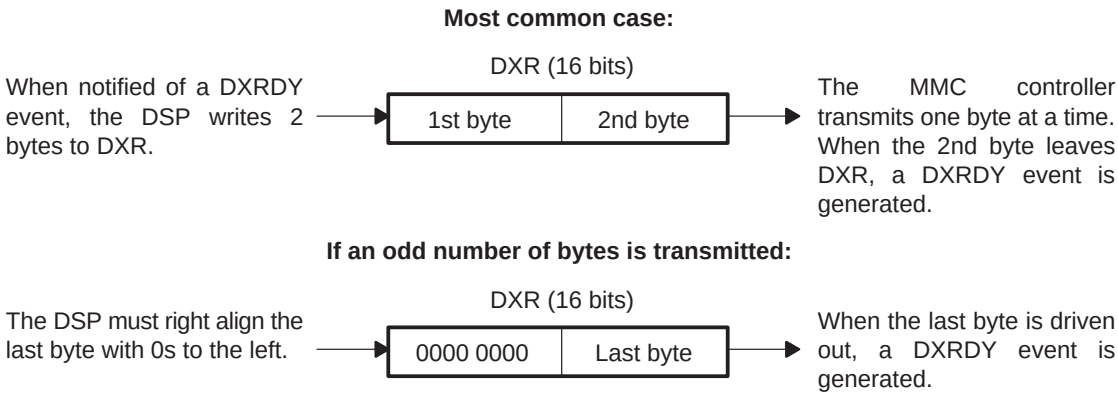


Figure 10–5. Data Flow in the Data Transmit Register (MMCDXR)



10.2 Native Mode

The interface of the MMC controller can use a native protocol or an SPI protocol (see section 10.5 on page 10-33). To use the native protocol, select the native mode by clearing the SPIEN bit of MMCCTL.

10.2.1 Native Mode Interface

Figure 10–6 summarizes the native mode interface.

In the native mode:

- The MMC controller supports one or more MultiMediaCards (MMCs) or Secure Digital Memory Cards (SD cards). If multiple cards are connected, the MMC controller selects one at a time by using an identification broadcast on the data line.
- The following MMC controller pins are used:
  - CMD: This pin is used for two-way control communication with the memory card or cards connected to the interface. On CMD, the MMC controller drives commands followed by arguments, and the memory card drives responses to the commands.
  - DAT0 or DAT[3:0]: Only one data line (DAT0) is used for an MMC. Four data lines are needed for an SD card. You configure the number of DAT pins (the data bus width) when you initialize the WIDTH bit of MMCCTL. For a comparison of the MMC and SD configurations, see Figure 10–7 (page 10-11).
  - CLK: This pin provides a clock signal to time transfers between the controller and the memory card(s).
- Because the command and data lines are separate, the next command to the card may be sent at the same time as data associated with the previous command. Thus, the native mode allows sequential and multiple-block read/write operations.

Figure 10–6. Native Mode Interface

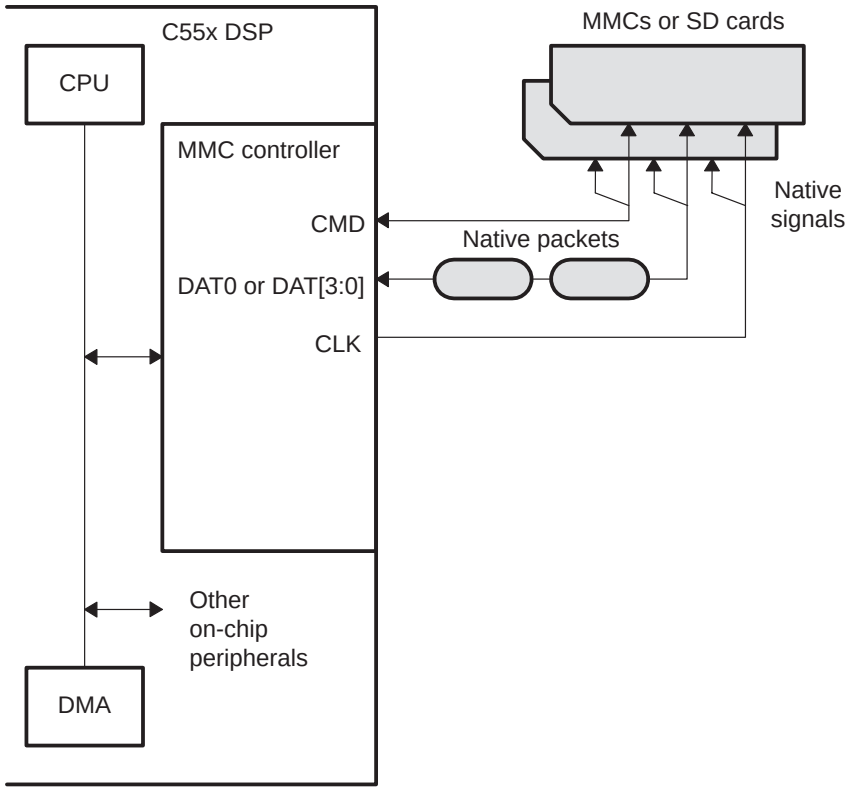
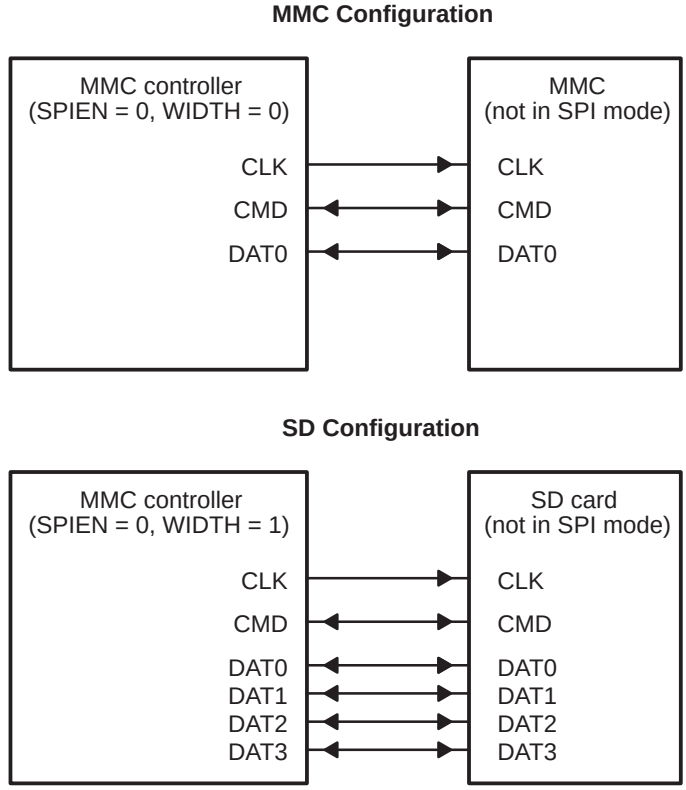


Figure 10–7. MMC Configuration Versus SD Configuration



10.2.1.1 Native Mode Write Sequence

Figure 10–8 and Table 10–4 describe the signal activity when the MMC controller is in the native mode and is writing data to a memory card. The same block length must be defined in the MMC controller and in the card. In a successful write sequence:

1) The controller sends a write command to the card.

2) The card sends a response to acknowledge the command.

3) The controller sends a block of data to the card.

4) The card sends the CRC status to the controller.

5) The card sends low BUSY bits until all the data has been programmed into the flash memory inside the card.

Figure 10–8. Native Mode Write Sequence

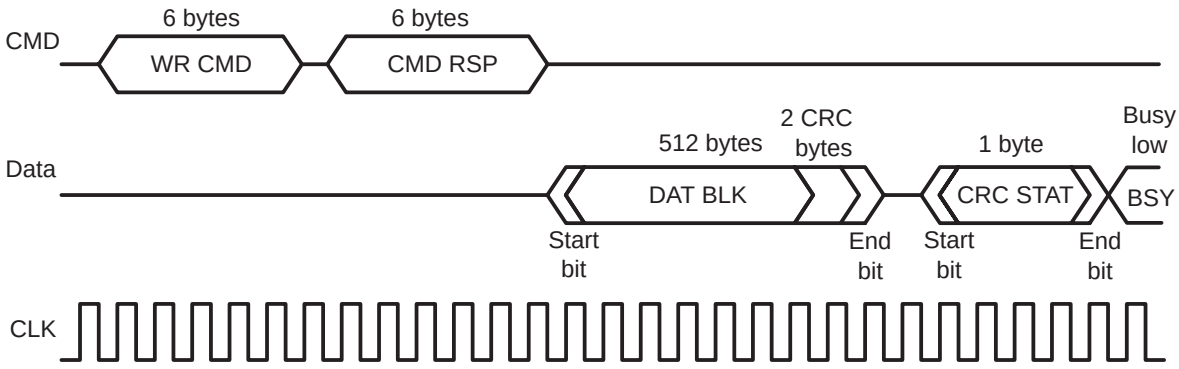


Table 10–4. Native Mode Write Sequence

Portion of the Sequence	Description
WR CMD	Write command. A 6-byte WRITE_BLOCK command token is sent from the DSP to the card.
CMD RSP	Command response. The card sends to the DSP a 6-byte response of type R1 to acknowledge the WRITE_BLOCK command.
DAT BLK	Data block. The DSP writes a block of data to the card. The data content is preceded by one start bit and is followed by two CRC bytes and one end bit.
CRC STAT	CRC status. The card sends to the DSP a one byte of CRC status information, which indicates to the DSP whether the data has been accepted by the card or rejected due to a CRC error. The CRC status content is preceded by one start bit and followed by one end bit.
BSY	Busy bits. The CRC status information is followed by a continuous stream of low busy bits until all of the data has been programmed into the flash memory on the card.

10.2.1.2 Native Mode Read Sequence

Figure 10–9 and Table 10–5 describe the signal activity when the MMC controller is in the native mode and is reading data from a memory card. The same block length must be defined in the MMC controller and in the card. In a successful read sequence:

1) The controller sends a read command to the card.

2) The card sends a response to acknowledge the command.

3) The card sends a block of data to the DSP.

Figure 10–9. Native Mode Read Sequence

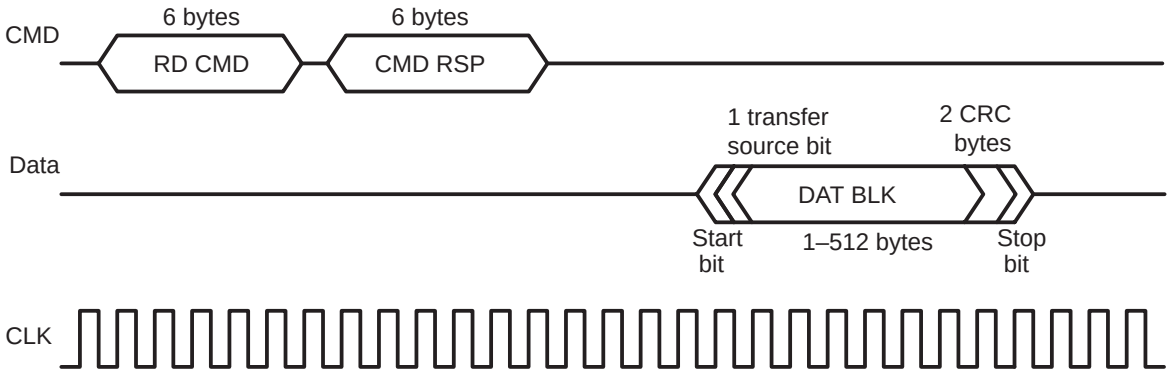


Table 10–5. Native Mode Read Sequence

Portion of the Sequence	Description
RD CMD	Read command. A 6-byte READ_SINGLE_BLOCK command token is sent from the DSP to the card.
CMD RSP	Command response. The card sends to the DSP a 6-byte response of type R1 to acknowledge the READ_SINGLE_BLOCK command.
DAT BLK	Data block. The card sends a block of data to the DSP. The data content is preceded by a start bit and then a transfer source bit. The data content is followed by 2 CRC bytes and then a stop bit.

10.2.2 Card Identification Operation

Before the MMC controller can start data transfers to or from memory cards in the native mode, it has to first identify how many cards are present on the bus and configure them.

For each card that responds to the ALL_SEND_CID broadcast command, the controller reads that card's unique card identification address (CID) and then assigns it a relative address (RCA). This address is much shorter than the CID and is used by the controller to identify the card in all future commands that involve the card.

Only one card completes the response to ALL_SEND_CID at any one time. The absence of any response to ALL_SEND_CID indicates that all cards have been identified and configured.

The procedure for a card identification operation is as follows (see also Figure 10–10):

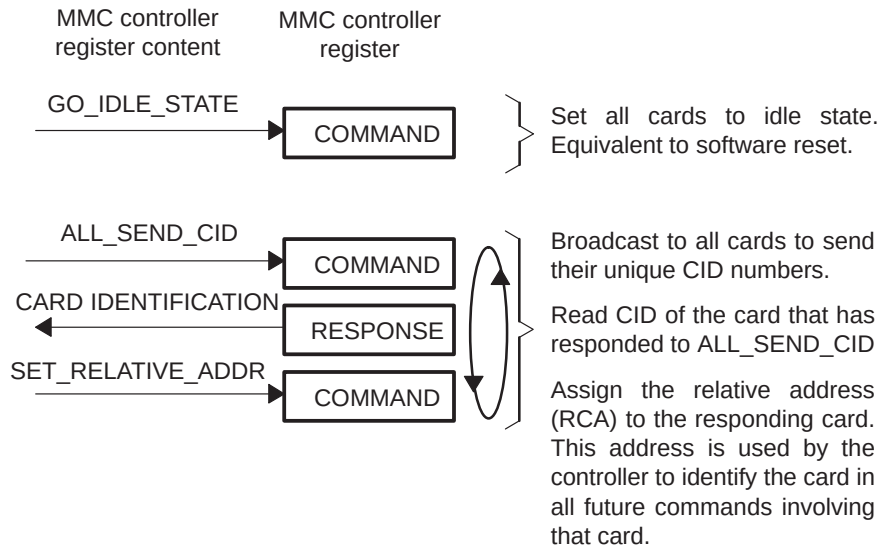
1) Use the command register (MMCCMD) to send the GO_IDLE_STATE broadcast command to the cards. This puts all cards in the idle state.

2) Use MMCCMD to send the ALL_SEND_CID broadcast command to the cards. This tells all cards to identify themselves.

3) Wait for a card to respond. If a card responds, go to step 4. Otherwise, stop.

4) Read the CID from the response registers (MMCRSP7–MMCRSP0), and assign a relative address to the card by sending the SET_RELATIVE_ADDR command.

Figure 10–10. Card Identification (Native Mode)



10.2.3 Native Mode Single-Block Write Operation

In the native mode, to write a single block of data to a memory card, use the following procedure. Figure 10–11 illustrates this procedure with some additional details. Note that the data block length must be 512 bytes, and the same block length must be defined in the MMC controller and in the card.

1) Write the card's relative address to the argument registers. Load the high part of the address to MMCARGH and the low part of the address to MMCARGL.

2) Use the command register (MMCCMD) to send a SELECT/DESELECT_CARD broadcast command. This selects the addressed card and deselects the others.

3) Write the destination start address to the argument registers. Load the high part to MMCARGH and the low part to MMCARGL.

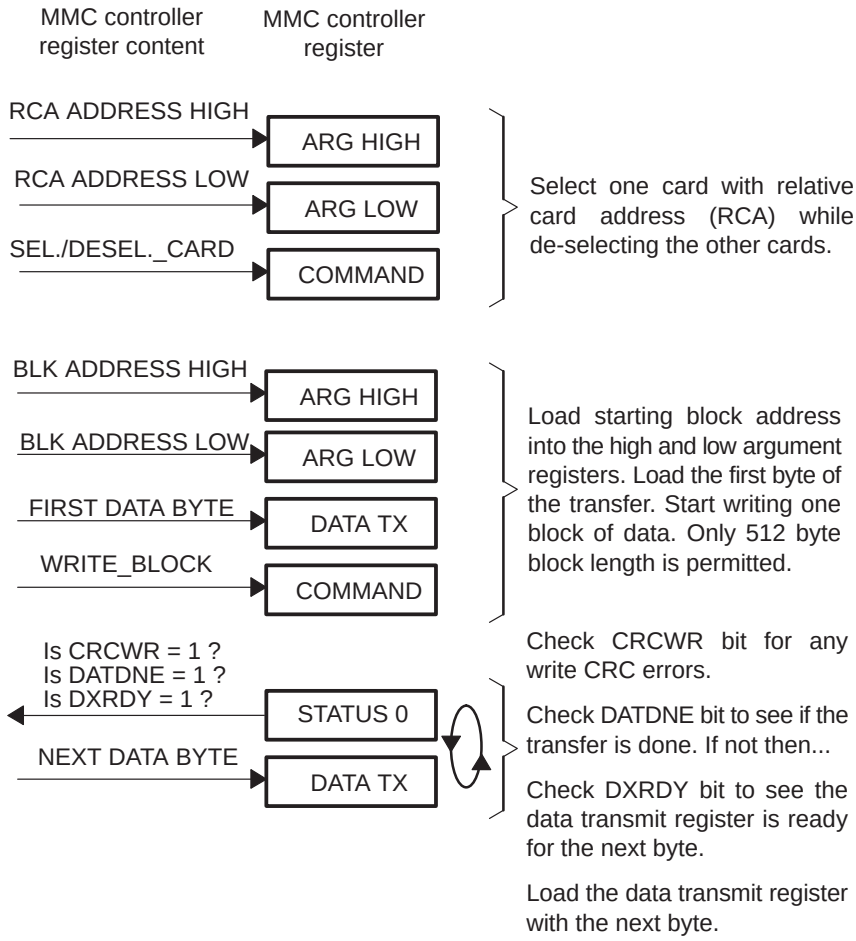
4) Write the first byte of the data block to the data transmit register (MMCDXR).

5) Send the WRITE_BLOCK command to the card.

6) Use status register 0 to check for errors and to determine when the byte has been successfully transmitted. If all of the data has not been written and if the previous byte has been transmitted, go to step 7. If all of the data has been written, stop.

7) Write the next byte of the data block to MMCDXR, and go to step 6.

Figure 10–11. Native Mode Single-Block Write Operation



10.2.4 Native Mode Single-Block Read Operation

In the native mode, to read a single block of data from a memory card, use the following procedure. The procedure is also illustrated in Figure 10–12 with some additional details. The same block length must be defined in the MMC controller and in the card.

1) Write the card's relative address to the argument registers. Load the high part of the address to MMCARGH and the low part of the address to MMCARGL.

2) Send a SELECT/DESELECT_CARD broadcast command via the command register (MMCCMD). This selects the addressed card and deselects the others.

3) Write the source start address to the argument registers. Load the high part to MMCARGH and the low part to MMCARGL.

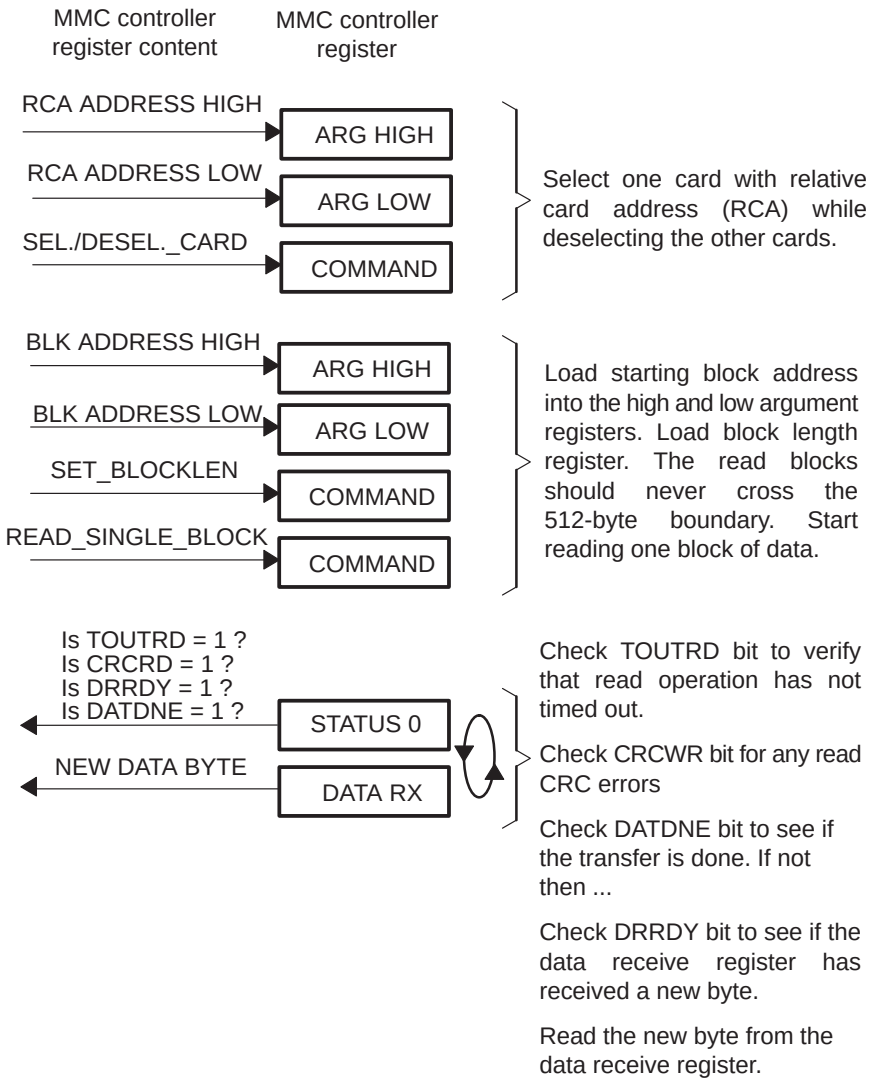
4) Send a SET_BLOCKLEN command via MMCCMD (if the block length is different than the length used in the previous operation).

5) Send a READ_SINGLE_BLOCK command via MMCCMD.

6) Use status register 0 to check for errors and to determine when a new byte has been successfully received. If all of the data has not been read and if a new byte has been received, go to step 7. If the all of the data has been read, stop.

7) Read the new byte of data from the data receive register (MMCDRR), and go to step 6.

Figure 10–12. Native Mode Single-Block Read Operation



10.2.5 Native Mode Multiple-Block Read Operation

In the native mode, to read a multiple blocks of data from a memory card, see the following procedure. The procedure is also illustrated in Figure 10–13 with some additional details. The same block length must be defined in the MMC controller and in the card.

Note:

The procedure in this section uses a STOP_TRANSMISSION command to end the block transfer. This assume that the value in the number-of-blocks register (MMCNBLK) is 0. A multiple-block operation can terminate itself if you load MMCNBLK with the exact number of blocks you want transferred.

1) Write the card's relative address to the argument registers. Load the high part of the address to MMCARGH and the low part of the address to MMCARGL.

2) Send a SET_BLOCKLEN command via MMCCMD (if the block length is different than the length used in the previous operation). The block length must be 512 bytes.

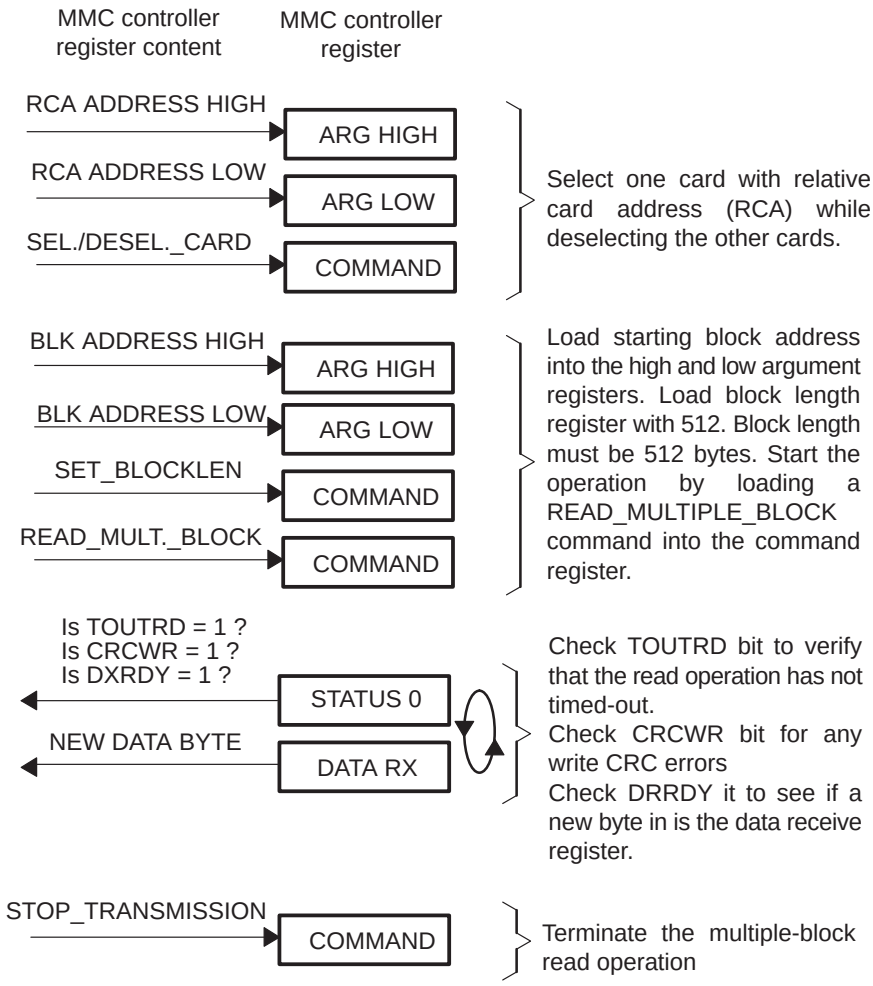
3) Send a READ__MULT_BLOCKS command via MMCCMD.

4) Use status register 0 to check for errors and to determine when a new byte has been successfully received. If more bytes are to be read, go to step 5. If the all of the data has been read, go to step 6.

5) Read the new byte of data from the data receive register (MMCDRR), and go to step 4.

6) Send a STOP_TRANSMISSION command via MMCCMD.

Figure 10–13. Native Mode Multiple-Block Read Operation



10.3 Native Mode Initialization

The general procedure for initializing the MMC controller is given in the following steps. Details about the registers or register bit fields to be configured in the native mode are in the subsections that follow the procedure.

1) Place the MMC controller in its reset state by setting MMCCTL(CMDRST) and MMCCTL(DATRST). With the same register write operation, write the desired values to other bits in MMCCTL.

2) Write to other registers to complete the MMC controller configuration.

3) Clear MMCCTL(CMDRST) and MMCCTL(DATRST) to release the MMC controller from its reset state. Make sure you do not change the values you wrote to the other bits of MMCCTL in step 1.

4) Enable the CLK pin so that the memory clock is sent to the memory card.

10.3.1 Initializing the MMC Control Register (MMCCTL)

When the MMC controller for the native mode, the bit fields named in Figure 10–14 affect the operation of the controller. The subsections that follow the figure help you decide how to initialize each of the fields.

Figure 10–14. MMCCTL Fields Used During Native Mode Initialization

15	14	13	12	11	10	9	8
							DMAEN
							R/W–0

7	6	5	4	3	2	1	0
DATEG		SPIEN			WIDTH	CMDRST	DATRST
R/W–0		R/W–0			R/W–0	R/W–0	R/W–0

Note: R/W-x = Read/Write-Reset value

10.3.1.1 Enable/Disable DMA Events

Register(Field)	*symval*	Value	Description
MMCCTL(DMAEN)		0	Disable DMA events.
		1	Enable DMA events.

Use DMAEN to disable for enable the MMC controller DMA events, which are described in section 10.1.5 on page 10-7.

10.3.1.2 Select a Type of Edge Detection (If Any) for the DAT3 Pin

Register(Field)	*symval*	Value	Description
MMCCTL(DATEG)		00	Disable DAT3 edge detection.
		01	Enable DAT3 rising edge detection.
		10	Enable DAT3 falling edge detection.
		11	Enable DAT3 dual edge detection (detect both edges).

The DATEG control bit of MMCCTL enables or disables general-purpose edge detection on the DAT3 pin. If you enable edge detection and an edge is detected, the DATEG flag bit of MMCST0 is set. In addition, if DATEG = 1 in MMCIE, an interrupt request is generated.

10.3.1.3 Enable Native Mode

Register(Field)	*symval*	Value	Description
MMCCTL(SPIEN)		0	Enable native mode./Disable SPI mode.
		1	Disable native mode./Enable SPI mode.

The SPIEN bit of MMCCTL determines whether the SPI mode is on. To enable the native mode, turn the SPI mode off (SPIEN = 0).

10.3.1.4 Select a Data Bus Width

Register(Field)	*symval*	Value	Description
MMCCTL(WIDTH)		0	1-bit data bus (DAT0 pin)
		1	4-bit data bus (pins DAT0–DAT3)

In the native mode, the MMC controller must know how wide the data bus must be for the memory card that is connected. If an MMC is connected, specify a 1-bit data bus (WIDTH = 0). If an SD card is connected, specify a 4-bit data bus (WIDTH = 1).

10.3.1.5 Reset/Enable the MMC Controller

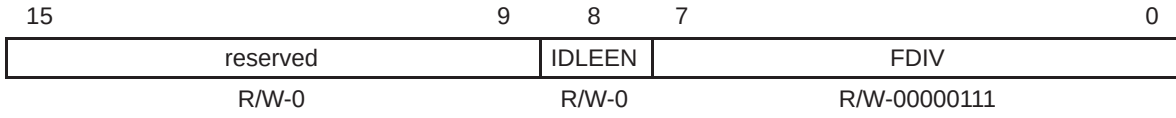
Register(Field)	*symval*	Value	Description
MMCCTL(CMDRST)		0	Enable the CMD (command) logic of the MMC controller.
		1	Place the CMD logic of the MMC controller in its reset state.
MMCCTL(DATRST)		0	Enable the DAT (data) logic of the MMC controller.
		1	Place the DAT logic of the MMC controller in its reset state.

To place the MMC controller in its reset state and disable it, set the CMDRST and DATRST bits of MMCCTL. The first step of the MMC controller initialization process is to disable both sets of logic. When initialization is complete but before you enable the CLK pin, enable the MMC controller by clearing the CMDRST and DATRST bits.

10.3.2 Initializing the Clock Control Registers (MMCFCLK and MMCCLK)

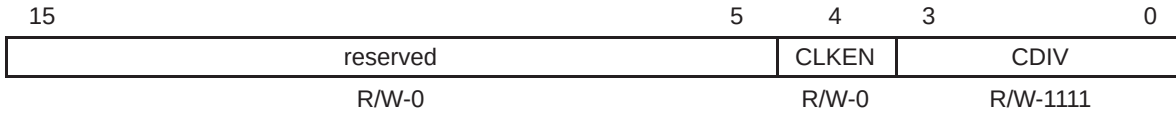
Figure 10–15 and Figure 10–16 show the bit fields in the function clock control register (MMCFCLK) and the memory clock control register (MMCCLK). The following subsections describe how decide on initialization values for these fields.

Figure 10–15. MMCFCLK



Note: R/W-x = Read/Write-Reset value

Figure 10–16. MMCCLK



Note: R/W-x = Read/Write-Reset value

10.3.2.1 Set the Function Clock and the Memory Clock

Register(Field)	*symval*	Value	Description
MMCFCLK(FDIV)		0–255	Divide-down value for the function clock.
MMCCLK(CDIV)		0–15	Divide-down value for the memory clock.

To generate the function clock (the clock for activity inside the MMC controller), the MMC controller divides down the CPU clock as shown in the following equation. When you initialize MMCFCLK, you specify FDIV, a divide-down value in the range 0 through 255.

$$\text{function clock frequency} = \frac{\text{CPU clock frequency}}{(\text{FDIV} + 1)}$$

The memory clock (the clock for the attached memory card) is a divided-down version of the function clock; see the following equation. When you initialize MMCCLK, you specify CDIV, a divide-down value in the range 0 through 15.

$$\text{memory clock frequency} = \frac{\text{function clock frequency}}{(\text{CDIV} + 1)}$$

For more information about the function clock and the memory clock , see section 10.1.3 on page 10-4.

10.3.2.2 Determine Whether the Function Clock Stops in Response to an IDLE Instruction

Register(Field)	*symval*	Value	Description
MMCFCLK(IDLEEN)		0	The MMC controller cannot be made idle.
		1	If PERI = 1 (see just below), the MMC controller is idle (the function clock is stopped) after the IDLE instruction is executed.
ICR(PERI)		0	Any peripheral in the PERIPH idle domain will be active after the IDLE instruction is executed.
		1	Any peripheral in the PERIPH idle domain can be idle after the IDLE instruction is executed, depending on the state of that peripheral's idle enable bit.

The MMC controller is one of the peripheral devices in the PERIPH idle domain. For details on controlling the various idle domains of the DSP, see Chapter 8, *Idle Configurations*. If you want the MMC controller to go idle in response to an IDLE instruction, make the following preparations:

1) Write 1 to the idle enable (IDLEEN) bit in MMCFCLK. This tells the DSP to stop the function clock of the MMC controller when the PERIPH domain becomes idle.

2) Write a 1 to the PERI bit in ICR (see page 8.7). This tells the DSP to make the PERIPH domain idle when an IDLE instruction is executed.

10.3.2.3 Enable/Disable the CLK Pin

Register(Field)	*symval*	Value	Description
MMCCLK(CLKEN)		0	Disable the CLK pin; drive a constant, low signal on the pin.
		1	Enable the CLK pin, so that it reflects the memory clock signal.

The CLKEN bit determines whether the memory clock appears on the CLK pin.

10.3.3 Initialize the Interrupt Enable Register (MMCIE)

Register(Field)	*symval*	Value	Description
MMCIE(11–0)		000h–FFFh	Determines which of the MMC interrupt requests will be forwarded to the CPU.

The bits in MMCIE individually enable or disable the interrupt requests described in section 10.1.4 on page 10-5. Figure 10–17 shows the bit fields of MMCIE apply to the native mode. Set one of these bits to enable the associated interrupt request. Clear one of these bit to disable the associated interrupt request. Load 0s into the bits not used in the native mode.

Figure 10–17. MMCIE Fields Used in the Native Mode

15	14	13	12	11	10	9	8
				DATEG	DRRDY	DXRDY	
				R/W–0	R/W–0	R/W–0	

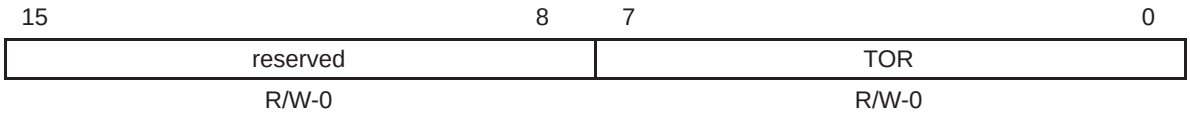
7	6	5	4	3	2	1	0
CRCRS	CRCRD	CRCWR	TOUTRS	TOUTRD	RSPDNE	BSYDNE	DATDNE
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

Note: R/W-x = Read/Write-Reset value

10.3.4 Initialize the Time-Out Registers (MMCTOR and MMCTOD)

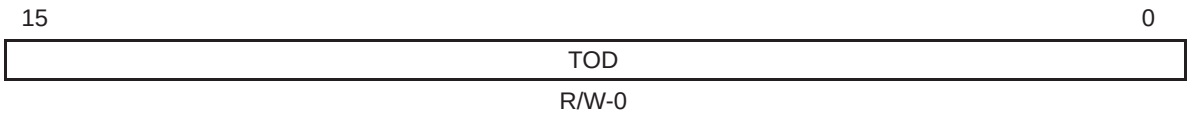
Specify the time-out period for responses (TOR, see Figure 10–18) and the time-out period for read data (TOD, see Figure 10–19) as described in the following subsections.

Figure 10–18. MMCTOR



Note: R/W-x = Read/Write-Reset value

Figure 10–19. MMCTOD



Note: R/W-x = Read/Write-Reset value

10.3.4.1 Set the Time-Out Period for a Response

Register(Field)	*symval*	Value	Description
MMCTOR(7–0)		0	Do not check for a response time-out condition.
		n = 1–255	If there is no response from the memory card in n CLK cycles, record a time-out condition.

When the MMC controller sends a command a memory card, it often must wait for a response. The controller can wait indefinitely or for up to 255 memory clock cycles. If you load 0 into MMCTOR during initialization, the controller will wait for a response indefinitely. If you load a nonzero value into MMCTOR, the controller automatically stops waiting after the specified number of cycles and then records a response time-out condition. If the associated interrupt request is enabled, the controller also sends an interrupt request to the CPU.

10.3.4.2 Set the Time-Out Period for a Data Read Operation

Register(Field)	*symval*	Value	Description
MMCTOD(15–0)		0	Do not check for a data-read time-out condition.
		n = 1–65535	If no data is received from the memory card in n CLK cycles, record a time-out condition.

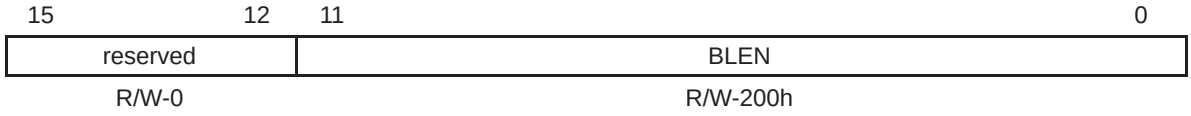
When the MMC controller requests data from a memory card, it can wait indefinitely for that data, or it can stop waiting after a programmable number of cycles. If you load 0 into MMCTOD during initialization, the controller will wait indefinitely. If you load a nonzero value *n* into MMCTOD, the controller will wait

n memory clock cycles and then record a data-read time-out condition in MMCST0. If the associated interrupt request is enabled, the controller also sends an interrupt request to the CPU.

10.3.5 Initialize the Data Block Registers (MMCBLEN and MMCNBLK)

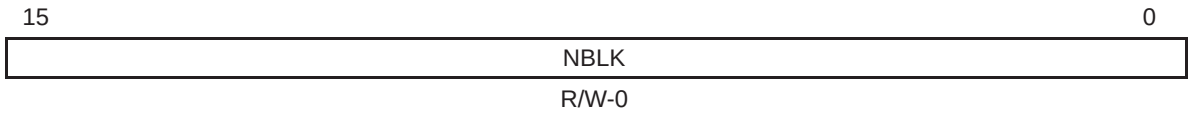
Specify the number of bytes in a data block (MMCBLEN, see Figure 10–20) and the number of blocks in a multiple-block transfer (MMCNBLK, see Figure 10–21). Details about these values are in the following subsections.

Figure 10–20. MMCBLEN



Note: R/W-x = Read/Write-Reset value

Figure 10–21. MMCNBLK



Note: R/W-x = Read/Write-Reset value

10.3.5.1 Set the Data Block Length

Register(Field)	*symval*	Value	Description
MMCBLEN(11–0)		1–512	Number of bytes in a data block

In MMCBLEN, you must define the size for each block of data transferred between the MMC controller and a memory card. The valid size depends on the type of read/write operation. A length of 0 bytes is prohibited.

10.3.5.2 If Necessary, Specify the Number of Blocks in a Multiple-Block Transfer

Register(Field)	*symval*	Value	Description
MMCNBLK(15–0)		0	Transfer an infinite number of blocks.
		n = 1–65535	Transfer n blocks.

For multiple-block transfers, you must specify how many blocks of data are to be transferred between the MMC controller and a memory card. You can specify an infinite number of blocks by loading 0 into MMCNBLK. When MMCNBLK = 0, the MMC controller transfer blocks until you end the transferring with a STOP_TRANSMISSION command. If you need a specific number of blocks transferred, load MMCNBLK with a value from 1 through 65535.

10.4 Monitoring Activity in the Native Mode

This section describes registers and specific register bits that you can use to obtain the status of the MMC controller in the native mode.

10.4.1 Detecting Edges and Level Changes on the DAT3 Pin

Register(Field)	*symval*	Value	Description
MMCST0(DATEG)		0	No edge detected on DAT3 pin
		1	Edge detected on DAT3 pin
MMCST1(DAT)		0	Low signal level on DAT3
		1	High signal level on DAT3

Detecting edges. The MMC controller sets the DATEG flag of status register 0 (MMCST0) if DAT3 edge detection is enabled (DATEG is nonzero in MMCCTL) and the specified edge is detected. The CPU can also be notified of the DAT3 by an interrupt if you enable the interrupt request in the interrupt enable register (DATEG = 1 in MMIE).

Detecting level changes. The DAT bit of status register 1 tracks the signal level on the DAT3 pin.

10.4.2 Determining Whether New Data is Available in MMCDRR

Register(Field)	*symval*	Value	Description
MMCST0(DRRDY)		0	MMCDRR not ready.
		1	MMCDRR ready. New data has arrived and can be read by the CPU or by the DMA controller.

The MMC controller sets the DRRDY flag of MMCST0 when new data arrives in the data receive register (MMCDRR). The CPU can also be notified of the event by an interrupt if you enable the interrupt request (DRRDY = 1 in MMIE).

10.4.3 Verifying That MMCDXR is Ready to Accept New Data

Register(Field)	*symval*	Value	Description
MMCST0(DXRDY)		0	MMCDXR not ready.
		1	MMCDXR ready. The data in MMCDXR has been transmitted; MMCDXR can accept new data from the CPU or from the DMA controller.

The MMC controller sets the DXRDY flag of MMCST0 when data leaves the data transmit register (DXRDY). The CPU can also be notified of the event by an interrupt if you enable the interrupt request (DXRDY = 1 in MMIE).

10.4.4 Checking for CRC Errors

Register(Field)	*symval*	Value	Description
MMCST0(CRCRS)		0	No response CRC error
		1	Response CRC error detected
MMCST0(CRCRD)		0	No read-data CRC error
		1	Read-data CRC error detected
MMCST0(CRCWR)		0	No write-data CRC error
		1	Write-data CRC error detected

The MMC controller sets one of these flags in response to the corresponding CRC error. The CPU can also be notified of the CRC error by an interrupt if you enable the interrupt request (CRCRS/CRCRD/CRCWR = 1 in MMIE).

10.4.5 Checking for Time-Out Events

Register(Field)	*symval*	Value	Description
MMCST0(TOUTRS)		0	No response time-out event
		1	Response time-out event detected
MMCST0(TOUTRD)		0	No read-data time-out event
		1	Read-data time-out event detected

The MMC controller sets one of these flags in response to a the corresponding time-out event. The CPU can also be notified of the time-out event by an interrupt if you enable the interrupt request (TOUTRS/TOUTRD = 1 in MMIE).

10.4.6 Determining When a Response/Command is Done

Register(Field)	*symval*	Value	Description
MMCST0(RSPDNE)			**If the command requires a response:**
		0	Response not done
		1	Response fully received with no CRC error
			If no response required:
		0	Command not done
		1	Command has been sent

The MMC controller sets the RSPDNE flag when the response is done (or, in the case of commands that do not require a response, when the command is done). The CPU can also be notified of the done condition by an interrupt if you enable the interrupt request (RSPDNE = 1 in MMIE).

10.4.7 Determining Whether the Memory Card is Busy

Register(Field)	*symval*	Value	Description
MMCST0(BSYDNE)		0	The memory card is busy.
		1	The memory card is no longer sending a busy signal.
MMCST1(BUSY)		0	The memory card has not sent a busy signal.
		1	The memory card is busy.

The card sends a busy signal either as an expected part of an R1b response or to indicate that the card is still programming the last write data into its flash memory. The MMC controller has two flags to tell you whether the memory card is sending a busy signal. The two flags are complements of each other:

- ❏ BSYDNE is set if the card did not send or is not sending a busy signal. As with the other bits in status register 0, this bit has an associated interrupt that you can enable (BSYDNE = 1 in MMCIE).
- ❏ BUSY is set when a busy signal is received from the card.

10.4.8 Determining Whether a Data Transfer is Done

Register(Field)	*symval*	Value	Description
MMCST0(DATDNE)			**When reading from memory card:**
		0	Read operation not done
		1	Data fully received with no CRC error
			When writing to memory card:
		0	Write operation not done
		1	Data fully transmitted

The MMC controller sets the DATDNE flag when all the bytes of a data transfer have been transmitted/received. You can poll this bit to determine when to stop writing to the data transmit register (for a write operation) or when to stop reading from the data receive register (for a read operation). The CPU can also be notified of the time-out event by an interrupt if you enable the interrupt request (TOUTRS/TOUTRD = 1 in MMIE).

10.4.9 Checking For a Data Transmit Empty Condition

Register(Field)	*symval*	Value	Description
MMCST1(DXEMP)		0	No data transmit empty condition
		1	Data transmit empty condition

Typically, this bit is not used to control data transfers; rather, it is checked during recovery from an error condition. There is no interrupt associated with the transmit empty condition.

During transmission, a data value is passed from the data transmit register (MMCDXR) to the data transmit shift register. Then value is passed from this shift register to the memory card one bit at a time. The DXEMP bit indicates when this shift register is empty; there are no bits available to shift out to the memory card.

10.4.10 Checking for a Data Receive Full Condition

Register(Field)	*symval*	Value	Description
MMCST1(DRFUL)		0	No data receive full condition
		1	Data receive full condition

Typically, this bit is not used to control data transfers; rather, it is checked during recovery from an error condition. There is no interrupt associated with the data receive full condition.

During reception, the data receive shift register accepts a data value one bit at a time. Then the whole value is passed from this shift register to the data receive register (MMCDRR). The DRFUL bit indicates when this shift register is full; no new bits can be shifted in from the memory card. Typically, this bit is used only during recovery from the error condition.

10.4.11 Checking the Status of the CLK Pin

Register(Field)	*symval*	Value	Description
MMCST1(CLKSTP)		0	CLK is active. The memory clock signal is being driven on the pin.
		1	CLK is held low. Possible reasons: Manual stop (CLKEN = 0), data receive full condition, or data transmit empty condition.

Read CLKSTP to determine whether the memory clock has been stopped on the CLK pin.

10.4.12 Getting the Remaining Block Count During a Multiple-Block Transfer

Register(Field)	*symval*	Value	Description
MMCNBLC(15–0)		n = 1–65535	There are n blocks left to be transferred.

During a the transfer of multiple data blocks, the block counter (MMCNBLC) tells you how many blocks are left to be transferred.

10.5 SPI Mode

The interface of the MMC controller can use a native (MMC/SD) protocol or an SPI protocol. To use the SPI protocol, select the SPI mode by setting the SPIEN bit of MMCCTL. In addition, make sure the memory card is in its SPI mode.

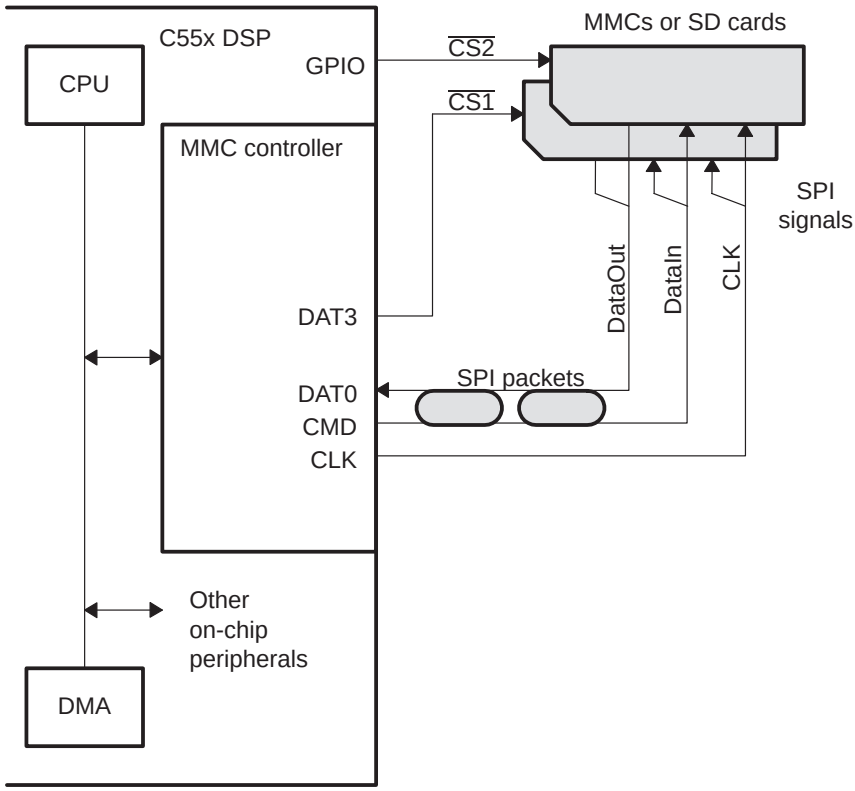
10.5.1 SPI Mode Interface

Figure 10–6 summarizes the SPI mode interface.

In the SPI mode:

- The MMC controller supports one or more MultiMediaCards (MMCs) or Secure Digital Memory Cards (SD cards). The MMC controller must access the cards one at a time using dedicated card select signals. One card select pin is available from the MMC controller. If multiple cards are connected, use general-purpose I/O (GPIO) pins of the DSP to provide additional card select signals.
- The following MMC controller pins are used:
  - DAT3 (for $\overline{CS}$): This pin is available as a card select line. The signal on this pin can be controlled and monitored via MMC controller register bits.
  - DAT0 (for DataOut): The controller uses this pin to receive serial data and responses from the selected memory card.
  - CMD (for DataIn): The controller uses this pin to transmit commands and serial data to the selected memory card.
  - CLK: This pin provides a clock signal to time transfers between the controller and the memory card(s).
- Because commands and data share the same line, they cannot overlap in time. Thus, the SPI mode does not support sequential and multiple-block read/write operations. Only single-block read/write operations can be performed

Figure 10–22. SPI Mode Interface



10.5.1.1 SPI Mode Write Sequence

Figure 10–23 and Table 10–6 describe the signal activity when the MMC controller is in the SPI mode and is writing data to a memory card. The same block length must be defined in the MMC controller and in the card. In a successful write sequence:

1) The controller sends a write command to the card.

2) The card sends a response to acknowledge the command.

3) The controller sends a block of data to the card.

4) The card sends a response to indicate acceptance of the data.

5) The card sends BUSY tokens until all the data has been programmed into the flash memory inside the card.

The card select signal is active until the MMC controller receives the data response.

Figure 10–23. SPI Mode Write Sequence

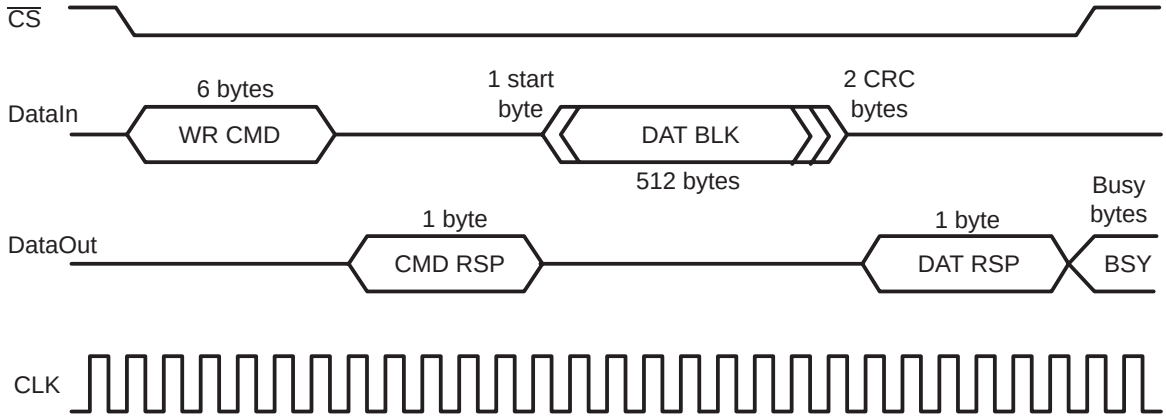


Table 10–6. SPI Mode Write Sequence

Portion of the Sequence	Description
WR CMD	Write command. A 6-byte WRITE_BLOCK command token is sent from the DSP to the card.
CMD RSP	Command response. The card sends to the DSP a 1-byte command response of type R1 to acknowledge the WRITE_BLOCK command.
DAT BLK	Data block. The DSP writes a block of data to the card. The data content is preceded by one start byte and is followed by two CRC bytes.
DAT RSP	Data response. The card sends to the DSP a data response token, indicating to the DSP whether the data has been accepted by the card or rejected due to a CRC error.
BSY	Busy token(s). The data response token is followed by 1-byte busy tokens until all of the data has been programmed into the flash memory inside the card.

10.5.1.2 SPI Mode Read Sequence

Figure 10–24 and Table 10–7 describe the signal activity when the MMC controller is in the SPI mode and is reading data from a memory card. The same block length must be defined in the MMC controller and in the card. In a successful read sequence:

1) The controller sends a read command to the card.

2) The card sends a response to acknowledge the command.

3) The card sends a block of data or a data error token to the DSP.

The card select signal is active throughout the sequence.

Figure 10–24. SPI Mode Read Sequence

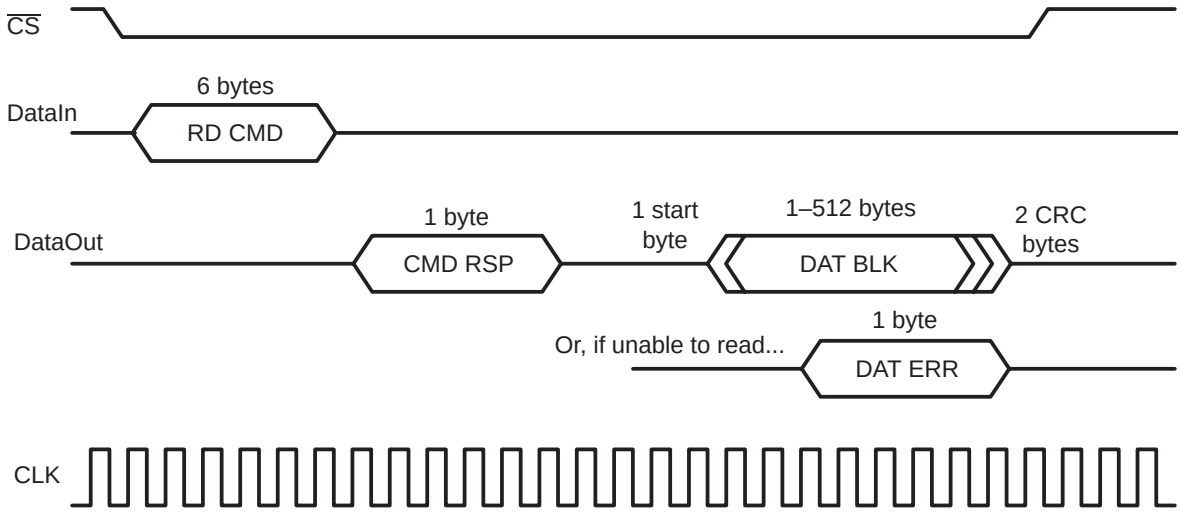


Table 10–7. SPI Mode Read Sequence

Portion of the Sequence	Description
RD CMD	Read command. A 6-byte READ_SINGLE_BLOCK command token is sent from the DSP to the card.
CMD RSP	Command response. The card sends to the DSP a 1-byte command response of type R1 to acknowledge the READ_SINGLE_BLOCK command.
DAT BLK or DAT ERR	Data block or data error token. If the read operation is successful, the card sends a block of data to the DSP. The data content is preceded by one start byte and is followed by two CRC bytes. If the read operation is not successful, the card instead sends a 1-byte data error token.

10.5.2 SPI Mode Single-Block Write Operation

In the SPI mode, to write a single block of data to a memory card, use the following procedure. Figure 10–25 illustrates this procedure with some additional details. The block length must be 512 bytes, and the same block length must be defined in the MMC controller and in the card.

1) Write 1 to the CSEN bit of the control register (MMCCTL). This drives the the card select signal low. (If a general-purpose I/O pin is the card select pin for the card, write to the appropriate GPIO bit.)

2) Write the destination start address to the argument registers. Load the high part to MMCARGH and the low part to MMCARGL.

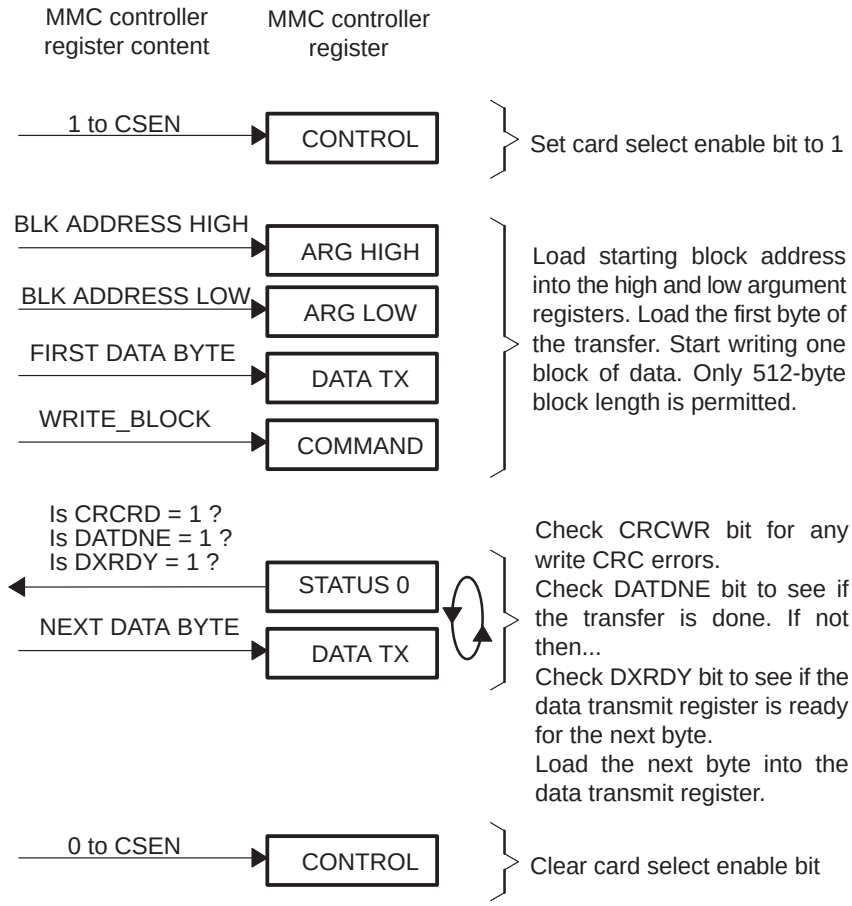
3) Write the first byte of the data block to the data transmit register (MMCDXR).

4) Send the WRITE_BLOCK command to the card via MMCCMD.

5) Use status register 0 to check for errors and to determine when the byte has been successfully transmitted. If all of the data has not been written and if the previous byte has been transmitted, go to step 6. If the all of the data has been written, stop. Also, deselect the card by clearing the CSEN bit.

6) Write the next byte of the data block to MMCDXR, and go to step 5.

Figure 10–25. SPI Mode Single-Block Write Operation



10.5.3 SPI Mode Single-Block Read Operation

In the SPI mode, to read a single block of data from a memory card, use the following procedure. The procedure is also illustrated in Figure 10–26 with some additional details. The block length must be in the range of 1–512 bytes, and the same block length must be defined in the MMC controller and in the card.

1) Write 1 to the CSEN bit of the control register (MMCCTL). This drives the the card select signal low. (If a general-purpose I/O pin is the card select pin for the card, write to the appropriate GPIO bit.)

2) Write the source start address to the argument registers. Load the high part to MMCARGH and the low part to MMCARGL.

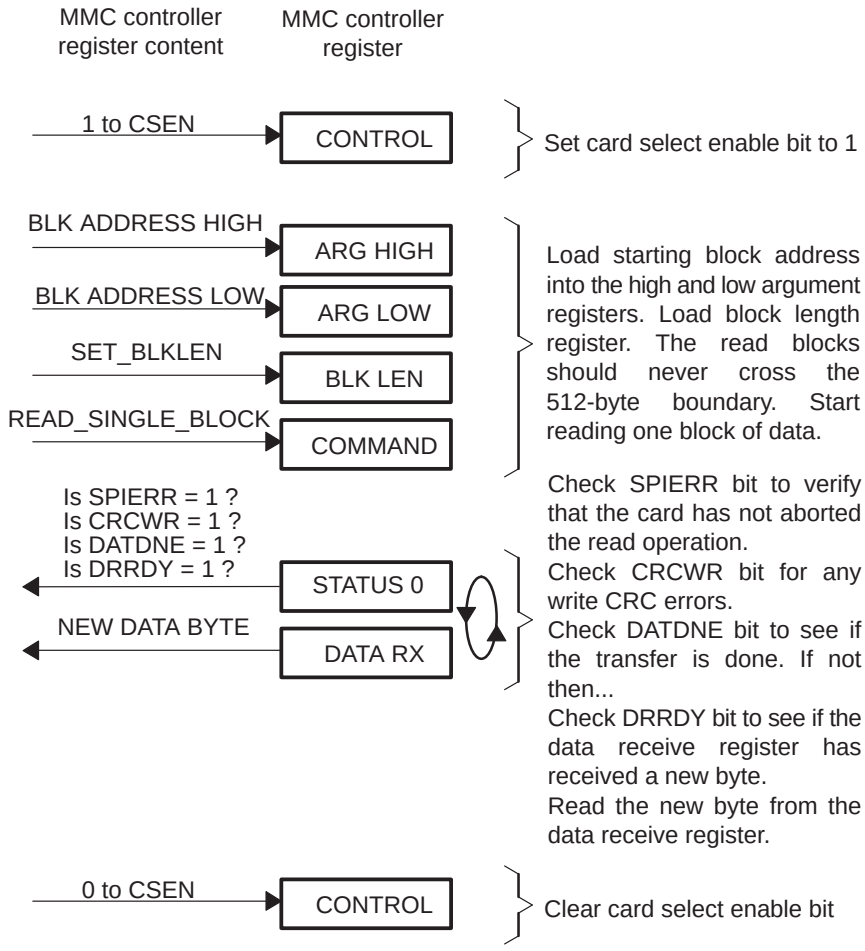
3) Send a SET_BLOCKLEN command via MMCCMD (if the block length is different than the length used in the previous operation).

4) Send a READ_SINGLE_BLOCK command via MMCCMD.

5) Use status register 0 to check for errors and to determine when a new byte has been successfully received. If all of the data has not been read and if a new byte has been received, go to step 6. If the all of the data has been read, stop. Also, deselect the card by clearing the CSEN bit.

6) Read the new byte of data from the data receive register (MMCDRR), and go to step 5.

Figure 10–26. SPI Mode Single-Block Read Operation



10.6 SPI Mode Initialization

The general procedure for initializing the MMC controller is given in the following steps. Details about the registers or register bit fields to be configured in the SPI mode are in the subsections that follow the procedure.

1) Place the MMC controller in its reset state by setting MMCCTL(CMDRST) and MMCCTL(DATRST). With the same register write operation, write the desired values to other bits in MMCCTL.
2) Write to other registers to complete the MMC controller configuration.
3) Clear MMCCTL(CMDRST) and MMCCTL(DATRST) to release the MMC controller from its reset state. Make sure you do not change the values you wrote to the other bits of MMCCTL in step 1.
4) Enable the CLK pin so that the memory clock is sent to the memory card.

10.6.1 Initializing the MMC Control Register (MMCCTL)

When the MMC controller is in the SPI mode, the bit fields named in Figure 10–14 affect the operation of the controller. The subsections that follow the figure help you decide how to initialize each of the fields.

Figure 10–27. MMCCTL Fields Used During SPI Mode Initialization

15	14	13	12	11	10	9	8
				CLKPRE	CLKPST	NACSKP	DMAEN
				R/W–0	R/W–0	R/W–0	R/W–0

7	6	5	4	3	2	1	0
DATEG		SPIEN	CSEN	SPICRC		CMDRST	DATRST
R/W–0		R/W–0	R/W–0	R/W–0		R/W–0	R/W–0

Note: R/W-x = Read/Write-Reset value

10.6.1.1 If Needed, Add a Delay Before Driving the $\overline{CS}$ Signal Low

Register(Field)	*symval*	Value	Description
MMCCTL(CLKPRE)		0	Do not insert clock cycles.
		1	Insert 8 CLK cycles before $\overline{CS}$ goes low.

10.6.1.2 If Needed, Add a Delay After Driving the $\overline{CS}$ Signal High

Register(Field)	*symval*	Value	Description
MMCCTL(CLKPST)		0	Do not insert clock cycles.
		1	Insert 8 CLK cycles after $\overline{CS}$ goes high.

10.6.1.3 Enable/Disable NAC Cycle Skip

Register(Field)	*symval*	Value	Description
MMCCTL(NACSKP)		0	Disable first NAC cycle skip
		1	Enable first NAC cycle skip

10.6.1.4 Enable/Disable DMA Events

Register(Field)	*symval*	Value	Description
MMCCTL(DMAEN)		0	Disable DMA events.
		1	Enable DMA events.

Use DMAEN to disable or enable the MMC controller DMA events, which are described in section 10.1.5 on page 10-7.

10.6.1.5 Select a Type of Edge Detection (If Any) for the DAT3 Pin

Register(Field)	*symval*	Value	Description
MMCCTL(DATEG)		00	Disable DAT3 edge detection.
		01	Enable DAT3 rising edge detection.
		10	Enable DAT3 falling edge detection.
		11	Enable DAT3 dual edge detection (detect both edges).

The use of the DATEG bit is applicable if the DAT3 pin is not used for the card select signal.

The DATEG control bit of MMCCTL enables or disables general-purpose edge detection on the DAT3 pin. If you enable edge detection and an edge is detected, the DATEG flag bit of MMCST0 is set. In addition, if DATEG = 1 in MMCIE, an interrupt request is generated.

10.6.1.6 Enable SPI Mode

Register(Field)	*symval*	Value	Description
MMCCTL(SPIEN)		0	Disable SPI mode/Enable native mode.
		1	Enable SPI mode./Disable native mode.

To enable the SPI mode, set this bit (SPIEN = 1).

10.6.1.7 Drive the Card Select Signal High/Low

Register(Field)	*symval*	Value	Description
MMCCTL(CSEN)		0	Drive DAT3 high to deselect the card.
		1	Drive DAT3 low to select the card.

If you are using the DAT3 pin as a card select line, set CSEN to select the memory card and clear CSEN to deselect the memory card. If you are using general-purpose I/O pins as card select lines, select and deselect the cards by writing to the appropriate GPIO bits.

10.6.1.8 Enable/Disable CRC Checking

Register(Field)	*symval*	Value	Description
MMCCTL(SPICRC)		0	Disable CRC checking.
		1	Enable CRC checking.

10.6.1.9 Reset/Enable the MMC Controller

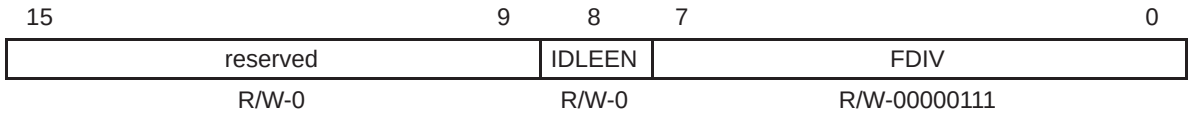
Register(Field)	*symval*	Value	Description
MMCCTL(CMDRST)		0	Enable the CMD (command) line portion of the MMC controller.
		1	Place the CMD line portion of the MMC controller in its reset state.
MMCCTL(DATRST)		0	Enable the DAT (data) line portion of the MMC controller.
		1	Place the DAT line portion of the MMC controller in its reset state.

To place the MMC controller in its reset state and disable it, set the CMDRST and DATRST bits of MMCCTL. The first step of the MMC controller initialization process is to disable both sets of logic. When initialization is complete but before you enable the CLK pin, enable the MMC controller by clearing the CMDRST and DATRST bits.

10.6.2 Initializing the Clock Control Registers (MMCFCLK and MMCCLK)

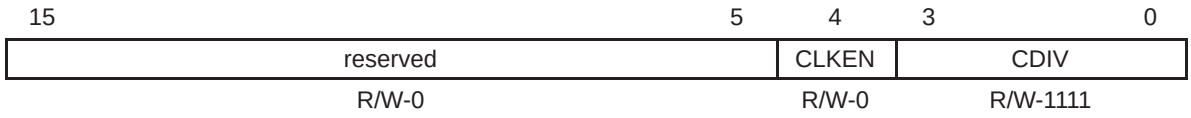
Figure 10–15 and Figure 10–16 show the bit fields in the function clock control register (MMCFCLK) and the memory clock control register (MMCCLK). The following subsections describe how to choose initialization values for these fields.

Figure 10–28. MMCFCLK



Note: R/W-x = Read/Write-Reset value

Figure 10–29. MMCCLK



Note: R/W-x = Read/Write-Reset value

10.6.2.1 Set the Function Clock and the Memory Clock

Register(Field)	*symval*	Value	Description
MMCFCLK(FDIV)		0–255	Divide-down value for the function clock.
MMCCLK(CDIV)		0–15	Divide-down value for the memory clock.

To generate the function clock (the clock for activity inside the MMC controller), the MMC controller divides down the CPU clock as shown in the following equation. When you initialize MMCFCLK, you specify FDIV, a divide-down value in the range 0 through 255.

$$\text{function clock frequency} = \frac{\text{CPU clock frequency}}{(\text{FDIV} + 1)}$$

The memory clock (the clock for the attached memory card) is a divided-down version of the function clock; see the following equation. When you initialize MMCCLK, you specify CDIV, a divide-down value in the range 0 through 15.

$$\text{memory clock frequency} = \frac{\text{function clock frequency}}{(\text{CDIV} + 1)}$$

For more information about the function clock and the memory clock , see section 10.1.3 on page 10-4.

10.6.2.2 Determine Whether the Function Clock Stops in Response to an IDLE Instruction

Register(Field)	*symval*	Value	Description
MMCFCLK(IDLEEN)		0	The MMC controller cannot be made idle.
		1	If PERI = 1 (see just below), the MMC controller is idle (the function clock is stopped) after the IDLE instruction is executed.
ICR(PERI)		0	Any peripheral in the PERIPH idle domain will be active after the IDLE instruction is executed.
		1	Any peripheral in the PERIPH idle domain can be idle after the IDLE instruction is executed, depending on the state of that peripheral's idle enable bit.

The MMC controller is one of the peripheral devices in the PERIPH idle domain. For details on controlling the various idle domains of the DSP, see Chapter 8, *Idle Configurations*. If you want the MMC controller to go idle in response to an IDLE instruction, make the following preparations:

1) Write 1 to the idle enable (IDLEEN) bit in MMCFCLK. This tells the DSP to stop the function clock of the MMC controller when the PERIPH domain becomes idle.
2) Write a 1 to the PERI bit in ICR (see page 8.7). This tells the DSP to make the PERIPH domain idle when an IDLE instruction is executed.

10.6.2.3 Enable/Disable the CLK Pin

Register(Field)	*symval*	Value	Description
MMCCLK(CLKEN)		0	Disable the CLK pin; drive a constant, low signal on the pin.
		1	Enable the CLK pin, so that it shows the memory clock signal.

The CLKEN bit determines whether the memory clock appears on the CLK pin.

10.6.3 Initialize the Interrupt Enable Register (MMCIE)

Register(Field)	*symval*	Value	Description
MMCIE(11–0)		000h–FFFh	Determines which of the MMC interrupt requests will be forwarded to the CPU.

The bits in MMCIE individually enable or disable the interrupt requests described in section 10.1.4 on page 10-5. Figure 10–30 shows the bit fields of MMCIE apply to the SPI mode. Set one of these bits to enable the associated interrupt request. Clear one of these bit to disable the associated interrupt request. Load 0s into the bits not used in the SPI mode.

Figure 10–30. MMCIE Fields Used in the SPI Mode

15	14	13	12	11	10	9	8
				DATEG	DRRDY	DXRDY	SPIERR
				R/W–0	R/W–0	R/W–0	R/W–0

7	6	5	4	3	2	1	0
CRCRS	CRCRD	CRCWR			RSPDNE	BSYDNE	DATDNE
R/W–0	R/W–0	R/W–0			R/W–0	R/W–0	R/W–0

Note: R/W-x = Read/Write-Reset value

10.6.4 Initialize the Block Length Register (MMCBLEN)

Register(Field)	*symval*	Value	Description
MMCBLEN(11–0)		1–512	Number of bytes in a data block.

In MMCBLEN (see Figure 10–31), you must define the size for each block of data transferred between the MMC controller and a memory card. The valid size depends on the type of read/write operation. A length of 0 bytes is prohibited.

Figure 10–31. MMCBLEN

15 12	11 0
reserved	BLEN
R/W-0	R/W-200h

10.7 Monitoring Activity in the SPI Mode

This section describes registers and specific register bits that you can use to obtain the status of the MMC controller in the SPI mode.

10.7.1 Detecting Edges and Level Changes on the DAT3 Pin

Register(Field)	*symval*	Value	Description
MMCST0(DATEG)		0	No edge detected on DAT3 pin
		1	Edge detected on DAT3 pin
MMCST1(DAT)		0	Low signal level on DAT3
		1	High signal level on DAT3

Detecting edges. The MMC controller sets the DATEG flag of status register 0 (MMCST0) if DAT3 edge detection is enabled (DATEG is nonzero in MMCCTL) and the specified edge is detected. The CPU can also be notified of the DAT3 by an interrupt if you enable the interrupt request in the interrupt enable register (DATEG = 1 in MMIE).

Detecting level changes. The DAT bit of status register 1 tracks the signal level on the DAT3 pin.

10.7.2 Determining Whether New Data is Available in MMCDRR

Register(Field)	*symval*	Value	Description
MMCST0(DRRDY)		0	MMCDRR not ready.
		1	MMCDRR ready. New data has arrived and can be read by the CPU or by the DMA controller.

The MMC controller sets the DRRDY flag of MMCST0 when new data arrives in the data receive register (MMCDRR). The CPU can also be notified of the event by an interrupt if you enable the interrupt request (DRRDY = 1 in MMIE).

10.7.3 Verifying That MMCDXR is Ready to Accept New Data

10.7.4 Verifying That MMCDXR is Ready to Accept New Data

Register(Field)	*symval*	Value	Description
MMCST0(DXRDY)		0	MMCDXR not ready.
		1	MMCDXR ready. The data in MMCDXR has been transmitted; MMCDXR can accept new data from the CPU or from the DMA controller.

The MMC controller sets the DXRDY flag of MMCST0 when data leaves the data transmit register (DXRDY). The CPU can also be notified of the event by an interrupt if you enable the interrupt request (DXRDY = 1 in MMIE).

10.7.5 Checking for an SPI Data Error

Register(Field)	*symval*	Value	Description
MMCST0(SPIERR)		0	No SPI data error
		1	SPI data error detected. A data error token has been received.
MMCETOK(7–0)		00h–FFh	SPI data error token from the memory card

If an SPI read operation is successful, the card sends a block of data to the DSP. If the read operation is not successful, the card instead sends a 1-byte data error token, which is stored in the MMCETOK register of the MMC controller. When the token arrives, the SPIERR bit is set. The CPU can also be notified of the error by an interrupt if you enable the interrupt request (SPIERR = 1 in MMIE).

10.7.6 Checking for CRC Errors

Register(Field)	*symval*	Value	Description
MMCST0(CRCRS)		0	No response CRC error
		1	Response CRC error detected
MMCST0(CRCRD)		0	No read-data CRC error
		1	Read-data CRC error detected
MMCST0(CRCWR)		0	No write-data CRC error
		1	Write-data CRC error detected

The MMC controller sets one of these flags in response to the corresponding CRC error. The CPU can also be notified of the CRC error by an interrupt if you enable the interrupt request (CRCRS/CRCRD/CRCWR = 1 in MMIE).

10.7.7 Determining When a Response/Command is Done

Register(Field)	*symval*	Value	Description
MMCST0(RSPDNE)			**If the command requires a response:**
		0	Response not done
		1	Response fully received with no CRC error
			If no response required:
		0	Command not done
		1	Command has been sent

The MMC controller sets the RSPDNE flag when the response is done (or, in the case of commands that do not require a response, when the command is done). The CPU can also be notified of the done condition by an interrupt if you enable the interrupt request (RSPDNE = 1 in MMIE).

10.7.8 Determining Whether the Memory Card is Busy

Register(Field)	*symval*	Value	Description
MMCST0(BSYDNE)		0	The memory card is busy.
		1	The memory card is no longer sending a busy signal.
MMCST1(BUSY)		0	The memory card has not sent a busy signal.
		1	The memory card is busy.

The card sends a busy signal either as an expected part of an R1b response or to indicate that the card is still programming the last write data into its flash memory. The MMC controller has two flags to tell you whether the memory card is sending a busy signal. The two flags are complements of each other:

- BSYDNE is set if the card did not send or is not sending a busy signal. As with the other bits in status register 0, this bit has an associated interrupt that you can enable (BSYDNE = 1 in MMCIE).
- BUSY is set when a busy signal is received from the card.

10.7.9 Determining Whether a Data Transfer is Done

Register(Field)	*symval*	Value	Description
MMCST0(DATDNE)			**When reading from memory card:**
		0	Read operation (if one is in progress) not done
		1	Data fully received with no CRC error
			When writing to memory card:
		0	Write operation (if one is in progress) not done
		1	Data fully transmitted

The MMC controller sets the DATDNE flag when all the bytes of a data transfer have been transmitted/received. You can poll this bit to determine when to stop writing to the data transmit register (for a write operation) or when to stop reading from the data receive register (for a read operation). The CPU can also be notified of the time-out event by an interrupt if you enable the interrupt request (TOUTRS/TOUTRD = 1 in MMIE).f

10.7.10 Checking For a Data Transmit Empty Condition

Register(Field)	*symval*	Value	Description
MMCST1(DXEMP)		0	No data transmit empty condition
		1	Data transmit empty condition

Typically, this bit is not used to control data transfers; rather, it is checked during recovery from an error condition. There is no interrupt associated with the transmit empty condition.

During transmission, a data value is passed from the data transmit register (MMCDXR) to the data transmit shift register. Then value is passed from this shift register to the memory card one bit at a time. The DXEMP bit indicates when this shift register is empty; there are no bits available to shift out to the memory card.

10.7.11 Checking for a Data Receive Full Condition

Register(Field)	*symval*	Value	Description
MMCST1(DRFUL)		0	No data receive full condition
		1	Data receive full condition

Typically, this bit is not used to control data transfers; rather, it is checked during recovery from an error condition. There is no interrupt associated with the data receive full condition.

During reception, the data receive shift register accepts a data value one bit at a time. Then the whole value is passed from this shift register to the data receive register (MMCDRR). The DRFUL bit indicates when this shift register is full; no new bits can be shifted in from the memory card. Typically, this bit is used only during recovery from the error condition.

10.7.12 Checking the Status of the CLK Pin

Register(Field)	*symval*	Value	Description
MMCST1(CLKSTP)		0	CLK is active. The memory clock signal is being driven on the pin.
		1	CLK is held low. Possible reasons: Manual stop (CLKEN = 0), data receive full condition, or data transmit empty condition.

Read CLKSTP to determine whether the memory clock has been stopped on the CLK pin.

10.8 MMC Controller Registers

The MMC controller memory-mapped registers are listed in Table 10–8. The x's in the Address column indicate the part of the addresses that is device dependent. For example, on a TMS320VC5509 DSP, the start address is 4800h for MMC controller 0, and the MMCCLK register is at address 4802h.

Table 10–8. MMC Controller Memory-Mapped Registers

Address (Hex)	Name	Description
xx00	MMCFCLK	MMC Function Clock Control Register
xx01	MMCCTL	MMC Control Register
xx02	MMCCLK	MMC Clock Control Register
xx03	MMCST0	MMC Status Register 0
xx04	MMCST1	MMC Status Register 1
xx05	MMCIE	MMC Interrupt Enable Register
xx06	MMCTOR	MMC Response Time–Out Register
xx07	MMCTOD	MMC Data Read Time–Out Register
xx08	MMCBLEN	MMC Block Length Register
xx09	MMCNBLK	MMC Number of Blocks Register
xx0A	MMCNBLC	MMC Number of Blocks Counter
xx0B	MMCDRR	MMC Data Receive Register
xx0C	MMCDXR	MMC Data Transmit Register
xx0D	MMCCMD	MMC Command Register
xx0E	MMCARGL	MMC Argument Register Low
xx0F	MMCARGH	MMC Argument Register High
xx10	MMCRSP0	MMC Response Register 0
xx11	MMCRSP1	MMC Response Register 1
xx12	MMCRSP2	MMC Response Register 2
xx13	MMCRSP3	MMC Response Register 3
xx14	MMCRSP4	MMC Response Register 4
xx15	MMCRSP5	MMC Response Register 5
xx16	MMCRSP6	MMC Response Register 6
xx17	MMCRSP7	MMC Response Register 7
xx18	MMCDRSP	MMC Data Response Register
xx19	MMCETOK	MMC SPI Error Token Register
xx1A	MMCCIDX	MMC Command Index Register

10.8.1 MMC Control Register (MMCCTL)

Figure 10–32. MMC Control Register (MMCCTL)

15	14	13	12	11	10	9	8
Reserved				CLKPRE	CLKPST	NACSKP	DMAEN
				R/W–0	R/W–0	R/W–0	R/W–0

7	6	5	4	3	2	1	0
DATEG		SPIEN	CSEN	SPICRC	WIDTH	CMDRST	DATRST
R/W–0		R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

Note: R/W-x = Read/Write-Reset value

Table 10–9. MMC Control Register (MMCCTL) Field Values

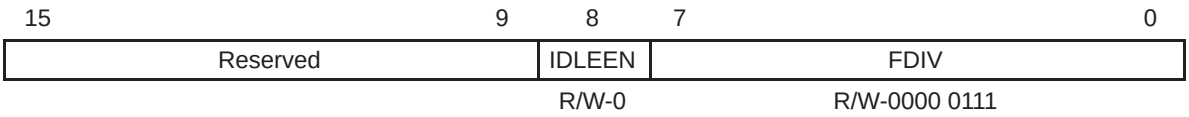
Bit	*field*	*symval*	Value	Description
15–12	Reserved			Reserved. The reserved bit location is always read as zero.
11	CLKPRE			Pre-$\overline{CS}$ low CLK cycles (only in SPI mode).
			0	Do not insert clock cycles.
			1	Insert 8 CLK cycles before $\overline{CS}$ goes low.
10	CLKPST			Post-$\overline{CS}$ high CLK cycles (only in SPI mode).
			0	Do not insert clock cycles.
			1	Insert 8 CLK cycles after $\overline{CS}$ goes high.
9	NACSKP			First NAC cycle skip enable (only in SPI mode).
			0	Disable first NAC cycle skip.
			1	Enable first NAC cycle skip.
8	DMAEN			DMA event enable bit.
			0	Disable DMA events.
			1	Enable DMA events.
7–6	DATEG			DAT3 Edge Detection Select bits.
			00	DAT3 edge detection is disabled.
			01	DAT3 rising edge detection is enabled.
			10	DAT3 falling edge detection is enabled.
			11	DAT3 dual edge detection is enabled (both edges detected).

Table 10–9. MMC Control Register (MMCCTL) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
5	SPIEN			SPI Mode Enable bit.
			0	Native mode is selected.
			1	SPI mode is selected.
4	CSEN			Card select enable (only in SPI mode).
			0	Drive $\overline{CS}$ (DAT3) high to deselect the card.
			1	Drive $\overline{CS}$ (DAT3) low to select the card.
3	SPICRC			CRC Checking (only in SPI mode).
			0	CRC checking is disabled.
			1	CRC checking is enabled.
2	WIDTH			Data Bus Width (only in native mode).
			0	Data bus has 1 bit (only DAT0 is used).
			1	Data bus has 4 bits (DAT0–3 are used).
1	CMDRST			CMD (command) logic reset
			0	CMD logic of the MMC controller is enabled.
			1	CMD logic of the MMC controller is in the reset state.
0	DATRST			DAT (data) logic reset
			0	DAT logic of the MMC controller is enabled.
			1	DAT logic of the MMC controller is in the reset state.

10.8.2 MMC Function Clock Control Register (MMCFCLK)

Figure 10–33. MMC Function Clock Control Register (MMCFCLK)



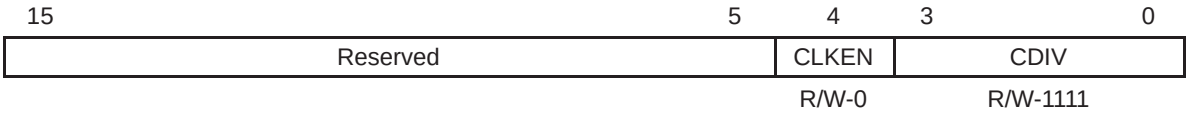
Note: R/W-x = Read/Write-Reset value

Table 10–10. MMC Function Clock Control Register (MMCFCLK) Field Values

Bit	*field*	*symval*	Value	Description
15–9	Reserved			Reserved. The reserved bit location is always read as zero.
8	IDLEEN			IDLE enable bit.
			0	The function clock cannot be stopped by an IDLE instruction.
			1	If an IDLE instruction makes the PERIPH domain idle, the MMC controller is idle (the function clock is stopped).
7–0	FDIV			Divide-down value for the function clock. The CPU clock is divided as follows to create the function clock: function clock frequency = CPU clock frequency/(FDIV + 1)

10.8.3 MMC Clock Control Register (MMCCLK)

Figure 10–34. MMC Clock Control Register (MMCCLK)



Note: R/W-x = Read/Write-Reset value

Table 10–11. MMC Clock Control Register (MMCCLK) Field Values

Bit	*field*	*symval*	Value	Description
15–5	Reserved			Reserved. The reserved bit location is always read as zero.
4	CLKEN			CLK enable bit.
			0	CLK pin is disabled and fixed low.
			1	CLK pin is enabled; it reflects the memory clock signal.
3–0	CDIV		0–15	Divide-down value for the memory clock. The function clock is divided down as follows to produce the memory clock: memory clock frequency = function clock frequency/(CDIV + 1)

10.8.4 MMC Status Register 0 (MMCST0)

The transition from 0 to 1 of each bit in MMCST0 may cause interrupt signal to the CPU. It depends on MMCIE. Each status bit is cleared when reading by the CPU except DRRDY and DXRDY.

Figure 10–35. MMC Status Register 0 (MMCST0)

15	14	13	12	11	10	9	8
Reserved				DATEG	DRRDY	DXRDY	SPIERR
				R–0	R–0	R–0	R–1

7	6	5	4	3	2	1	0
CRCRS	CRCRD	CRCWR	TOUTRS	TOUTRD	RSPDNE	BSYDNE	DATDNE
R–0	R–0	R–0	R–0	R–0	R–0	R–0	R–0

Note: R/W-x = Read/Write-Reset value

Table 10–12. MMC Status Register 0 (MMCST0) Field Values

Bit	*field*	*symval*	Value	Description
15–12	Reserved			Reserved. The reserved bit location is always read as zero.
11	DATEG			DAT3 edge detect bit.
			0	No DAT3 edge is detected.
			1	DAT3 edge has been detected.
10	DRRDY			Data receive ready bit. DRRDY is cleared to 0 when the DAT logic is reset (DATRST = 1), when a command is sent with data receive/transmit clear (DCLR = 1), or when data is read from MMCDRR.
			0	MMCDRR not ready
			1	MMCDRR ready. New data has arrived and can be read by the CPU or by the DMA controller.
9	DXRDY			Data transmit ready bit. DXRDY is cleared to 0 when the DAT logic is reset (DATRST = 1), when a command is sent with data receive/transmit clear (DCLR = 1), or when data is written to MMCDXR.
			0	MMCDXR not ready.
			1	MMCDXR ready. The data in MMCDXR has been transmitted; MMCDXR can accept new data from the CPU or from the DMA controller.

Table 10–12. MMC Status Register 0 (MMCST0) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
8	SPIERR			Data error bit (in SPI mode)
			0	No data error is received.
			1	Data error token has been received.
7	CRCRS			Response CRC error bit
			0	No CRC error is detected.
			1	CRC error has been detected.
6	CRCRD			Read-data CRC error bit
			0	No CRC error is detected.
			1	CRC error has been detected.
5	CRCWR			Write-data CRC error bit
			0	No CRC error is detected.
			1	CRC error has been detected.
4	TOUTRS			Response time-out bit
			0	No time-out event
			1	A time-out event has occurred while the MMC controller was waiting for a response to a command.
3	TOUTRD			Read-data time-out bit
			0	No time-out event
			1	A time-out event has occurred while the MMC controller was waiting for data.
2	RSPDNE			Command/response done bit.
				If the command requires a response:
			0	Response not done
			1	Response fully received with no CRC error
				If no response required:
			0	Command not done
			1	Command has been sent

Table 10–12. MMC Status Register 0 (MMCST0) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
1	BSYDNE			Busy done bit used for commands with R1b response (CMD38) to indicate card is busy. BSYDNE is set to indicate card is no longer busy.
			0	The memory card is busy.
			1	The memory card is no longer sending a busy signal.
0	DATDNE			Data transfer done bit.
				When reading from memory card:
			0	Read operation not done
			1	Data fully received with no CRC error
				When writing to memory card:
			0	Write operation not done
			1	Data fully transmitted

10.8.5 MMC Status Register 1 (MMCST1)

There are no interrupts associated with events that set the flags in MMCST1.

Figure 10–36. MMC Status Register 1 (MMCST1)

15	5	4	3	2	1	0
Reserved		DAT	DXEMP	DRFUL	CLKSTP	BUSY
		R-0	R-0	R-0	R-0	R-0

Note: R/W-x = Read/Write-Reset value

Table 10–13. MMC Status Register 1 (MMCST1) Field Values

Bit	*field*	*symval*	Value	Description
15–5	Reserved			Reserved. The reserved bit location is always read as zero.
4	DAT			DAT3 status bit.
			0	Low signal level on DAT3 pin
			1	High signal level on DAT3 pin

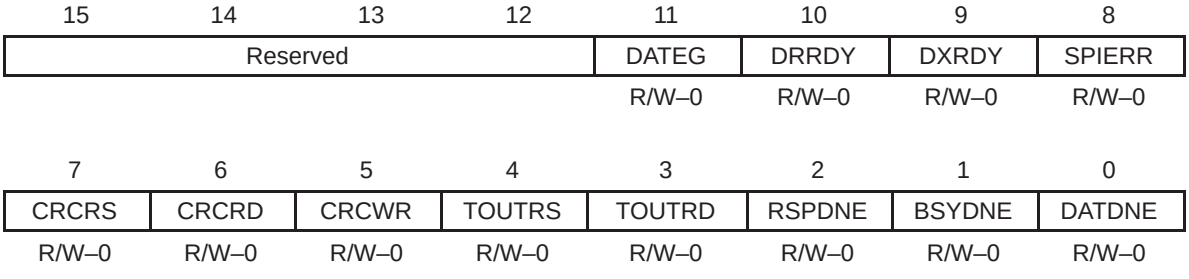
Table 10–13. MMC Status Register 1 (MMCST1) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
3	DXEMP			Data transmit empty
			0	No data transmit empty condition. The data transmit shift register is not empty.
			1	Data transmit empty condition. The data transmit shift register is empty. No bits are available to be shifted out to the memory card.
2	DRFUL			Data receive full
			0	No data receive full condition. The data receive shift register is not full.
			1	Data receive full condition. The data receive shift register is full. No new bits can be shifted in from the memory card.
1	CLKSTP			Clock stop status bit.
			0	CLK is active. The memory clock signal is being driven on the pin.
			1	CLK is held low. Possible reasons: Manual stop (CLKEN = 0), data receive full condition, or data transmit empty condition.
0	BUSY			Busy bit
			0	No busy signal detected from memory card
			1	Busy signal detected (memory card is busy)

10.8.6 MMC Interrupt Enable Register (MMCIE)

This register is used to enable or disable status interrupts. To disable an interrupt, clear the corresponding bit in MMCIE to 0; to enable it, set the bit to 1.

Figure 10–37. MMC Interrupt Enable Register (MMCIE)



Note: R/W-x = Read/Write-Reset value

Table 10–14. MMC Interrupt Enable Register (MMCIE) Field Values

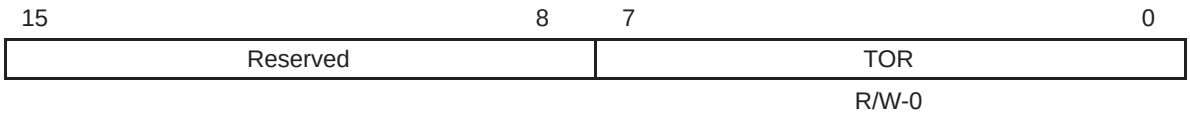
Bit	*field*	*symval*	Value	Description
15–12	Reserved			Reserved. The reserved bit location is always read as zero.
11	DATEG			DAT3 edge detect interrupt enable
			0	DAT3 edge detect interrupt is disabled.
			1	DAT3 edge detect interrupt is enabled.
10	DRRDY			Data receive ready interrupt enable
			0	Data receive ready interrupt is disabled.
			1	Data receive ready interrupt is enabled.
9	DXRDY			Data transmit ready interrupt enable
			0	Data transmit ready interrupt is disabled.
			1	Data transmit ready interrupt is enabled.
8	SPIERR			SPI data error interrupt enable (in SPI mode)
			0	SPI data error interrupt is disabled.
			1	SPI data error interrupt is enabled.
7	CRCRS			Response CRC error interrupt enable
			0	Response CRC error interrupt is disabled.
			1	Response CRC error interrupt is enabled.
6	CRCRD			Read-data CRC error interrupt enable
			0	Read-data CRC error interrupt is disabled.
			1	Read-data CRC error interrupt is enabled.
5	CRCWR			Write-data CRC error interrupt enable
			0	Write-data CRC error interrupt is disabled.
			1	Write-data CRC error interrupt is enabled.
4	TOUTRS			Response time-out interrupt enable
			0	Response time-out interrupt is disabled.
			1	Response time-out interrupt is enabled.

Table 10–14. MMC Interrupt Enable Register (MMCIE) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
3	TOUTRD			Read-data time-out interrupt enable
			0	Read-data time-out interrupt is disabled.
			1	Read-data time-out interrupt is enabled.
2	RSPDNE			Response/command done interrupt enable
			0	Response/command done interrupt is disabled.
			1	Response/command done interrupt is enabled.
1	BSYDNE			Busy done interrupt enable
			0	Busy done interrupt is disabled.
			1	Busy done interrupt is enabled.
0	DATDNE			Data transfer done interrupt enable
			0	Data transfer done interrupt is disabled.
			1	Data transfer done interrupt is enabled.

10.8.7 MMC Response Time-Out Register (MMCTOR)

Figure 10–38. MMC Response Time-Out Register (MMCTOR)



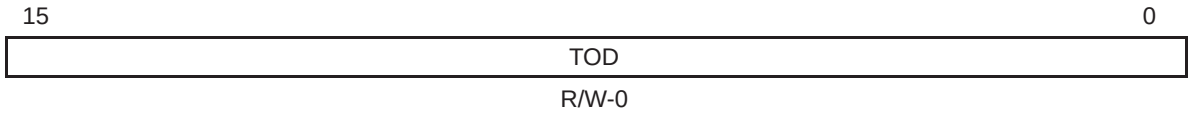
Note: R/W-x = Read/Write-Reset value

Table 10–15. MMC Response Time-Out Register (MMCTOR) Field Values

Bit	*field*	*symval*	Value	Description
15–8	Reserved			Reserved. The reserved bit location is always read as zero.
7–0	TOR			Time-out period for response (native mode)
			00h	No time-out.
			01h–FFh	1 CLK clock cycle to 255 CLK clock cycles.

10.8.8 MMC Data Read Time-Out Register (MMCTOD)

Figure 10–39. MMC Data Read Time-Out Register (MMCTOD)



Note: R/W-x = Read/Write-Reset value

Table 10–16. MMC Data Read Time-Out Register (MMCTOD) Field Values

Bit	*field*	*symval*	Value	Description
15–0	TOD			Time-out period for data read (native mode)
			0000h	No time-out.
			0001h–FFFFh	1 CLK clock cycles to 65535 CLK clock cycles

10.8.9 MMC Block Length Register (MMCBLEN)

MMCBLEN specifies the data block length in bytes. This value must be same as CSD register settings in the memory card. The default value in this register after a DSP reset is 512.

Figure 10–40. MMC Block Length Register (MMBLEN)

Note: R/W-x = Read/Write-Reset value

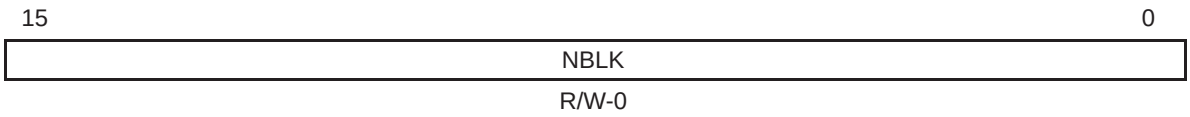
Table 10–17. MMC Block Length Register (MMCBLEN) Field Values

Bit	*field*	*symval*	Value	Description
15–12	Reserved			Reserved. The reserved bit location is always read as zero.
11–0	BLEN		1–512	Block length value specifies the byte count of a date block. The value 0 is prohibited.

10.8.10 MMC Number of Blocks Register (MMCNBLK)

MMCNBLK is used for specifying number of blocks for a multiple-block transfer (only possible in the native mode).

Figure 10–41. MMC Number of Blocks Register (MMCNBLK)



Note: R/W-x = Read/Write-Reset value

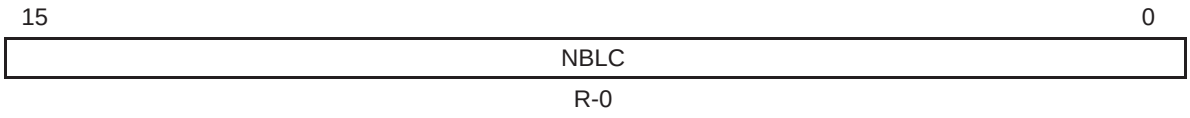
Table 10–18. MMC Number of Blocks Register (MMCNBLK) Field Values

Bit	*field*	*symval*	Value	Description
15–0	NBLK			Number of blocks value specifies the total number of blocks to be transferred.
			0	Infinite number of blocks.
			1–65535	Number of blocks value specifies the total number of blocks to be transferred. The value 0000h indicates an infinite number of blocks.

10.8.11 MMC Number of Blocks Counter Register (MMCNBLC)

MMCNBLC is a down counter for tracking the number of blocks left to be transferred.

Figure 10–42. MMC Number of Blocks Counter Register (MMCNBLC)



Note: R/W-x = Read/Write-Reset value

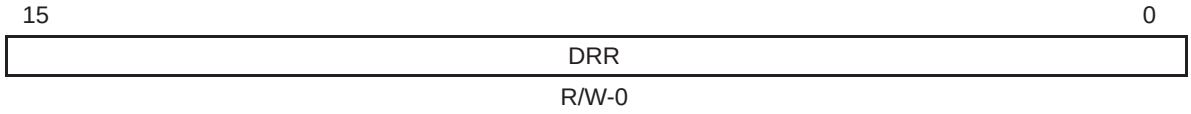
Table 10–19. MMC Number of Blocks Counter Register (MMCNBLC) Field Values

Bit	*field*	*symval*	Value	Description
15–0	NBLC		0–65535	Number of blocks left to be transferred.

10.8.12 MMC Data Receive Register (MMCDRR)

Data comes into the MMC controller via MMCDRR. The CPU or the DMA controller can read data from this register.

Figure 10–43. MMC Data Receive Register (MMCDRR)

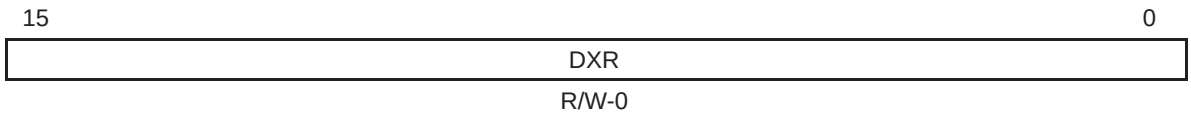


Note: R/W-x = Read/Write-Reset value

10.8.13 MMC Data Transmit Register (MMCDXR)

Data exits the MMC controller via MMCDXR. The CPU or the DMA controller can write data to this register.

Figure 10–44. MMC Data Transmit Register (MMCDXR)



Note: R/W-x = Read/Write-Reset value

10.8.14 MMC Command Register (MMCCMD)

Writing to MMCCMD causes the MMC controller to send the command. Thus, the argument registers (MMCARGH and MMCARGL) have to be loaded properly before a write to MMCCMD. For the format of a command (index, arguments, and other bits), see the description for the argument registers (page 10-65).

The CMD field of MMCMD specifies the type of command to be sent. The other fields define the operation (command, response, additional activity) for the MMC controller.

The content of MMCCMD is kept after the transfer to the transmit shift register.

Figure 10–45. MMC Command Register (MMCCMD)

15	14	13	12	11	10 9	8	7 6	5 0
DCLR	INIT	DATA	STREAM	WRITE	RSPFMT	BSYEXP	Reserved	CMD
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Note: R/W-x = Read/Write-Reset value

Table 10–20. MMC Command Register (MMCCMD) Field Values

Bit	*field*	*symval*	Value	Description
15	DCLR			Data receive/transmit clear
			0	Do not clear the data receive ready (DRRDY) and data transmit ready (DXRDY) bits.
			1	Clear DRRDY and DXRDY.
14	INIT			Initialization clock cycles
			0	Do not insert initialization clock cycles.
			1	Insert initialization clock cycles; insert 80 CLK cycles before sending the command specified in the CMD field. These dummy clock cycles are required for resetting a card after power on.
13	DATA			Data transfer indicator
			0	No data transfer
			1	A data transfer associated with the command.
12	STREAM			Stream enable
			0	If DATA = 1, the data transfer is a block transfer.
			1	If DATA = 1, the data transfer is a stream transfer.

Table 10–20. MMC Command Register (MMCCMD) Field Values (Continued)

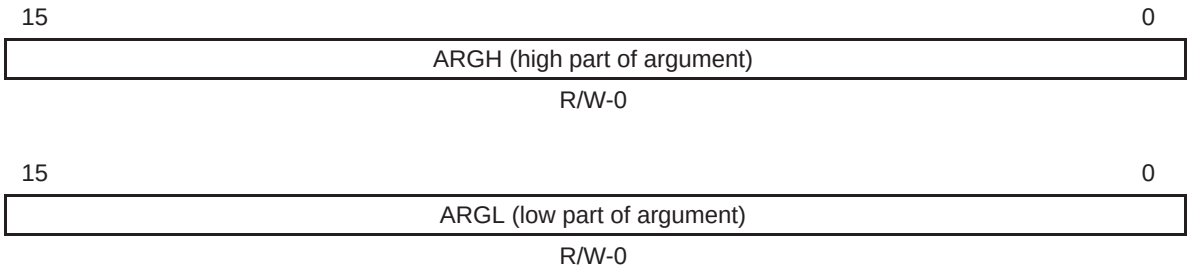
Bit	*field*	*symval*	Value	Description
11	WRITE			Write enable
			0	If DATA = 1, the data transfer is a read operation.
			1	If DATA = 1, the data transfer is a write operation.
10–9	RSPFMT			Response format (expected type of response to the command)
			00	No response
			01	R1/R4/R5/R6 Native mode: 48 bits with CRC SPI mode: 8 bits with CRC (if CRC enabled)
			10	R2 Native mode: 136 bits with CRC SPI mode: 16 bits with CRC (if CRC enabled)
			11	R3 Native mode: 48 bits with no CRC SPI mode: Not applicable
8	BSYEXP			Busy expected. If an R1b (R1 with busy) response is expected, set RSPFMT = 01 and BSYEXP = 1.
			0	No busy signal expected.
			1	Busy signal expected.
7–6	Reserved			Reserved. These reserved bits are always read as 0s.
5–0	CMD		xxxxx	Command index for the command to be sent to the memory card.

10.8.15 MMC Argument Registers (MMCARGH and MMCARGL)

MMCARGH and MMCARGL (see 10-66) are used for specifying the arguments to be sent with the command specified in MMCCMD. Writing to MMCCMD causes the MMC controller to send a command; load MMCARGH and MMCARGL before writing to MMCCMD. Also, **make sure you do not modify the argument registers while they are being used for an operation.** The contents of the argument registers are kept after transfer to the shift register.

Table 10–21 shows the format for a command.

Figure 10–46. MMC Argument Registers (MMCARGH and MMCARGL)



Note: R/W-x = Read/Write-Reset value

Table 10–21. Command Format

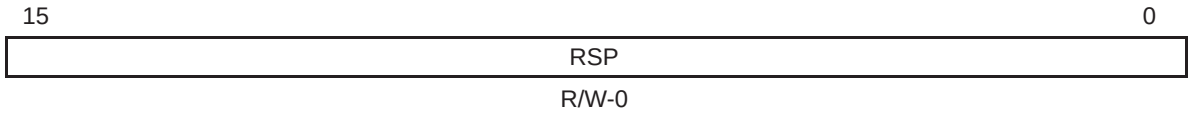
Register	Bit Position	Description
	47	Start bit
	46	Transmission bit
MMCCMD(5–0)	45–40	Command index
MMCARGH	39–24	Argument, high part
MMCARGL	23–8	Argument, low part
	7–1	CRC7
	0	End bit

10.8.16 MMC Response Registers (MMCRSPn)

Each command has a preset response type. When the MMC controller receives a response, it is stored in some or all of the eight response registers (MMCRSP7–MMCRSP0). The response registers are updated as the responses arrive, even if the CPU has not read the previous contents.

As shown in Figure 10–47, each of the response registers holds up to 16 bits. The tables that follow the figure show which registers are used for each type of response. Table 10–22 and Table 10–23 show response formats for the native mode. Note that in the native mode, the first byte of the response is a command index byte and stored in the MMC command index register (MMCIDX; see page). Table 10–24 and Table 10–25 show SPI mode response formats.

Figure 10–47. Format of an MMC Response Register (MMCRSPn)



Note: R/W-x = Read/Write-Reset value

Table 10–22. R1/R4/R5/R6 or R3 Response in the Native Mode

Register	Bit Position of Response
MMCIDX	47–40
MMCRSP7	39–24
MMCRSP6	23–8
MMCRSP5–0	— (Last byte of this type of response is not recorded)

Table 10–23. R2 Response in the Native Mode

Register	Bit Position of Response
MMCIDX	135–128
MMCRSP7	127–112
MMCRSP6	111–96
MMCRSP5	95–80
MMCRSP4	79–64
MMCRSP3	63–48
MMCRSP2	47–32
MMCRSP1	31–16
MMCRSP0	15–0

Table 10–24. R1 Response in SPI Mode

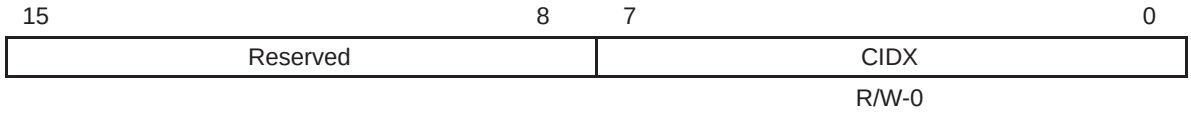
Register	Bit Position of Response
MMCRSP7	7–0 (stored in least significant byte of MMCRSP7)
MMCRSP6–0	—

Table 10–25. R2 Response in SPI Mode

Register	Bit Position of Response
MMCRSP7	15–0
MMCRSP6–0	—

10.8.17 MMC Command Index Register (MMCCIDX)

Figure 10–48. MMC Command Index Register (MMCCIDX)



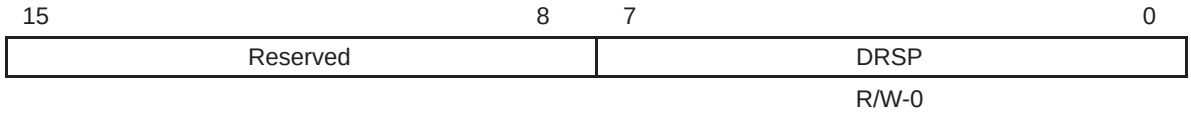
Note: R/W-x = Read/Write-Reset value

Table 10–26. MMC Command Index Register (MMCCIDX) Field Values

Bit	*field*	*symval*	Value	Description
15–8	Reserved			Reserved. The reserved bit location is always read as zero.
7–0	CIDX			Command index byte of a response, consists of start bit, transmission bit, and command index; stored when it is received in the native mode.

10.8.18 MMC Data Response Register (MMCDRSP)

Figure 10–49. MMC Data Response Register (MMCDRSP)



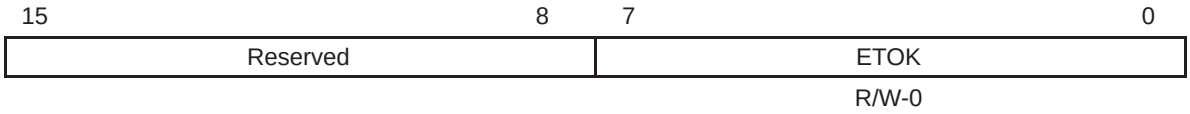
Note: R/W-x = Read/Write-Reset value

Table 10–27. MMC Data Response Register (MMCDRSP) Field Values

Bit	*field*	*symval*	Value	Description
15–8	Reserved			Reserved. The reserved bit location is always read as zero.
7–0	DRSP			Native mode: During a write operation, the CRC status token is stored in this register (see section 10.2.1.1 on page 10-11). SPI mode: During a write operation, the data response byte is stored in this register (see section 10.5.1.1 on page 10-34).

10.8.19 MMC SPI Error Token Register (MMCETOK)

Figure 10–50. MMC SPI Error Token Register (MMCETOK)



Note: R/W-x = Read/Write-Reset value

Table 10–28. MMC SPI Error Token Register (MMCETOK) Field Values

Bit	*field*	*symval*	Description
15–8	reserved		Reserved. These reserved bits are always read as 0s.
7–0	ETOK		Native mode: Not used SPI mode: If a read operation is not successful, the card responds with a 1-byte data error token (see section 10.5.1.2 on page 10-35). The MMC controller stores the token in this field.

Chapter 11

Real-Time Clock (RTC)

This chapter discusses the real-time clock (RTC) of the TMS320C55x™ (C55x™) DSPs. To determine whether a particular C55x DSP has an RTC, see the data sheet for that DSP.

Topic **Page**

11.1 Introduction to the Real-Time Clock (RTC)

The real-time clock (RTC) provides the following features:

- ❏ 100-year calendar up to year 2099
- ❏ Peripheral bus interface
- ❏ 32 bytes of configuration/status registers
- ❏ Counts seconds, minutes, hours, day of the week, date, month, and year with leap year compensation
- ❏ Binary-coded-decimal (BCD) representation of time, calendar, and alarm
- ❏ 12-hour (with AM and PM in 12-hour mode) or 24-hour clock modes
- ❏ Second, minute, hour, or day alarm interrupt
- ❏ Update Cycle interrupt
- ❏ Periodic interrupt
- ❏ Single interrupt port
- ❏ Supports external 32 kHz oscillator
- ❏ Separate power supply

The RTC has a separate clock domain and power supply. The clock is derived from the external 32 kHz crystal (RTCX1 and RTCX2). Figure 11–1 shows a block diagram of the RTC. Table 11–1 lists and describes the signals.

Figure 11–1. Real-Time Clock Block Diagram

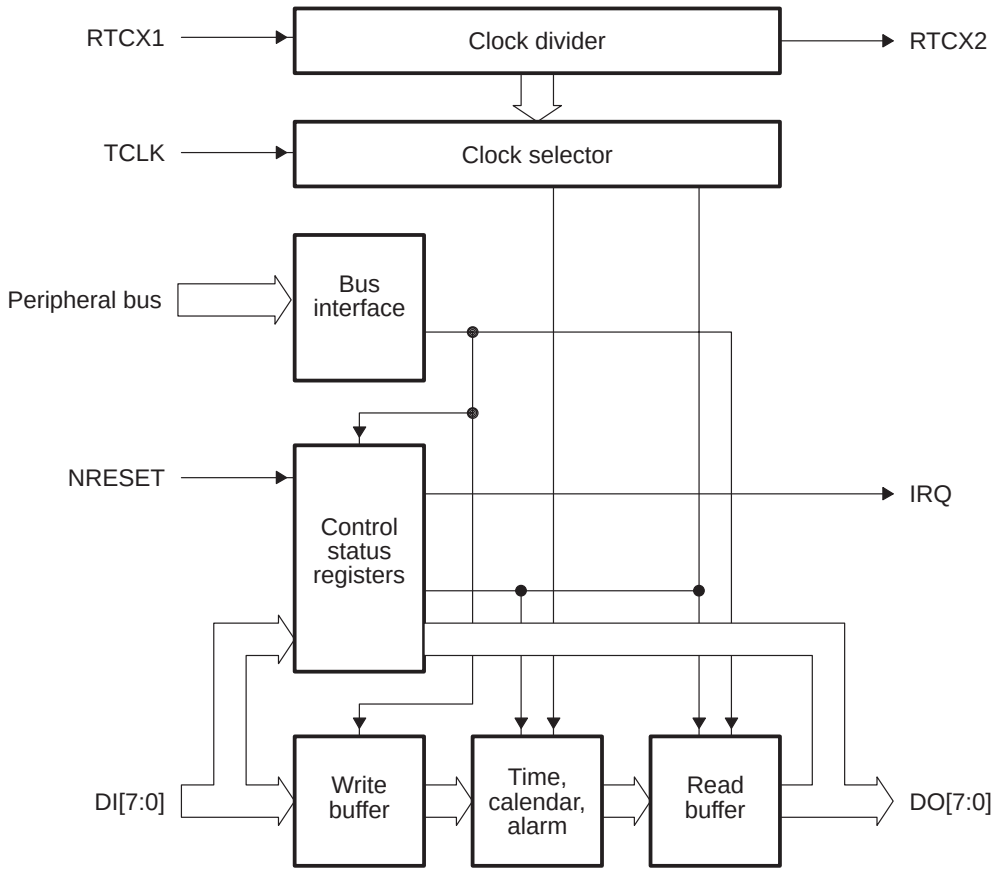


Table 11–1. Real-Time Clock Signal Descriptions

Name	I/O	Signal Descriptions
DI(7:0)	Input	Data input bus.
DO(7:0)	Output	Data output bus.
IRQ	Output	Interrupt request. IRQ is an active-high output that goes high when one of the three interrupts (periodic, alarm, or update-ended) is present and the corresponding interrupt enable bit, in the interrupt enable register, is set. IRQ remains high until the CPU reads the interrupt flag register. NRESET also clears pending interrupts.

Table 11–1. Real-Time Clock Signal Descriptions (Continued)

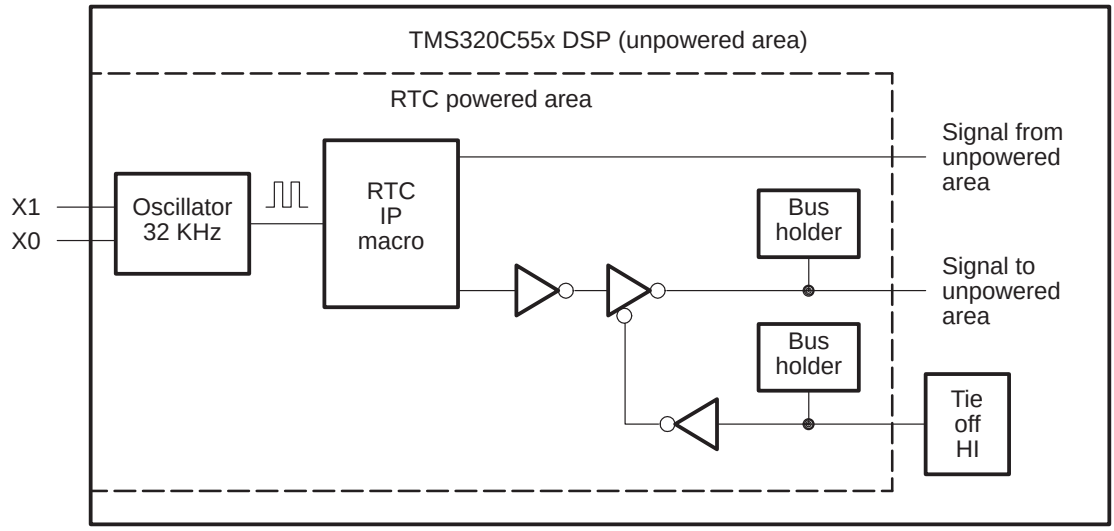
Name	I/O	Signal Descriptions
NRESET	Input	Reset. NRESET has no effect on the clock, timer, or calendar. When NRESET is low, the following occurs: ❏ Periodic Interrupt Enable (PIE) bit is cleared to 0. ❏ Alarm Interrupt Enable (AIE) bit is cleared to 0. ❏ Update-Ended Interrupt Enable (UIE) bit is cleared to 0. ❏ Interrupt Request Status Flag (IRQF) bit is cleared to 0. ❏ Periodic Interrupt Flag (PF) bit is cleared to 0. ❏ Alarm Interrupt Flag (AF) bit is cleared to 0. ❏ Update-Ended Interrupt Flag (UF) bit is cleared to 0. ❏ IRQ signal is in the high state. Note that these bits are not stable until NRESET is applied.
PS	Input	Power sense. Indication for power detection from the core.
RTCX1	Input	32.768 kHz time base.
RTCX2	Output	32.768 kHz time base.
TCLK	Input	Test clock. TCLK is only used for a test.

11.2 Real-Time Clock Power

The RTC has its own power supply; therefore, it will operate while the rest of the TMS320C55x DSP is powered off or has 0 voltage in its core and I/O pins. When the RTC detects the core is not powered, its outputs into the core are in a tristate mode and it uses busholders at the inputs of the RTC from the core, so they are not floating. Figure 11–2 shows a block diagram of this circuit. The input signals have a bus holder with a tristate inverter (IV2x1) to hold the state with the bus holder. The output does not require this special consideration because the output signal connects to a gate that is protected by an insulator and does not leak into the core logic.

The RTC can be idled by not supplying its 32 kHz oscillator signal and it can be turned off by not providing power to the RTC power supply pins, RCV_{DD} and RDV_{DD}.

Figure 11–2. Real-Time Clock Power Isolation Block Diagram



11.3 Real-Time Clock Registers

The RTC registers are listed in Table 11–2.

Table 11–2. Real-Time Clock Registers

Address (Hex)	Name	Description
1800h	RTCSEC	Seconds Register
1801h	RTCSECA	Seconds Alarm Register
1802h	RTCMIN	Minutes Register
1803h	RTCMINA	Minutes Alarm Register
1804h	RTCHOUR	Hours Register
1805h	RTCHOURA	Hours Alarm Register
1806h	RTCDAYW	Day of the Week Register
1807h	RTCDAYM	Day of the Month (date) Register
1808h	RTCMONTH	Month Register
1809h	RTCYEAR	Year Register
180Ah	RTCPINTR	Periodic Interrupt Selection Register
180Bh	RTCINTEN	Interrupt Enable Register
180Ch	RTCINTFL	Interrupt Flag Register
180Dh–1BFFh		Reserved

11.3.1 RTC Time, Alarm, and Calendar Registers

The time, calendar, and alarm registers are set or initialized by writing to the appropriate register bytes. The contents of the time, calendar, and alarm registers are in binary-coded-decimal (BCD) format; that is, the use of 4 binary bits representing a single-decimal digit. Before writing to the time and calendar registers, the SET bit in the interrupt enable register (RTCINTEN) should be set to prevent updates while an access is being attempted. The SET bit should be cleared after the data has been written to allow the RTC to update the time, calendar, and alarm bytes. When writing to the time, calendar, and alarm registers, all data must be in BCD format. Once initialized, the RTC makes all updates in BCD mode. Table 11–3 shows the BCD format of the time, calendar, and alarm registers.

Table 11–3. Real-Time Clock Registers Decimal Range and BCD Format

Address (Hex)	Name	Function	Decimal Range	BCD Format
1800h	RTCSEC	Seconds	0–59	00–59
1801h	RTCSECA	Seconds alarm	0–59	00–59
			don't care	C0–FF
1802h	RTCMIN	Minutes	0–59	00–59
1803h	RTCMINA	Minutes alarm	0–59	00–59
			don't care	C0–FF
1804h	RTCHOUR	12-hour mode	1–12	01–12 (AM), 81–92 (PM)
		24-hour mode	0–23	00–23
1805h	RTCHOURA	12-hour mode alarm	1–12	01–12 (AM), 81–92 (PM)
			don't care	C0–FF
		24-hour mode alarm	0–23	00–23
			don't care	C0–FF
1806h	RTCDAYW	Day of the Week (Sunday = 1)	1–7	1–7
		Day of the Week Alarm (Sunday = 1)	1–7	1–7
			don't care	8–F
1807h	RTCDAYM	Day of the month (Date)	1–31	01–31
1808h	RTCMONTH	Month (January = 01)	1–12	01–12
1809h	RTCYEAR	Year	0–99	00–99

The 12-hour or 24-hour time modes cannot be changed without reinitializing the hours register (RTCHOUR) and hours alarm register (RTCHOURA). When the 12-hour format is selected, the MSB of RTCHOUR selects either AM or PM; AM when the MSB is cleared to 0, PM when the MSB is set to 1.

The time, calendar, and alarm bytes are always accessible because they are double buffered. However, if the read of the time and calendar bytes occurs during an update, a problem may exist and the data being read may not be correct.

There are three (seconds, minutes, and hours) alarm registers that can be used in one of two ways:

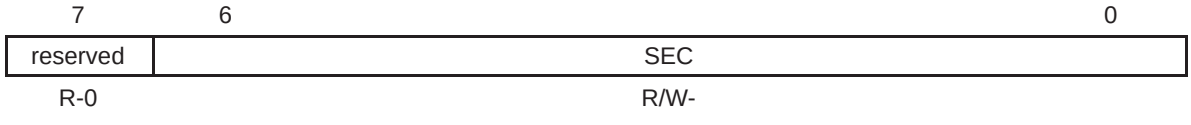
- Write an alarm time in the appropriate hours, minutes, and seconds for an alarm condition, the alarm interrupt occurs at the specified time of each day if the alarm enable bit is set.
- Insert a “don’t care” state in one or more of the three alarms bytes.

The “don’t care” state is any hexadecimal value from C0 to FF written to an alarm register; the two most-significant bits of each alarm register when set to 1 set the “don’t care” condition. For example, an alarm is generated each hour when the “don’t care” bits are set in the hours alarm register (RTCHOURA). Similarly, an alarm is generated every minute when the “don’t care” bits are set in RTCHOURA and the minutes alarm register (RTCMINA). The “don’t care” bits set in all three alarm registers (RTCHOURA, RTCMINA, and RTCSECA) generates an interrupt every second.

The DAEN bit in the day of the week register (RTCDAYW) allows you to extend the alarm interrupt up to once per week. When the DAEN bit is set to 1, the upper-half bits (bits 7–4) of RTCDAYW are assigned for the day-of-the-week alarm. Similar to other alarm registers, the DAR bits represent the alarm value and bit 7 represents the “don’t care” condition. Only bits 7–4 of RTCDAYW are transferred when the SET bit, in RTCINTEN, is cleared to 0. When the DAEN bit is cleared to 0, the day-of-the-week alarm is disabled and the DAY bits of RTCDAYW are transferred. The seconds, minutes, or hours data is kept with same data format, when PS falls.

11.3.1.1 RTC Seconds Register (RTCSEC)

Figure 11–3. RTC Seconds Register (RTCSEC)



Note: R/W-x = Read/Write-Reset value

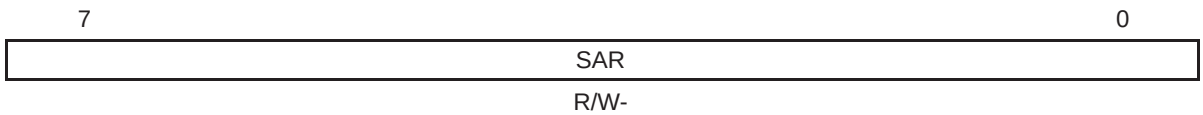
Table 11–4. RTC Seconds Register (RTCSEC) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7	reserved			Reserved. The reserved bit location is always read as zero.
6–0	SEC		00–59	Seconds select bits. This BCD value sets the second of the time.

† For CSL C macro implementation, use the notation RTC_RTCSEC_*field_symval*

11.3.1.2 RTC Seconds Alarm Register (RTCSECA)

Figure 11–4. RTC Seconds Alarm Register (RTCSECA)



Note: R/W-x = Read/Write-Reset value

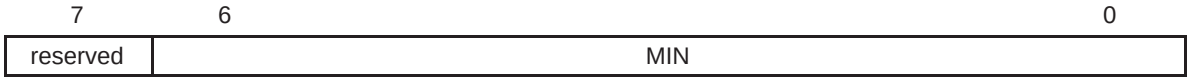
Table 11–5. RTC Seconds Alarm Register (RTCSECA) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7–0	SAR		00–59	Seconds alarm select bits. This BCD value sets the second of the alarm time. When bits 7 and 6 are set to 1, a “don’t care” condition is set. The “don’t care” bits set in all three alarm registers (RTCHOURA, RTCMINA, and RTCSECA) generates an interrupt every second.

† For CSL C macro implementation, use the notation RTC_RTCSECA_*field_symval*

11.3.1.3 RTC Minutes Register (RTCMIN)

Figure 11–5. RTC Minutes Register (RTCMIN)



Note: R/W-x = Read/Write-Reset value

Table 11–6. RTC Minutes Register (RTCMIN) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7	reserved			Reserved. The reserved bit location is always read as zero.
6–0	MIN		00–59	Minutes select bits. This BCD value sets the minute of the time.

† For CSL C macro implementation, use the notation RTC_RTCMIN_*field_symval*

11.3.1.4 RTC Minutes Alarm Register (RTCMINA)

Figure 11–6. RTC Minutes Alarm Register (RTCMINA)

Note: R/W-x = Read/Write-Reset value

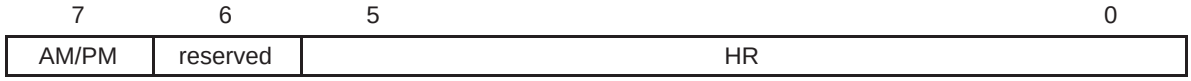
Table 11–7. RTC Minutes Alarm Register (RTCMINA) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7–0	MAR		00–59	Minutes alarm select bits. This BCD value sets the minute of the alarm time. When bits 7 and 6 are set to 1, a “don’t care” condition is set. The “don’t care” bits set in the hours and minutes alarm registers (RTCHOURA and RTCMINA) generates an interrupt every minute.

† For CSL C macro implementation, use the notation RTC_RTCMINA_*field_symval*

11.3.1.5 RTC Hours Register (RTCHOUR)

Figure 11–7. RTC Hours Register (RTCHOUR)



Note: R/W-x = Read/Write-Reset value

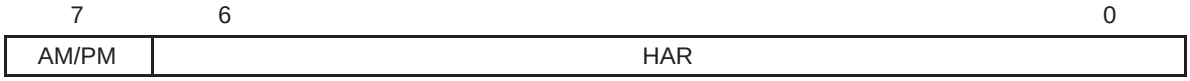
Table 11–8. RTC Hours Register (RTCHOUR) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7	AM/PM			AM/PM select bit.
			0	Time is set for AM.
			1	Time is set for PM.
6	reserved			Reserved. The reserved bit location is always read as zero.
5–0	HR			Hours select bits. This BCD value sets the hour of the time.
				For 12-hour mode (TM = 0 in RTCINTEN):
			01–12	This BCD value in conjunction with the AM/PM bit sets the hour of the time. For AM, AM/PM bit must be cleared to 0; for PM, AM/PM bit must be set to 1.
				For 24-hour mode (TM = 1 in RTCINTEN):
			00–23	This BCD value in conjunction with the AM/PM bit sets the hour of the time. AM/PM bit must be cleared to 0.

† For CSL C macro implementation, use the notation RTC_RTCHOUR_*field_symval*

11.3.1.6 RTC Hours Alarm Register (RTCHOURA)

Figure 11–8. RTC Hours Alarm Register (RTCHOURA)



Note: R/W-x = Read/Write-Reset value

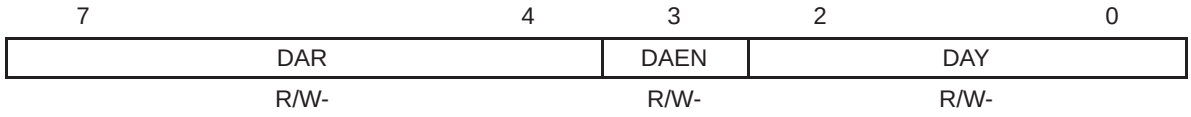
Table 11–9. RTC Hours Alarm Register (RTCHOURA) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7	AM/PM			AM/PM select bit.
			0	Alarm time is set for AM or is in 24-hour mode.
			1	Alarm time is set for PM.
6–0	HAR			Hours alarm select bits. This BCD value sets the hour of the alarm time. When bits 6 and 5 are set to 1, a “don’t care” condition is set. The “don’t care” bits generate an interrupt every hour.
				For 12-hour mode (TM = 0 in RTCINTEN):
			01–12	This BCD value in conjunction with the AM/PM bit sets the hour of the alarm time. For AM, AM/PM bit must be cleared to 0; for PM, AM/PM bit must be set to 1.
				For 24-hour mode (TM = 1 in RTCINTEN):
			00–23	This BCD value in conjunction with the AM/PM bit sets the hour of the alarm time. AM/PM bit must be cleared to 0.

† For CSL C macro implementation, use the notation RTC_RTCHOURA_*field_symval*

11.3.1.7 RTC Day of the Week and Day Alarm Register (RTCDAYW)

Figure 11–9. RTC Day of the Week and Day Alarm Register (RTCDAYW)



Note: R/W-x = Read/Write-Reset value

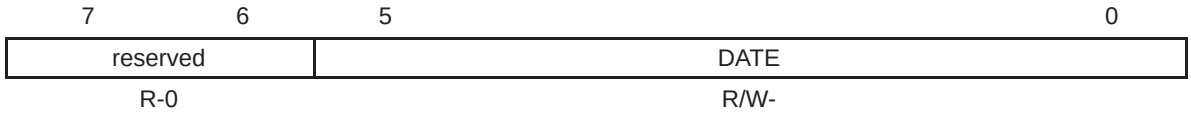
Table 11–10. RTC Day of the Week and Day Alarm Register (RTCDAYW) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7–4	DAR		1–7	Day-of-the-week alarm select bits. This BCD value sets the day-of-the-week alarm (Sunday = 1). When bit 7 is set to 1, a “don’t care” condition is set. The “don’t care” bit generates an interrupt every week.
3	DAEN			Day-of-the-week alarm enable bit.
			0	Day-of-the-week alarm is disabled.
			1	Day-of-the-week alarm is enabled. Day-of-the-week alarm is set to BCD value of DAR bits.
2–0	DAY		1–7	Day-of-the-week select bits. This BCD value sets the day of the week (Sunday = 1).

† For CSL C macro implementation, use the notation RTC_RTCDAYW_*field_symval*

11.3.1.8 RTC Day of the Month (Date) Register (RTCDAYM)

Figure 11–10. RTC Day of the Month (Date) Register (RTCDAYM)



Note: R/W-x = Read/Write-Reset value

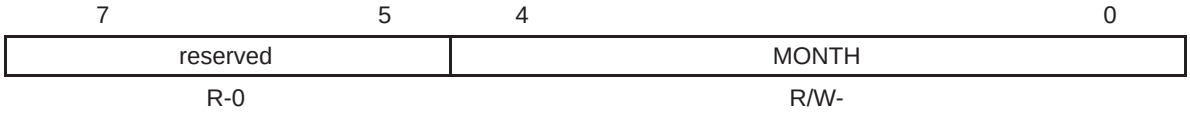
Table 11–11. RTC Day of the Month (Date) Register (RTCDAYM) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7–6	reserved			Reserved. The reserved bit location is always read as zero.
5–0	DATE		01–31	Date select bits. This BCD value sets the date of the calendar.

† For CSL C macro implementation, use the notation RTC_RTCDAYM_*field_symval*

11.3.1.9 RTC Month Register (RTCMONTH)

Figure 11–11.RTC Month Register (RTCMONTH)



Note: R/W-x = Read/Write-Reset value

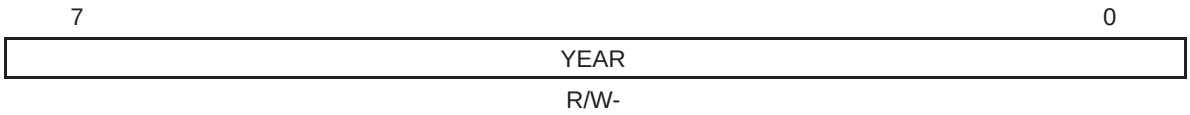
Table 11–12. RTC Month Register (RTCMONTH) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7–5	reserved			Reserved. The reserved bit location is always read as zero.
4–0	MONTH		01–12	Month select bits. This BCD value sets the month of the calendar (January = 01).

† For CSL C macro implementation, use the notation RTC_RTCMONTH_*field_symval*

11.3.1.10 RTC Year Register (RTCYEAR)

Figure 11–12. RTC Year Register (RTCYEAR)



Note: R/W-x = Read/Write-Reset value

Table 11–13. RTC Year Register (RTCYEAR) Field Values

Bit	*field*†	*symval*†	BCD Value	Description
7–0	YEAR		00–99	Year select bits. This BCD value sets the year of the calendar.

† For CSL C macro implementation, use the notation RTC_RTCYEAR_*field_symval*

11.3.2 RTC Interrupt Registers

The RTC has three interrupt control registers that can be accessed at any time.

11.3.2.1 RTC Periodic Interrupt Selection Register (RTCPINTR)

Figure 11–13. RTC Periodic Interrupt Selection Register (RTCPINTR)

7	6	5	4	0
UIP	reserved		RS	
R/W-	R-0		R/W-	

Note: R/W-x = Read/Write-Reset value

Table 11–14. RTC Periodic Interrupt Selection Register (RTCPINTR) Field Values

Bit	*field*†	*symval*†	Value	Description
7	UIP			Update-in-progress bit.
			0	Update cycle will not occur for at least 244 µs.
			1	After UIP goes high, update cycle will occur in 244 µs.
6–5	reserved			Reserved. The reserved bit location is always read as zero.
4–0	RS			Periodic interrupt rate select bits.
			00000	None
			00001	3.90625 ms
			00010	7.8125 ms
			00011	122.070 µs
			00100	244.141 µs
			00101	488.281 µs
			00110	976.5625 µs
			00111	1.953125 ms
			01000	3.90625 ms
			01001	7.8125 ms
			01010	15.625 ms
			01011	31.25 ms

† For CSL C macro implementation, use the notation RTC_RTCPINTR_*field_symval*

Table 11–14. RTC Periodic Interrupt Selection Register (RTCPINTR) Field Values (Continued)

Bit	*field*†	*symval*†	Value	Description
			01100	62.5 ms
			01101	125 ms
			01110	250 ms
			01111	500 ms
			10000 –11111	1 minute

† For CSL C macro implementation, use the notation RTC_RTCPINTR_*field_symval*

Update-In-Progress (UIP) Bit

The update-in-progress (UIP) bit is a status flag that can be monitored. When the UIP bit is set to 1, the update cycle will occur soon. When the UIP bit is cleared to 0, the update cycle will not occur for at least 244 µs. All time and calendar data is accessible when the UIP bit is 0. The UIP bit is a read-only bit and is not affected by an NRESET signal. Writing a 1 to the SET bit in the interrupt enable register (RTCINTEN) clears the UIP bit.

The UIP bit is used to determine how soon the update cycle will occur. Because the RTC has a register-based time and calendar, the RTC does not require an additional clock period to transfer data from time and calendar counters. This means the time and calendar data can be read at any time other than at the update cycle time.

Periodic Interrupt Rate Select (RS) Bits

The RS bits select 1 of the 13 taps out of the 16-stage divider or disable the divider output. The selected tap is used for generating a periodic interrupt. Bit 4 of RS affects only the 1 minute periodic interrupt and the other RS bits are ignored when bit 4 is set to 1. These bits are read/write bits and are not affected by an NRESET signal.

11.3.2.2 RTC Interrupt Enable Register (RTCINTEN)

Figure 11–14. RTC Interrupt Enable Register (RTCINTEN)

7	6	5	4	3	2	1	0
SET	PIE	AIE	UIE	reserved		TM	reserved
R/W-	R/W-	R/W-	R/W-	R-0		R/W-	R-0

Note: R/W-x = Read/Write-Reset value

Table 11–15. RTC Interrupt Enable Register (RTCINTEN) Field Values

Bit	*field*†	*symval*†	Value	Description
7	SET			SET bit isolates or connects the write and read buffers from the time, calendar, and alarm registers. Loading new data to time, calendar, or alarm and updating time and calendar data to read buffer are blocked while the SET bit is set to 1. The SET bit is a read/write bit that is not affected by an NRESET signal.
			0	The write and read buffers are connected to the time, calendar, and alarm registers.
			1	The write and read buffers are isolated from the time, calendar, and alarm registers so a read or write operation can be executed independent with the update cycle.
6	PIE			Periodic interrupt enable bit allows the periodic interrupt flag (PF) bit in the interrupt flag register (RTCINTFL) to cause an active IRQ. The PIE bit is a read/write bit that is cleared by an NRESET signal.
			0	Periodic interrupts are disabled.
			1	Periodic interrupts are enabled.
5	AIE			Alarm interrupt enable (AIE) bit allows the alarm interrupt flag (AF) bit in the interrupt flag register (RTCINTFL) to cause an active IRQ. The AIE bit is a read/write bit that is cleared by an NRESET signal.
			0	Alarm interrupts are disabled.
			1	Alarm interrupts are enabled.
4	UIE			Update-ended interrupt enable (UIE) bit allows the update-ended flag (UF) bit in the interrupt flag register (RTCINTFL) to cause an active IRQ. The UIE bit is a read/write bit that is cleared by an NRESET signal.
			0	Update-ended interrupts are disabled.
			1	Update-ended interrupts are enabled.

† For CSL C macro implementation, use the notation RTC_RTCINTEN_*field_symval*

Table 11–15. RTC Interrupt Enable Register (RTCINTEN) Field Values (Continued)

Bit	*field*†	*symval*†	Value	Description
3–2	reserved			Reserved. The reserved bit location is always read as zero.
1	TM			Time mode bit indicates whether the hour byte is in 24-hour mode or 12-hour mode. The TM bit is a read/write bit that is not affected by an NRESET signal.
			0	12-hour mode
			1	24-hor mode
0	reserved			Reserved. The reserved bit location is always read as zero.

† For CSL C macro implementation, use the notation RTC_RTCINTEN_*field_symval*

11.3.2.3 RTC Interrupt Flag Register (RTCINTFL)

Figure 11–15. RTC Interrupt Flag Register (RTCINTFL)

7	6	5	4	3 – 0
IRQF	PF	AF	UF	reserved
R-	R-	R-	R-	R-0

Note: R/W-x = Read/Write-Reset value

Table 11–16. RTC Interrupt Flag Register (RTCINTFL) Field Values

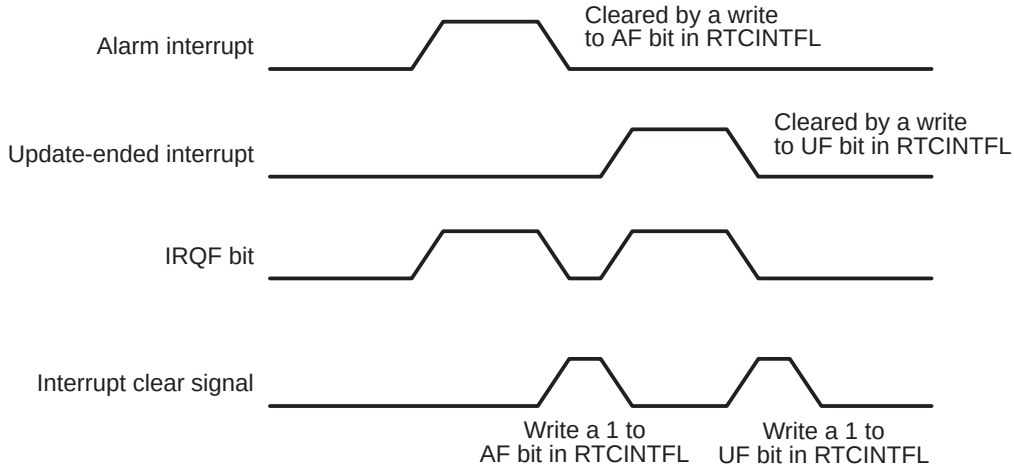
Bit	*field*	Value	Description
7	IRQF		Interrupt request status flag bit indicates if an interrupt has occurred.
		0	No interrupt flags are set.
		1	One or more of the interrupt flags and the corresponding enables are set. Any time the IRQF bit is set, the IRQ signal is driven active high. To clear an interrupt flag, write a 1 to the interrupt flag bit that caused the interrupt.
6	PF		Periodic interrupt flag bit indicates if a periodic interrupt has occurred.
		0	No periodic interrupt occurred.
		1	Periodic interrupt has occurred. The PF bit is set based on the status of the PIE bit in the interrupt enable register (RTCINTEN) and cleared by an NRESET signal or by writing a 1 into this bit.

Table 11–16. RTC Interrupt Flag Register (RTCINTFL) Field Values (Continued)

Bit	*field*	Value	Description
5	AF		Alarm interrupt flag bit indicates if an alarm interrupt has occurred.
		0	No alarm interrupt occurred.
		1	Alarm interrupt has occurred. The AF bit is set based on the status of the AIE bit in the interrupt enable register (RTCINTEN) and cleared by an NRESET signal or by writing a 1 into this bit.
4	UF		Update-ended interrupt flag bit indicates if an update-ended interrupt has occurred.
		0	No update-ended interrupt occurred.
		1	Update-ended interrupt has occurred. The UF bit is set based on the status of the UIE bit in the interrupt enable register (RTCINTEN) and cleared by an NRESET signal or by writing a 1 into this bit.
3–0	reserved		Reserved. The reserved bit location is always read as zero.

Figure 11–16 shows the events when both an alarm interrupt and an update-ended interrupt occur and how they are cleared by writing a 1 into their corresponding bit fields of the interrupt flag register (RTCINTFL). In Figure 11–16 the alarm interrupt occurs, setting the IRQF bit and AF bit in RTCINTFL. The AF bit is cleared by writing a 1 to the AF bit in RTCINTFL. After the alarm interrupt occurs, the update-ended interrupt occurs, setting the IRQF bit and UF bit in RTCINTFL. The UF bit is cleared by writing a 1 to the UF bit in RTCINTFL. The IRQF bit is cleared when a 1 is written to AF bit and UF bit.

Figure 11–16. Clearing Alarm Interrupt and Update-Ended Interrupts



11.4 Real-Time Clock Interrupts

The RTC provides three separate interrupts. The alarm interrupt can be programmed to occur at rates from once per second to once per week. The periodic interrupt can be selected for rates from 500 ms to 122 μs.

11.4.1 Interrupt Enable and Flag Bits

Three bits (PIE, AIE, and UIE) in the interrupt enable register (RTCINTEN) enable the interrupts. Writing a logic 1 to an interrupt enable bit allows that interrupt to be initiated when the event occurs. If an interrupt flag is already set when an interrupt is enabled, the IRQ signal is immediately set to an active level, although the interrupt initiating the event may have occurred much earlier. Pending interrupts should be cleared before first enabling new interrupts.

When an interrupt event occurs, the corresponding flag bit (PF, AF, or UF) is set in the interrupt flag register (RTCINTFL). These flag bits are set independent of the state of the corresponding enable bit in RTCINTEN. The flag bit can be used in a polling mode without enabling the corresponding enable bits. However, care should be taken when using the flag bits since they are cleared each time RTCINTFL is read. Double latching is used to insure that bits are stable throughout the read cycle. If a new interrupt occurs during the read cycle, the interrupt is set in the corresponding flag bit after the read cycle is completed. All three bits can be set when reading RTCINTFL.

If one of the three flag bits becomes active and the corresponding enable bit is set, the IRQ signal becomes active high. The IRQ signal is asserted as long as at least one of the three interrupt flag bits and enable bits are set. The IRQF bit in RTCINTFL is set to 1 when IRQ is driven active high. The IRQF bit indicates that one or more interrupts have been initiated by the RTC. Reading RTCINTFL clears the IRQF bit.

The RTC also provides ALARMIRQ that is driven active, when the AF bit in RTCINTFL is set active.

11.4.2 Periodic Interrupt

The periodic interrupt causes the IRQ signal to go to an active state from once every 1 minute to once every 122 μs. The periodic interrupt rate is selected using the RS bits in the periodic interrupt selection register (RTCPINTR), see Table 11–17. Changing the four LSBs of the RS bits affects the periodic interrupt rate. When bit 4 of the RS bits is set to 1, the periodic interrupt is asserted every minute regardless of the other RS bit values. The minute interrupt is triggered when the second register is changed from 59 to 00. The periodic interrupts are enabled by the PIE bit in RTCINTEN.

Table 11–17. RTC Periodic Interrupt Rates Based on RS Bits

RTCPINTR Bits					
RS4	**RS3**	**RS2**	**RS1**	**RS0**	**Periodic Interrupt Rate**
0	0	0	0	0	None
0	0	0	0	1	3.90625 ms
0	0	0	1	0	7.8125 ms
0	0	0	1	1	122.070 µs
0	0	1	0	0	244.141 µs
0	0	1	0	1	488.281 µs
0	0	1	1	0	976.5625 µs
0	0	1	1	1	1.953125 ms
0	1	0	0	0	3.90625 ms
0	1	0	0	1	7.8125 ms
0	1	0	1	0	15.625 ms
0	1	0	1	1	31.25 ms
0	1	1	0	0	62.5 ms
0	1	1	0	1	125 ms
0	1	1	1	0	250 ms
0	1	1	1	1	500 ms
1	X	X	X	X	1 minute

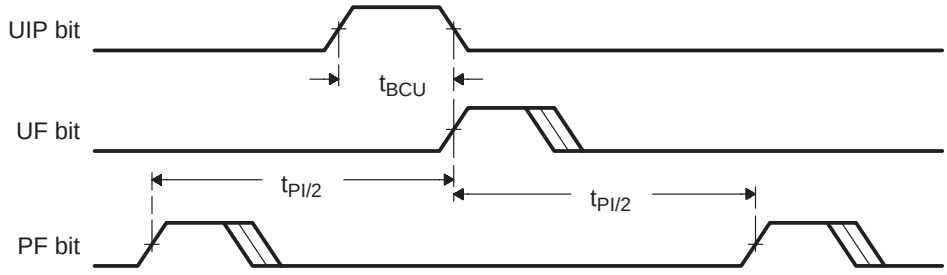
11.5 Real-Time Clock Update Cycle

The RTC executes an update cycle once per second regardless of the SET bit value in the interrupt enable register (RTCINTEN). When the SET bit is set to 1, the time, calendar, and alarm bytes are double buffered and become separate from the actual time, calendar, and alarm registers. This allows the RTC to maintain accuracy independent of reading or writing to the buffers. The update cycle also compares each alarm byte with the corresponding time byte.

There are three methods that the RTC uses to avoid any possibility of accessing inconsistent time and calendar data.

- The first method uses the update-ended interrupt enable (UIE) bit in RTCINTEN. If the update-ended interrupt is enabled, an interrupt occurs after every update cycle that indicates that over 999 ms are available to read valid time and calendar data. If this interrupt is used, the IRQF bit in the interrupt flag register (RTCINTFL) should be cleared before leaving the interrupt routine.
- The second method uses the update-in-progress (UIP) bit in the periodic interrupt selection register (RTCPINTR) to determine if the update cycle is in progress. The UIP bit pulses once per second, and after the UIP bit goes high, the update occurs 244 μs later. If a low is read on the UIP bit, the update does not occur for at least 244 μs.
- The third method uses the periodic interrupt flag (PF) bit in RTCINTFL to determine if the update cycle is in progress. The UIP bit in RTCPINTR is set between the setting of the PF bit in RTCINTFL (see Figure 11–17). The periodic interrupt that is greater than t_{BCU} allows determining the update cycle. The read should be completed within $t_{PI/2}$ after the PF bit is set.

Figure 11–17. Setting of the Update-In-Progress (UIP) Bit in RTCPINTR



11.6 Real-Time Clock Operating Modes

The RTC works with three operation modes: a normal mode, a battery-backup mode, and a test mode. No mode affects the time, calendar, and alarm registers.

11.6.1 Normal Mode

In a normal mode, the power is fully supplied and all registers can be accessed.

11.6.2 Low-Power Mode

In low-power mode, the TMS320C55x device is not powered, but the RTC is powered. In this mode, all ports should be isolated from nonpowered sections of the device. The calendar timer is working in this mode.

11.6.3 Test Mode

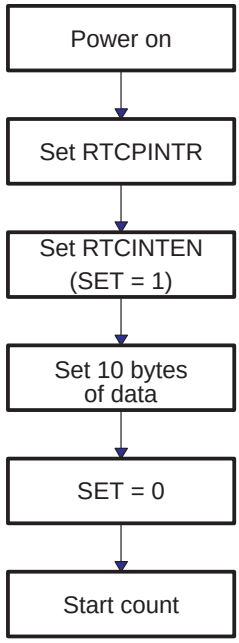
Three test modes are provided for TI-use only and are used for testing the RTC.

11.7 Real-Time Clock Programming Sequence

11.7.1 Initialization

The 10 bytes of data in the time, calendar, and alarm registers have to be initialized with data after setting the appropriate bits in both the periodic interrupt selection register (RTCPINTR) and the interrupt enable register (RTCINTEN), see Figure 11–18. The SET bit in RTCINTEN has to be set to 1 before the 10 bytes of data are initialized, then the counting starts when the SET bit is cleared to 0.

Figure 11–18. Initialization Flow Diagram



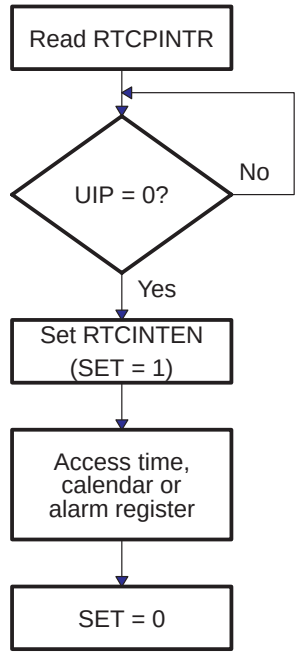
11.7.2 Read/Write Time, Calendar, or Alarm Register

Reading or writing of the registers should be followed by a detecting update cycle and the data mode must be same for all registers.

11.7.2.1 Using UIP Bit to Access Registers

Figure 11–19 shows how to access the time, calendar, or alarm register by using the update-in-progress (UIP) bit in the periodic interrupt selection register (RTCPINTR). If the UIP bit is cleared to 0, an update cycle will not occur for at least 244 μs.

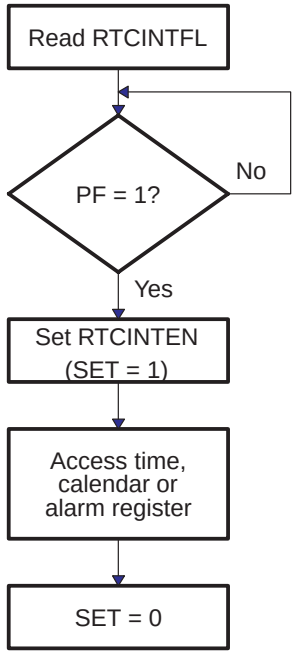
Figure 11–19. Using UIP Bit to Access Registers



11.7.2.2 Using PF Bit to Access Registers

Figure 11–20 shows how to access the time, calendar, or alarm register by using the periodic interrupt flag (PF) bit in the interrupt flag register (RTCINTFL). If the PF bit is set to 1, an update cycle will not occur for at least a time of $t_{PI/2}$.

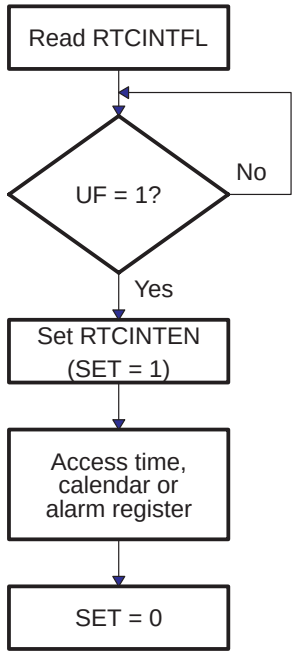
Figure 11–20. Using PF Bit to Access Registers



11.7.2.3 Using UF Bit to Access Registers

Figure 11–21 shows how to access the time, calendar, or alarm register by using the update ended interrupt flag (UF) bit in the interrupt flag register (RTCINTFL). If the UF bit is set to 1, an update cycle will not occur for at least 1 second.

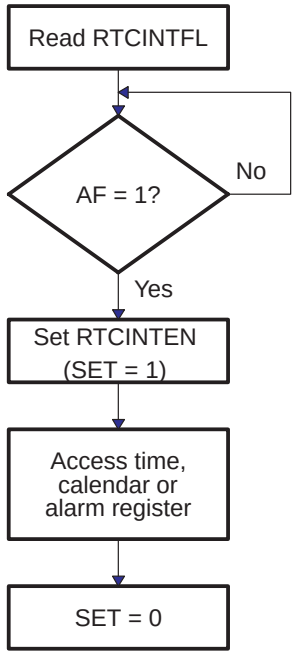
Figure 11–21. Using UF Bit to Access Registers



11.7.2.4 Using AF Bit to Access Registers

Figure 11–22 shows how to access the time, calendar, or alarm register by using the alarm interrupt flag (AF) bit in the interrupt flag register (RTCINTFL). If the AF bit is set to 1, an update cycle will not occur for at least 1 second.

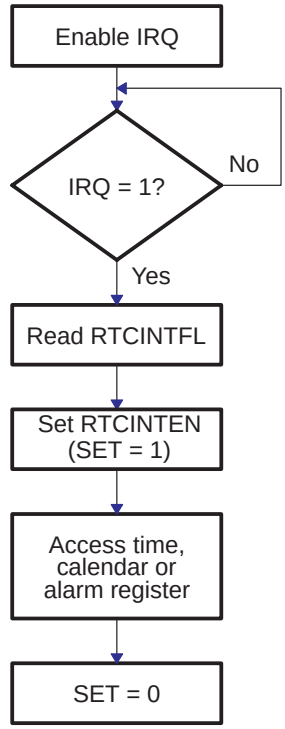
Figure 11–22. Using AF Bit to Access Registers



11.7.2.5 Using IRQ Signal or IRQF Bit to Access Registers

Figure 11–23 shows how to access the time, calendar, or alarm register by using the status of the IRQ pin. If the IRQ pin is active high, an update cycle will not occur. Any time the interrupt request status flag (IRQF) bit in the interrupt flag register (RTCINTFL) is set to 1, the IRQ signal is driven active high.

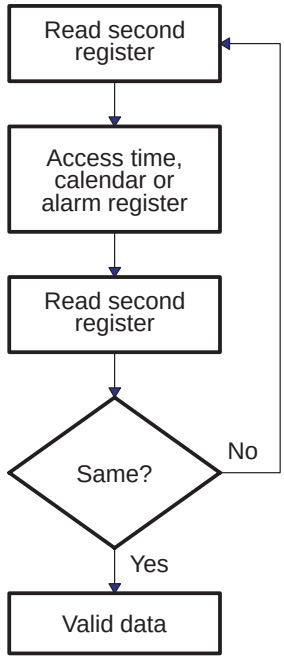
Figure 11–23. Using IRQ Signal or IRQF Bit to Access Registers



11.7.2.6 Using No Flags to Access Registers

Since the timer, calendar, or alarm register is protected, it can be read at any time regardless of the update cycle. Figure 11–24 shows that the register should be read at least twice, to avoid the possibility of reading unstable data that could occur while the update cycle has occurred.

Figure 11–24. Using No Flags to Access Registers



Chapter 12

System Control Registers

This chapter describes the boot mode register and system register of the TMS320C55x™ DSP.

Topic **Page**

12.1 Boot Mode Register

The boot mode register (BOOT_MOD) is a read-only register in the peripheral bus controller of the DSP. This register allows you to see the boot mode selected by the three BOOTM pins of the DSP. For example, the following table shows the boot modes available on the TMS320VC5510 DSPs. Only the three lowest bits of BOOT_MOD are used; other bits are undefined (shown as x in the table). For the I/O address of BOOT_MOD, see the data sheet for your TMS320C55x DSP.

Table 12–1. TMS320VC5510 Boot Modes Selected With the BOOTM Pins and Reflected in BOOT_MOD

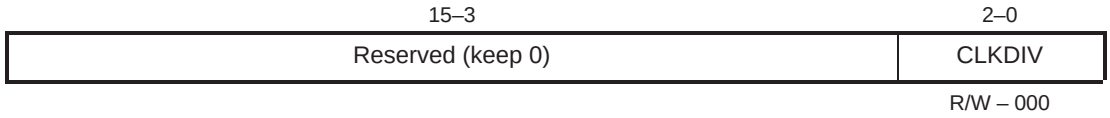
BOOTM2 At Reset	BOOTM1 At Reset	BOOTM0 At Reset	BOOT_MOD After Reset	Boot Mode Selected
Low	Low	Low	xxxx xxxx xxxx x000b	No boot
Low	Low	High	xxxx xxxx xxxx x001b	Reserved (do not use)
Low	High	Low	xxxx xxxx xxxx x010b	Reserved (do not use)
Low	High	High	xxxx xxxx xxxx x011b	16-bit asynchronous memory boot
High	Low	Low	xxxx xxxx xxxx x100b	32-bit asynchronous memory boot
High	Low	High	xxxx xxxx xxxx x101b	EHPI boot
High	High	Low	xxxx xxxx xxxx x110b	McBSP 0 boot, 16-bit serial word length
High	High	High	xxxx xxxx xxxx x111b	McBSP 0 boot, 8-bit serial word length

12.2 System Register

The system register (SYSR) includes the CLKDIV bits, which determine how much the CPU clock is divided down for the CLKOUT pin (for more details, see section 2.6 on page 2-9). SYSR is an I/O-mapped register. Figure 12–1 and Table 12–2 describe the bit fields of SYSR.

Figure 12–1. System Register (SYSR)

SYSR



Legend:

R/W Read/write access
– X X is the value after a DSP reset

Table 12–2. SYSR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15–3	Reserved	Bits 15–3 of SYSR are not available for your use. Whenever you modify SYSR, write 0s to bits 15–3.	–
2–0	CLKDIV	Clock divide-down value for CLKOUT. CLKDIV determines the amount by which the CPU clock is divided down to produce the signal on the CLKOUT pin. If you want the CLKOUT frequency and the CPU clock frequency to be the same, make CLKDIV = 000b.	000b
		000b CLKOUT frequency = 1/1 × CPU clock frequency	
		001b CLKOUT frequency = 1/2 × CPU clock frequency	
		010b CLKOUT frequency = 1/3 × CPU clock frequency	
		011b CLKOUT frequency = 1/4 × CPU clock frequency	
		100b CLKOUT frequency = 1/5 × CPU clock frequency	
		101b CLKOUT frequency = 1/6 × CPU clock frequency	
		110b CLKOUT frequency = 1/7 × CPU clock frequency	
		111b CLKOUT frequency = 1/8 × CPU clock frequency	

Chapter 13

Timer

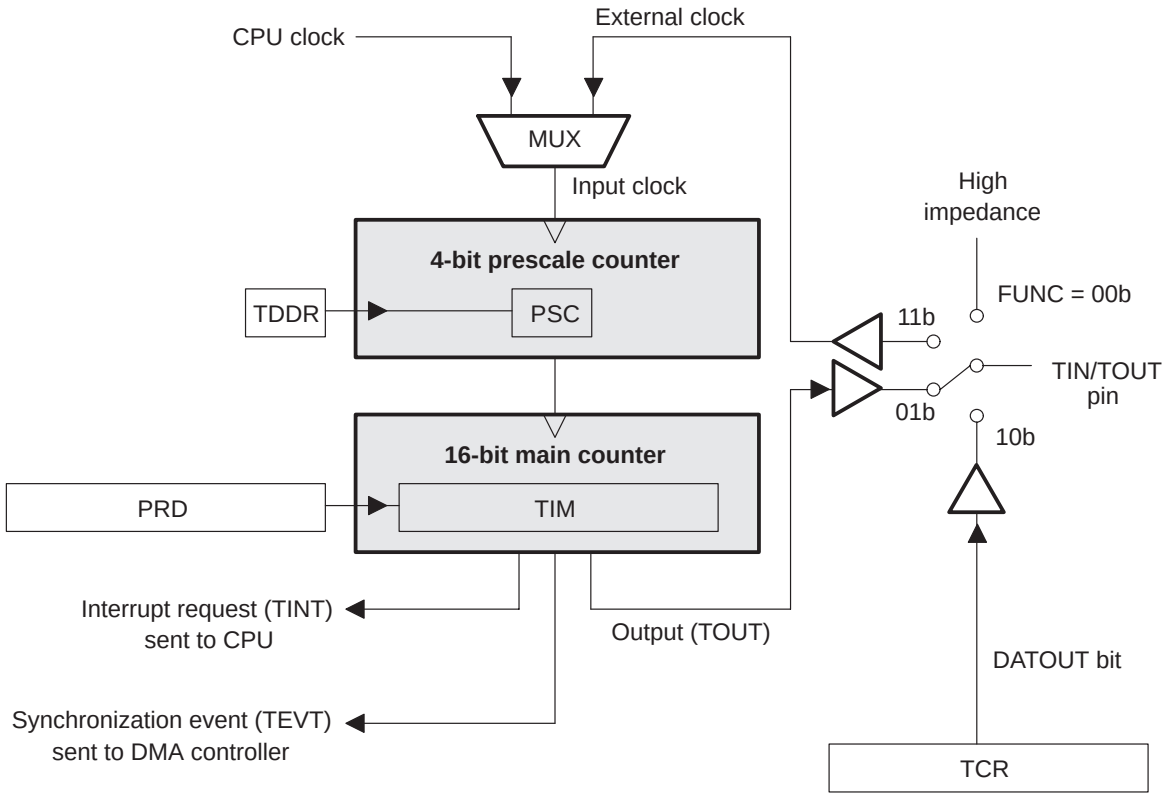
This chapter describes the type of timer available on the TMS320C55x™ DSPs. The C55x™ DSP has two identical but independent software-programmable timers. Two uses for the timers are to generate periodic interrupts and to provide periodic signals to devices outside the C55x DSP.

Topic **Page**

13.1 Introduction to the Timer

Each timer has up to a 20-bit dynamic range provided by two counters: a 4-bit prescale counter and a 16-bit main counter. Figure 13–1 shows a high-level diagram of the timer.

Figure 13–1. Conceptual Block Diagram of the Timer



The timer has two count registers (PSC and TIM) and two period registers (TDDR and PRD). During timer initialization or during timer reloading, the contents of the period registers are copied into the count registers. A timer control register (TCR) controls and monitors the operation of the timer and the timer pin (TIN/TOUT). Depending on the value of the FUNC bits in TCR, the pin can be a general-purpose output (connected to the DATOUT bit of TCR), a timer output, or a clock input—or it can be in the high impedance state. (The details of the timer registers are in section 13.10 on page 13-15. More information about the timer pin is in section 13.2.)

The prescale counter is driven by an input clock, which may be the CPU clock or an external clock. PSC is decremented by 1 every input clock cycle. One cycle after PSC reaches 0, the TIM is decremented by 1. One cycle after TIM reaches 0, the timer sends an interrupt request (TINT) to the CPU, a synchronization event (TEVT) to the DMA controller, and (if applicable) an output to the timer pin. The rate at which timer sends these signals is

$$\text{TINT rate} = \frac{\text{Input clock rate}}{(\text{TDDR} + 1) \times (\text{PRD} + 1)}$$

13.2 Timer Pin

Each timer has one pin. The pin can be configured in the ways described in Table 13–1. The two FUNC bits in the timer control register (TCR) define the function of the timer pin and determine the required clock source for the timer. As described in the table, in some cases, the signal on the pin is affected by other TCR bits (POLAR, C/P, and PWID, or DATOUT).

Sections 13.2.1 through 13.2.4 show different uses for the timer pin (different settings for the FUNC bits).

Note:

There are limitations on changing the FUNC bits. For details, see section 13.6 on page 13-11.

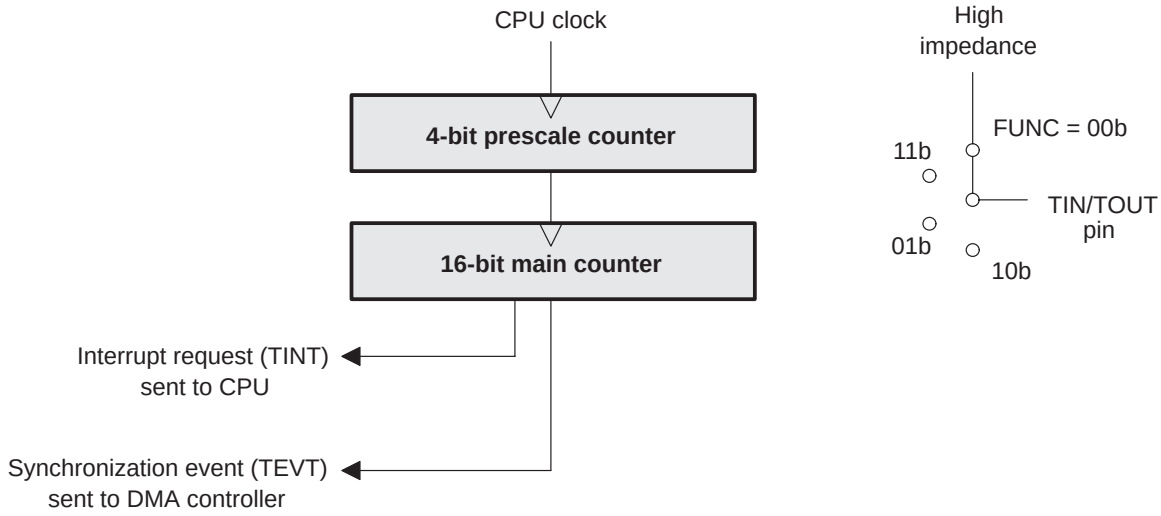
Table 13–1. Configuring the Timer Pin With the FUNC Bits

FUNC Bits	Timer Pin Function	Clock Source
00b	None The pin is in the high impedance state.	Internal (from DSP clock generator)
01b	Timer output The signal on pin changes each time the main counter decrements to 0. The signal polarity is selected by the POLAR bit, and the signal toggles or pulses, depending on the C/P bit. If pulsing is selected, the pulse width is defined by the PWID bits.	Internal (from DSP clock generator)
10b	General-purpose output The signal level on the pin reflects the value in the DATOUT bit.	Internal (from DSP clock generator)
11b	External clock input The pin receives a clock signal from a source outside the DSP.	External

13.2.1 Timer Pin in the High Impedance State

When FUNC = 00b (see Example 13–1), it is independent of the timer, and it neither receives nor drives a signal. The timer input clock must be the CPU clock.

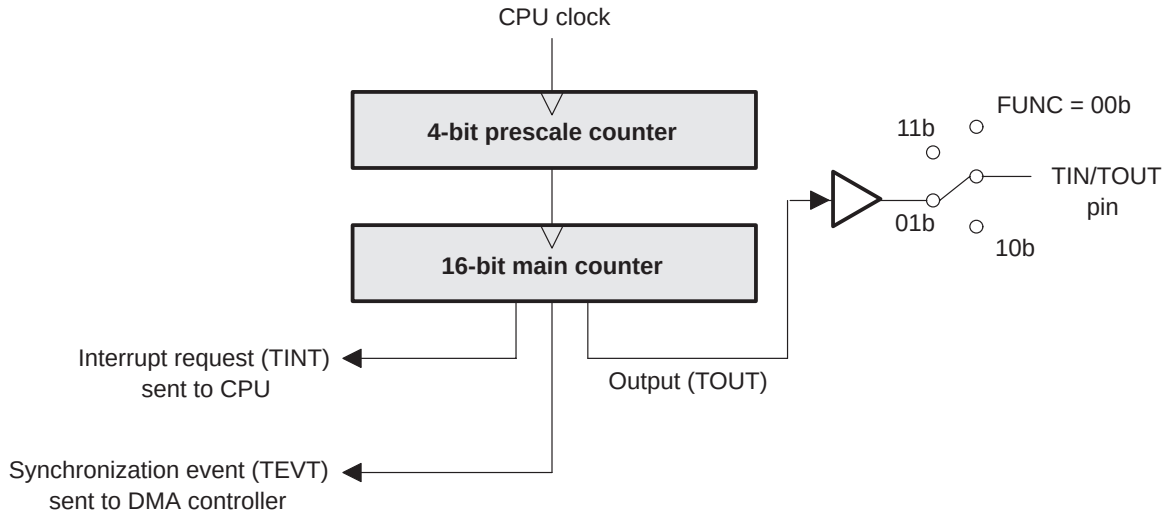
Example 13–1. Timer Pin in High Impedance State



13.2.2 Timer Pin Configured to Reflect the Timer Output (TOUT)

Example 13–2 shows the case when FUNC = 01b. The timer pin is used for the timer output and, therefore, cannot be used for an external clock source. The input clock must be the CPU clock.

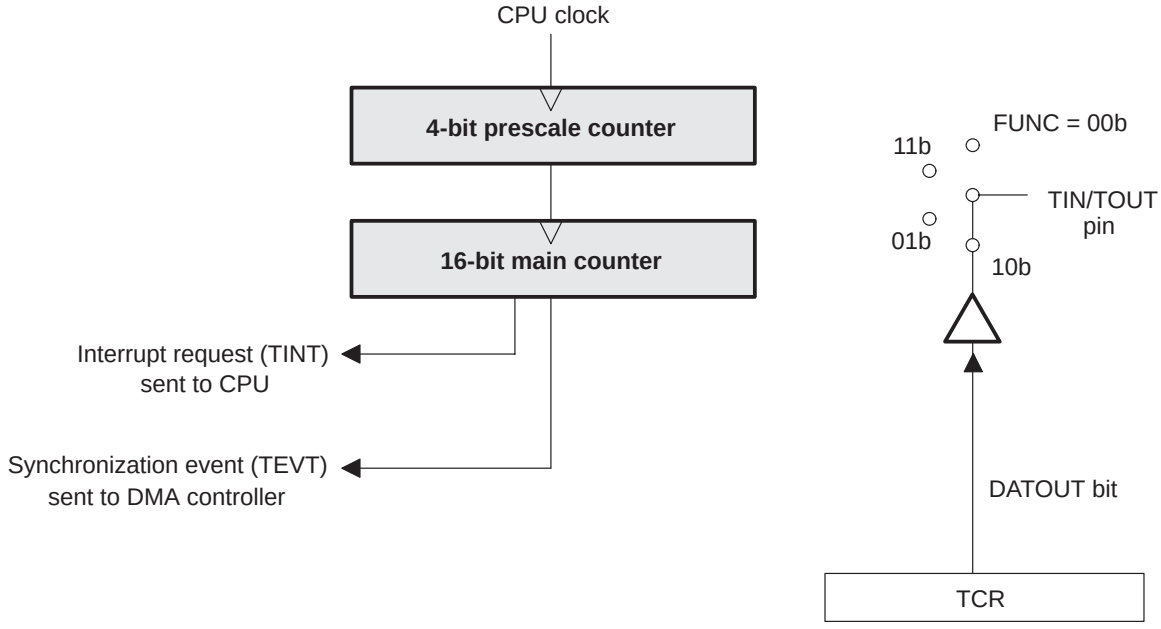
Example 13–2. Timer Pin Configured to Reflect Timer Output (TOUT)



13.2.3 Timer Pin Used as a General-Purpose Output

Example 13–3 shows the case when FUNC = 10b. The timer pin is configured as a general-purpose output; it is independent of the timer. The pin reflects the value of the DATOUT bit. To bring the output signal low, write 0 to DATOUT. To bring the output signal high, write 1 to DATOUT. The timer input clock must be the CPU clock.

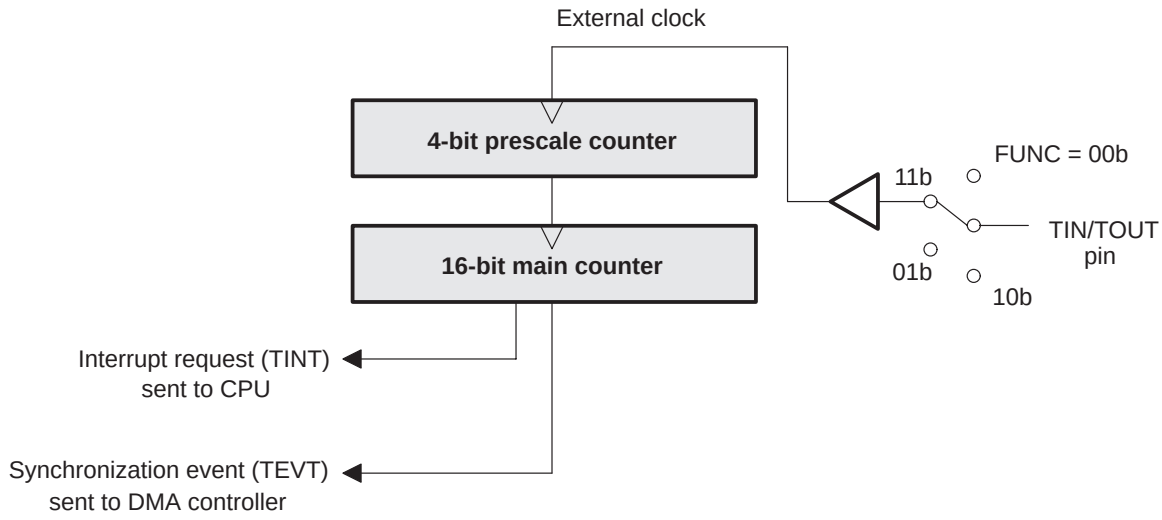
Example 13–3. Timer Pin Used as a General-Purpose Output



13.2.4 Timer Pin Used for an External Input Clock

Example 13–4 shows the case when FUNC = 11b. The timer pin is used by an external clock source and, therefore, cannot be used for the timer output. The only outputs of the timer are the interrupt request and the DMA synchronization event.

Example 13–4. Timer Pin Used for External Input Clock



13.3 Timer Interrupt

Each timer has a timer interrupt signal (TINT). For a given timer, the timer interrupt request is sent to the CPU when the main count register (TIM) counts down to 0. The timer interrupt rate is:

$$\text{TINT rate} = \frac{\text{Input clock rate}}{(\text{TDDR} + 1) \times (\text{PRD} + 1)}$$

TINT automatically sets a flag in one of the interrupt flag registers (IFR0 and IFR1). You can enable or disable the interrupt in one of the interrupt enable registers (IER0 and IER1) and in one of the debug interrupt enable registers (DBIER0 and DBIER1). When you are not using the timer, disable the timer interrupt so that it does not cause unexpected interrupts.

13.4 Initializing a Timer

Use the following procedure to initialize a timer. Example 13–5 illustrates how the procedure can be implemented in the C55x assembly language.

1) Make sure that the timer is stopped (TSS = 1), that timer loading is enabled (TLB = 1), and that the other control bits in TCR are set properly. While TLB = 1, the count registers (TIM and PSC) are loaded from the period registers (PRD and TDDR).

2) Load the desired prescale counter period (in input clock cycles) by writing to TDDR in PRSC.

3) Load the desired main counter period (in input clock cycles) into PRD.

4) Turn off timer loading (TLB = 0) and start the timer (TSS = 0). When the timer starts, TIM holds the value that was loaded into PRD, and PSC holds the value that was loaded into TDDR.

Example 13–5. Initializing Timer 0 on a TMS320VC5510 DSP

```
MOV #9D70h, port(#1002h    ; Load TCR0 such that:
                           ; IDLE_EN = 1 (Allow idle mode)
                           ; FUNC = 01b (Pin reflects timer output)
                           ; TLB = 1 (Load TIM and PSC until TLB = 0)
                           ; FREE = 1 (Timer not stopped by breakpoints)
                           ; PWID = 01b (Pulses on pin last 2 CPU clock cycles)
                           ; ARB = 1 (Reload TIM and PSC when TIM reaches 0)
                           ; TSS = 1 (Stop timer)
                           ; C/P = 0 (Pulse mode selected for pin)
                           ; POLAR = 0 (Signal on pin starts low, pulses high)
                           ; Other bits contain 0s

                           ; Timer to generate output every 196608 clock cycles:
MOV #2h, port(#1003h       ; TDDR = 2 in PRSC0
MOV #FFFFh, port(#1001h)   ; PRD0 = FFFFh = 65535

MOV #9960h, port(#1002h)   ; Modify TCR0 such that:
                           ; TLB = 0 (Stop loading TIM and PSC)
                           ; TSS = 0 (Start timer)
                           ; Other bits are unchanged
```

13.5 Stopping/Starting a Timer

Use the TSS bit of the timer control register (TCR) to stop the timer (TSS = 1) or start the timer (TSS = 0).

13.6 Changing the Timer Pin Function/Clock Source

Table 13–2 explains when and how you can select each of the four pin function/clock source configurations. After the table is a procedure for using an external clock source. All of the bits mentioned in this section are in the timer control register (see page 13-16).

A DSP reset forces FUNC = 00b. You can keep it at 00b or choose any other configuration. If FUNC = 01b or 10b, you can only toggle between these two values until you reset the DSP.

Table 13–2. Selecting and Switching Pin Function/Clock Source Configurations

Current Configuration	**Can Be Directly Changed To …**	**Comments**
FUNC=00b (default) Pin: In high impedance state Clock Source: Internal	FUNC=01b, 10b, or 11b	If you want to switch to an external clock (FUNC = 11b), see the procedure that follows this table.
FUNC = 01b Pin: Timer output Clock Source: Internal	FUNC=10b Pin: General-purpose output Clock Source: Internal	To change to a value other than FUNC = 10b, you must reset the DSP. Otherwise, the change creates an error condition that sets the ERR_TIM error flag in the timer control register (TCR). To recover from the error condition, reset the DSP.
FUNC = 10b Pin: General-purpose output Clock Source: Internal	FUNC = 01b Pin: Timer output Clock Source: Internal	To change to a value other than FUNC = 01b, you must reset the DSP. Otherwise, the change creates an error condition that sets the ERR_TIM error flag in TCR. To recover from the error condition, reset the DSP.
FUNC = 11b Pin: Timer clock input Clock Source: External	No other configuration	To change from FUNC = 11b to another configuration, you must reset the DSP. Otherwise, the change creates an error condition that sets the ERR_TIM error flag in TCR. To recover from the error condition, reset the DSP.

13.6.1 Using an External Clock Source (Changing From FUNC = 00b to FUNC = 11b)

FUNC = 11b is the only configuration that supports an external clock source. The following procedure explains how to use FUNC = 11b.

1) Reset the DSP. (This forces FUNC = 00b.)

2) Write 11b to the FUNC bits.

3) Poll the INT/EXT bit in TCR until it is 1. When INT/EXT = 1, the timer is ready to run using an external clock source from the timer pin. (Reading INT/EXT clears the bit automatically.)

4) Complete the other steps in timer initialization (see page 13-10).

13.7 Reloading the Timer Count Registers

The timer count registers can be reloaded manually or automatically. To reload them manually, write to the I/O addresses of the registers. The following example loads PSC and TIM for timer 1 on the TMS320VC5510 DSP:

```
MOV #0202h, port(#2403h  ; PSC = 2 (and TDDR = 2) in PRSC1
MOV #FFh, port(#2400h)   ; TIM1 = 00FFh
```

The TLB and ARB bits in the timer control register (TCR) provide two different methods to have the registers reloaded automatically. While TLB = 1, PSC is loaded from TDDR, and TIM is loaded from PRD. This method helps with timer initialization (see page 13-10).

If ARB = 1, the count registers are reloaded from the period registers each time TIM reaches 0. This method allows the timer to count continually without input from your program. If ARB = 0, the timer stops counting the next time TIM reaches 0.

13.8 Timer Emulation Mode

You can program the timer to respond in one of three ways to a software breakpoint that occurs during emulation:

- ❏ Stop when the main counter decrements to 0.
- ❏ Stop immediately.
- ❏ Do not stop.

To program a response, use the SOFT and FREE bits of the timer control register (see page 13-16).

13.9 Timers at Reset

A DSP reset forces the following conditions for each timer by resetting the time registers. For details on the registers, see page 13-15.

- ❏ The timer is stopped (TSS = 1).
- ❏ The count for the prescale counter is 0.
- ❏ The count for the main counter is FFFFh.
- ❏ The timer is set to count down once (ARB = 0) rather than repeatedly.
- ❏ The timer cannot be forced into its idle mode by the idle instruction (IDLE_EN = 0).
- ❏ A software breakpoint during emulation will cause the timer to stop immediately (SOFT = FREE = 0).
- ❏ The timer pin is in the high impedance state, and the clock source is internal (FUNC = 00b).

13.10 Timer Registers

For each timer, the DSP contains the registers listed . For the I/O address of each register, see the data sheet for your TMS320C55x DSP.

Table 13–3. Registers of a Timer

Register	Description	For Details, See ...
TIM	Main count register	Section 13.10.1
PRD	Main period register	Section 13.10.1
PRSC	Timer prescale register	Section 13.10.1
TCR	Timer control register	Section 13.10.2 (page 13-16)

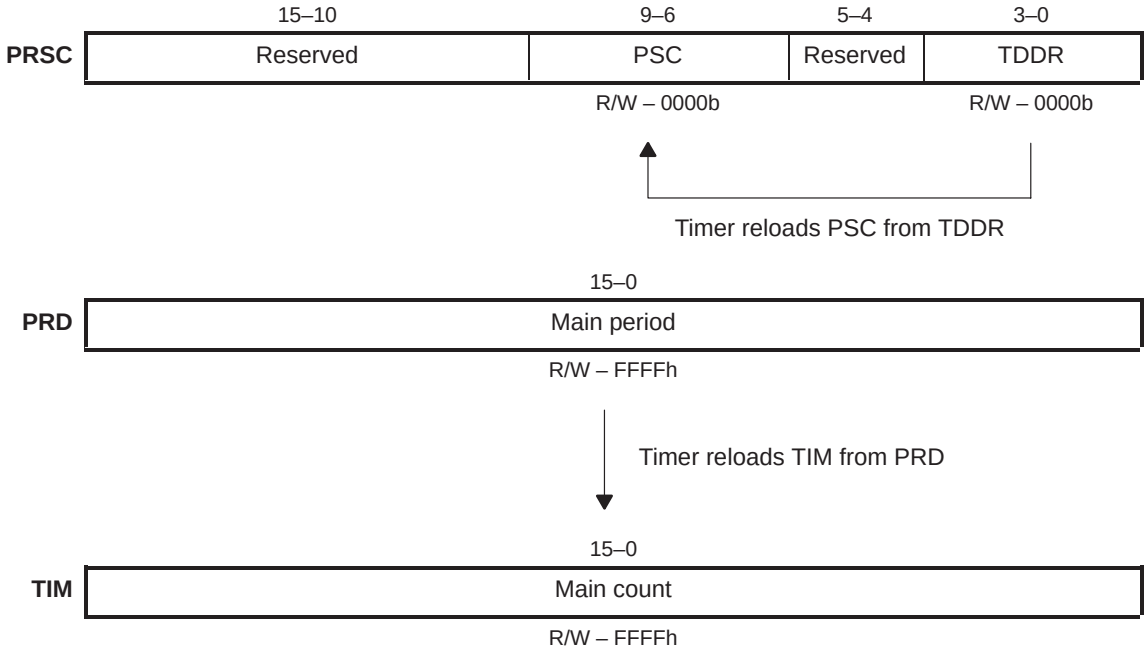
13.10.1 Period and Count Registers (TDDR, PSC, PRD, TIM)

Figure 13–2 and Table 13–4 summarize the period and count registers available for each timer. Each timer has two counters: a 4-bit prescale counter and a 16-bit main counter. Each of the two counters has a count register and a period register. During timer operation, the count registers are decremented. The timer can automatically reload each count register by copying the content of the associated period register.

Table 13–4. Period and Count Registers for Each Timer Counter

Counter	Register	Description
Prescale counter	PSC	Prescale count register. Bits 9–6 of the timer prescale register (PRSC).
	TDDR	Timer divide-down register (prescale period register). Bits 3–0 of PRSC.
Main counter	TIM	Main count register
	PRD	Main period register

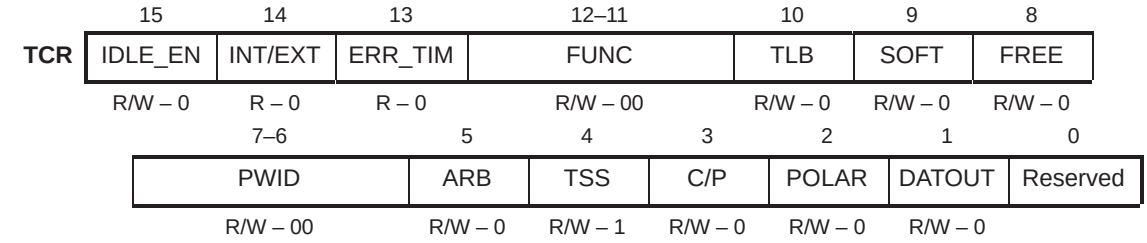
Figure 13–2. Period and Count Registers of a Timer



13.10.2 Timer Control Register (TCR)

Each timer has a timer control register (TCR) of the form shown in Figure 13–3. describes the bits of TCR. Using specific bits in TCR, you can configure, start, stop, load, and reload the associated timer. Other bits in the TCR control the functionality of the associated timer output pin.

Figure 13–3. Timer Control Register (TCR)



Legend:

R Read only access
R/W Read/write access
– X X is the value after a DSP reset.

Table 13–5. TCR Bit Descriptions

Bit(s)	Name	Description	Reset Value
15	IDLE_EN	Idle enable bit for the timer. If the PERIPH idle domain is configured to be idle and IDLE_EN = 1, the timer stops and enters a low-power (idle) state. 0 The timer cannot be placed in an idle state. 1 If the PERIPH domain is idle (PERIS = 1 in the idle status register), the timer is stopped in a low-power state. The idle domains are described beginning on page 8-2. The idle registers are described in section 8.7 (page 8-9).	0
14	INT/EXT	Internal-to-external clock change indicator. When changing the timer clock source from internal to external, a program can check this bit to determine when the timer is adjusted and is ready to use an external clock source. 0 Timer not ready to use an external clock source 1 Timer ready to use an external clock source	0
13	ERR_TIM	Timer-pin error flag. Some changes to the FUNC bits create an error condition that is reflected in ERR_TIM. When TIM_ERR = 1, reset the DSP and reinitialize the timer. 0 No error detected or TIM_ERR has been read 1 Error detected in a write to the FUNC bits. One of the following changes was attempted: Change from FUNC = 01b to FUNC = 00b or 11b Change from FUNC = 10b to FUNC = 00b or 11b Change from FUNC = 11b to any other value	0

Table 13–5. TCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
12–11	FUNC	Function bits for the timer pin. The two FUNC bits define the function of the timer pin and determine the required clock source for the timer. If the internal clock source is chosen, the CPU clock drives the counting of the timer. If an external clock source is chosen, the timer is driven by a clock signal coming in on the timer pin. 00b **Pin Function:** None. The pin is in the high impedance state. **Clock Source:** Internal (from DSP clock generator) 01b **Pin Function:** Timer output. The signal on pin changes each time the main counter decrements to 0. The signal polarity is selected by the POLAR bit. The signal toggles or pulses, depending on the C/P bit. If pulsing is selected, the pulse width is defined by the PWID bits. **Clock Source:** Internal (from DSP clock generator) 10b **Pin Function:** General-purpose output. The signal level on the pin reflects the value in the DATOUT bit. **Clock Source:** Internal (from DSP clock generator) 11b **Pin Function:** External clock input. The pin receives a clock signal from a source outside the DSP. **Clock Source:** External	00b
10	TLB	Timer load bit. While TLB = 1, the count registers are loaded from the period registers. 0 TLB has been cleared. 1 Until TLB = 0, load TIM from PRD, and load PSC from TDDR.	0

Table 13–5. TCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
9 8	SOFT FREE	Timer emulation mode bits. SOFT and FREE determine the response of the timer when a breakpoint is encountered in the high-level language debugger. If FREE = 1, the timer does not stop in response to a software breakpoint, regardless of the value of SOFT. If FREE = 0, SOFT determines how the timer responds to a breakpoint. **SOFT FREE When A Breakpoint Is Encountered ...** 0 0 The timer stops immediately. (Reset condition) 0 1 The timer continues to run. 1 0 The timer stops after the main count register (TIM) decrements to 0. 1 1 The timer continues to run.	00b
7–6	PWID	Timer-output pulse width bit. PWID determines the width of each pulse on the timer pin under the following conditions: ❑ The timer pin is configured to show the timer output (FUNC = 01b). ❑ The pulse mode is selected (C/P = 0). The pulse width is defined in CPU clock periods. 00b 1 CPU clock period 01b 2 CPU clock periods 10b 4 CPU clock periods 11b 8 CPU clock periods	00b
5	ARB	Auto-reload bit. When ARB = 1, the count registers are automatically reloaded from the period registers whenever the main count register (TIM) is decremented past 0. 0 ARB has been cleared. 1 Each time TIM reaches 0, reload TIM from PRD, and reload PSC from TDDR.	0
4	TSS	Timer stop status bit. Use TSS to stop the timer, start the timer, or determine whether the timer is stopped. 0 Start the timer./The timer is running. 1 Stop the timer./The timer is stopped. (Reset condition)	1

Table 13–5. TCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
3	C/P	Timer clock mode/pulse mode bit. When the timer pin is configured to show the timer output (FUNC = 01b), C/P determines whether the signal on the pin is pulsed or toggled. 0 Pulse mode. A pulse is driven on the timer pin each time the main count register (TIM) reaches 0. The width of the pulse is defined by the PWID bits. The polarity of the pulse is defined by the POLAR bit. 1 Clock mode. The signal on the timer pin has a 50% duty cycle. The signal toggles (from high to low or from low to high) each time TIM reaches 0.	0
2	POLAR	Timer-output polarity bit. When the timer pin is configured to show the timer output (FUNC = 01b), POLAR determines the polarity of the signal on the pin. The specific effect of this bit depends on whether pulse mode (C/P = 0) or clock mode (C/P = 1) is selected. 0 The signal on the timer pin starts low. Then ... Pulse mode: Each time the main count register (TIM) reaches 0, a high pulse is driven on the timer pin. The width of the pulse is defined by the PWID bits. Between pulses the signal is low. Clock mode: The first time TIM reaches 0, the signal on the timer pin toggles high. During subsequent countdowns: If the signal is high, it toggles low; if the signal is low, it toggles high. 1 The signal on the timer pin starts high. Then ... Pulse mode: Each time TIM reaches 0, a low pulse is driven on the timer pin. The width of the pulse is defined by the PWID bits. Between pulses, the signal is high. Clock mode: The first time TIM reaches 0, the signal on the timer pin toggles low. During subsequent countdowns: If the signal is low, it is toggled high; if the signal is high, it is toggled low.	0

Table 13–5. TCR Bit Descriptions (Continued)

Bit(s)	Name	Description	Reset Value
1	DATOUT	Data output bit. When the timer pin is configured as a general-purpose output pin (FUNC = 10b), use DATOUT to control the signal level on the pin. 0 Drive the signal on the timer pin low. 1 Drive the signal on the timer pin high.	0
0	Reserved	This bit is not available for your use.	–

Chapter 14

USB Module

If your TMS320C55x™ DSP contains a USB module, you can use the C55x™ DSP to create a full speed USB slave device that is compliant with Universal Serial Bus Specification Version 1.1. This chapter explains the architecture of the module and how to program the module.

Topic **Page**

14.1 USB Concepts Overview

This section explains USB concepts and terminology used in this chapter.

14.1.1 Terminology

In a USB system, the host is the master. The host initiates all data transfers between itself and attached USB devices. Therefore, the direction of a data transfer is described relative to the host:

OUT transfer — A transfer of data from the host to a device:

Host → Device

IN transfer — A transfer of data from a device to the host:

Host ← Device

Each IN or OUT transfer can be one of the following types. The types of transfers on a USB are:

Control transfer — A data transfer that is used by the USB host to send commands to a USB device, including commands to enumerate the device when it is first attached. Control transfers include error checking.

Bulk transfer — A data transfer that is used by the host for large amounts of data that are not time-critical. Can use when transfer time is not critical. The host only allocates bus time for bulk transfers when the time is not need by transfers of the other types. Bulk transfers include error checking. A device such as a printer is a good application for this type of transfer.

Interrupt transfer — The data transfer used when a USB device must send or receive moderate amounts of data periodically with minimum latency. Interrupt transfers include error checking. Typical devices that use this type of transfer are keyboards and joysticks.

Isochronous transfer — A data transfer available to support USB devices that need to send or receive data in real time at a constant rate. Isochronous transfers can handle more data than interrupt transfers, but no error checking is performed. A device such as a digital speaker is a typical application for isochronous transfers.

Note:

From an implementation standpoint, bulk and interrupt transfers are treated the same way in the C55x USB module. The only difference is that, in the case of an interrupt transfer, the transfer is initiated periodically by the host, whereas a bulk transfer is initiated by the host whenever the bus is not used for other transfers.

For data transfer between a USB host and a USB device, the data passes through an endpoint in the device:

Endpoint — A designated storage location within a USB device. Each endpoint in a device is uniquely identified by its number and its direction (IN or OUT).

OUT Endpoint — An endpoint that holds data received from the USB host. To use data from the host, the USB device must read the data from an OUT endpoint.

Each device must have an OUT endpoint 0 to be used for control transfers.

IN Endpoint — An endpoint that holds data to be sent to the USB host. To send data to the host, the USB device must write to an IN endpoint.

Each device must have an IN endpoint 0 to be used for control transfers.

The overall characteristics of the USB device and the type of each endpoint must be reported to the host when the device is attached to the bus for the first time. This process is called enumeration.

The USB bandwidth is shared by multiple USB devices. Data is transferred on the bus at regular (1-millisecond) intervals. Each of these intervals is called a **frame**, and the host divides up the frame for all the devices on the bus. As each new USB device is recognized and successfully configured by the host, it gains a portion of the frame. The size of the portion depends on factors such as the type of transfer (for example, isochronous versus bulk) and the amount of bandwidth that is not being used by other devices that are already on the bus.

14.1.2 Data Toggle Mechanism

For non-isochronous transfers, the USB uses a data toggle mechanism to detect transmission errors, to ensure that the transmitter and the receiver of USB data are synchronizing throughout a transfer. The data toggle mechanism requires two data packet types (DATA0 and DATA1) and two toggle bits

(one in the transmitter and one in the receiver). Each packet transmitted is a DATA0 packet or a DATA1 packet, depending on the value of the transmitter's toggle bit (0 = DATA0; 1 = DATA1). If the receiver is synchronized, its toggle bit matches that of the transmitter, and the receiver expects the data type that was transmitted. Once the packet is successfully received, the receiver complements its toggle bit and sends an acknowledgement to the transmitter. When the acknowledgement arrives at the transmitter, the transmitter complements its toggle bit.

A USB transfer may be comprised of a number of transactions, but the first packet of the first transaction is a DATA0 packet. Subsequent packets alternate in type (DATA1, DATA0, DATA1, and so on).

14.2 Introduction to the USB Module

The USB module described in this section is a USB 1.1-compliant, full speed slave device.

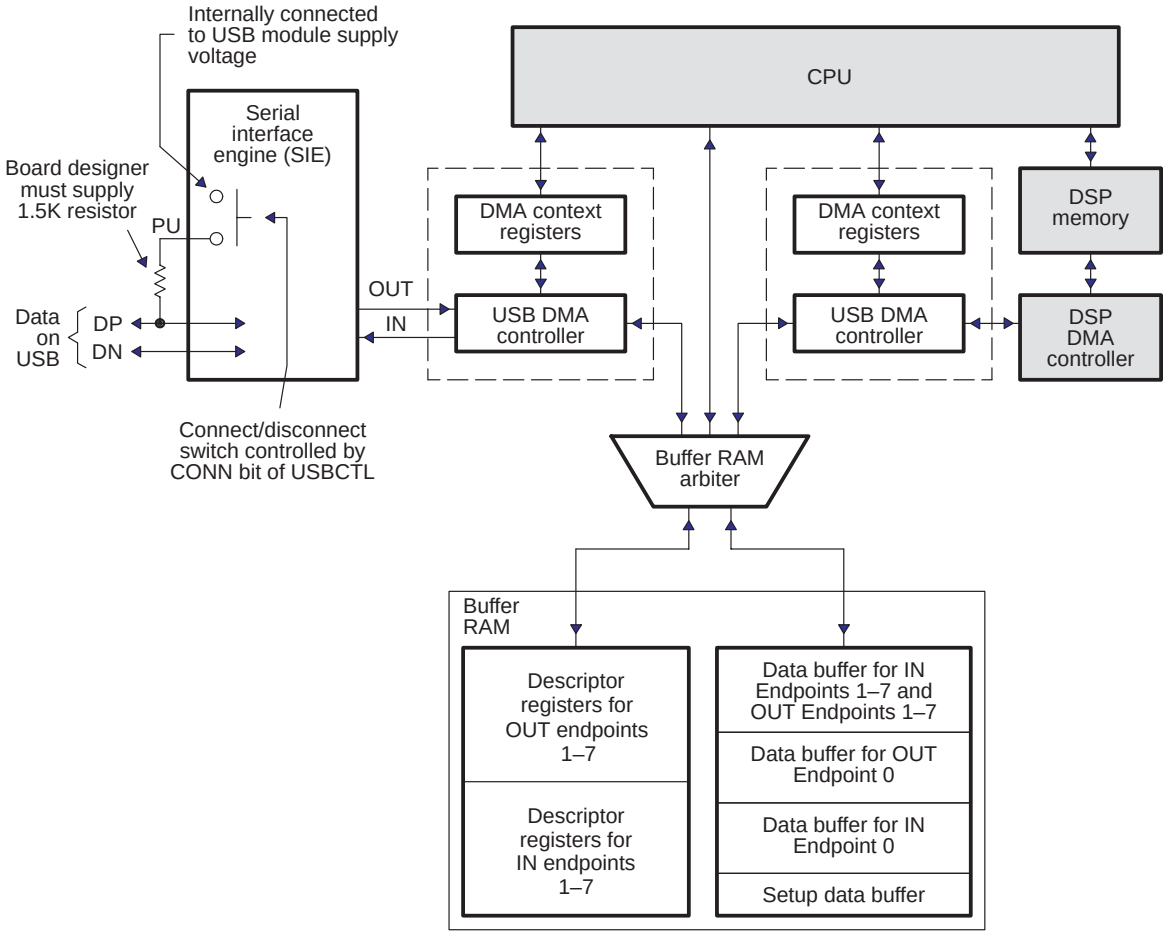
The C55x USB module has 16 endpoints:

- Two control endpoints (for control transfers only): OUT endpoint 0 and IN endpoint 0.
- Fourteen general-purpose endpoints (for other types of transfers): OUT endpoints 1–7 and IN endpoints 1–7. Each of these endpoints has:
  - Support for bulk, interrupt, and isochronous transfers.
  - An optional double-buffer scheme for fast data throughput.
  - A dedicated DMA channel. A DMA controller inside the USB module can pass data between the general-purpose endpoints and the DSP memory while the CPU performs other tasks. (This DMA controller does not access the control endpoints.)

14.2.1 Block Diagram of the USB Module

Figure 14–1 contains a conceptual block diagram of the USB module. The shaded blocks in the figure are outside the USB module. Following the figure is a list that describes each of the main components of the module.

Figure 14–1. Conceptual Block Diagram of the USB Module



❑ **Interface pins:**

Pin	Description
DP	Connect this pin to the line of the USB connector that carries the positive differential data.
DN	Connect this pin to the line of the USB connector that carries the negative differential data.
PU	Use this pin to connect a 1.5 Kohm pullup resistor to the DP line. A software-controlled switch can connect the pullup resistor to an internal 3.6-V source. When the CPU sets the connect bit of USBCTL (CONN = 1), the switch closes and completes the pullup circuit, causing the USB host to detect the USB module as a new device on the bus, and to start the enumeration process. To disconnect the device from the USB system, clear the CONN bit. The switch will open and disconnect the pullup resistor.

❑ **Serial interface engine (SIE).** The SIE is the USB protocol handler. It parses the USB bit stream for data packets that are meant for the USB device. For an OUT transfer, the SIE converts the serial data to parallel data and passes them to the USB buffer manager. For an IN transfer, the SIE converts parallel data from the UBM to serial data, and transmits them over the USB.

The SIE also handles error-checking tasks. For an OUT transfer, the SIE does the error checking and transfers only the good data to the UBM. For an IN transfer, the SIE generates the necessary error-checking information before sending the data on the bus.

❑ **USB buffer manager (UBM) and the control and status registers.** The UBM controls data flow between the SIE and the buffer RAM. Most of the control registers are used to control the behavior of the UBM, and most of the status registers are modified by the UBM, to notify the CPU when any events occur.

❑ **Buffer RAM.** The buffer RAM contains registers that are mapped in the DSP I/O space. In the RAM are:

- Relocatable buffer space for each of the general-purpose endpoints (3.5K bytes). A general-purpose endpoint can have one data buffer (X buffer) or two data buffers (X buffer and Y buffer).
- A fixed data buffer for OUT endpoint 0 (64 bytes)
- A fixed data buffer for IN endpoint 0 (64 bytes)

- A fixed data buffer for a setup packet (8 bytes)
- Descriptor registers. For each of the general-purpose endpoints, there are eight registers that determine the endpoint characteristics.

❑ **USB DMA controller and its context registers.** This DMA controller can transfer data between the DSP memory and the X and Y buffers of the general-purpose endpoints. Each of these endpoints has a dedicated DMA channel and a dedicated set of DMA context registers for controlling and monitoring activity in that channel. The CPU can read from or write to each of these context registers by accessing the appropriate 16-bit address in I/O space.

The USB DMA controller accesses memory via the auxiliary port of the DSP DMA controller that is described in Chapter 3. This auxiliary port is shared by the USB DMA controller and the enhanced host port interface (EHPI), but the USB DMA controller is given the higher priority.

A state machine in the USB DMA controller handles data transfers in the host-DMA mode (see section 14.5 on page 14-40).

❑ **Buffer RAM arbiter.** The 8-bit-wide buffer RAM can be accessed by the UBM, by the USB DMA controller, and by the DSP CPU. The buffer RAM arbiter provides a fair access scheme to arbitrate accesses from these three requesters.

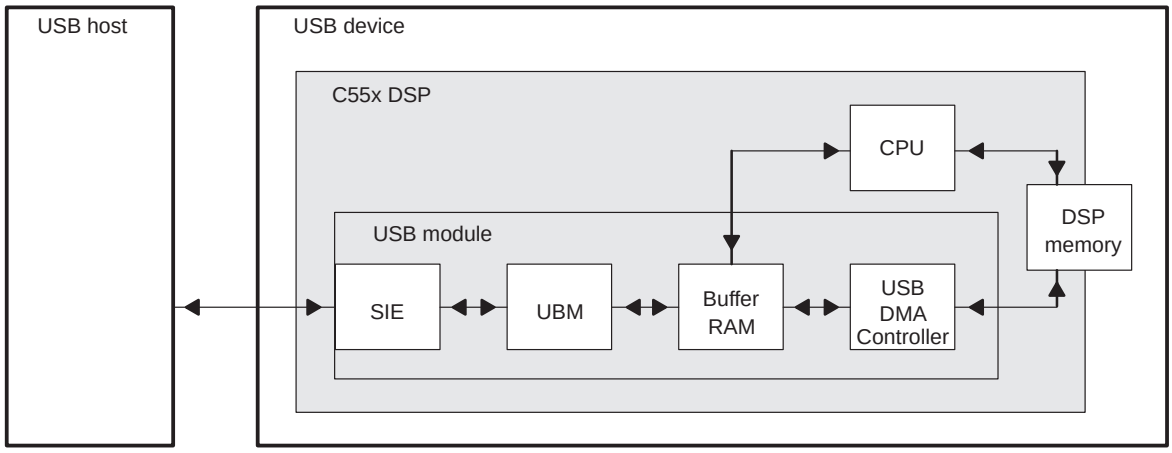
The USB DMA controller only accesses the X and Y buffers of the general-purpose endpoints. The controller uses 24-bit byte addresses to access DSP memory.

The CPU can access the buffer RAM, including the descriptor registers, via I/O space. The CPU writes 16-bit values to I/O space. However, when the CPU writes to the RAM, the high eight bits are ignored, and when the CPU reads from the RAM, the high eight bits are 0s.

14.2.2 Transferring Data Between the USB Host and the DSP Memory

Figure 14–2 shows, at a high-level, how data travels between the USB host and the DSP memory when a C55x DSP handles the USB activity for a USB device. During IN transfers, the SIE (serial interface engine) converts the parallel data from the UBM into a serial data stream for the host. During OUT transfers, the SIE converts the host's serial data into a parallel format for the UBM. The UBM either moves data from the SIE to the buffer RAM or from the buffer RAM to the SIE. Before the UBM transfers data to the SIE, the CPU or the USB DMA controller must put the data into the buffer RAM. When the CPU or the DMA controller is ready to move data to the DSP memory, it must wait for the UBM to move the data from the SIE to the buffer RAM.

Figure 14–2. Path for Data Transferred Between the Host and the DSP Memory



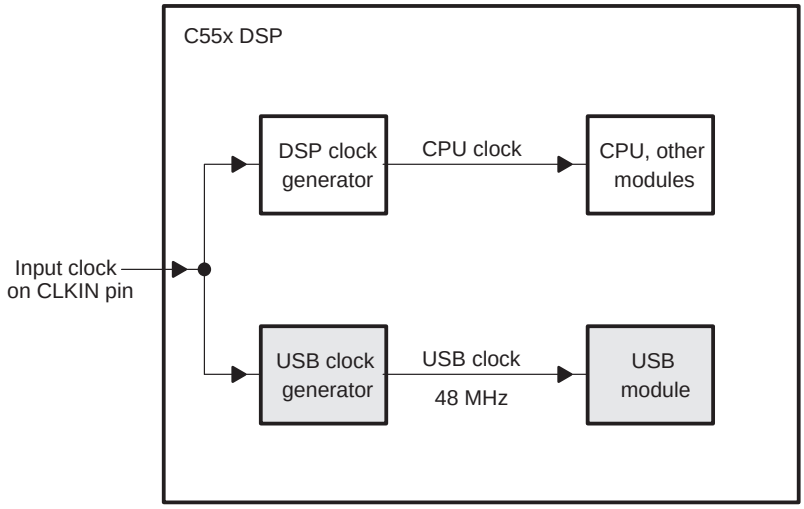
14.2.3 Clock Generation for the USB Module

As shown in Figure 14–3, the USB module has a dedicated clock generator that is independent of the DSP clock generator. Both generators receive their input from the CLKIN pin. The DSP clock generator supplies the CPU clock that is used by the CPU and most of the other modules inside the DSP. The USB clock generator supplies the clock needed to operate the USB module. Because the generators are independent, if an IDLE instruction turns off the DSP clock generator, the USB module can keep running.

Note:

The USB module requires a 48-MHz clock. The clock on the CLKIN pin may vary, but you must program the USB clock generator to produce a 48-MHz clock.

Figure 14–3. Clock Generation for the USB Module



To understand the USB clock generator, see the description of the DSP clock generator in Chapter 2. The two clock generators are nearly identical in function. The USB clock mode register, USBCLKMD (shown in Figure 14–4), has the same bit fields of the DSP clock mode register (CLKMD), which is described in section 2.8 on page 2-12. However, the reset value of the BYPASS DV bits in USBCLKMD is *not* determined by a pin. These bits are always reset to 01b, providing a USB clock that is half the speed of the input clock. For the I/O address of USBCLKMD, check the data sheet for your C55x DSP.

Figure 14–4. USB Clock Mode Register (USBCLKMD)

15	14	13	12	11–7
Rsvd	IAI	IOB	TEST (keep 0)	PLL MULT
	R/W – 0	R/W – 1		R/W – 00000

6–5	4	3–2	1	0
PLL DIV	PLL ENABLE	BYPASS DIV	BREAKLN	LOCK
R/W – 00	R/W – 0	R/W – 01	R – 1	R – 0

Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset.

14.3 USB Buffer Manager (UBM)

When data is to be moved to or from the buffer RAM, the UBM accesses one of the following buffers in the buffer RAM:

Buffers For Transfers To DSP Memory	Buffers Transfers From DSP Memory
Control endpoint buffers	
OUT endpoint 0 buffer	IN endpoint 0 buffer
General-purpose endpoint buffers	
OUT endpoint 1 buffer (X or Y)	IN endpoint 1 buffer (X or Y)
OUT endpoint 2 buffer (X or Y)	IN endpoint 2 buffer (X or Y)
OUT endpoint 3 buffer (X or Y)	IN endpoint 3 buffer (X or Y)
OUT endpoint 4 buffer (X or Y)	IN endpoint 4 buffer (X or Y)
OUT endpoint 5 buffer (X or Y)	IN endpoint 5 buffer (X or Y)
OUT endpoint 6 buffer (X or Y)	IN endpoint 6 buffer (X or Y)
OUT endpoint 7 buffer (X or Y)	IN endpoint 7 buffer (X or Y)

Each of the general-purpose endpoints can be configured to have a single buffer (X) or a double buffer (two buffers, X and Y). This is controlled by the double buffer mode (DBUF) bit in USBxCNFn. If there are two buffers, the UBM keeps track of which buffer to use. If the endpoint is in the non-isochronous mode, the UBM uses the X buffer for a DATA0 packet and the Y buffer for a DATA1 packet.

Each of the endpoint buffers is associated with a programmable count register of the following format:

7	6–0
NAK	CT (bytes)

The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. While the NAK bit is set (NAK = 1), the UBM sends a NAK in response to a host request at that particular endpoint. The UBM does not access the buffer until NAK is cleared (NAK = 0). The role of the NAK bit is summarize in the flow chart of Figure 14–5.

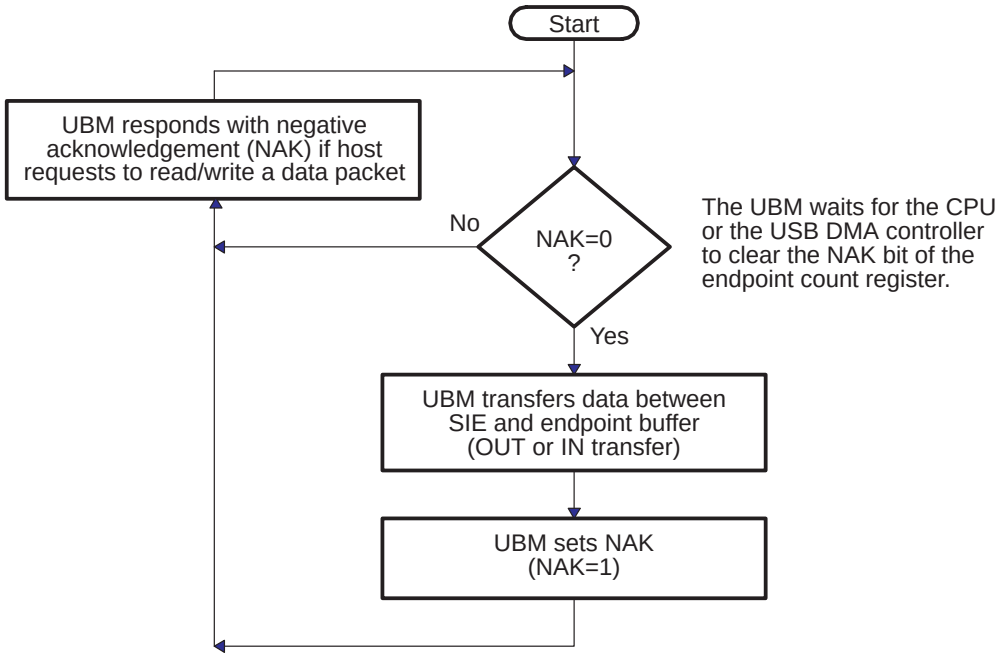
When an OUT packet is received from the serial interface engine (SIE), the UBM writes the incoming data to the appropriate endpoint buffer. Then the UBM sets the NAK bit to prevent the host from overwriting the packet before it is read by the CPU or the USB DMA controller. When an IN token is received, the UBM transfers data from the buffer to the SIE. Then the UBM sets the NAK bit so that the host will not receive the same packet multiple times before a new packet is loaded into the buffer. When NAK is cleared by the CPU or by the USB DMA controller, the UBM can access the buffer again.

The CT (count) field indicates the number of bytes in a transfer between the SIE and an endpoint buffer. For an IN transfer, you must initialize the CT field to tell the UBM how many bytes to read from the buffer. In addition, you must clear NAK when you want the UBM to start. In the case of an OUT transfer, the UBM updates the CT field after moving a new data packet from the SIE to the endpoint buffer.

Note:

In isochronous transfers, the count can be as large as 1023 bytes, requiring a 10-bit CT field. Thus, for isochronous transfers, the count value is extended by three high bits from another register (see section 14.8.3.4 on page 14-75).

Figure 14–5. Role of a NAK bit in UBM Activity (at an Individual Endpoint)



14.4 USB DMA Controller

The USB module contains a dedicated DMA controller that can transfer data between the DSP memory and the data buffers of the general-purpose endpoints (OUT endpoints 1–7 and IN endpoints 1–7). This DMA controller cannot access the control endpoints (OUT endpoint 0 and IN endpoint 0).

Note:

The information in this section assumes that you have *not* enabled the host-DMA mode. For details about this mode, see section 14.5 on page 14-40.

14.4.1 Advantage of Using the USB DMA Controller

The USB DMA controller transfers data between the endpoint buffers and the DSP memory with minimal CPU involvement. The CPU tells the USB DMA controller to begin a data transfer; then it can continue with other tasks while the controller moves the data. The controller notifies the CPU of the transfer status via GO and RLD status flags and interrupts (see section 14.6.3 on page 14-50).

14.4.2 Things To Consider

Keep the following facts in mind when you use the USB DMA controller:

- Each of the general-purpose endpoints must be in double-buffer mode (DBUF = 1 in USBOCNF1–USBOCNF7 and in USBICNF1–USBICNF7). The USB DMA controller assumes there an X buffer and a Y buffer for each general-purpose endpoint, and it accesses the buffers alternately, beginning with the X buffer.

- The USB DMA controller accesses the DSP memory via the auxiliary port of the DSP DMA controller that is described in Chapter 3. This auxiliary port is also used by the enhanced host port interface (EHPI). The USB module has the higher priority and thus can delay EHPI memory accesses.

- The DSP DMA controller must share the external memory interface (EMIF) with other parts of the DSP. The EMIF handles requests from throughout the DSP according to a preset priority ranking. When the USB DMA controller must access external memory, the DSP DMA controller sends a request to the EMIF and waits to be serviced. For a table of EMIF request types and their priorities, see section 5.3 on page 5-8.

If the USB DMA controller is not used:

- ❏ Make sure the software does not write to the DMA control register (USBODCTLn or USBIDCTLn) of any endpoint. Writing 1 to the GO bit of any DMA control register initiates a DMA transfer in the controller. In addition, if the controller finishes a DMA transfer and finds that the RLD bit is 1, the controller will perform another transfer.
- ❏ Do not enable RLD and GO interrupt requests in the registers USBODIE and USBIDIE.

14.4.3 Interaction Between the CPU and the USB DMA Controller

Table 14–1 shows how the CPU and the USB DMA controller interact. Each action of the CPU, of course, depends on the instructions in your code. Figure 14–6 (a) (page 14-16) shows how DMA activity is affected by the GO and RLD bits set by the CPU and how it is affected by the NAK bit in an endpoint buffer count register. Figure 14–6 (b) (page 14-17) shows how the CPU (via your code) can handle GO and RLD reports from the controller.

After a DMA transfer, the GO bit of USBxDCTLn is cleared if the RLD bit of USBxDCTLn is 0. If RLD is 1, and neither OVF nor STP is set in USBxDCTLn, the controller performs a DMA reload operation (see section 14.4.5 on page 14-18): The contents of the primary address and size registers are swapped with the contents of the reload address and size registers. After a reload operation, the controller automatically starts a new DMA transfer.

Table 14–1. CPU-Initiated DMA Transfers

Action of the CPU (Executing Your Code)	Action of the USB DMA Controller
Initialize the DMA context registers. Each general-purpose endpoint has eight DMA context registers (see section 14.4.7).	Behave according to the contents of the DMA context registers.
Issue a go command (set the GO bit). The GO bit is in the DMA control register for the endpoint (USBODCTLn or USBIDCTLn). Before initiating a new transfer, poll the GO bit to make sure the previous transfer (or series of transfers) is complete (GO = 0).	Respond to a go command. When the CPU sets the GO bit, the controller begins polling the NAK bit in the X-/Y-buffer count register. When NAK = 1, the controller begins the DMA transfer, unless the endpoint is in the isochronous mode (ISO = 1) . When ISO = 1, the controller also waits for a start-of-frame packet (SOF) on the bus.
Set or clear the RLD (reload) bit as desired. The RLD bit is in the DMA control register for the endpoint. Set RLD if you want to tell the controller to begin another transfer after the current transfer is complete. Make sure you initialize the reload address and size registers first.	Behave according to the value of RLD. Once a DMA transfer is complete, the controller checks the RLD bit. If RLD = 0, the controller stops, clears GO, and waits for the CPU to set GO again. If RLD = 1, the controller performs a DMA reload operation, clears RLD, and begins another transfer (if NAK = 1).
Issue a stop command (optional). To stop the controller before it would normally stop itself, set the STP bit (STP = 1) in the DMA control register for the endpoint.	Respond to a stop command. The controller normally stops when it has completed a transfer and the RLD bit is 0. However, if the CPU sets the STP bit for the endpoint, the controller stops its activity on the next packet boundary or at the end of the current DMA transfer, whichever happens first. As it stops, the controller clears the STP and GO bits.
Enable/disable interrupts, and respond to interrupts. Using the GO and RLD interrupt enable registers, individually enable/disable GO and RLD interrupts. If an interrupt is enabled, it is passed to the CPU as a USB interrupt. The interrupt service routine (ISR) can read USBINTSRC (see page 14-85) to determine the interrupt source. Then the ISR can execute the appropriate subroutine.	Generate interrupts. When the controller completes a transfer and RLD = 0, the controller clears the GO bit and sets the GO interrupt flag. The RLD interrupt flag is set when the controller completes a reload operation and clears the RLD bit. When an interrupt flag is set, the corresponding interrupt (if enabled) is sent to the CPU. For information about the GO and RLD flags and interrupts, see section 14.6.3 (page 14-50).
Read status information. To monitor the activity of the controller, read the status bits in the DMA control register and the flag bits in the interrupt flag registers.	Record status information for the CPU. The controller modifies bits in the DMA control register and in the interrupt flag registers to notify the CPU of specific actions or errors.

Figure 14–6. Activity for CPU-Initiated DMA Transfers

(a) USB DMA Controller Flow (at an Individual Endpoint)

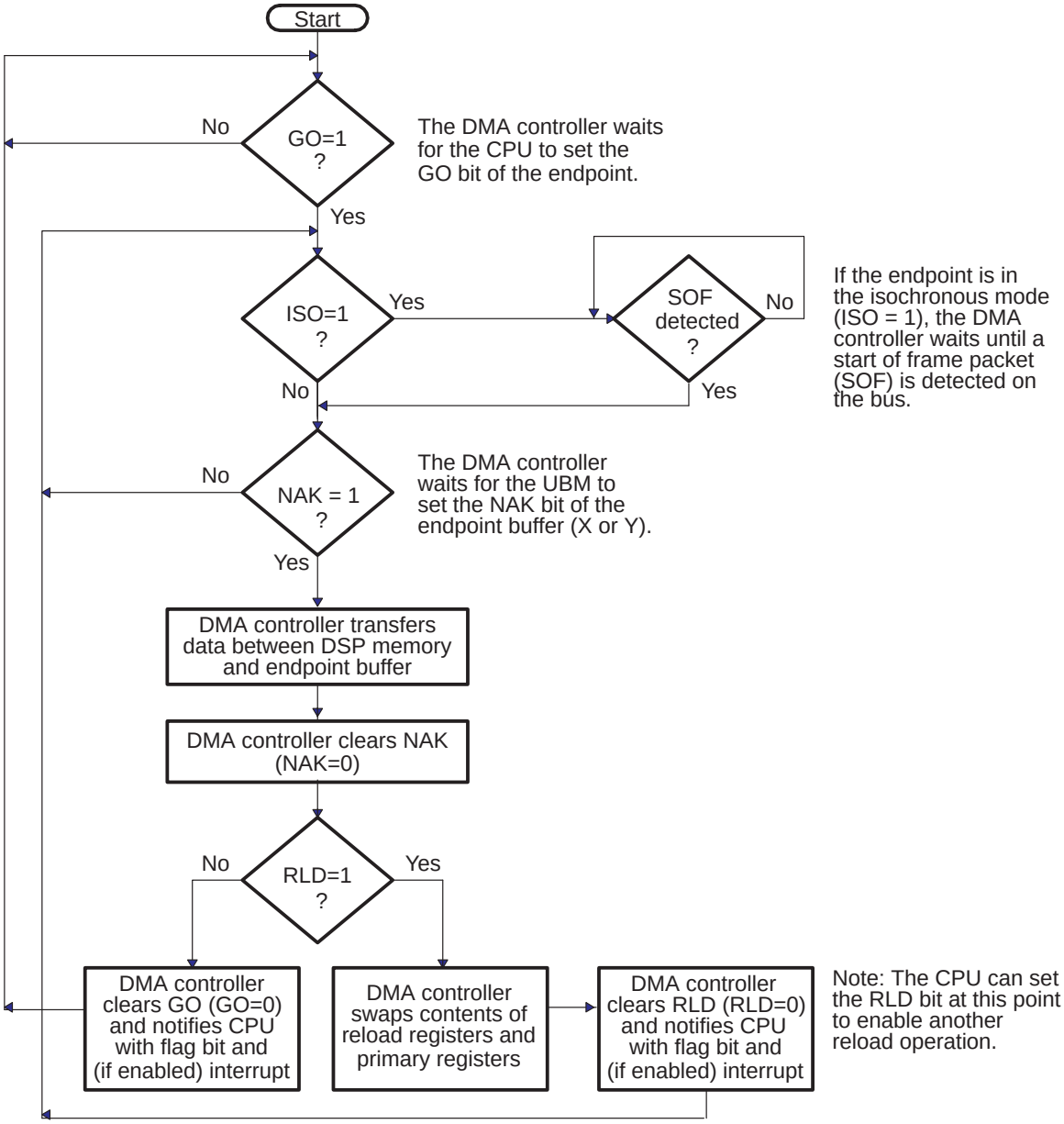
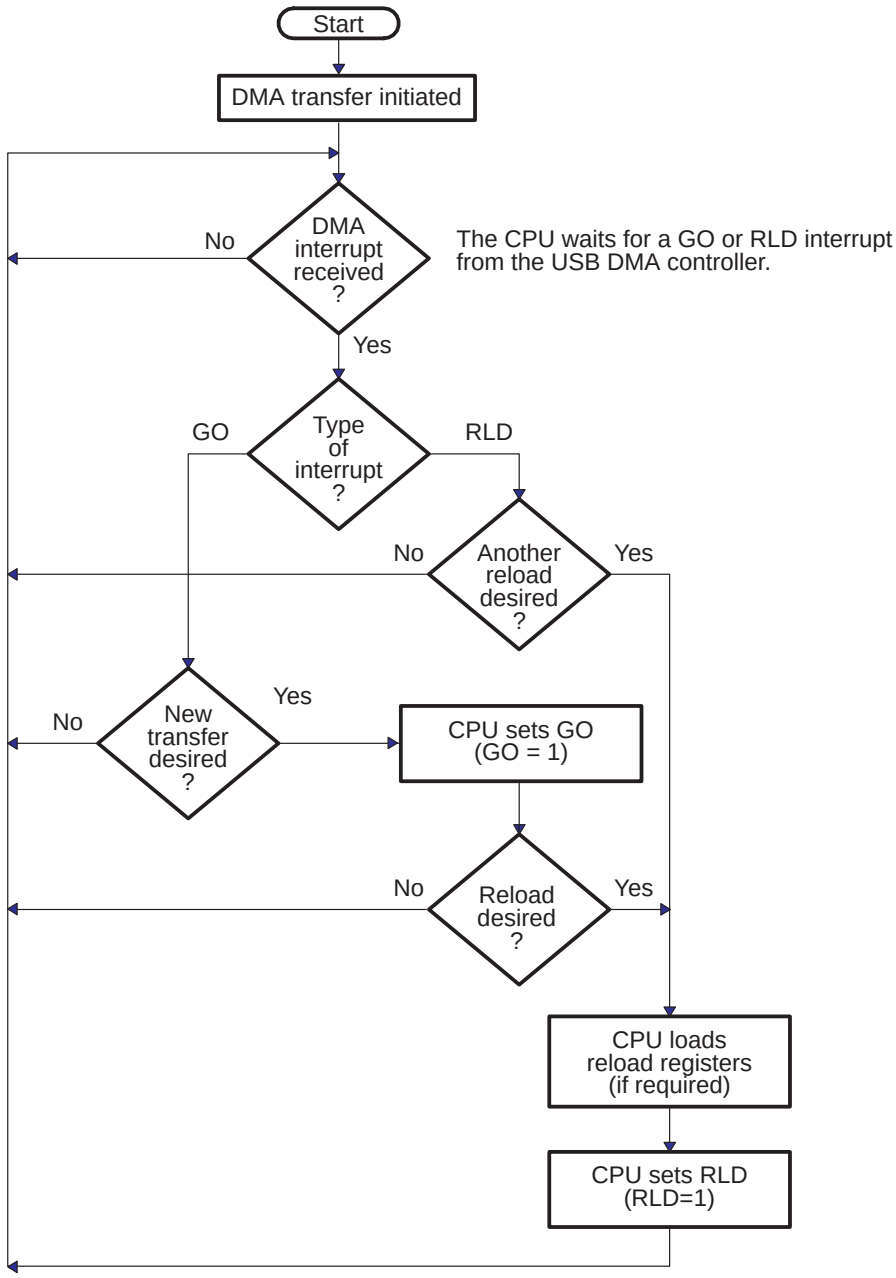


Figure 14–6. Activity for CPU-Initiated DMA Transfers (Continued)

(b) CPU Flow (at an Individual Endpoint)



14.4.4 Automatic Alternating Accesses of the X and Y Buffers

For non-isochronous USB transfers, the USB DMA controller automatically tracks the data packet type and determines which of the endpoint's buffers to access, X or Y:

Data Packet Type	USB DMA Controller Accesses ...
DATA0	X buffer OUT transfer: The controller reads data from the X buffer. IN transfer: The controller writes data to the X buffer.
DATA1	Y buffer OUT transfer: The controller reads data from the Y buffer. IN transfer: The controller writes data to the Y buffer.

For isochronous USB transfers, the USB DMA controller uses the X buffer first and then alternates between the Y buffer and the X buffer.

14.4.5 DMA Reload Operation (Automatic Register Swapping)

For each endpoint n (n = 1, 2, 3, 4, 5, 6, or 7), the USB DMA controller has a set of primary registers and a set of reload registers for the DMA transfer size and the DSP memory address (see Table 14–2). The primary registers are used for the current DMA transfer, and the reload registers are used to queue up an address and size for the next transfer.

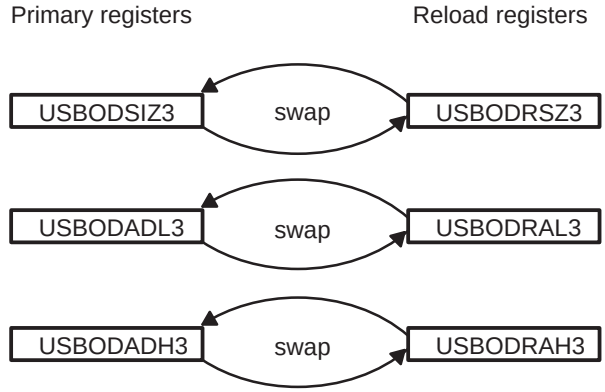
Table 14–2. Primary USB DMA Size and Address Registers and the Corresponding Reload Registers

Endpoint	Primary Register	Register Contains ...	Reload Register
OUT endpoint n	USBODSIZn	DMA transfer size in bytes	USBODRSZn
	USBODADLn	Low 16 bits of DSP memory address	USBODRALn
	USBODADHn	High 8 bits of DSP memory address	USBODRAHn
IN endpoint n	USBIDSIZn	DMA transfer size in bytes	USBIDRSZn
	USBIDADLn	Low 16 bits of DSP memory address	USBIDRALn
	USBIDADHn	High 8 bits of DSP memory address	USBIDRAHn

As mentioned in Table 14–1 on page 14-15, when the USB DMA controller completes a DMA transfer, it checks the RLD (reload) bit of the appropriate DMA control register. If RLD = 1, the controller performs a DMA reload operation: The controller swaps the contents of the primary registers and the reload registers. Example 14–1 shows a reload operation involving the registers for OUT endpoint 3.

This register swapping saves CPU time if you repeatedly toggle between the same two blocks in memory. Rather than putting new values into the reload registers between transfers, you can set the reload registers once and initiate a reload operation each time you want the controller to access the other block.

Example 14–1. DMA Reload Operation for OUT Endpoint 3



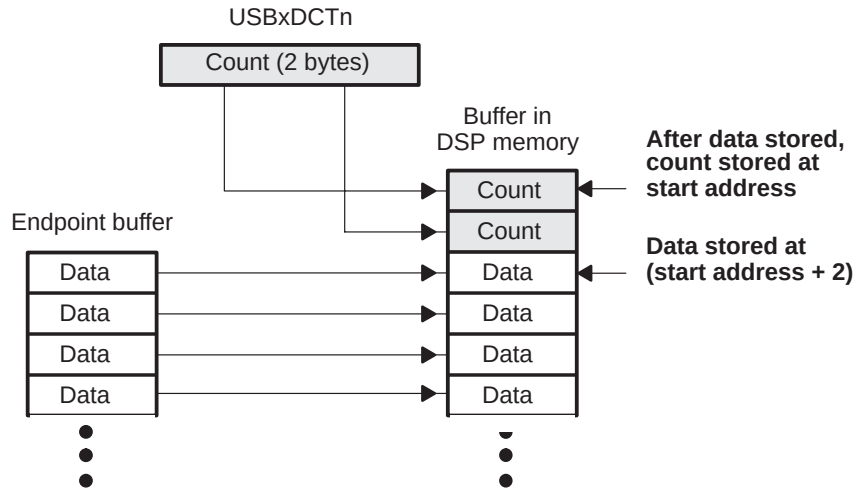
14.4.6 Transfer Count Saved to DSP Memory For an OUT Transfer

For each new DMA transfer, the USB DMA controller ensures that the transfer count for the endpoint starts at 0. When you give a go command (GO = 1), the controller clears the endpoint's count register (USBxDCTn) before moving the data. Likewise, after the controller completes a DMA reload operation (see section 14.4.5), it clears the count register before beginning the next DMA transfer.

At times, an OUT transfer will end with a short packet. If the USB DMA controller performs a DMA reload operation and immediately starts the next transfer, the count register is cleared before you can read the number of bytes in the packet. To prevent this loss of information, the controller copies the count to the DSP memory after every read from an endpoint buffer.

Figure 14–7 shows the positions of the data and the count in DSP memory. The start address is the address programmed in the primary address registers (USBxDADHn and USBDADLn). When the controller moves the data, it begins writing at (start address + 2). When all of the data has been moved, the controller stores the 2-byte count at the start address.

Figure 14–7. Storage of Transfer Count For an OUT Transfer



To properly read data from an OUT transfer, follow these guidelines:

- ❏ When you define the size of the buffer in DSP memory, include an additional two bytes for the DMA transfer count. Specifically, the buffer must be two bytes larger than the size you programmed in USBxDSIZn.
- ❏ When you read the data, keep in mind that the data starts two bytes after the start address you specified in USBxDADHn and USBxDADLn.

14.4.7 Configuring the USB DMA Controller

To configure an endpoint that will be accessed with CPU-initiated DMA transfers, use the instructions in the following paragraphs. In the register and bit names that appear in these paragraphs, a lowercase x can be O (for OUT) or I (for IN), and a lowercase n can be 1, 2, 3, 4, 5, 6, or 7 (indicating the endpoint number). For example, one of the possible values for USBxDCTLn is USBIDCTL4, which represents the DMA control register for IN endpoint 4.

14.4.7.1 Set the Transfer Size

Register(Field)	*symval*	Value	Description
USBxDSIZn(15–0)		1–65535	Number of bytes to be transferred
USBxDCTn(15–0)		1–65535	Number of bytes that have been transferred

For an endpoint n, you must tell the USB DMA controller how many bytes to transfer between the DSP memory and the endpoint. Write the number of bytes (up to 64K bytes) to USBxDSIZn.

The count value in USBxDCTn is cleared before each new DMA transfer and is updated with the number of bytes transferred at the end of the transfer. If you specified a DMA reload operation (RLD = 1), the controller automatically clears USBxDCTn before beginning the next DMA transfer.

For information about using the GO and RLD bits, see section 14.4.3 on page 14-14.

14.4.7.2 Set the DSP Memory Address

Register(Field)	*symval*	Value	Description
USBxDADHn(15–0)		0000h–00FFh	High 8 bits of the DSP memory address
USBxDADLn(15–0)		0000h–FFFFh	Low 16 bits of the DSP memory address

Because each endpoint has a dedicated DMA channel, the USB DMA controller knows the location of the buffer for endpoint n, but you must tell the controller which address to use when accessing the DSP memory. The controller accesses bytes in the endpoint buffer.

The address you specify must be a **byte address** with 24 bits. Load the 8 high bits of the address into USBxDADHn. (Bits 15–8 of USBxDADHn must contain 0s). Load the 16 low bits of the address into USBxDADLn.

In addition, the address must be **16-bit aligned**. Make sure the least significant bit (LSB) of USBxDADL is 0.

14.4.7.3 Enable/Disable a DMA Reload Operation and, If Necessary, Initialize the Reload Registers

Register(Field)	*symval*	Value	Description
USBxDCTLn(RLD)		0	No pending DMA reload operation. (Writing 0 to RLD has no effect.)
		1	Enable DMA reload operation.
USBxDRSZn(15–0)		0–65535	Reload-size value for USBxDSIZn
USBxDRAHn(15–0)		0000h–00FFh	Reload-address value for USBxDADHn
USBxDRALn(15–0)		0000h–FFFFh	Reload-address value for USBxDADLn

The USB DMA controller checks RLD at the end of each DMA transfer to determine whether to stop or to begin another transfer. If you want the controller to begin another transfer after the first, initialize the reload registers (USBxDRSZn, USBxDRAHn, and USBxDRALn) and then set the RLD bit. When the controller is done with the first transfer and it finds RLD = 1, it performs a DMA reload operation (see section 14.4.5 on page 14-18) and begins

the next transfer. If the controller is stopped (GO = 0), setting RLD has no effect.

Each time the controller performs a DMA reload operation, it clears RLD and notifies the CPU. To notify the CPU, the controller sets the endpoint's RLD interrupt flag bit in USBxDRIF. In addition, if the endpoint's DMA interrupts are enabled in USBxDIE, the controller sends an interrupt request to the CPU. To keep DMA transfers continuous, the CPU can set the RLD bit again before the end of each DMA transfer (that is, before the GO bit is cleared to 0).

14.4.7.4 If Desired, Enable DMA Interrupt Requests

Register(Field)	*symval*	Value	Description
USBxDIE(xEn)		0	Disable DMA GO and RLD interrupt requests.
		1	Enable DMA GO and RLD interrupt requests.

If DMA interrupts for an endpoint are enabled, the USB DMA controller can generate a GO interrupt request each time it clears the GO bit of USBxDCTLn; that is, it can notify the CPU that the controller has stopped. Similarly, the controller can use a RLD interrupt request to notify the CPU that a DMA reload operation is done (when the controller clears the RLD bit of USBxDCTLn).

Both of these DMA interrupt requests are enabled or disabled by a bit in one of the DMA interrupt enable registers. The interrupt enable bits for OUT endpoints 1–7 are in USBODIE; those for IN endpoints 1–7 are in USBIDIE. You enable GO and RLD interrupt requests for an endpoint by writing to the corresponding interrupt enable bit. For example, to enable DMA interrupt requests for IN endpoint 6, write a 1 to USBIDIE(IE6). For OUT endpoint 2, write a 1 to USBODIE(OE2).

For more details about the DMA interrupt requests, see section 14.6.3 on page 14-50.

14.4.7.5 Determine Whether to Reverse the Endianness (Orientation) of Data During DMA Transfers

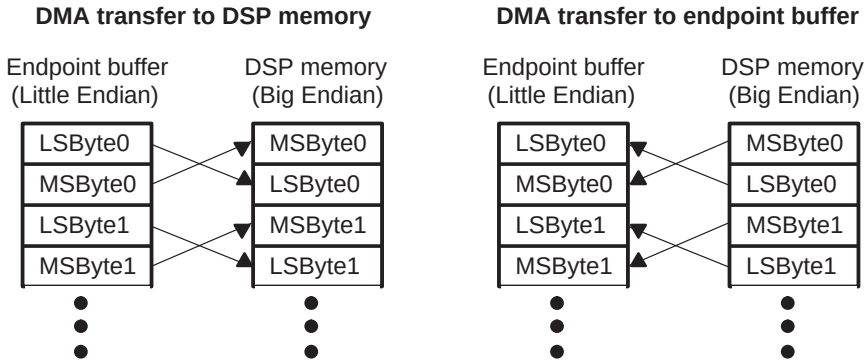
Register(Field)	*symval*	Value	Description
USBxDCTLn(END)		0	Do not change the order of the bytes in the next DMA transfer.
		1	Reverse the endianness of each word moved in the next DMA transfer.

In the Big Endian data orientation for words, the first byte is the most significant byte (MSByte) of the word. In the Little Endian orientation, the first byte is the

least significant byte (LSByte). The C55x CPU assumes that data in memory has the Big Endian orientation. When the UBM transfers data between the SIE and the endpoint buffer, the UBM does not change the order of any data bytes. However, by using the END bit, you can tell the USB DMA controller to swap the byte orientation before writing to the endpoint buffer or after reading from the endpoint buffer.

Figure 14–8 shows the effect of making END = 1 for an endpoint: The USB DMA controller reverses the endianness of the words it transfers between the DSP memory and the endpoint buffer. When END = 0 for an endpoint, the USB DMA controller does not change the order of the bytes.

Figure 14–8. The Effect of END = 1 on USB DMA Transfers



14.4.7.6 Enable/Disable Concatenation

Register(Field)	*symval*	Value	Description
USBxDCTLn(CAT)		0	Disable concatenation.
		1	Enable concatenation.

When the packet sent or received at an endpoint is larger than the DMA transfer size, you may want to set CAT = 1, so that the USB DMA controller does multiple DMA transfers for a single OUT or IN transfer on the USB. For more details, see the description for the CAT bit in section 14.8.2.1 (page 14-60).

14.4.7.7 Select Whether to Require Short Packets

Register(Field)	*symval*	Value	Description
USBxDCTLn(SHT)		0	Short packets not required
		1	Short packets required

Typically, a USB transfer ends with a short packet, a packet that is shorter than the maximum allowable size for the endpoint. If a maximum-size packet is the

last packet, the transmitter (host or slave device) can send one more packet, of zero length, to indicate that there is no more data. The SHT bit tells the USB DMA controller whether to wait for/generate 0-byte packets when the transfer ends with a maximum-size packet. For more details, see the description for the SHT bit in section 14.8.2.1 (page 14-60).

14.4.7.8 Select Whether a Missing Packet is an Error During Isochronous Transfers

Register(Field)	*symval*	Value	Description
USBxDCTLn(EM)		0	Do not halt the USB DMA controller in response to a missing packet. Treat a missing packet as a zero-length packet.
		1	A missing packet is an error and will stop the USB DMA controller.

If the USB DMA controller is handling data packets for an isochronous endpoint, this bit determines how the controller will respond if no packet is received in/transmitted from the endpoint buffer during the current USB frame. For more details, see the description for the EM bit in section 14.8.2.1 (page 14-60).

14.4.8 Monitoring CPU-Initiated DMA Transfers

To monitor an endpoint that is involved in CPU-initiated DMA transfers, use the instructions in the following paragraphs. In the register and bit names that appear in these paragraphs, a lowercase x can be O (for OUT) or I (for IN), and a lowercase n can be 1, 2, 3, 4, 5, 6, or 7 (indicating the endpoint number). For example, one of the possible values for USBxDCTLn is USBODCTL5, which represents the DMA control register for OUT endpoint 5.

14.4.8.1 Checking the Transfer Count (USBxDCTn)

Register(Field)	*symval*	Value	Description
USBxDCTn(15–0)		0–65535	Indicates how many bytes have been transferred

The USB DMA controller clears USBxDCTn before each new DMA transfer, including those transfers following DMA reload operations. USBxDCTn is updated with the number of bytes transferred at the end of a transfer. You can read this register to determine how many bytes were moved.

14.4.8.2 Determining Whether a DMA Transfer is In Progress/Done

Register(Field)	*symval*	Value	Description
USBxDCTLn(GO)		0	The USB DMA controller is idling (ready for the next DMA transfer).
		1	A DMA transfer is in progress.

Register(Field)	*symval*	Value	Description
USBxDGIF(xEn)		0	No DMA GO interrupt is pending.
		1	The USB DMA controller has completed the current DMA transfers or series of transfers and has cleared the GO bit. This event also generates a GO interrupt if the interrupt is enabled by USBxDIE(xEn).

When the CPU sets the GO bit, the DMA controller begins a DMA transfer. At the end of the transfer, if the RLD bit is 1, the controller does not clear GO. Instead the controller performs a DMA reload operation (see section 14.4.5 on page 14-18) and begins a new transfer with the new address and size. By using repeated reload operations, you can have the controller perform a series of transfers.

When the controller completes a transfer and finds RLD = 0, it clears GO to 0. In addition, it sets the endpoint's GO flag in USBxDGIF and generate an interrupt request. For example, if the controller is done at OUT endpoint 4, it sets the OE4 bit in USBODGIF. If the corresponding interrupt request is enabled by the OE4 bit in USBODGIE, an interrupt request is generated.

14.4.8.3 Determining Whether a DMA Reload Operation is In Progress/Done

Register(Field)	*symval*	Value	Description
USBxDCTLn(RLD)		0	USB DMA controller is done with the previously requested reload operation.
		1	USB DMA controller is waiting to complete a reload operation.
USBxDRIF(xEn)		0	No DMA RLD interrupt is pending.
		1	The USB DMA controller has completed the DMA reload operation and has cleared RLD. This event also generates a RLD interrupt if the interrupt is enabled by USBxDIE(xEn).

When the USB DMA controller completes a DMA transfer, it checks the RLD bit. If RLD = 1, the controller performs a DMA reload operation (see section 14.4.5 on page 14-18). When the DMA reload operation is done, the controller clears RLD. In addition, it sets the endpoint's RLD flag in USBxDRIF and can generate an interrupt request. For example, if the controller complete a DMA reload operation for OUT endpoint 4, it sets the OE4 bit in USBODRIF. If the corresponding interrupt request is enabled by the OE4 bit in USBODRIE, an interrupt request is generated.

14.4.8.4 Checking for an Overflow or Underflow Condition

Register(Field)	*symval*	Value	Description
USBxDCTLn(OVF)		0	No overflow/underflow detected
		1	Overflow/underflow detected

Essentially, an overflow condition occurs when too many bytes are arriving in an endpoint buffer, and an underflow condition occurs when not enough bytes are available to be read from the endpoint buffer. For more details, see the description for the OVF bit in section 14.8.2.1 (page 14-60).

14.4.8.5 Watching for Missed Packets During Isochronous Transfers

Register(Field)	*symval*	Value	Description
USBxDCTLn(PM)		0	No missing packet.
		1	Missing packet: A packet did not arrive in the previous USB frame for the endpoint.

If the USB DMA controller is handling data packets for an isochronous endpoint, and you can determine how the controller will respond if no packet is received in/transmitted from the endpoint buffer during the current USB frame. If you want the controller to consider a missing packet an error condition, set the EM bit in USBxDCTLn. If EM = 1, you can watch for missing packets by monitoring the PM bit in USBxDCTLn. EM and PM are described in section 14.8.2.1 (page 14-60).

14.4.9 USB DMA State Tables and State Diagrams

This section contains the following state tables to summarize the status of the USB DMA controller under various conditions:

- Table 14–3: Non-isochronous IN DMA Transfer (page 14-27)
- Table 14–4: Non-isochronous OUT DMA Transfer (page 14-29)
- Table 14–5: Isochronous IN DMA Transfer (page 14-31)
- Table 14–6: Isochronous OUT DMA Transfer (page Table 14–6)

This section also includes the following state diagrams to support the isochronous transfer state tables:

- Figure 14–9: Missing Packet Response for Isochronous OUT DMA Transfer (page 14-38)
- Figure 14–10: Missing Packet Response for Isochronous IN DMA Transfer (page 14-39)

Table 14–3. State Table: Non-Isochronous IN DMA Transfer

Description		Initial State						End State						
DMA transfer state	**Programmer view**	**Bytes free in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Update DMA count**	**Send packet (clear NAK)**	**DMA transfer activity**
Normal transfer in progress		> 0	> 0	x	x	x	x							In progress
Endpoint buffer full	Stop requested	0	x	1	x	x	x	1		1	1			Idle
DMA transfer completion	Stop requested	> 0	0	1	x	x	x	1		1	1			Idle
Endpoint buffer full, more data remaining in DMA transfer		0	> 0	0	x	x	x					1	Max	Idle
DMA transfer completion, endpoint buffer full		0	0	0	0	1	x	1	0	1		1	Max	Idle
					1				1	0				
DMA transfer completion, endpoint buffer full		0	0	0	0	0	0	1	0	1		1	Max	Idle
					1				1	0				
DMA transfer completion, endpoint buffer full	Short packet requested	0	0	0	x	0	1					1	Max	Idle until current packet moves out, then prepare a 0-byte packet

Table 14–3. State Table: Non-Isochronous IN DMA Transfer (Continued)

Description		**Initial State**						**End State**						
DMA transfer state	**Programmer view**	**Bytes free in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Update DMA count**	**Send packet (clear NAK)**	**DMA transfer activity**
DMA transfer completion, endpoint buffer **not** full		> 0	0	0	0	0	x	1	0	1		1	Short	Idle
					1				1	0				
DMA transfer completion, endpoint buffer **not** full	CAT requested, next buffer is **not** ready yet	> 0	0	0	0	1	x	1		1		1		Pause
DMA transfer completion, endpoint buffer **not** full	CAT requested, next buffer is ready	> 0	0	0	1	1	x	1	1			1		In progress

Table 14–4. State Table: Non-Isochronous OUT DMA Transfer

Description		Initial State							End State							
DMA transfer state	**Programmer view**	**Received packet size**	**Bytes in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA count**	**Clear NAK for next packet**	**DMA transfer activity**
Normal transfer in progress		x	> 0	> 0	x	x	x	x								In progress
Packet transfer completion	Stop requested	x	0	x	1	x	x	x	1		1	1		1	1	Idle
DMA transfer completion	Stop requested	x	> 0	0	1	x	x	x	1		1	1				Idle
Packet transfer completion, more data remaining in DMA transfer		Max	0	> 0	0	x	x	x						1	1	Idle
Packet transfer completion, more data remaining in DMA transfer		Short	0	> 0	0	0	x	x	1	0	1			1	1	Idle
						1				1	0					
DMA transfer completion, packet fits exactly		Max	0	0	0	0	x	0	1	0	1			1	1	Idle
						1				1	0					
DMA transfer completion, packet fits exactly		Max	0	0	0	0	1	x	1	0	1			1	1	Idle
						1				1	0					

Table 14–4. State Table: Non-Isochronous OUT DMA Transfer (Continued)

Description		Initial State							End State							
DMA transfer state	**Programmer view**	**Received packet size**	**Bytes in endpt buffer**	**Bytes in DMA transfer**	**STP**	**RLD**	**CAT**	**SHT**	**End of current DMA transfer**	**Reload and swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA count**	**Clear NAK for next packet**	**DMA transfer activity**
DMA transfer completion, packet fits exactly	Short packet requested	Max	0	0	0	x	0	1						1	1	Idle, expecting a 0–byte packet
DMA transfer completion, packet fits exactly	Short packet requested	Short	0	0	0	0	x	x	1	0	1			1	1	Idle
						1				1	0					
DMA transfer completion, more data remaining in endpoint buffer	Overflow condition	x	> 0	0	0	x	0	x	1		1		1	1		Idle
DMA transfer completion, more data remaining in endpoint buffer	CAT requested, next buffer is not ready yet	x	> 0	0	0	0	1	x	1		1			1		Pause
DMA transfer completion, more data remaining in endpoint buffer	CAT requested, next buffer is ready	x	> 0	0	0	1	1	x	1	1				1		In progress

Table 14–5. State Table: Isochronous IN DMA Transfer

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
DMA failed to keep up with USB		0	x	x	x	x	x	x	x	1		1		1			Idle
Normal transfer in progress		1	> 0	> 0	x	x	x	x	x								In progress
Endpoint buffer full, more data remaining in DMA transfer	Stop requested	1	0	> 0	x	1	x	x	x	1		1	1		1	Max	Idle
DMA transfer completion, endpoint buffer is **not** full	Stop requested	1	> 0	0	x	1	x	x	x	1		1	1				Idle
DMA transfer completion, endpoint buffer is full	Stop requested	1	0	0	x	1	x	x	x	1		1	1		1	Max	Idle

† Entries in this column are taken from the missing packet response state diagram of Figure 14–9 (page 14-38).

Table 14–5. State Table: Isochronous IN DMA Transfer (Continued)

Description		**Initial State**								**Final State**							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
Endpoint buffer full, more data remaining in DMA transfer, host missed an IN request earlier		1	0	> 0	1	0	x	x	x	1		1			0	Max	Idle
Endpoint buffer full, more data remaining in DMA transfer		1	0	> 0	0	0	x	x	x						1	Max	Idle
DMA transfer completion, endpoint buffer is **not** full	Current buffer is the last of the transfer	1	> 0	0	0	0	0	0	x	1	0	1			1	Non-max packet	Idle
							1				1	0					

† Entries in this column are taken from the missing packet response state diagram of Figure 14–9 (page 14-38).

Table 14–5. State Table: Isochronous IN DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, endpoint buffer is **not** full	CAT requested, next buffer is not ready yet (underflow condition)	1	> 0	0	0	0	0	1	x	1		1		1	1	Non-max packet	Idle
DMA transfer completion, endpoint buffer is **not** full	CAT requested, next buffer is ready	1	> 0	0	0	0	1	1	x	1					1		Start next transfer, fill up rest of endpoint buffer
DMA transfer completion, endpoint buffer full, missing packet error seen		1	0	0	1	0	x	x	x	1		1					Idle
DMA transfer completion, endpoint buffer full		1	0	0	0	0	0	0	0	1	0	1			1	Max	Idle
							1				1	0					

† Entries in this column are taken from the missing packet response state diagram of Figure 14–9 (page 14-38).

Table 14–5. State Table: Isochronous IN DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes free in Endpt Buffer**	**Bytes in DMA transfer**	**Missing Packet Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Send Packet (clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, endpoint buffer full	Short (0-byte) packet requested	1	0	0	0	0	x	0	1						1	Short (zero)	Idle
DMA transfer completion, endpoint buffer full	CAT requested	1	0	0	0	0	0	1	x	1	0	1			1	Max	Idle
							1				1	0					

† Entries in this column are taken from the missing packet response state diagram of Figure 14–9 (page 14-38).

Table 14–6. State Table: Isochronous OUT DMA Transfer

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes in Endpt Buffer**	**Bytes in DMA transfer**	**Normal, Short, Ignore, Missing Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA failed to keep up with USB		0	x	x	x	x	x	x	x	1		1		1			Idle
Normal transfer in progress		1	> 0	> 0	x	x	x	x	x								In Progress
Endpoint buffer is empty	Stop requested	1	0	x	x	1	x	x	x	1		1	1		1	1	Idle
Endpoint buffer empty, more data remaining in DMA transfer		1	0	> 0	Normal	0	x	x	x						1	1	Idle
		1	0	> 0	Short	0	0	x	x	1	0	1			1	1	Idle
							1				1	0					
		1	0	> 0	Ignore	0	x	x	x							1	Idle
		1	0	> 0	Missing Error	0	x	x	x	1		1			1	1	Idle

† Entries in this column are taken from the missing packet response state diagram of Figure 14–10 (page 14-39).

Table 14–6. State Table: Isochronous OUT DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes in Endpt Buffer**	**Bytes in DMA transfer**	**Normal, Short, Ignore, Missing Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, more data remaining in the endpoint buffer.	Stop requested	1	> 0	0	Normal	1	x	x	x	1		1	1			1	Idle
	Overflow condition	1	> 0	0	Normal	0	x	0	x	1		1		1	1	1	Idle
	CAT requested, next buffer is not ready yet (overflow condition)	1	> 0	0	Normal	0	0	1	x	1		1		1	1	1	Idle
	CAT requested, next buffer is ready	1	> 0	0	Normal	0	1	1	x	1	1				1	0	In progress

† Entries in this column are taken from the missing packet response state diagram of Figure 14–10 (page 14-39).

Table 14–6. State Table: Isochronous OUT DMA Transfer (Continued)

Description		Initial State								Final State							
DMA Transfer State	**Program-mer view**	**DMA transfer complete before SOF**	**Bytes in Endpt Buffer**	**Bytes in DMA transfer**	**Normal, Short, Ignore, Missing Error †**	**STP**	**RLD**	**CAT**	**SHT**	**End of Current DMA transfer**	**Reload and Swap**	**Reset GO**	**Reset STP**	**Set OVF**	**Update DMA Count**	**Receive Packet (Clear NAK)**	**DMA Transfer Activity**
DMA transfer completion, endpoint buffer is empty	Stop requested	1	0	0	x	1	x	x	x	1		1	1		1	1	Idle
	CAT requested	1	0	0	Normal	0	0	1	x	1	0	1			1	1	Idle
							1				1	0					
		1	0	0	Normal	0	0	0	0	1	0	1			1	1	Idle
							1				1	0					
	Short (0-byte) Packet expected	1	0	0	Normal	0	x	0	1						1	1	Expect a short packet next
DMA expecting a short (0-byte) packet to end the transfer		1	0	0	Missing Error	0	x	x	x	1		1			1	1	Idle
		1	0	0	Short	0	0	x	x	1	0	1			1	1	Idle
							1				1	0					
		1	0	0	Ignore	0	x	x	x							1	In progress

† Entries in this column are taken from the missing packet response state diagram of Figure 14–10 (page 14-39).

Figure 14–9. Missing Packet Response for Isochronous IN DMA Transfer (Supports Table 14–5 on Page 14-31)

GO: Go command to USB DMA controller
PM: Previous packet missed
EM: Error if missing packet

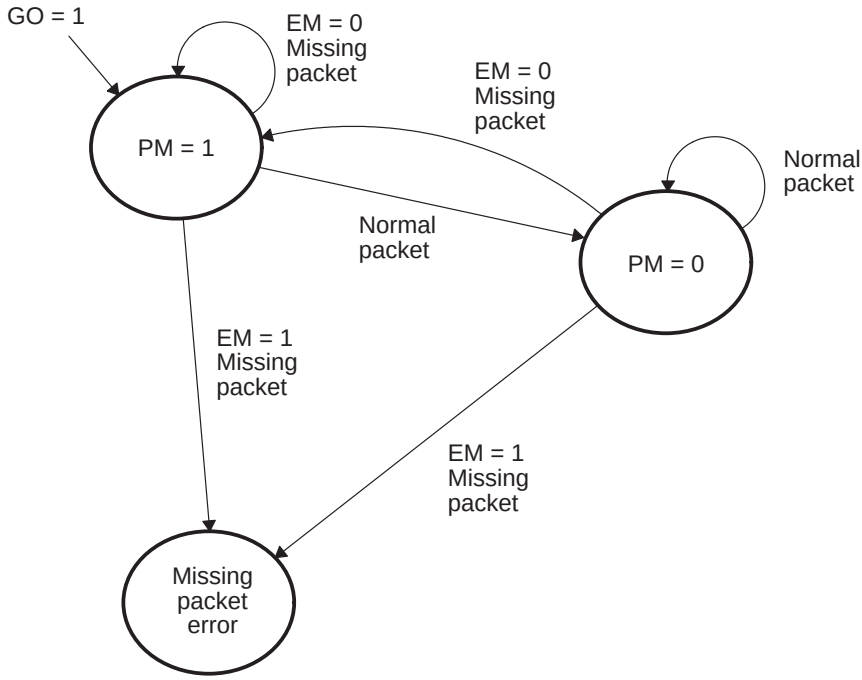
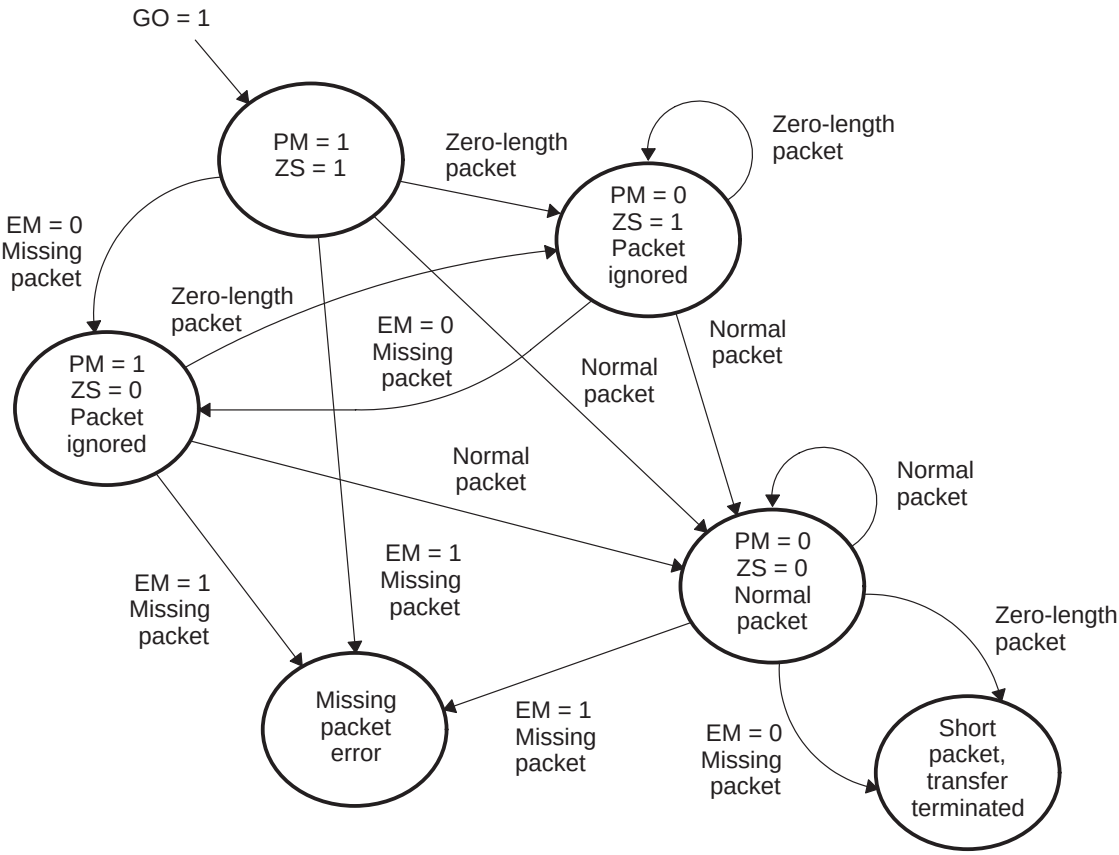


Figure 14–10. Missing Packet Response for Isochronous OUT DMA Transfer (Supports Table 14–6 on Page 14-35)

GO: Go command to USB DMA controller
PM: Previous packet missed
EM: Error if missing packet
ZS: Zero-length packet status



14.5 Host-DMA Mode: Host-Initiated Direct Memory Accesses

In the host-DMA mode, the host can initiate direct memory accesses through two general-purpose endpoints (one OUT endpoint and one IN endpoint). On the USB, the host-DMA endpoints appear as bulk endpoints (endpoints that expect bulk transfers). During enumeration, the device should report that these endpoints are bulk endpoints, and the host should communicate with these endpoints as it would with any bulk endpoint.

Each host-DMA transfer is performed on a packet-by-packet basis. Each transfer begins with an OUT packet containing a 5 byte host-DMA protocol header (described in section 14.5.1). If data is moving from the host to the device, the data being transferred follows the protocol header in the same packet. If the data is moving from the device to the host, the data being transferred is sent in a subsequent IN packet.

When the CPU has relinquished DMA control to the host, the CPU should not write to the GO bit or the RLD bit. However, the CPU can be notified by interrupt flags and (if enabled) interrupts when a host-DMA transfer is complete or if an error has occurred. You do not need to initialize any of the DMA context registers because the details of each transfer are in the header protocol sent by the host.

Note:

Once the two bulk endpoints are selected for this purpose, the endpoints cannot be used for any other purpose until a USB software reset (write 1 to the SOFTRST bit of USBGCTL) or a DSP reset (drive the $\overline{\text{RESET}}$ pin low) is performed.

14.5.1 Protocol Header For the Host-DMA Mode

Each time the host communicates with host-DMA endpoints, the host must first send a 5-byte protocol header to describe the desired transfer to the USB DMA controller. Figure 14–11 shows the values the host must send in the header. Byte 1 tells the USB DMA controller about the data transfer that is requested. Table 14–7 gives the details about this first byte. Bytes 2–5 must contain the DSP memory address as shown in the following figure. Because a C55x DSP uses 24-bit addresses, byte 5 of the header protocol must be 0s.

Figure 14–11. Bytes of the Host-DMA Protocol Header

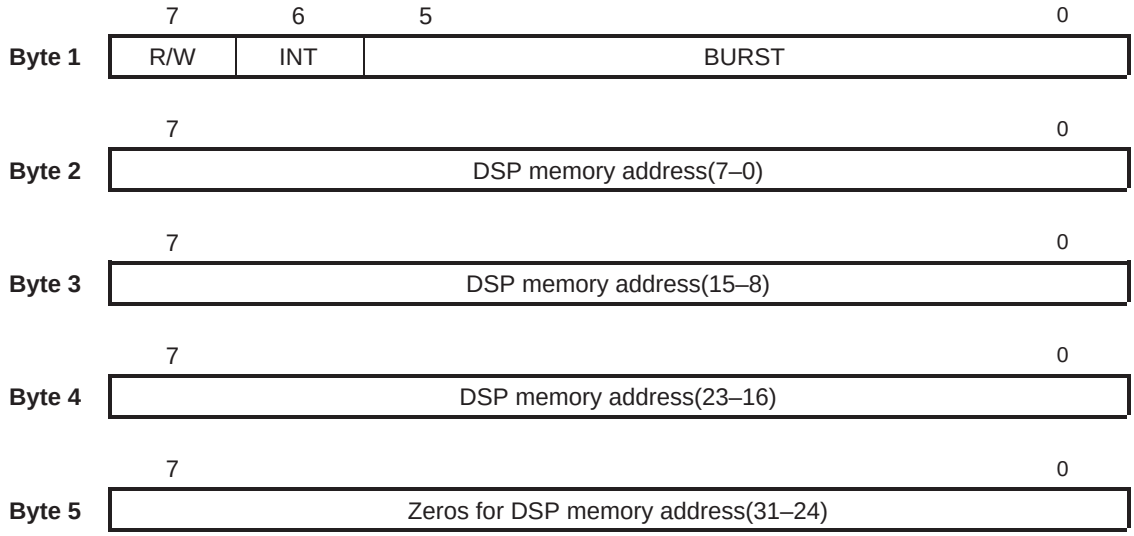


Table 14–7. The Fields of Byte 1 of the Host-DMA Protocol Header

Bit	*field*	*symval*	Value	Function
7	R/W		0	Host write operation: The host wants the USB DMA controller to write data to the DSP memory.
			1	Host read operation: The host wants the USB DMA controller to read data from the DSP memory.
6	INT		0	No host interrupt
			1	Host interrupt requested: The host wants the USB DMA controller to generate a host interrupt (HINT) at the end of the transfer.
5–0	BURST		n =1–59	The host wants n bytes transferred by the USB DMA controller. The host can send as few as one byte (BURST = 1) and as many as 59 bytes (BURST = 59).

14.5.2 Initiating a Host Read Operation (Data to Host)

When the host wants to read data from the DSP memory, the host must initiate two bulk transfers on the USB in the following order:

1) **OUT transfer to program the DMA state machine.** In the first transfer, the host sends a protocol header (5 bytes), which includes a bit to indicate that the host wants to read data. The UBM receives the header from the SIE and then passes the header to the selected OUT endpoint buffer. The USB DMA controller reads the header, configures itself, and then moves the requested data from the memory to the selected IN endpoint buffer.

2) **IN transfer to get the data.** In the second transfer, the host sends a request for the data. The SIE passes up to 64 bytes to the host. The first 5 bytes is a copy of the protocol header that was sent in the previous OUT transfer. The next 1 to 59 bytes are the data the UBM has retrieved from the selected IN endpoint buffer.

14.5.3 Initiating a Host Write Operation (Data to DSP Memory)

To write data to the DSP memory, the host must initiate one bulk transfer: an OUT transfer that sends the 5-byte protocol header followed by up to 59 bytes of data (up to 64 bytes total). After the UBM has passed all the bytes to the selected OUT endpoint buffer, the USB DMA controller interprets the header and then moves the 1 to 59 data bytes from the buffer to the DSP memory.

14.5.4 Configuring the USB Module for the Host-DMA Mode

The following tables and paragraphs explain the steps to follow to prepare the USB module for the host-DMA mode. First, choose two of the general-purpose endpoints to serve as the host-DMA endpoints, and program these two endpoints as bulk endpoints with the double buffer mode enabled. Second, select the endpoints in USBHEPSEL and (if desired) enabled the interrupts for the host-DMA mode. Finally, enable the host-DMA mode.

14.5.4.1 Make Sure the Chosen Endpoints are in the Non-Isochronous Mode

Register(Field)	*symval*	Value	Description
USBxCNFn(ISO)		0	Non-isochronous mode
		1	Isochronous mode

The host-DMA endpoints must be configured as bulk endpoints, and for bulk transfers, an endpoint must be in the non-isochronous mode. Clear the ISO bit of the appropriate endpoint configuration registers. For example, if your host-DMA endpoints are OUT endpoint 4 and IN endpoint 4, make sure that USBOCNF4(ISO) = 0 and USBICNF4(ISO) = 0. The endpoint configuration registers are described on pages 14-70 (for IN endpoints) and 14-72 (for OUT endpoints).

14.5.4.2 Make Sure the Chosen Endpoints are in the Double Buffer Mode

Register(Field)	*symval*	Value	Description
USBxCNFn(DBUF)		0	Single buffer mode
		1	Double buffer mode

Whenever the USB DMA controller accesses an endpoint buffer, the controller assumes that the endpoint buffer is equally divided between an X buffer and a Y buffer. To meet this requirement for your host-DMA endpoints, set the DBUF bits in the appropriate endpoint configuration registers. For example, if your host-DMA endpoints are OUT endpoint 1 and IN endpoint 3, make sure that USBOCNF1(DBUF) = 1 and USBICNF3(DBUF) = 1. The endpoint configuration registers are described on pages 14-70 (for IN endpoints) and 14-72 (for OUT endpoints).

14.5.4.3 Set the Maximum Packet Size for the Chosen Endpoints (64 Bytes Recommended)

Register(Field)	*symval*	Value	Description
USBxSIZn(SIZ)		0–64	Size for buffer X/Y (maximum packet size)

In the double buffer mode (required for the USB DMA controller), the USB DMA controller accesses one of the buffers at a time (X or Y), based on the data toggle sequence (DATA0, DATA1, DATA0, and so on). The 7-bit SIZ field in USBxSIZn determines the maximum packet size: the number of bytes the active buffer (X or Y) can hold at one time. Because the host-DMA mode allows transfers up to 64 bytes long, it is recommended that you load 64 into SIZ for each of the host-DMA endpoints. If you want more than 59 data bytes transferred, send them in multiple host-DMA transfers. The fields of the buffer size register are described in section 14.8.3.5 (page 14-78).

14.5.4.4 Select the Chosen Endpoints in USBHEPSEL

Register(Field)	*symval*	Value	Description
USBHEPSEL(IEP)		1–7	Indicates which IN endpoint will be used as the host-DMA endpoint for IN transfers.
USBHEPSEL(OEP)		1–7	Indicates which OUT endpoint will be used as the host-DMA endpoint for OUT transfers.

With USBHEPSEL you must select one of the OUT endpoints and one of the IN endpoints to be used for host-initiated DMA transfers. Once you select the host-DMA endpoints in this register, these two endpoints cannot be used for any other purpose until you initiate a USB software reset (write 1 to the SOFTRST bit of USBGCTL) or a DSP reset (drive the $\overline{\text{RESET}}$ pin low). The fields of USBHEPSEL are described in section 14.8.6.2 on page 14-94.

14.5.4.5 Enable or Disable the Interrupts Associated with the Host-DMA Mode

Register(Field)	*symval*	Value	Description
USBHCTL(HIE)		0	Disable host interrupt requests.
		1	Enable host interrupt requests.
USBHCTL(HERRIE)		0	Disable host error interrupt requests.
		1	Enable host error interrupt requests.
USBHSTAT(HIF)		0	Host-DMA transfer not done
		1	Host-DMA transfer done. The USB DMA controller has completed a DMA transfer between the DSP memory and a host-DMA endpoint. This flag is set only if the host requests the interrupt by setting the INT bit in the first byte of the protocol header.
USBHSTAT(HERRIF)		0	No host error
		1	A host error has occurred: During an OUT transfer, the size of the data transferred from the host did not match the size specified in the protocol header.

If you want the host to be able to generate an interrupt request at the end of a host-DMA transfer, set the host interrupt enable (HIE) bit of the host control register (USBHCTL). Likewise, if you want the CPU to be notified of a host transfer error, set the host error interrupt enable (HERRIE) bit of USBHCTL.

The host status register (USBHSTAT) has flags that the CPU can poll regardless of whether the interrupts are enabled. The HIF bit indicates that the USB DMA controller has completed a host-DMA transfer and the host requested an interrupt at the end of the DMA transfer (in the protocol header). The HERRIF bit indicates that a host error has occurred.

14.5.4.6 Enable the Host-DMA Mode

Register(Field)	*symval*	Value	Description
USBHCTL(EN)		0	Disable host-DMA mode.
		1	Enable host-DMA mode (allow the host to initiate DMA transfers at the endpoints selected in USBHEPSEL).
USBHSTAT(DIS)		0	Host-DMA mode enabled.
		1	Host-DMA mode disabled.

A reset clears the EN bit of USBHCTL. Set the EN bit to enable the host-DMA mode. To verify that the mode is on, you can check the DIS bit of USBHSTAT. If DIS = 1, the mode is disabled. The fields of USBHCTL and USBHSTAT are described in on pages 14-93 and 14-94, respectively.

14.6 Interrupt Activity in the USB Module

The interrupt requests generated by the USB module can be grouped into the following main categories:

- ❏ USB bus interrupt requests (see section 14.6.1 on page 14-47)
- ❏ Endpoint interrupt requests (see section 14.6.2 on page 14-48)
- ❏ USB DMA interrupt requests (see section 14.6.3 on page 14-50)
- ❏ Host-DMA mode interrupt requests (see section 14.6.4 on page 14-52)

As shown in Figure 14–12, all requests are multiplexed through an arbiter to a single USB interrupt request for the CPU. When the arbiter receives multiple interrupt requests at the same time, it services them one at a time according to a predefined priority ranking. The priority of each request is included in the description of the interrupt source register (USBINTSRC) on page 14-95.

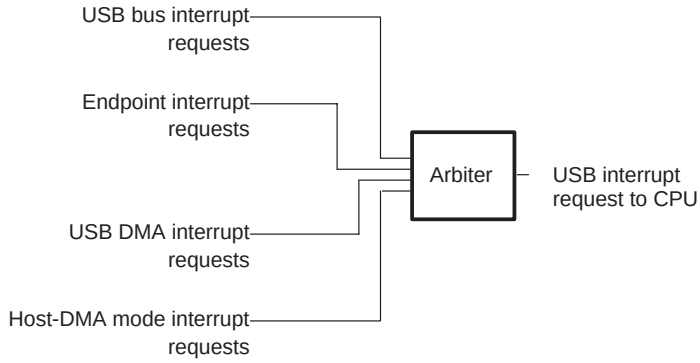
The USB interrupt is one of the maskable interrupts of the CPU. As with any maskable interrupt request, if it is properly enabled in the CPU, the CPU executes the corresponding interrupt service routine (ISR). The ISR for the USB interrupt can determine the interrupt source by reading the interrupt source register, USBINTSRC (see page 14-95). Then the ISR can branch to the appropriate subroutine.

After the CPU reads USBINTSRC, the following events occur:

1) The interrupt flag for the source interrupt is cleared in the corresponding interrupt flag register. *Exception:* The STPOW and SETUP bits in USBIF are not cleared when USBINTSRC 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 USBINTSRC, and forwards the interrupt request to the CPU.

Figure 14–12. Possible Sources of a USB Interrupt Request



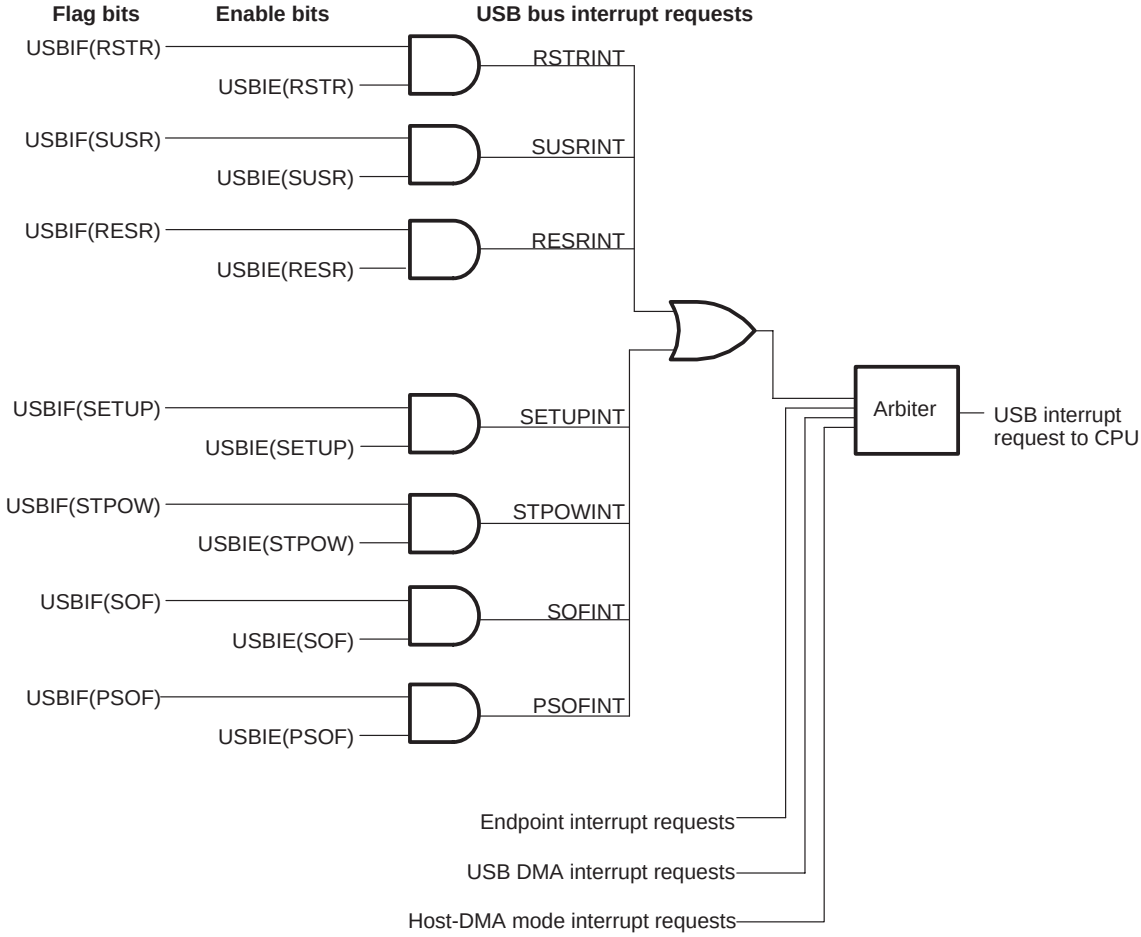
14.6.1 USB Bus Interrupt Requests

The USB module can generate a number of interrupt requests that are related to activity on the USB (see Table 14–8). As shown in Figure 14–13, each of the interrupt requests has a flag bit in the USB interrupt flag register (USBIF) and an enable bit in the USB interrupt enable register (USBIE). When one of the specified events occurs, its flag bit is set. If the corresponding enable bit is 0, the the interrupt request is blocked. If the enable bit is 1, the request is forwarded to the CPU as a USB interrupt.

Table 14–8. Descriptions of the USB Bus Interrupt Requests

USB Bus Interrupt Request	Interrupt Source
RSTRINT	A reset condition is detected on the USB.
SUSRINT	A suspend condition is detected on the USB.
RESRINT	Activity on the USB resumes, ending a suspend condition.
SETUPINT	A setup packet arrived. (Setup data is stored in the setup packet buffer.)
STPOWINT	A setup overwrite has occurred; that is, a new setup packet arrived before the previous setup packet was read from the setup packet buffer.
SOFINT	A start-of-frame (SOF) packet is detected on the USB.
PSOFINT	The pre-SOF (PSOF) timer has finished counting down. If you want an interrupt to occur n (1 to 255) clock cycles before each SOF packet, load n into the pre-SOF interrupt timer count register, USBPSOFTMR (see page 14-97). The counter runs at 750 kHz (12MHz/16).

Figure 14–13. Enable Paths of USB Bus Interrupt Requests



14.6.2 Endpoint Interrupt Requests

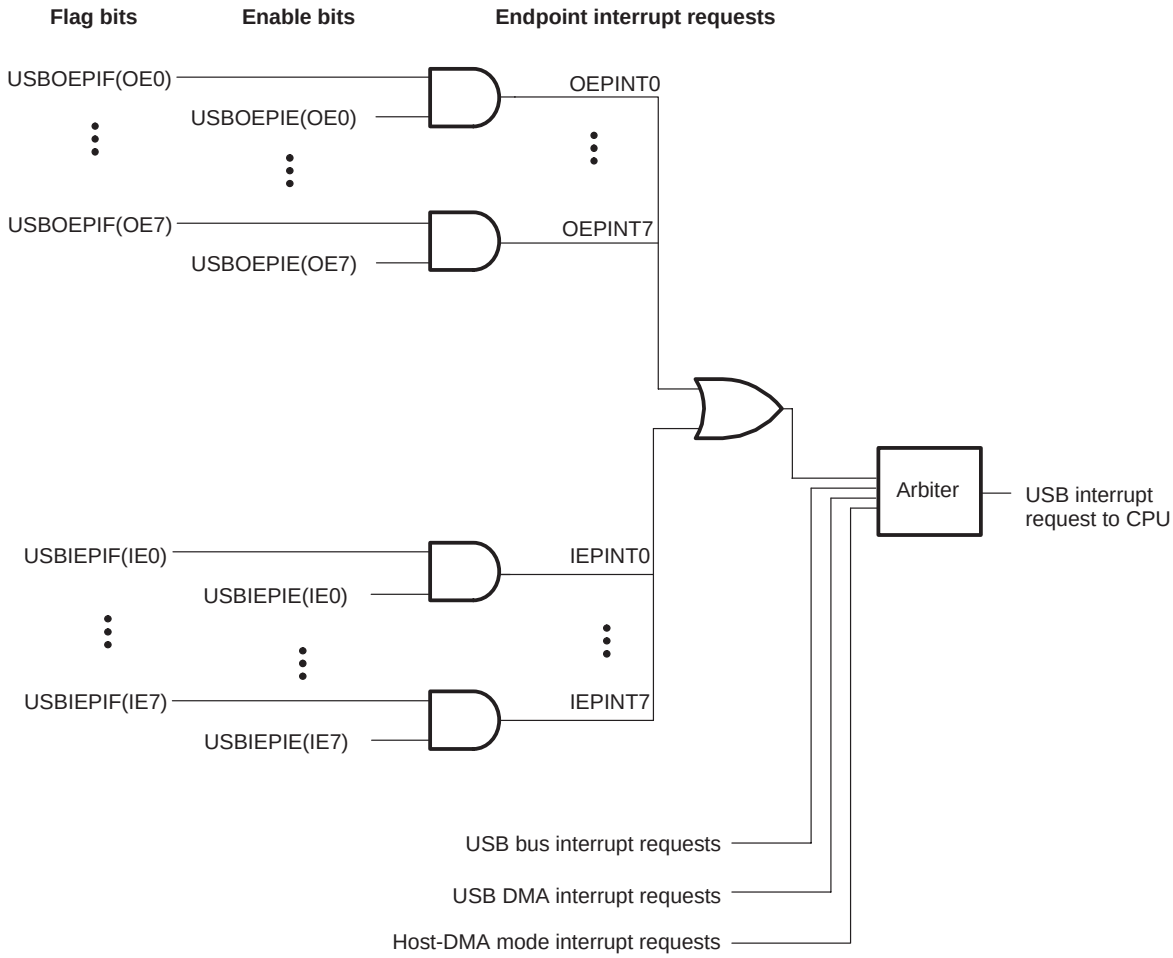
For each endpoint, the UBM can generate an interrupt request every time data moves in or out of the endpoint buffer. As shown in Figure 14–14:

- Each OUT endpoint has a flag bit in the OUT endpoint interrupt flag register (USBOEPIF) and an enable bit in the OUT endpoint interrupt enable register (USBOEPIE).
- Each IN endpoint has a flag bit in the IN endpoint interrupt flag register (USBIEPIF) and an enable bit in the IN endpoint interrupt enable register (USBIEPIE).
- For either type of endpoint, when both the flag bit and the enable bit are set, an interrupt request is passed to the CPU.

Software must set the enable bit, but the flag bit is set by the UBM when a specific event occurs:

- **For an OUT endpoint (data from host):** When the UBM receives a valid data packet, it writes the data to the appropriate OUT endpoint buffer. The UBM then sets the NAK bit of the endpoint's count register, to keep the host from writing to the buffer before the data is read. When the NAK bit is set, the associated interrupt flag bit is also set.
- **For an IN endpoint (data to host):** When the USB module receives an IN packet, the UBM reads the data from the appropriate IN endpoint buffer. Then the UBM sets the NAK bit of the endpoint's count register, to keep the host from reading again before new data is placed in the buffer. When the NAK bit is set, the associated interrupt flag bit is also set.

Figure 14–14. Enable Paths for the Endpoint Interrupt Requests



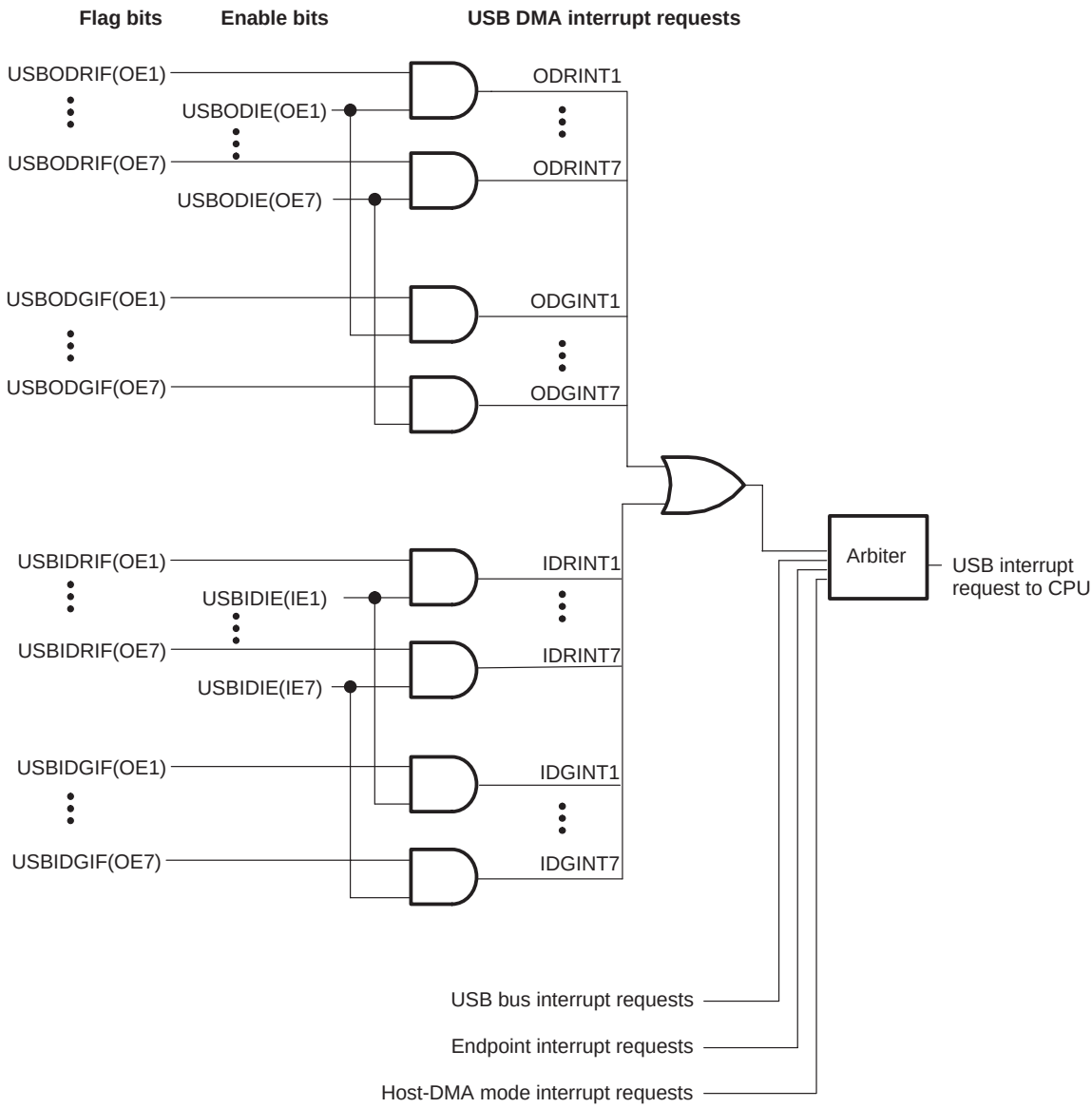
14.6.3 USB DMA Interrupt Requests

The USB module can generate an interrupt request every time the USB DMA controller clears the GO bit or RLD (reload) bit for one of the general-purpose endpoints (OUT endpoints 1–7 and IN endpoints 1–7). As shown in Figure 14–15:

- Each OUT endpoint has:
  - One flag bit in the OUT endpoint DMA GO interrupt flag register (USBODGIF).
  - Another flag bit in the OUT endpoint DMA RLD interrupt flag register (USBODRIF).
  - A single enable bit in the OUT endpoint DMA interrupt enable register (USBODIE). This bit enables or disables both GO and RLD interrupt requests.
- Each IN endpoint has:
  - One flag bit in the IN endpoint DMA GO interrupt flag register (USBIDGIF)
  - Another flag bit in the IN endpoint DMA RLD interrupt flag register (USBIDRIF)
  - A single enable bit in the OUT endpoint DMA interrupt enable register (USBIDIE). This bit enables or disables both GO and RLD interrupt requests.
- For either type of endpoint, when either or both of the flag bits are set and the enable bit is set, an interrupt request is passed to the CPU.

Software must set the enable bit, but the flag bit is set by the DMA controller for a specific event. To see how the GO and RLD bits are used to control DMA activity, see section 14.4.3 on page 14-14.

Figure 14–15. Enable Paths for the USB DMA Interrupt Requests



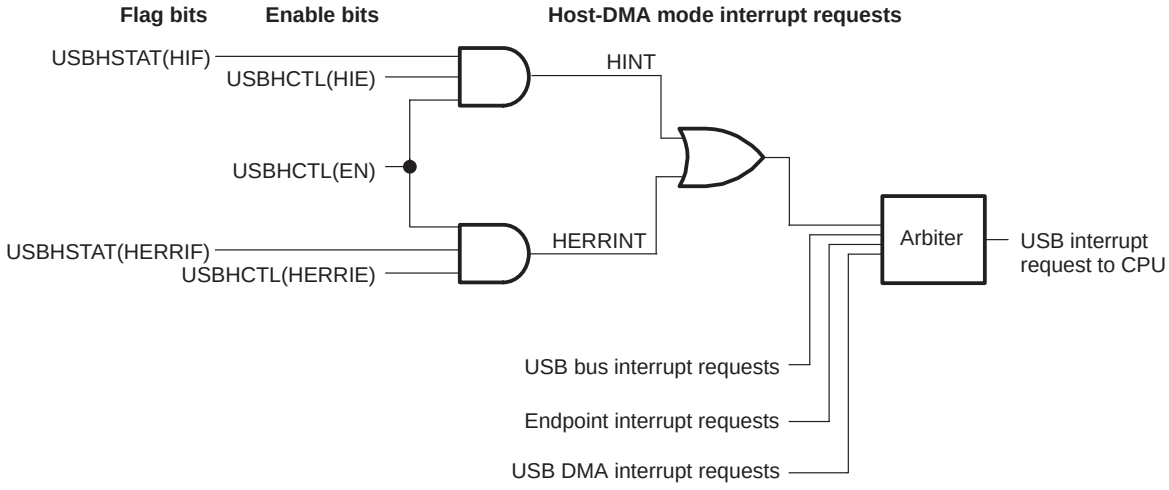
14.6.4 Host-DMA Mode Interrupt Requests

If the host-DMA mode (see section 14.5 on page 14-40) is enabled, a special state machine in the USB DMA controller can generate the following types of interrupts. The flag bits, enable bits, and interrupt requests are shown in Figure 14–16. Notice these interrupt requests are dependent on the EN bit in USBHCTL; when EN = 1, the host-DMA mode is enabled.

- **Host interrupt (HINT).** If the USB DMA controller completes a transfer between the USB host and the DSP memory (via an endpoint buffer) and the host requested an interrupt at the end of the transfer, the state machine sets the HIF flag bit in the host status register (USBHSTAT). If the HIE enable bit is 1 in the host control register (USBHCTL), an interrupt request is passed to the CPU.

- **Host error interrupt (HERRINT).** During an OUT transfer, if the size of the data transferred from the host does not match the size specified in the protocol header, the state machine sets the HERRIF flag bit in USBHSTAT. If the HERRIE enable bit is 1 in USBHCTL, an interrupt request goes to the CPU.

Figure 14–16. Enable Paths for the Host-DMA Mode Interrupt Requests



14.7 Power, Emulation, and Reset Considerations

This section is a summary of the effects of power control, emulation, and reset operations on the USB module.

14.7.1 Putting the USB Module into Its Idle Mode

The USB module is one of the peripheral devices in the PERIPH idle domain. For details on controlling the various idle domains of the DSP, see Chapter 8, *Idle Configurations*. If you want the USB module to become idle in response to an IDLE instruction, make the following preparations:

1) Write 1 to the idle enable (IDLEEN) bit in USBIDLECTL (see page 14-103). This tells the DSP to make the USB module idle when the PERIPH domain becomes idle.

2) Write a 1 to the PERI bit in ICR (see page 8.7). This tells the DSP to make the PERIPH domain idle in response to an IDLE instruction.

14.7.2 USB Module Indirectly Affected By Certain Idle Configurations

As mentioned in section 14.7.1, the USB module can be affected by any idle configuration that turns off the PERIPH idle domain. In addition, activity in the USB module can be affected by other idle configurations. For example:

- ❏ Idle configurations that turn off the CPU, preventing the CPU from controlling and monitoring USB activity. (If enabled, an interrupt from the USB module will wake the CPU.)
- ❏ Idle configurations that turn off the DSP DMA controller, preventing the USB module from accessing the DSP memory
- ❏ Idle configurations that turn off the EMIF, preventing the DSP DMA controller from accessing external memory

For more details about idle configurations, see Chapter 8.

14.7.3 USB Module During Emulation

During emulation, the USB module is not halted by a breakpoint. However, the CPU is halted and, therefore, unable to respond to USB interrupts or other requests.

In addition, the USB DMA controller cannot access memory if the DSP DMA controller is programmed to halt when a breakpoint is encountered in the debugger software. The FREE bit of DMA_GCR controls the emulation behavior of the DSP DMA controller. If FREE = 0 (the reset value), a breakpoint suspends DMA transfers. If FREE = 1, DMA transfers are not interrupted by a breakpoint. The DSP DMA controller is described in Chapter 3.

14.7.4 Resetting the USB Module

There are three ways to reset the USB module:

- Write 1 to the USB software reset bit (SOFTRST) in the USB global control register (USBGCTL). This resets the USB module but does not hold it in reset. Immediately after the reset operation, the USB module is free to run. The reset operation disconnects the USB module from the bus. **Note:** The reset triggered by setting the SOFTRST bit does not affect the USB control register (USBCTL).
- Write 0 to the USB reset bit (USBRST) in the USB idle control register (USBIDLECTL). This resets the USB module and holds it in reset until you write 1 to USBRST. During the reset operation, all of the USB module registers assume their power-on default values (shown in the register figures of section 14.8 on page 14-55). One important effect is that the USB module is disconnected from the USB (CONN = 0 in USBCTL).
- Initiate a DSP reset by driving the $\overline{\text{RESET}}$ pin low. The entire DSP is reset and is held in the reset state until you drive the pin high. When all DSP registers assume their reset values, the USBRST bit is forced to 0, which causes a USB reset.

14.8 USB Module Registers

This section cover the following topics

Topic	See ...
High-level summary of the USB registers	Section 14.8.1
DMA context registers	Section 14.8.2 on page 14-59
Descriptor registers for IN and OUT Endpoints 1–7	Section 14.8.3 on page 14-68
Descriptor registers for IN and OUT Endpoints 0	Section 14.8.4 on page 14-82
Interrupt registers	Section 14.8.5 on page 14-85
Host-DMA mode registers	Section 14.8.6 on page 14-93
General control and status registers	Section 14.8.7 on page 14-95

14.8.1 High-Level Summary of USB Module Registers

Table 14–9 lists the registers that are part of the USB module. These registers are in the I/O space of the C55x DSP. There are two additional registers that also reside in I/O space but are not part of the USB module:

- USBCLKMD, which controls the operation of the the dedicated USB clock generator (see section 14.2.3 on page 14-9)
- USBIDLECTL, which contains bits to put the USB module into its idle mode or into reset (see section 14.8.7.8 on page 14-103)

On each C55x DSP that contains a USB module, the set of USB module registers may start at a different base address, but the individual registers are at the same offset from the base address. To form a register's address, add the base address and the offset shown in the first column of Table 14–9. For example, the base address on a TMS320VC5509 DSP is 5800h, and the OUT endpoint 1 DMA context registers begin at I/O address 5808h.

Table 14–9. High-Level Summary of the USB Module Registers

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
DMA context registers (for details, see section 14.8.2 on page 14-59)			
Base address + 0000h			Reserved
Base address + 0008h	8	16	OUT endpoint 1 DMA context block
Base address + 0010h	8	16	OUT endpoint 2 DMA context block
Base address + 0018h	8	16	OUT endpoint 3 DMA context block
Base address + 0020h	8	16	OUT endpoint 4 DMA context block
Base address + 0028h	8	16	OUT endpoint 5 DMA context block
Base address + 0030h	8	16	OUT endpoint 6 DMA context block
Base address + 0038h	8	16	OUT endpoint 7 DMA context block
Base address + 0040h			Reserved
Base address + 0048h	8	16	IN endpoint 1 DMA context block
Base address + 0050h	8	16	IN endpoint 2 DMA context block
Base address + 0058h	8	16	IN endpoint 3 DMA context block
Base address + 0060h	8	16	IN endpoint 4 DMA context block
Base address + 0068h	8	16	IN endpoint 5 DMA context block
Base address + 0070h	8	16	IN endpoint 6 DMA context block
Base address + 0078h	8	16	IN endpoint 7 DMA context block

Table 14–9. High-Level Summary of the USB Module Registers (Continued)

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
USB buffer RAM			
Base address + 0080h	3584	8	Space for X and Y data buffers for OUT endpoints 1–7 and IN endpoints 1–7
Base address + 0E80h	64	8	OUT endpoint 0 buffer
Base address + 0EC0h	64	8	IN endpoint 0 buffer
Base address + 0F00h	8	8	Setup packet buffer
Descriptor registers (for details, see section 14.8.3 on page 14-68)			
Base address + 0F08h	8	8	OUT endpoint 1 descriptor block
Base address + 0F10h	8	8	OUT endpoint 2 descriptor block
Base address + 0F18h	8	8	OUT endpoint 3 descriptor block
Base address + 0F20h	8	8	OUT endpoint 4 descriptor block
Base address + 0F28h	8	8	OUT endpoint 5 descriptor block
Base address + 0F30h	8	8	OUT endpoint 6 descriptor block
Base address + 0F38h	8	8	OUT endpoint 7 descriptor block
Base address + 0F40h			Reserved
Base address + 0F48h	8	8	IN endpoint 1 descriptor block
Base address + 0F50h	8	8	IN endpoint 2 descriptor block
Base address + 0F58h	8	8	IN endpoint 3 descriptor block
Base address + 0F60h	8	8	IN endpoint 4 descriptor block
Base address + 0F68h	8	8	IN endpoint 5 descriptor block
Base address + 0F70h	8	8	IN endpoint 6 descriptor block
Base address + 0F78h	8	8	IN endpoint 7 descriptor block

Table 14–9. High-Level Summary of the USB Module Registers (Continued)

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Control and status registers (for details, see section 14.8.7 on page 14-95)			
Base address + 0F80h	1		IN endpoint 0 configuration register
Base address + 0F81h	1	8	IN endpoint 0 count register
Base address + 0F82h	1	8	OUT endpoint 0 configuration register
Base address + 0F83h	1	8	OUT endpoint 0 count register
Base address + 0F84h			Reserved
Base address + 0F91h	1	8	Global control register
Base address + 0F92h	1	8	Interrupt source register
Base address + 0F93h	1	8	Endpoint interrupt flag register for IN endpoints
Base address + 0F94h	1	8	Endpoint interrupt enable register for OUT endpoints
Base address + 0F95h	1	8	DMA RLD (reload) interrupt flag register for IN endpoints
Base address + 0F96h	1	8	DMA RLD interrupt flag register for OUT endpoints
Base address + 0F97h	1	8	DMA GO interrupt flag register for IN endpoints
Base address + 0F98h	1	8	DMA GO interrupt flag register for OUT endpoints
Base address + 0F99h	1	8	DMA interrupt enable register for IN endpoints
Base address + 0F9Ah	1	8	DMA interrupt enable register for OUT endpoints
Base address + 0F9Bh	1	8	Endpoint interrupt enable register for IN endpoints
Base address + 0F9Ch	1	8	Endpoint interrupt enable register for OUT endpoints
Base address + 0F9Dh			Reserved
Base address + 0FA0h	1	8	Host control register
Base address + 0FA1h	1	8	Host endpoint select register
Base address + 0FA2h	1	8	Host status register
Base address + 0FA3h			Reserved

Table 14–9. High-Level Summary of the USB Module Registers (Continued)

I/O Address (Word Address)	Number of Registers	Width (Bits)	Description
Base address + 0FF8h	1	8	Frame number register, low part
Base address + 0FF9h	1	8	Frame number register, high part
Base address + 0FFAh	1	8	Pre-SOF interrupt timer register
Base address + 0FFBh			Reserved
Base address + 0FFCh	1	8	USB control register
Base address + 0FFDh	1	8	USB interrupt enable register
Base address + 0FFEh	1	8	USB interrupt flag register
Base address + 0FFFh	1	8	USB device address register

14.8.2 DMA Context Registers

Each of the general-purpose endpoints (OUT endpoints 1–7 and IN endpoints 1–7) has a dedicated DMA channel and a dedicated block of DMA context registers for controlling and monitoring transfer activities in that channel. This section describes function of each of the context registers, which are summarized in Table 14–10.

For each endpoint, the block of DMA context registers starts at a different base address, but the individual registers are at the same offset from the base address. The first column of Table 14–10 shows the offsets.

Table 14–10. USB DMA Context Registers for Each OUT or IN Endpoint n (n = 1, 2, 3, 4, 5, 6, or 7)

Offset From Context Block's Base Address (Words)	USB DMA Context Register		
	OUT endpoint n	IN endpoint n	Description
0	USBODCTLn	USBIDCTLn	Control register
1	USBODSIZn	USBIDSIZn	Size register (transfer size in bytes)
2	USBODADLn	USBIDADLn	Address register, low part (byte address for a location in DSP memory)
3	USBODADHn	USBIDADHn	Address register, high part (byte address for a location in DSP memory)
4	USBODCTn	USBIDCTn	Count register (transfer count in bytes)
5	USBODRSZn	USBIDRSZn	Reload-size register (reload value for USBxDSIZn, x = O or I)
6	USBODRALn	USBIDRALn	Reload-address register, low part (reload value for USBxDADLn, x = O or I)
7	USBODRAHn	USBIDRAHn	Reload-address register, high part (reload value for USBxDADHn, x = O or I)

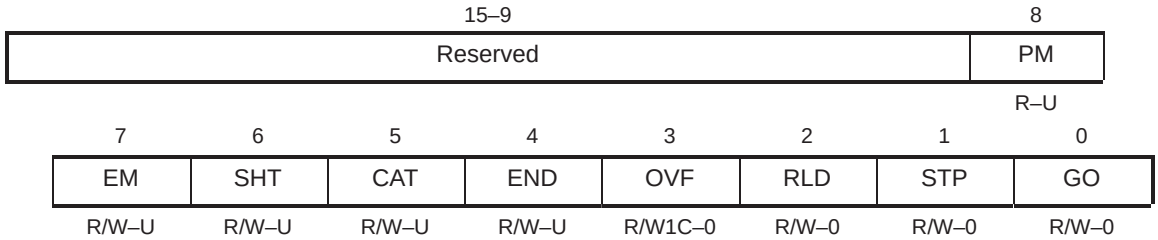
14.8.2.1 USB DMA Control Register (USBxDCTLn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

This register controls the operation of the endpoint DMA channel. The control bits in this register affect the DMA state changes described in section 14.4.9 on page 14-26.

The state of bits 4–6 is captured when a 1 is written to the GO bit and is not captured during the DMA transfer. When the DMA transfer completes, if the RLD bit is set, the state of bits 4–6 is captured again and a new transfer is started. If the RLD bit is set when a DMA transfer ends, the captured USBxDADLn, USBxDADHn, and USBxDSIZn values are swapped with the values stored in USBxDRALn, USBxDRAHn, and USBxDRSZn, respectively.

Figure 14–17. USB DMA Control Register (USBxDCTLn)

USBxDCTLn



Legend:

R Read-only access
R/W Read access/Write access
R/W1C Read access/Write 1 to clear this bit
–X X is the value after a DSP reset (X = U indicates that the DSP reset value is undefined).

Table 14–11. USBxDCTLn Bit Descriptions

Bit	*field*	*symval*	Value	Description
15–9	Reserved			These bits are not available for use.
8	PM			Previous packet missing. This status bit indicates that a packet did not occur during the previous frame. The software should consider this bit as a don't care when EM=0.
			0	A packet did occur on the previous frame for this endpoint.
			1	A packet did NOT occur on the previous frame for this endpoint.
7	EM			Error on missing packet. This control bit determines if, during an isochronous transfer, missing a packet during a frame should be considered an error condition.
			0	Missing packets are treated the same as zero-length packets.
			1	Missing packets will cause the GO bit to be cleared and the DMA to be halted. The error status will show in the PM bit. This event will only occur when PM goes from 0 to 1.

Table 14–11. USBxDCTLn Bit Descriptions (Continued)

Bit	*field*	*symval*	Value	Description
6	SHT			Short packet control. This bit only takes effect on a start or reload condition.
				For IN transfers:
			0	If the size of the last packet in the transfer matches the maximum packet size, do not insert an additional, 0-byte packet.
			1	If the size of the last packet in the transfer matches the maximum packet size, insert a zero-length (0-byte) packet to terminate the transaction with a short packet.
				For OUT transfers:
			0	If the size of the last packet in the transfer matches the maximum packet size, do not wait for a 0-byte packet to indicate the end of the transfer.
			1	If the size of the last packet in the transfer matches the maximum packet size, wait for an additional, 0-byte packet as an indication of the end of the transfer.
5	CAT			Concatenation control. This bit only takes effect on a start or reload condition.
				For IN transfers:
			0	If the transfer size is not enough to fill a maximum-size packet, allow the USB module to transfer a short packet.
			1	Concatenate DMA transfers. If the transfer size is not enough to fill a packet, then perform the next DMA transfer to fill the packet before allowing the USB module to send the data out.
				For OUT transfers:
			0	If the packet size exceeds the number of bytes remaining in the DMA transfer, record an overflow in the OVF bit.
			1	Concatenate DMA transfers. If the packet size exceeds the number of bytes remaining in the DMA transfer, do not record an overflow. Instead, record the current position in the buffer. When the next DMA transfer starts, read the rest of the data, beginning at the recorded position.

Table 14–11. USBxDCTLn Bit Descriptions (Continued)

Bit	*field*	*symval*	Value	Description
4	END			Endianness (byte orientation). This bit only takes effect on a start or reload condition.
			0	Little Endian (first byte is least significant byte in word)
			1	Big Endian (first byte is most significant byte in word)
3	OVF			Overflow/Underflow. For conditions that set this flag, see the state tables in section 14.4.9 (page 14-26).
				For isochronous IN transfers:
			0	Read – No underflow condition
			1	Read – Underflow condition Write – Write 1 to clear this flag.
				For isochronous and non-isochronous OUT transfers:
			0	Read – No overflow condition
			1	Read – Overflow condition Write – Write 1 to clear this flag.
2	RLD			Reload control. User writes a 1 to reload the address and size registers from reload registers when there are no pending transfers. The current address and size are automatically swapped with the reload address and size
			0	Do not use the reload registers.
			1	Reload and swap the address and size registers.
1	STP			Stop DMA transfer. This bit stops the DMA transfer on the next packet boundary. The data in the UBM Buffer is not affected.
			0	DMA functions normally
			1	DMA stops on the next packet boundary or at the end of the current DMA transfer (whichever occurs first), without clearing the UBM state or sending the current packet. At this time the GO and STP bits are reset.

Table 14–11. USBxDCTLn Bit Descriptions (Continued)

Bit	field	symval	Value	Description
0	GO			Start DMA transfer. When written with 1, this bit starts the DMA transfer for the endpoint. Writes of 0 have no effect. GO is cleared when a DMA transfer is no longer active.
			0	The USB DMA controller is idling (the controller is available for a new transfer).
			1	Read 1 – The USB DMA controller is performing a transfer. Write 1 – Start the endpoint DMA transfer (STP bit must be 0).

14.8.2.2 USB DMA Address Registers (USBxDADHn and USBxDADLn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

The USB DMA controller handles transfers between an endpoint buffer and the DSP memory. Because each of the USB DMA channels is dedicated to a particular endpoint, the start address for the endpoint buffer is known. Software must supply only a start address for the DSP memory.

The start address must be a **byte address**. Load the high 8 bits of the byte address to USBxDADHn and the low 16 bits to USBxDADLn. The DMA controller concatenates the two values to form a 24-bit address:

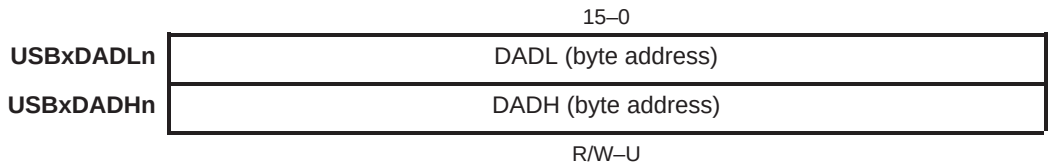
DSP memory address: DADH:DADL

In addition the address must be **16-bit aligned**. Make sure that the least significant bit (LSB) is 0.

The DMA starts an OUT transfer from (DADH:DADL) + 2 and continues to (DADH:DADL) + DSIZ + 2 or the transfer is otherwise terminated (for example, by a stop command via the STP bit or by a short OUT packet). The 16 bit word at (DADH:DADL) is then updated with the count of bytes actually transferred. The byte order of this word is architecture dependent and not dependent on the state of the END bit.

IN transfers start from (DADH:DADL) and continue to (DADH:DADL) + DSIZ.

Figure 14–18. USB DMA Address Registers (USBxDADLn and USBxDADHn)



Legend:

R/W Read/write access
–U The contents of the registers is undefined after a DSP reset.

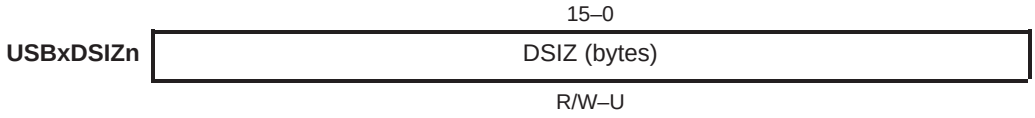
Table 14–12. USBxDADLn and USBxDADHn Bit Descriptions

Bit	*field*	*symval*	Value	Description
USBxDADLn(15–0)	DADL		0000h–FFFFh	Low part of the DSP memory start address. The start address must be 16-bit aligned; therefore, make sure bit 0 of this register is 0.
USBxDADHn(7–0)	DADH		0000h–00FFh	High part of the DSP memory start address. C55x DSP memory addresses have 24 bits; therefore, load the 8 high bits of DADH with 0s.

14.8.2.3 USB DMA Size Register (USBxDSIZn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

USBxDSIZn specifies the number of bytes for the DMA controller to transfer in a single DMA transfer. During a DMA reload operation (see section 14.4.5 on page 14-18), the content of USBxDSIZn is swapped with the content of USBxDRSZn (described on page 14-68).

Figure 14–19. DMA Size Register (USBxDSIZn)



Legend:

R/W Read/write access
–U The content of the register is undefined after a DSP reset.

Table 14–13. USBxDSIZn Bit Description

Bit	*field*	*symval*	Value	Description
15–0	DSIZ		1–65535	Number of bytes for the DMA controller to transfer

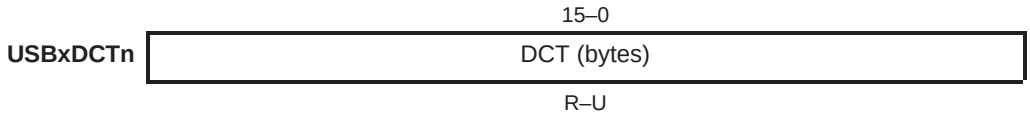
14.8.2.4 DMA Count Register (USBxDCTn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

USBxDCTn counts up, to track the number of bytes that have been transferred for OUT or IN endpoint n. The USB DMA controller automatically loads this register with 0 before beginning each DMA transfer. This includes the transfer that follows a DMA reload operation.

Note:

When the USB DMA controller stores data from an OUT transfer, it also stores USBxDCTn to the DSP memory (see section 14.4.6 on page 14-19).

Figure 14–20. DMA Count Register (USBxDCTn)



Legend:

R Read-only register
–U The content of the register is undefined after a DSP reset.

Table 14–14. USBxDCTn Bit Description

Bit	*field*	*symval*	Value	Description
15–0	DCT		0–65535	Indicates the number of bytes that have been transferred by the USB DMA controller **Note:** Before starting each new DMA transfer, the controller loads this register with 0.

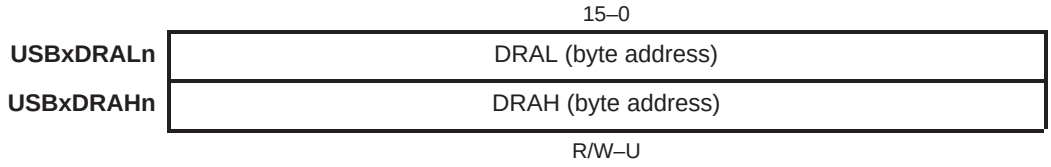
14.8.2.5 USB DMA Reload-Address Registers (USBxDRALn and USBxDRAHn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

USBxDRALn specifies the low 16 bits of the reload address, and USBxDRAHn specifies the high 8 bits of the reload address. If the RLD bit is set when the current DMA transfer is completed, then the contents of these reload registers are swapped with the contents of their corresponding primary registers:

- The content of USBxDRALn is swapped with the content of USBxDADLn.
- The content of USBxDRAHn is swapped with the content of USBxDADHn.

This register swapping is part of the DMA reload operation described in section 14.4.5 on page 14-18.

Figure 14–21. DMA Reload-Address Registers (USBxDRALn and USBxDRAHn)



Legend:

R/W Read/write access
–U The contents of the registers is undefined after a DSP reset.

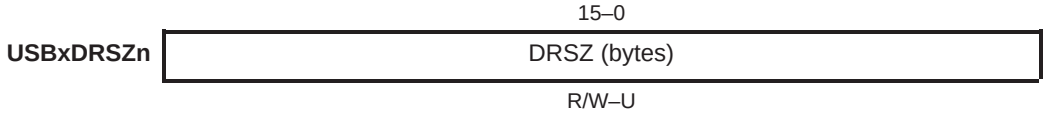
Table 14–15. USBxDRALn and USBxDRAHn Bit Descriptions

Bit	*field*	*symval*	Value	Description
USBxDRALn(15–0)	DRAL		0000h–FFFFh	Reload value for DADL The addresses used by the USB DMA controller must be 16-bit aligned; therefore, make sure bit 0 of this register is 0.
USBxDRAHn(15–0)	DRAH		0000h–00FFh	Reload value for DADH C55x DSP memory addresses have 24 bits; therefore, load the 8 high bits of DRAH with 0s.

14.8.2.6 DMA Reload-Size Register (USBxDRSZn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

Specifies the reload size. If the RLD bit is set when the current DMA transfer is completed, the content of USBxDRSZn is swapped with the content of USBxDSIZn. This register swapping is part of the DMA reload operation described in section 14.4.5 on page 14-18.

Figure 14–22. DMA Reload-Size Register (USBxDRSZn)



Legend:

R/W Read/write access
–U The content of the register is undefined after a DSP reset.

Table 14–16. USBxDRSZn Bit Description

Bit	*field*	*symval*	Value	Description
15–0	DRSZ		1–65535	Reload value for DSIZ

14.8.3 Descriptor Registers for IN and OUT Endpoints 1–7

The general-purpose endpoints (IN endpoints 1–7 and OUT endpoints 1–7) each have a block of eight descriptor registers to define the endpoint characteristics for the UBM. Table 14–17 shows the descriptor registers available for an OUT endpoint and for an IN endpoint. To access a descriptor register, find the base address of the descriptor block and add the offset shown in the first column of Table 14–17. The actual addresses can be found in the data sheet of the respective C55x DSP.

Table 14–17. Descriptor Registers for Each OUT or IN Endpoint n (n = 1, 2, 3, 4, 5, 6, or 7)

Offset From the Descriptor Block's Base Address (Words)	Endpoint Descriptor Register		
	OUT Endpoint n	IN Endpoint n	Description
0	USBOCNFn	USBICNFn	Endpoint n configuration register
1	USBOBAXn	USBIBAXn	X-buffer base address register (bits 11–4 of a byte address)

Table 14–17. Descriptor Registers for Each OUT or IN Endpoint n (n = 1, 2, 3, 4, 5, 6, or 7) (Continued)

Offset From the Descriptor Block's Base Address (Words)	Endpoint Descriptor Register		
	OUT Endpoint n	IN Endpoint n	Description
2	USBOCTXn	USBICTXn	X-buffer count register (transfer count in bytes)
3	USBOCTXHn	USBISIZHn	**OUT endpoint:** X-buffer count extension register (used for isochronous transfers only) **IN endpoint:** X-/Y-buffer size extension register (used for isochronous transfers only)
4	USBOSIZn	USBISIZn	X-/Y-buffer size register (transfer size in bytes)
5	USBOBAYn	USBIBAYn	Y-buffer base address register (bits 11–4 of a byte address)
6	USBOCTYn	USBICTYn	Y-buffer count register (transfer count in bytes)
7	USBOCTYHn	Reserved	**OUT endpoint:** Y-buffer count extension register (used for isochronous transfers only) **IN endpoint:** Not available for use

14.8.3.1 IN Endpoint n Configuration Register (USBICNFn) (n = 1, 2, 3, 4, 5, 6, or 7)

As shown in Figure 14–23, the function of bits 5–0 of USBICNFn depend on whether you have programmed the endpoint for non-isochronous transfers or for isochronous transfers. Table 14–18 describes the bits of USBICNFn, taking into account the optional function of bits 5–0.

Figure 14–23. IN Endpoint n Configuration Register (USBICNFn)

USBICNFn in Non-Isochronous Mode (ISO = 0)

7	6	5	4	3	2–0
UBME	ISO = 0	TOGGLE	DBUF	STALL	Reserved
R/W–U	R/W–U	R/W–U	R/W–U	R/W–U	

USBICNFn in Isochronous Mode (ISO = 1)

7	6	5–3	2–0
UBME	ISO = 1	CTXH	CTYH
R/W–U	R/W–U	R/W–U	R/W–U

Legend:

R/W Read access/Write access
–U The content of these bits is undefined after a DSP reset.

Table 14–18. USBICNFn Bit Descriptions

Bit	f*ield*	*symval*	Value	Description
7	UBME			UBM access enable
			0	The UBM cannot access this endpoint (the endpoint is inactive).
			1	The UBM can access this endpoint (the endpoint is active).
6	ISO			Isochronous mode enable
			0	Non-isochronous mode
			1	Isochronous mode

Table 14–18. USBICNFn Bit Descriptions (Continued)

Bit	f*ield*	*symval*	Value	Description
Bits 5–0 in Non-Isochronous Mode (ISO = 0)				
5	TOGGLE			Endpoint data toggle. This bit reflects the data toggle sequence (see section 14.1.2 on page 14-3). **Note:** You do not need to write to this bit; it is maintained by the UBM.
			0	The next data packet is DATA0.
			1	The next data packet is DATA1.
4	DBUF			Double buffer mode enable **Note:** The USB DMA controller requires the double buffer mode. If the DMA controller will be servicing the endpoint, make sure DBUF = 1 before you start the controller.
			0	Single buffer used (X buffer only)
			1	Double buffer mode. The USB DMA controller tracks the data toggle sequence to determine the active buffer. For a DATA0 packet, the controller uses the X buffer; for a DATA1 packet, the controller uses the Y buffer.
3	STALL			Endpoint stall. Set this bit to tell the USB host that the endpoint is stalled.
			0	No stall
			1	Endpoint stalled. A STALL handshake will be initiated in response to host access requests until the STALL bit is cleared.
2–0	Reserved			Write 0s to these bits.
Bits 5–0 in Isochronous Mode (ISO = 1)				
5–3	CTXH		000b–111b	IN endpoint X-buffer byte count – high bits.
2–0	CTYH		000b–111b	IN endpoint Y-buffer byte count – high bits.

14.8.3.2 OUT Endpoint n Configuration Register (USBOCNFn) (n = 1, 2, 3, 4, 5, 6, or 7)

As shown in Figure 14–24, the function of bits 5–0 of USBOCNFn depend on whether you have programmed the endpoint for non-isochronous transfers or for isochronous transfers. Table 14–19 describes the bits of USBOCNFn, taking into account the optional function of bits 5–0.

Figure 14–24. OUT Endpoint n Configuration Register (USBOCNFn)

USBOCNFn in Non-Isochronous Mode (ISO = 0)

7	6	5	4	3	2–0
UBME	ISO = 0	TOGGLE	DBUF	STALL	Reserved
R/W–U	R/W–U	R/W–U	R/W–U	R/W–U	

USBOCNFn in Isochronous Mode (ISO = 1)

7	6	5–3	2–0
UBME	ISO = 1	Reserved	SIZH
R/W–U	R/W–U	R/W–U	R/W–U

Legend:

R/W Read access/Write access
– U The content of these bits is undefined after a DSP reset.

Table 14–19. USBOCNFn Bit Descriptions

Bit	*field*	*symval*	Value	Description
7	UBME			UBM access enable
			0	The UBM cannot access this endpoint (the endpoint is inactive).
			1	The UBM can access this endpoint (the endpoint is active).
6	ISO			Isochronous mode enable
			0	Non-isochronous mode
			1	Isochronous mode

Table 14–19. USBOCNFn Bit Descriptions (Continued)

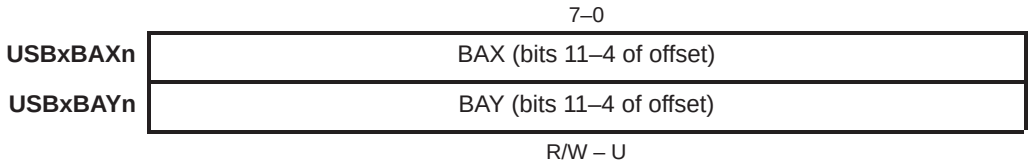
Bit	*field*	*symval*	Value	Description
Bits 5–0 in Non-Isochronous Mode (ISO = 0)				
5	TOGGLE			Endpoint data toggle. This bit reflects the data toggle sequence (see section 14.1.2 on page 14-3). **Note:** You do not need to write to this bit; it is maintained by the UBM.
			0	The next data packet is DATA0.
			1	The next data packet is DATA1.
4	DBUF			Double buffer mode enable **Note:** The USB DMA controller requires the double buffer mode. If the DMA controller will be servicing the endpoint, make sure DBUF = 1 before you start the controller.
			0	Single buffer used (X buffer only)
			1	Double buffer mode. The USB DMA controller tracks the data toggle sequence to determine the active buffer. For a DATA0 packet, the controller uses the X buffer; for a DATA1 packet, the controller uses the Y buffer.
3	STALL			Endpoint stall. Set this bit to tell the USB host that the endpoint is stalled.
			0	No stall
			1	Endpoint stalled. A STALL handshake will be initiated in response to host access requests until the STALL bit is cleared.
2–0	Reserved			These bits are not available for use.
Bits 5–0 in Isochronous Mode (ISO = 1)				
5–3	Reserved			Write 0s to these bits.
2–0	SIZH		000h–111b	OUT endpoint X-/Y-buffer size – high bits

14.8.3.3 Endpoint n Buffer Base Address Registers (USBxBAXn, USBxBAYn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

Each general-purpose endpoint has two buffer base address registers: one for its X buffer and one for its Y buffer (see Figure 14–25 and Table 14–20). By writing to one of these registers, you provide bits 11–4 of a 12-bit relative address. The USB module adds 0s for the bits 3–0 of the relative address. The address is relative to the start address of the USB module registers.

Consider Example 14–2, which follows Table 14–20. Rather than the absolute address, you load the offset shifted right by 4 bits. The 4-bit shift is required because the buffer base address registers must hold the 8 high bits. When the USB module uses those 8 bits, it extends them with four least significant 0s.

Figure 14–25. Endpoint Buffer Base Address Registers



Legend:

R/W Read/write access
– U The contents of the registers is undefined after a DSP reset.

Table 14–20. Endpoint Buffer Base Address Register Bit Descriptions

Bit	*field*	*symval*	Value	Description
For IN endpoint n:				
USBIBAXn(7–0)	BAX		00h–FFh	Bits 11–4 of the X-buffer base address for IN endpoint n. BIts 3–0 are 0s.
USBIBAYn(7–0)	BAY		00h–FFh	Bits 11–4 of the Y-buffer base address for IN endpoint n. Bits 3–0 are 0s.
For OUT endpoint n:				
USBOBAXn(7–0)	BAX		00h–FFh	Bits 11–4 of the X-buffer base address for OUT endpoint n. Bits 3–0 are 0s.
USBOBAYn(7–0)	BAY		00h–FFh	Bits 11–4 of the Y-buffer base address for OUT endpoint n. Bits 3–0 are 0s.

Example 14–2. Loading the Endpoint Buffer Base Addresses

IN endpoint 1, X buffer: Assigned to the 1st 64 bytes of the buffer RAM
IN endpoint 1, Y buffer: Assigned to the 2nd 64 bytes of the buffer RAM

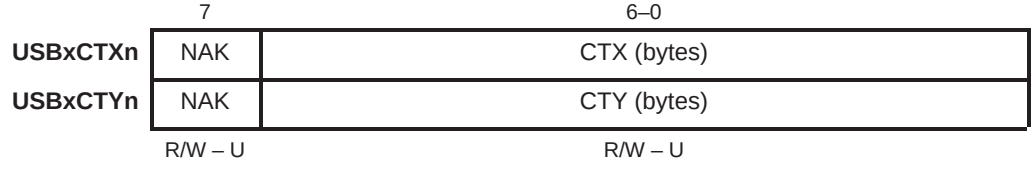
Buffer	I/O Address Seen by CPU	Value Loaded into Buffer Base Address Register
X	USB module registers base address + Offset for top of buffer RAM (80h)	USBIBAX1 = (80 >>4)
Y	USB module registers base address + Offset for 64 bytes further (C0h)	USBIBAY1 = (C0 >>4)

14.8.3.4 Endpoint n Count Registers (USBxCTXn, USBxCTYn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

Each general-purpose endpoint has two count registers (see Figure 14–26 and Table 14–21): one for its X buffer and one for its Y buffer. The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. While the NAK bit is set (NAK = 1), the UBM sends a NAK in response to host data requests at the endpoint. For more details about the role of the NAK bit, see section 14.3 on page 14-11.

Each buffer (X or Y) needs a count register to determine how many bytes the UBM should move out of the buffer (for an IN endpoint) or to record how many bytes the UBM has moved into the buffer (for an OUT endpoint). If the endpoint is in the non-isochronous mode (ISO = 0), the CTX/CTY field is the full count register. If the endpoint is in the isochronous mode (ISO = 1), the CTX/CTY field is the 7 low bits of a 10-bit count register. The 3 high bits come from another register, as described in Table 14–21.

Figure 14–26. Endpoint n Count Registers



Legend:

R/W Read/write access
– U The contents of these bits are undefined after a DSP reset.

Table 14–21. Endpoint Count n Register Bit Descriptions

Bit	*field*	*symval*	Value	Description
7	NAK			Negative acknowledgement
				For IN endpoint n:
			0	Data in the endpoint buffer is ready for an IN transfer.
			1	Data in the endpoint buffer is not ready. The UBM will send a NAK in response to an IN token.
				For OUT endpoint n:
			0	The endpoint buffer is ready for an OUT transfer.
			1	Either the endpoint buffer is not ready or it contains a valid data packet from the previous transfer. The UBM will send a NAK in response to an OUT token.
6–0	CTX/CTY			Count bits for buffer b (b = X or Y).
				In the non-isochronous mode:
			0–64	**For IN endpoint n:** When NAK = 0, this value indicates the number of bytes the UBM should move out of buffer b in response to an IN token. **For OUT endpoint n:** This value is a running count of the bytes the UBM has stored to buffer b. **Note:** If CTb holds a value greater than 64, the results are unpredictable.
				In the isochronous mode:
			000 0000b–111 1111b	**For IN endpoint n:** These bits are the 7 low bits of the 10-bit byte count for buffer b. As shown in Figure 14–27 (page 14-77), the 3 high bits are taken from USBICNFn. Load into these 10 bits the number of bytes you want the UBM to read from buffer b in response to an IN token. When an IN token arrives and NAK = 0, the UBM reads that many bytes, passing them to the serial interface engine (SIE) for the host. **For OUT endpoint n:** These bits are the 7 low bits of the 10-bit byte counter for buffer b. As shown in Figure 14–28 (page 14-78), the 3 high bits are taken from USBOCTXHn/USBOCTYHn. The 10-bit counter keeps a running count of the bytes the UBM has written to buffer b.

Figure 14–27. IN Endpoint n Extended Count Values in the Isochronous Mode (ISO = 1)

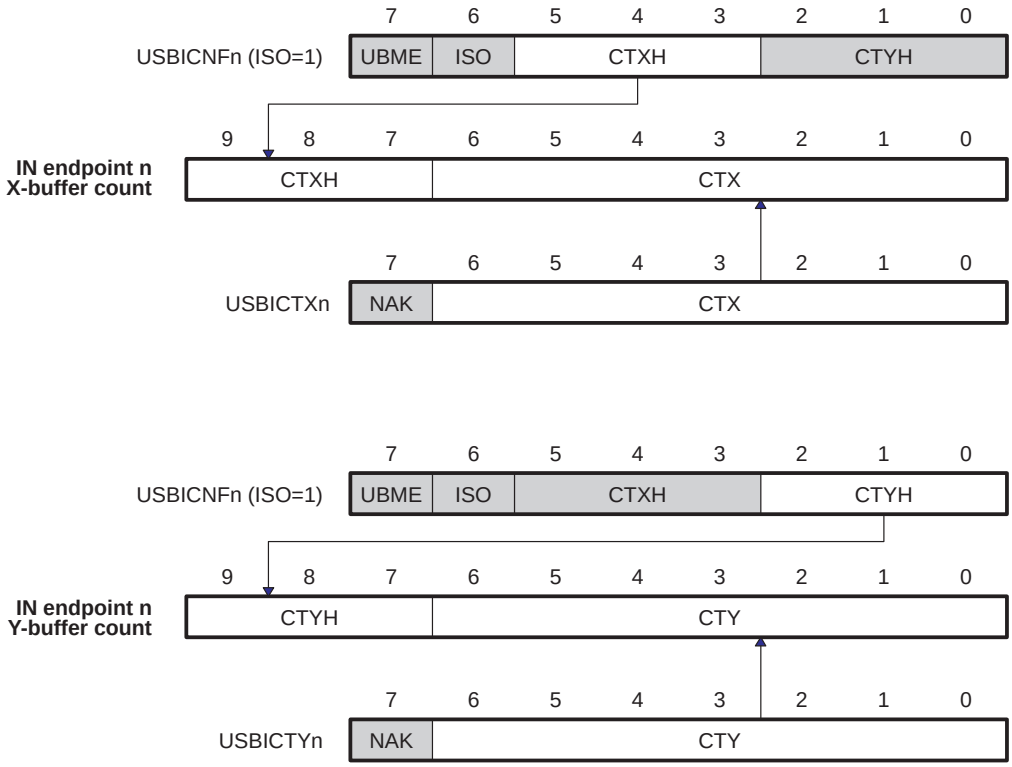
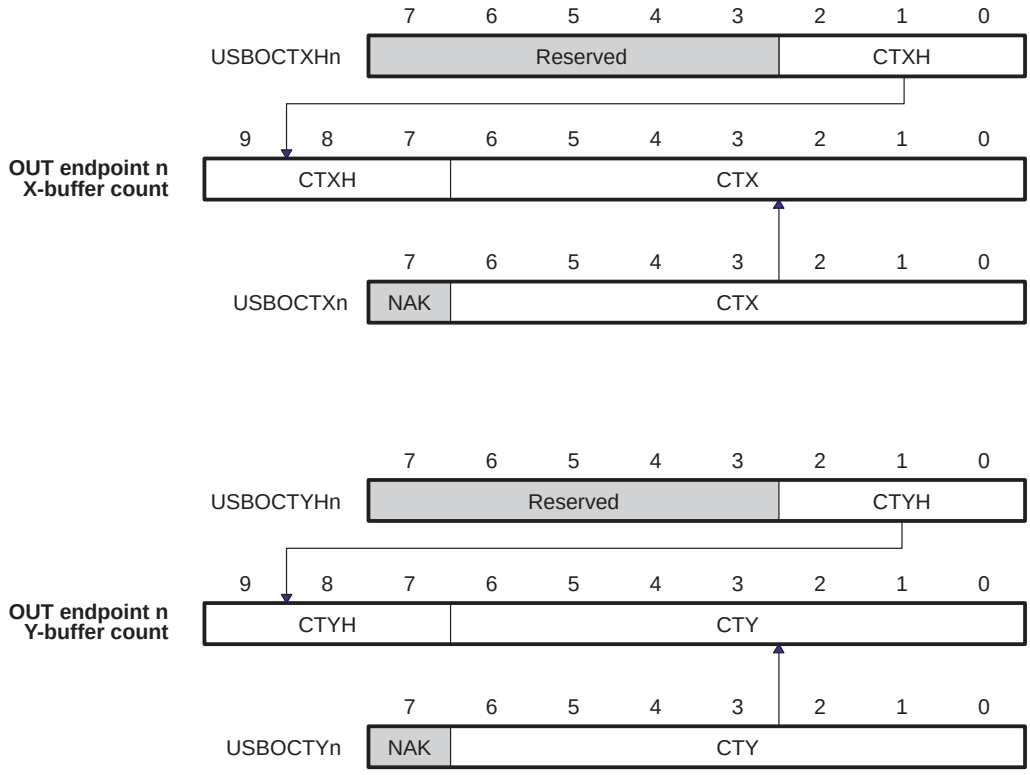


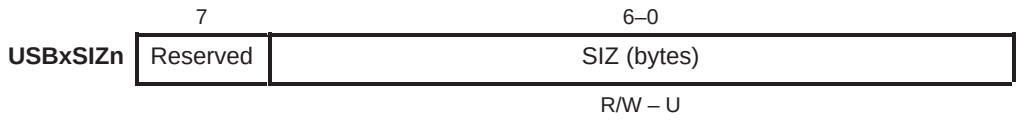
Figure 14–28. OUT Endpoint n Extended Count Values in the Isochronous Mode (ISO = 1)



14.8.3.5 Endpoint n X-/Y-Buffer Size Register (USBxSIZn) (x = O or I; n = 1, 2, 3, 4, 5, 6, or 7)

The SIZ field in USBxSIZn determines the maximum packet size: the number of bytes the endpoint buffer can hold at one time. In the double buffer mode (required for the USB DMA controller), the X buffer and the Y buffer are of equal size, and SIZ defines that size.

Figure 14–29. Endpoint n X-/Y-Buffer Size Register (USBxSIZn)



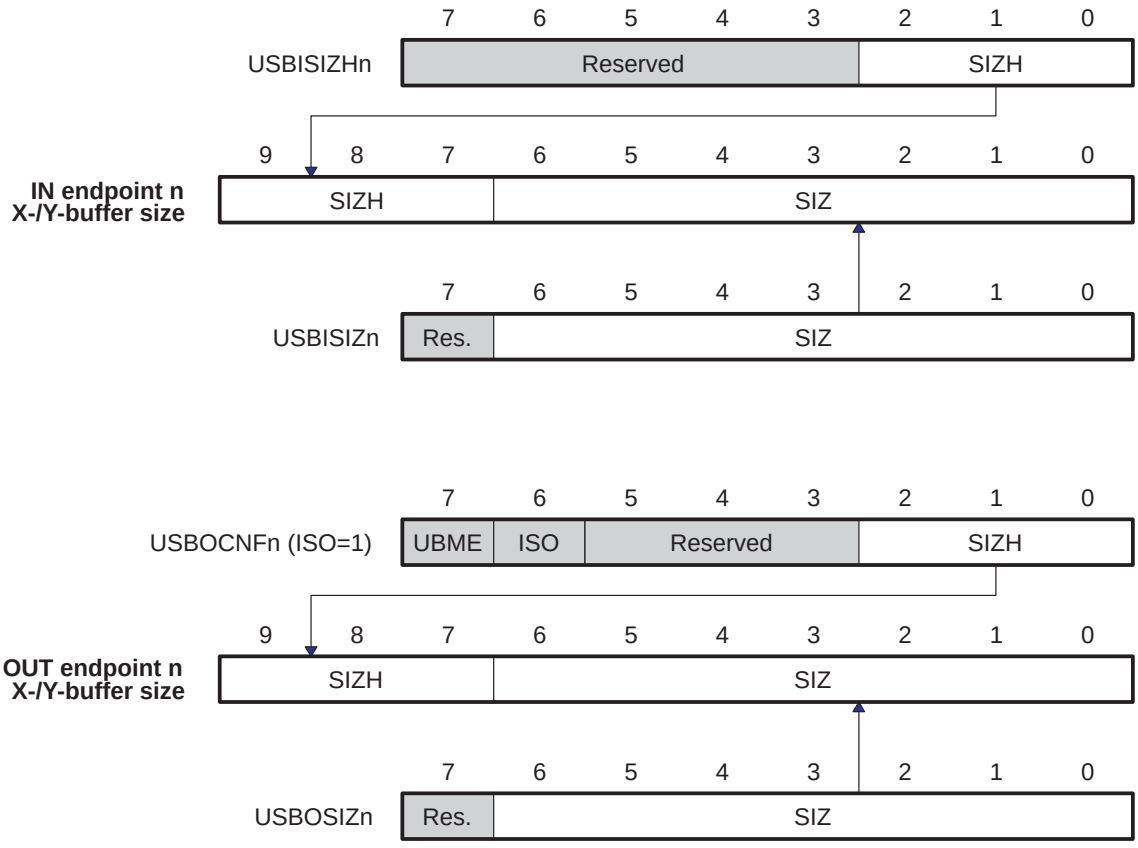
Legend:

R/W Read/write access
– U The contents of these bits are undefined after a DSP reset.

Table 14–22. Endpoint Size Register Bit Descriptions

Bit	***field***	*symval*	Value	**Description**
7	Reserved			This bit is not available for use.
6–0	SIZ			X-/Y-buffer size
				In the non-isochronous mode:
			8, 16, 32, or 64	The number of bytes in the buffer RAM allocated for the X buffer. In the double buffer mode (DBUF = 1), the same number of bytes is allocated for the Y buffer. **Note:** If SIZ holds a value greater than 64, the results are unpredictable.
				In the isochronous mode:
			000 0001b–111 1111b	The 7 low bits of the 10-bit buffer size. As shown in Figure 14–30 (page 14-80), the 3 high bits are taken from USBISIZHn for an IN endpoint or from USBOCNFn for an OUT endpoint. The 10-bit buffer size is used for the X buffer and, in the double buffer mode, is also used for the Y buffer.

Figure 14–30. Extended Size Values in the Isochronous Mode (ISO = 1)



14.8.3.6 Endpoint n Buffer Size and Count Extension Registers: USBISIZHn, USBICTXHn, USBOCTYHn (n = 1, 2, 3, 4, 5, 6, or 7)

These registers provide buffer size and count extensions for isochronous transfers between the USB module and a host processor. Specifically:

- If IN endpoint n is using the isochronous mode (ISO = 1 in USBICNFn), USBISIZHn supplies 3 high bits to extend the X-/Y-buffer size from a 7-bit value (SIZ) to a 10-bit value (SIZH:SIZ). The formation of this extended size value is shown in Figure 14–30 on page 14-80.

- If OUT endpoint n is using the isochronous mode (ISO = 1 in USBOCNFn):

  - USBOCTXHn supplies 3 high bits to extend the X-buffer byte count from a 7-bit value (CTX) to a 10-bit value (CTXH:CTX).

  - USBOCTYHn supplies 3 high bits to extend the Y-buffer byte count from a 7-bit value (CTY) to a 10-bit value (CTYH:CTY).

  The formation of these extended count values is shown in Figure 14–28 on page 14-78.

Figure 14–31. Endpoint Buffer Size and Count Extension Registers

USBISIZHn

7–3	2–0
Reserved	SIZH
	R/W – U

USBOCTXHn

7–3	2–0
Reserved	CTXH
	R/W – U

USBOCTYHn

7–3	2–0
Reserved	CTYH
	R/W – U

Legend:

R/W Read access/Write access
– U The content of these bits is undefined after a DSP reset.

Table 14–23. Endpoint Size and Count Extension Register Bit Descriptions

Bit	*field*	*symval*	Value	Description
USBISIZHn(2–0)	SIZH		000h–111h	In the isochronous mode (ISO = 1), these bits are the 3 high bits of the IN endpoint X-/Y-buffer byte count. The 7 low bits are in the SIZ bits of USBISIZn (see section 14.8.3.5 on page 14-78).
USBOCTXHn(2–0)	CTXH		000h–111h	In the isochronous mode (ISO = 1), these bits are the 3 high bits of the OUT endpoint X-buffer byte count. The 7 low bits are in the CTX bits of USBOCTXn (see section 14.8.3.4 on page 14-75).
USBOCTYHn(2–0)	CTYH		000h–111h	In the isochronous mode (ISO = 1), these bits are the 3 high bits of the OUT endpoint Y-buffer byte count. The 7 low bits are in the CTY bits of USBOCTYn (see section 14.8.3.4 on page 14-75).

14.8.4 Descriptor Registers for IN and OUT Endpoints 0

The control endpoints (IN endpoint 0 and OUT endpoint 0) each have two descriptor registers to define them: a configuration register and a count register, which are described here.

14.8.4.1 Endpoint 0 Configuration Register (USBxCNF0) (x = O or I)

IN endpoint 0 and OUT endpoint 0 each have a configuration register of the form shown in Figure 14–32. Table 14–24 describes the bit fields of this regiser.

Figure 14–32. Endpoint 0 Configuration Register (USBxCNF0)

USBxCNF0

7	6	5	4	3	2–0
UBME	Reserved	TOGGLE	Reserved	STALL	Reserved
R/W – 0		R – 0		R/W – 0	

Legend:

R Read-only access
R/W Read/write access
– 0 A DSP reset forces these bits to 0.

Table 14–24. USBxCNF0 Bit Descriptions

Bit	*field*	*symval*	Value	Description
7	UBME			UBM access enable
			0	The UBM cannot access this endpoint (the endpoint is inactive).
			1	The UBM can access this endpoint (the endpoint is active).
6	Reserved			This bit is not available for use.
5	TOGGLE			Endpoint data toggle. This bit reflects the data toggle sequence (see section 14.1.2 on page 14-3).
			0	The next data packet is DATA0.
			1	The next data packet is DATA1.
4	Reserved			This bit is not available for use.
3	STALL			Endpoint stall. Set this bit to tell the USB host that the endpoint is stalled.
			0	No stall
			1	Endpoint stalled. A STALL handshake will be initiated in response to host access requests until the STALL bit is cleared. The STALL bit is automatically cleared when the next setup packet arrives.
2–0	Reserved			These bits are not available for use.

14.8.4.2 Endpoint 0 Count Register (USBxCT0) (x = O or I)

IN endpoint 0 and OUT endpoint 0 each has one count register. As shown in Figure 14–33 and Figure 14–34, the two count registers have the same form but different reset values for their NAK bits and different accessibility for their CT0 (count) fields.

Table 14–25 describes the bit fields of an endpoint 0 count register. The NAK bit corresponds to the negative acknowledgement (NAK) of the USB protocol. While the NAK bit is set (NAK = 1), the UBM sends a NAK in response to host requests at the endpoint. For more details about the role of the NAK bit, see section 14.3 on page 14-11. The CT0 field determines how many bytes the UBM should move out of the endpoint buffer (for IN endpoint 0) or records how many bytes the UBM has moved into the endpoint buffer (for OUT endpoint 0).

Figure 14–33. Count Register for IN Endpoint 0 (USBICT0)

Legend:

R/W Read/write access
– X X is the value after a DSP reset.

Figure 14–34. Count Register for OUT Endpoint 0 (USBOCT0)

Legend:

R Read-only access
R/W Read/write access
– X X is the value after a DSP reset.

Table 14–25. USBxCT0 Bit Descriptions

Bit	***field***	*symval*	Value	**Description**
7	NAK			Negative acknowledgement
				For IN endpoint 0:
			0	Data in the endpoint buffer is ready for an IN transfer.
			1	Data in the endpoint buffer is not ready. The UBM will send a NAK in response to an IN token.
				For OUT endpoint 0:
			0	The endpoint buffer is ready for an OUT transfer.
			1	Either the endpoint buffer is not ready or it contains a valid data packet from the previous transfer. The UBM will send a NAK in response to an OUT token.

Table 14–25. USBxCT0 Bit Descriptions (Continued)

Bit	*field*	*symval*	Value	Description
6–0	CT0			Count bits
			0–64	**For IN endpoint 0:** When NAK = 0, this value indicates the number of bytes the UBM should move out of the endpoint buffer in response to an IN token. **For OUT endpoint 0:** This value is a running count of the bytes the UBM has stored to the endpoint buffer. **Note:** If CT0 holds a value greater than 64, the results are unpredictable.

14.8.5 Interrupt Registers

This section describes registers that identify the current USB interrupt source (USBINTSRC), hold flags for interrupt events (USBxEPIF, USBxDGIF, and USBxDRIF), enable/disable interrupt requests (USBxEPIE and USBxDIE).

14.8.5.1 Interrupt Source Register (USBINTSRC)

All interrupt requests generated in the USB module are multiplexed through an arbiter to a single USB interrupt request for the CPU. The interrupt service routine can determine the interrupt source by reading the interrupt source register (USBINTSRC). Then the ISR can branch to the appropriate subroutine.

When the CPU reads the INTSRC field (see Figure 14–35), it obtains a 7-bit interrupt source code. Table 14–27 shows the valid INTSRC codes and the corresponding interrupt sources.

When the interrupt arbiter receives multiple interrupt requests at the same time, it services them one at a time according to a predefined priority ranking. The INTSRC value also identifies the priority of an interrupt source (02h is highest, 52h is lowest).

For more details about the interrupt sources and how USBINTSRC is used, see section 14.6 on page 14-46.

Figure 14–35. Interrupt Source Register (USBINTSRC)

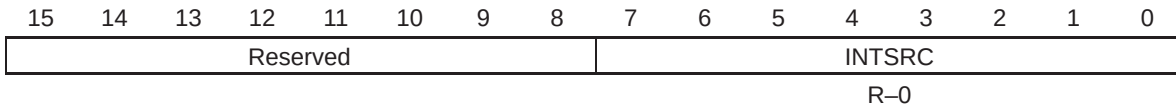


Table 14–26. Interrupt Source Register (USBINTSRC) Field Values

Bit	*field*	*symval*	Value	Function
15–8	Reserved			These bits are not available for use.
7–0	INTSRC		00h–52h	These 8 bits indicate the interrupt source, the event that caused an interrupt to the CPU. In addition, these bits indicate the priority of each interrupt source. The interrupt source corresponding to INTSRC = 02h has the highest priority, and the interrupt source corresponding to INTSRC = 52h has the lowest priority.

Table 14–27. Interrupt Sources Matched to INTSRC Values

INTSRC Value/ Priority	Interrupt Source	INTSRC Value/ Priority	Interrupt Source
00h	(no interrupt)		
02h	OUT endpoint 0	32h	OUT endpoint 1 DMA Reload
		33h	OUT endpoint 1 DMA Go
04h	IN endpoint 0	34h	OUT endpoint 2 DMA Reload
		35h	OUT endpoint 2 DMA Go
06h	RSTR interrupt	36h	OUT endpoint 3 DMA Reload
		37h	OUT endpoint 3 DMA Go
08h	SUSR interrupt	38h	OUT endpoint 4 DMA Reload
		39h	OUT endpoint 4 DMA Go
0Ah	RESR interrupt	3Ah	OUT endpoint 5 DMA Reload
		3Bh	OUT endpoint 5 DMA Go
0Ch	SETUP packet received	3Ch	OUT endpoint 6 DMA Reload
		3Dh	OUT endpoint 6 DMA Go
0Eh	STPOW packet received	3Eh	OUT endpoint 7 DMA Reload
		3Fh	OUT endpoint 7 DMA Go
10h	SOF		
11h	PSOF		
12h	OUT endpoint 1	42h	IN endpoint 1 DMA Reload

Table 14–27. Interrupt Sources Matched to INTSRC Values (Continued)

INTSRC Value/ Priority	Interrupt Source	INTSRC Value/ Priority	Interrupt Source
14h	OUT endpoint 2	43h	IN endpoint 1 DMA Go
16h	OUT endpoint 3	44h	IN endpoint 2 DMA Reload
18h	OUT endpoint 4	45h	IN endpoint 2 DMA Go
1Ah	OUT endpoint 5	46h	IN endpoint 3 DMA Reload
1Ch	OUT endpoint 6	47h	IN endpoint 3 DMA Go
1Eh	OUT endpoint 7	48h	IN endpoint 4 DMA Reload
		49h	IN endpoint 4 DMA Go
22h	IN endpoint 1	4Ah	IN endpoint 5 DMA Reload
24h	IN endpoint 2	4Bh	IN endpoint 5 DMA Go
26h	IN endpoint 3	4Ch	IN endpoint 6 DMA Reload
28h	IN endpoint 4	4Dh	IN endpoint 6 DMA Go
2Ah	IN endpoint 5	4Eh	IN endpoint 7 DMA Reload
2Ch	IN endpoint 6	4Fh	IN endpoint 7 DMA Go
2Eh	IN endpoint 7	50h	Host interrupt
		52h	Host error

14.8.5.2 Endpoint Interrupt Flag Register (USBxEPIF) (x = O or I)

For each endpoint, the USB module sets an interrupt flag in USBxEPIF every time the UBM moves data in/out of the endpoint's buffer. For example, if the UBM has finished moving data into the buffer for OUT endpoint 2, the OE2 flag is set in USBOEPIF. In addition to setting a flag, the USB module can generate an endpoint interrupt request. For more details about endpoint interrupt requests, see section 14.6.2 on page 14-48.

Figure 14–36. Endpoint Interrupt Flag Register for OUT Endpoints (USBOEPIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	OE0
R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0

Figure 14–37. Endpoint Interrupt Flag Register for IN Endpoints (USBIEPIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0

Table 14–28. Endpoint Interrupt Flag Register (USBxEPIF) Field Values

Bit	*field*	*symval*	Value	Function
15–8	Reserved		0	These bits are not available for use.
7–1	xE[7:0]			x = O (for OUT) or I (for IN).
			0	No interrupt pending
			1	Indicates that the corresponding endpoint generated an interrupt. It is set by the hardware and cleared either when the CPU reads the interrupt source register (USBINTSRC) with INTSRC equal to the corresponding interrupt or when the CPU writes a 1 to this bit.

14.8.5.3 Endpoint Interrupt Enable Register (USBxEPIE) (x = O or I)

If a bit gets set in the endpoint interrupt flag register (USBxEPIF) and the corresponding bit is set in USBxEPIE, an endpoint interrupt request is generated. For example, if the IE2 bit of USBIEPIF gets set and the IE2 bit of USBIEPIE is 1, an IN endpoint 2 interrupt request is generated. For more details about endpoint interrupt requests, see section 14.6.2 on page 14-48.

Figure 14–38. Endpoint Interrupt Enable Register for OUT Endpoints (USBOEPIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	OE0
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

Figure 14–39. Endpoint Interrupt Enable Register for IN Endpoints (USBIEPIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

Table 14–29. Endpoint Interrupt Enable Register (USBxEPIE) Field Values

Bit	*field*	*symval*	Value	Function
15–8	Reserved		0	These bits are not available for use.
7–1	xE[7:0]			Endpoint interrupt enable. x = O (for OUT) or I (for IN).
			0	Endpoint interrupt requests are disabled for the endpoint.
			1	Endpoint interrupt requests are enabled for the endpoint.

14.8.5.4 DMA GO Interrupt Flag Register (USBxDGIF) (x = O or I)

At the completion of a DMA transfer, if RLD = 0, the USB DMA controller clears the GO bit of the endpoint, and the corresponding GO interrupt flag is set in USBxDGIF. For example, when the controller goes idle after servicing OUT endpoint 6, the OE6 bit is set in USBODGIF. In addition to setting the flag, the USB module can generate a DMA GO interrupt request. For details on GO interrupt requests, see section 14.6.3 on page 14-50.

Figure 14–40. DMA GO Interrupt Flag Register for OUT Endpoints (USBODGIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	Reserved
R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	

Figure 14–41. DMA GO Interrupt Flag Register for IN Endpoints (USBIDGIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	Reserved
R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	

Table 14–30. DMA GO Interrupt Flag Register (USBxDGIF) Field Values

Bit	*field*	*symval*	Value	Function
15–8	Reserved		0	These bits are not available for use.
7–1	xE[7:1]			x = O (for OUT) or I (for IN). The USB DMA controller sets the DMA GO flag bit to indicate that the controller is ready for a new DMA transfer. The flag bit is cleared either when the CPU reads the interrupt source register (USBINTSRC) with INTSRC equal to the corresponding interrupt or when the CPU writes a 1 to this bit.
			0	No GO interrupt pending
			1	GO interrupt pending
0	Reserved		0	This bit is not available for use.

14.8.5.5 DMA RLD Interrupt Flag Register (USBxDRIF) (x = O or I)

At the completion of a DMA transfer, if RLD = 1, the USB DMA controller clears the RLD bit of the endpoint, and the corresponding RLD interrupt flag is set in USBxDRIF. For example, when the controller performs a DMA reload operation for IN endpoint 7, the IE7 bit is set in USBIDRIF. In addition to setting the flag, the USB module can generate a DMA RLD interrupt request. For details on RLD interrupt requests, see section 14.6.3 on page 14-50.

Figure 14–42. DMA RLD Interrupt Flag Register for OUT Endpoints (USBODRIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	Reserved
R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	

Figure 14–43. DMA RLD Interrupt Flag Register for IN Endpoints (USBIDRIF)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	Reserved
R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	

Table 14–31. DMA RLD Interrupt Flag Register (USBxDRIF) Field Values

Bit	*field*	*symval*	Value	Function
15–8	Reserved		0	These bits are not available for use.
7–1	xE[7:1]			x = O (for OUT) or I (for IN). The DMA RLD flag bit is set when the USB DMA controller completes a DMA reload operation (see section 14.4.5 on page 14-18). The flag bit is cleared either when the CPU reads the interrupt source register (USBINTSRC) with INTSRC equal to the corresponding interrupt or when the CPU writes a 1 to this bit.
			0	No RLD interrupt pending
			1	RLD interrupt pending
0	Reserved			This bit is not available for use.

14.8.5.6 DMA Interrupt Enable Register (USBxDIE) (X = O or I)

USBxDIE enables or disables both DMA GO interrupt requests and DMA RLD interrupt requests. Consider DMA interrupt activity for IN endpoint 2. If the USB module sets the IE2 GO flag in USBIDGIF and IE2 = 1 in USBIDIE, a DMA GO interrupt request is generated for the endpoint. Likewise, if the IE2 RLD flag is set in USBIDRIF and IE2 = 1 in USBIDIE, a DMA RLD interrupt is generated for the endpoint. For more details about DMA interrupt requests, see section 14.6.3 on page 14-50.

Figure 14–44. DMA Interrupt Enable Register for OUT Endpoints (USBODIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
OE7	OE6	OE5	OE4	OE3	OE2	OE1	Reserved
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	

Figure 14–45. DMA Interrupt Enable Register for IN Endpoints (USBIDIE)

15	14	13	12	11	10	9	8
Reserved							

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	IE1	Reserved
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	

Table 14–32. DMA Interrupt Enable Register (USBxDIE) Field Values

Bit	*field*	*symval*	Value	Function
15–8	Reserved		0	These bits are not available for use.
7–1	xE[7:1]			x = O (for OUT) or I (for IN). DMA interrupt enable. This bit enables or disables both the RLD interrupt and the GO interrupt for the endpoint.
			0	RLD and GO interrupt requests are disabled for the endpoint.
			1	RLD and GO interrupt requests are enabled for the endpoint.
0	Reserved		0	This bit is not available for use.

14.8.6 Host-DMA Mode Registers

This section describes the registers used to configure the host-DMA mode of the USB module and to monitor host-DMA transfers. For more information about this mode, see section 14.5 on page 14-40.

14.8.6.1 Host Control Register (USBHCTL)

USBHCTL has bits to enable or disable the following: the host-DMA mode, the host interrupt, and the host error interrupt.

Figure 14–46. Host Control Register (USBHCTL)

7	6	5	4	3	2	1	0
Reserved					HERRIE	HIE	EN
					R/W–0	R/W–0	R/W–0

Table 14–33. Host Control Register (USBHCTL) Field Values

Bit	*field*	*symval*	Value	Function
7–3	Reserved			This bits are not available for use.
2	HERRIE			Host error interrupt enable
			0	Host error interrupt disabled
			1	Host error interrupt enabled
1	HIE			Host interrupt enable
			0	Host interrupt disabled
			1	Host interrupt enabled
0	EN			Host-DMA mode enable
			0	Host-DMA mode disabled
			1	Host-DMA mode enabled

14.8.6.2 Host Endpoint Select Register (USBHEPSEL)

USBHEPSEL determines which two of the 14 general-purpose endpoints are to be used by the USB DMA controller for host-DMA transfers.

Figure 14–47. Host Endpoint Select Register (USBHEPSEL)

7	6	5	4	3	2	1	0
Reserved	IEP			Reserved	OEP		
	R/W–0				R/W–0		

Table 14–34. Host Endpoint Select Register (USBHEPSEL) Field Values

Bit	*field*	*symval*	Value	Function
7	Reserved			This bit is not available for use.
6–4	IEP		1–7	IN endpoint number. IEP determines which of the general-purpose IN endpoints will be the host-DMA endpoint for IN transfers in the host-DMA mode.
3	Reserved			This bit is not available for your use.
2–0	OEP		1–7	OUT endpoint number. OEP determines which of the general-purpose OUT endpoints will be the host-DMA endpoint for OUT transfers in the host-DMA mode.

14.8.6.3 Host Status Register (USBHSTAT)

Check USBHSTAT to determine whether the host-DMA mode is disabled (DIS bit). USBSTAT also provides interrupt flag bits for the host interrupt and the host error interrupt.

Figure 14–48. Host Status Register (USBHSTAT)

7	6	5	4	3	2	1	0
Reserved					HERRIF	HIF	DIS
					R/W1C–0	R/W1C–0	R–1

Table 14–35. Host Status Register (USBHSTAT) Field Values

Bit	*field*	*symval*	Value	Function
7–3	Reserved			These bits are not available for use.
2	HERRIF			Host error interrupt flag
			0	No host error interrupt pending
			1	A host error has occurred: During an OUT transfer, the size of the data transferred from the host did not match the size specified in the protocol header.

Table 14–35. Host Status Register (USBHSTAT) Field Values (Continued)

Bit	*field*	*symval*	Value	Function
1	HIF			Host interrupt flag
			0	No host interrupt pending
			1	The USB DMA controller has completed a DMA transfer between the DSP memory and a host-DMA endpoint. This flag is set only if the host requests the interrupt by setting the INT bit in the first byte of the protocol header.
0	DIS			Host-DMA mode disabled flag
			0	Host-DMA mode not disabled
			1	Host-DMA mode disabled

14.8.7 General Control and Status Registers

This section describes the USB registers that perform general control and status functions. The bits in these registers allow such functions as resetting the USB module, shutting down the USB module, responding to specific conditions on the USB, and tracking USB frames for isochronous transfers.

14.8.7.1 Global Control Registe (USBGCTL)

The SOFTRST bit in USBGCTL enables you to reset the USB module without resetting the entire DSP.

Figure 14–49. Global Control Register (USBGCTL)

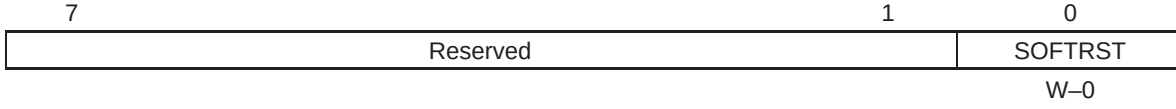


Table 14–36. Global Control Register (USBGCTL) Field Values

Bit	*field*	*symval*	Value	Function
7–1	Reserved			These bits are not available for use.
0	SOFTRST			Software reset
			1	Reset the USB module. The effects are the same as a hardware DSP reset (see page 14-54): All of the USB module registers assume their power-on default values except for USBCTL. Two effects are: (1) The USB module goes inactive, and (2) The USB module is disconnected from the USB. As soon as the reset is complete, SOFTRST is cleared to 0, and the USB is free to run.

14.8.7.2 Frame Number Registers (USBFNUMH, USBFNUML)

See Figure 14–50 and Figure 14–51. The register formed by the concatenation of the bits in USBFNUMH and USBFNUML can help you with isochronous transfers. This register reports the current USB frame number. The 11-bit value rolls over to 0 if it grows larger than 2047.

Figure 14–50. Frame Number (High Part) Register (USBFNUMH)

7	6	5	4	3	2	1	0
Reserved					FNUMH		
						R–0	

Figure 14–51. Frame Number (Low Part) Register (USBFNUML)

7	6	5	4	3	2	1	0
FNUML							
R–0							

Table 14–37. Frame Number Register (USBFNUMH and USBFNUML) Field Values

Register(Bit)	*field*	*symval*	Value	Function
USBFNUMH(7–3)	Reserved		0	Reserved. Reads as 0.
USBFNUMH(2–0)	FNUMH		0h–3h	Most significant 3 bits of the current USB frame number.
USBFNUML(7–0)	FNUML		00h–FFh	Least significant 8 bits of the current USB frame number.

14.8.7.3 PSOF Interrupt Timer Counter (USBPSOFTMR)

USBPSOFTMR is shown in Figure 14–52 and described in Table 14–38.

A start-of-frame (SOF) token is expected on the USB every 1 millisecond. If an endpoint is placed in the isochronous mode, the SOF token triggers a pending DMA transfer for that endpoint. To provide the minimum latency between the preparation of data buffers and the availability of those buffers to the USB DMA controller, the USB module can give advance notice of each SOF token. Advance notice can come from a pre-SOF (PSOF) interrupt request. If you load USBPSOFTMR with a value *n* and enable PSOF interrupt requests, the USB module generates a PSOF interrupt request n 750-kHz clock cycles ahead of the expected arrival of the SOF token.

Figure 14–52. PSOF Interrupt Timer Counter (USBPSOFTMR)

7	6	5	4	3	2	1	0
PSOFTMR							
R/W–0							

Table 14–38. PSOF Interrupt Timer Counter (USBPSOFTMR) Field Values

Bit	*field*	*symval*	Value	Function
7–0	PSOFTMR		0–255	Indicates the number of clocks a PSOF interrupt should proceed each SOF token. The clock is 750 kHz (USB 12 MHz clock divided by 16). **Note:** The time of the next SOF token is predicted and the prediction is not guaranteed to be precise.

14.8.7.4 USB Control Register (USBCTL)

USBCTL enables and controls the features that are described in Table 14–39. This register is not affected during a reset operation that is initiated with the SOFTRST bit. This register is affected by other reset operations.

Figure 14–53. USB Control Register (USBCTL)

7	6	5	4	3	2	1	0
CONN	FEN	RWUP	FRSTE	Reserved		SETUP	DIR
R/W–0	R/W–1	R/W–0	R/W–1			R/W–0	R/W–0

Table 14–39. USB Control Register (USBCTL) Field Values

Bit	*field*	*symval*	Value	Function
7	CONN			Connect/disconnect
			0	Upstream port is disconnected. The pull-up disabled.
			1	Upstream port is connected. Pull up enabled
6	FEN			Function enable
			0	The USB module is inactive.
			1	The USB module is active. If connected to the bus (see the description for the CONN bit), the module will communicate with the host.
5	RWUP			Device remote wakeup request. Writing a 1 to this bit generates a remote wake up signal on the bus. The USB module clears RWUP after the signal is sent.
			0	Writing a 0 to this bit has no effect.
			1	The CPU has asked the USB module to generate a remote wakeup signal on the bus.
4	FRSTE			Function reset/USB reset connection enable
			0	Function (module) reset is not connected to a USB reset. If a USB reset request is detected on the bus, the RSTR interrupt request is generated, but the USB module is not reset.
			1	Function (module) reset is connected to a USB reset. If a USB reset request is detected on the bus, the RSTR interrupt request is generated and the USB module is reset. All pending interrupts are cleared except the RSTR interrupt. The USB module is not disconnected from the bus.
3–2	Reserved		0	These bits are not available for use.
1	SETUP			Setup interrupt status. Software can write a 1 to this bit to indicate when a SETUP interrupt is being serviced. A write of 0 has no effect.
			0	The CPU has cleared this SETUP bit by writing 1 to the SETUP bit of the USB interrupt flag register (USBIF). A new setup packet will be accepted.
			1	Do not accept a new setup packet.The USB module will send a STALL in response to any setup packet until the SETUP bit is 0.

Table 14–39. USB Control Register (USBCTL) Field Values (Continued)

Bit	*field*	*symval*	Value	Function
0	DIR			Endpoint 0 data direction. When a setup packet arrives, the CPU must decode the packet and set or clear the DIR bit to reflect the direction of data flow. This bit also determines the endpoint's response to a 0-byte handshake packet. The USB module will not generate an endpoint interrupt upon completion of a control transfer handshake (a 0-byte transfer). The USB module stalls endpoint 0 if an OUT packet is expected (DIR = 0, OUT NAK = 0, IN NAK = 1) and an IN token arrives (early handshake), or the other way around.
			0	An OUT control transaction is expected. If an IN token (early handshake) arrives, respond with STALL.
			1	An IN control transaction is expected. If an OUT token (early handshake) arrives, respond with STALL.

14.8.7.5 USB Interrupt Flag Register (USBIF)

USBIF indicates the current status of the USB bus interrupts.

Notes:

1) The STPOW and SETUP bits are not automatically cleared when the CPU reads the interrupt source register (USBINTSRC). To clear one of these bits, write a 1 to it. The other bits in USBIF are automatically cleared when the corresponding interrupt value is read from USBINTSRC.

2) If a new setup token is received before SETUP is cleared, the USB module sets STPOW and an STPOW interrupt request (if enabled) is generated.

Figure 14–54. USB Interrupt Flag Register (USBIF)

7	6	5	4	3	2	1	0
RSTR	SUSR	RESR	SOF	PSOF	SETUP	Reserved	STPOW
R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0	R/W1C–0		R/W1C–0

Table 14–40. USB Interrupt Flag Register (USBIF) Field Values

Bit	*field*	*symval*	Value	Function
7	RSTR			Function reset request. This bit is set in response to host initiating a port reset. This bit is not affected by a USB module reset.
			0	No reset condition detected on the USB
			1	Reset condition detected on the USB
6	SUSR			Function suspend request
			0	No suspend condition detected on the USB
			1	Suspend condition detected on the USB
5	RESR			Function resume request
			0	No resume condition detected on the USB
			1	Activity on the USB resumes after a suspend condition.
4	SOF			Start of frame (SOF) notification
			0	No SOF packet detected on the USB
			1	SOF packet detected on the USB
3	PSOF		0	Pre-SOF (PSOF) notification. This bit is set a multiple of 16 USB clock cycles ahead of when the SOF token is expected. This allows the user to provide a transfer buffer for the USB DMA controller in time to prevent overflow or underflow conditions. This is especially helpful for isochronous transfers. The timing of this event is controlled by the content of the USBPSOFTMR register.
			0	No PSOF notification received
			1	PSOF notification received from PSOF timer
2	SETUP		0	SETUP packet received. Clearing this bit also clears the SETUP bit in the USB control register (USBCTL) and allows a new setup packet to move into the setup packet buffer.
			0	No setup packet received
			1	A new setup packet has arrived.
1	Reserved			This bit is not available for use.

Table 14–40. USB Interrupt Flag Register (USBIF) Field Values (Continued)

Bit	*field*	*symval*	Value	Function
0	STPOW			Setup overwrite
			0	No setup overwrite
			1	A setup overwrite has occurred; that is, SETUP = 1 in USBIF and another setup packet has arrived.

14.8.7.6 USB Interrupt Enable Register (USBIE)

The bits in USBIE enable or disable each of the interrupts associated with the bits in the USB interrupt flag register, USBIF (see section 14.8.7.5).

Figure 14–55. USB Interrupt Enable Register (USBIE)

7	6	5	4	3	2	1	0
RSTR	SUSR	RESR	SOF	PSOF	SETUP	Reserved	STPOW
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0		R/W–0

Table 14–41. USB Interrupt Enable Register (USBIE) Field Values

Bit	*field*	*symval*	Value	Function
7	RSRT			Function reset interrupt enable. This bit is not affected by a USB reset.
			0	Function reset interrupt disabled
			1	Function reset interrupt enabled
6	SUSR			Function suspend interrupt enable
			0	Function suspend interrupt disabled
			1	Function suspend interrupt enabled
5	RESR			Function resume interrupt enable
			0	Function resume interrupt disabled
			1	Function resume interrupt enabled
4	SOF			Start-of-frame (SOF) interrupt enable
			0	SOF interrupt disabled
			1	SOF interrupt enabled

Table 14–41. USB Interrupt Enable Register (USBIE) Field Values (Continued)

Bit	*field*	*symval*	Value	Function
3	PSOF			Pre-SOF (PSOF) interrupt enable
			0	PSOF interrupt disabled
			1	PSOF interrupt enabled
2	SETUP			SETUP interrupt enable
			0	SETUP interrupt disabled
			1	SETUP interrupt enabled
1	Reserved			This bit is not available for use.
0	STPOW			Setup overwrite interrupt enable
			0	STPOW interrupt disabled
			1	STPOW interrupt enabled

14.8.7.7 USB Device Address Register (USBADDR)

This register contains the address that uniquely identifies the USB module to the USB host.

Figure 14–56. USB Device Address Register (USBADDR)

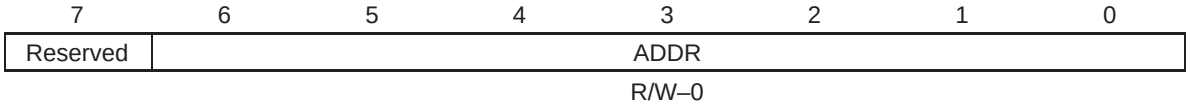


Table 14–42. USB Device Address Register (USBADDR) Field Values

Bit	*field*	*symval*	Value	Function
7	Reserved		0	Reserved = 0
6–0	ADDR		00h–3Fh	These seven bits hold the USB address assigned to the USB module. The CPU writes the address to this register as a result of a Set Address request from the host.

14.8.7.8 USB Idle Control Register (USBIDLECTL)

This register, shown in Figure 14–57, is not part of the USB module, but it contains the idle enable bit that enables the CPU to put the USB module in its idle mode. It also contains an idle status bit that indicates when the USB module is in the idle mode. Finally, USBIDLECTL contains the USBRST bit, which the CPU can use to hold the USB module in reset. The details about these bits are in Table 14–43.

This register resides in I/O space.

Figure 14–57. USB Idle Control Register (USBIDLECTL)

7	3	2	1	0
Reserved		USBRST	IDLESTAT	IDLEEN
		R/W–0	R–0	R/W–0

Table 14–43. USB Idle Control Register (USBIDLECTL) Field Values

Bit	*field*	*symval*	Value	Function
7–3	Reserved			These bits are not available for use.
2	USBRST			USB module reset
			0	Hold the USB module in reset.
			1	Bring the USB module out of reset.
1	IDLESTAT			Idle status
			0	The USB module has not been turned off by an IDLE instruction.
			1	The USB module is idle following an IDLE instruction.
0	IDLEEN			Idle enable
			0	The USB module cannot be affected by an IDLE instruction.
			1	Allow the USB module to be deactivated by an IDLE instruction. If the PERIPH domain has been deactivated by an IDLE instruction (PERIS = 1 in the idle status register), the USB module is inactive.

Chapter 15

Watchdog Timer

Some TMS320C55x™ (C55x™) DSPs include a watchdog timer. The watchdog timer prevents the system from locking up, if the software becomes trapped in loops with no controlled exit. To determine whether a particular C55x DSP has a watchdog timer, see the data sheet for that DSP.

Topic **Page**

15.1 Introduction to the Watchdog Timer

The watchdog timer output has a 4-way mux associated with it. The watchdog timer requires a special servicing sequence to be executed periodically. Without this periodic servicing, the watchdog timer counter reaches zero and times out. Consequently, an active-low pulse will be asserted on the watchdog timer output. The watchdog timer output can be selected to be not connected or to be connected to the local hardware reset, NMI (nonmaskable interrupt), or watchdog timer interrupt. This output selection allows maximum flexibility for utilizing the watchdog timer as required by the particular application.

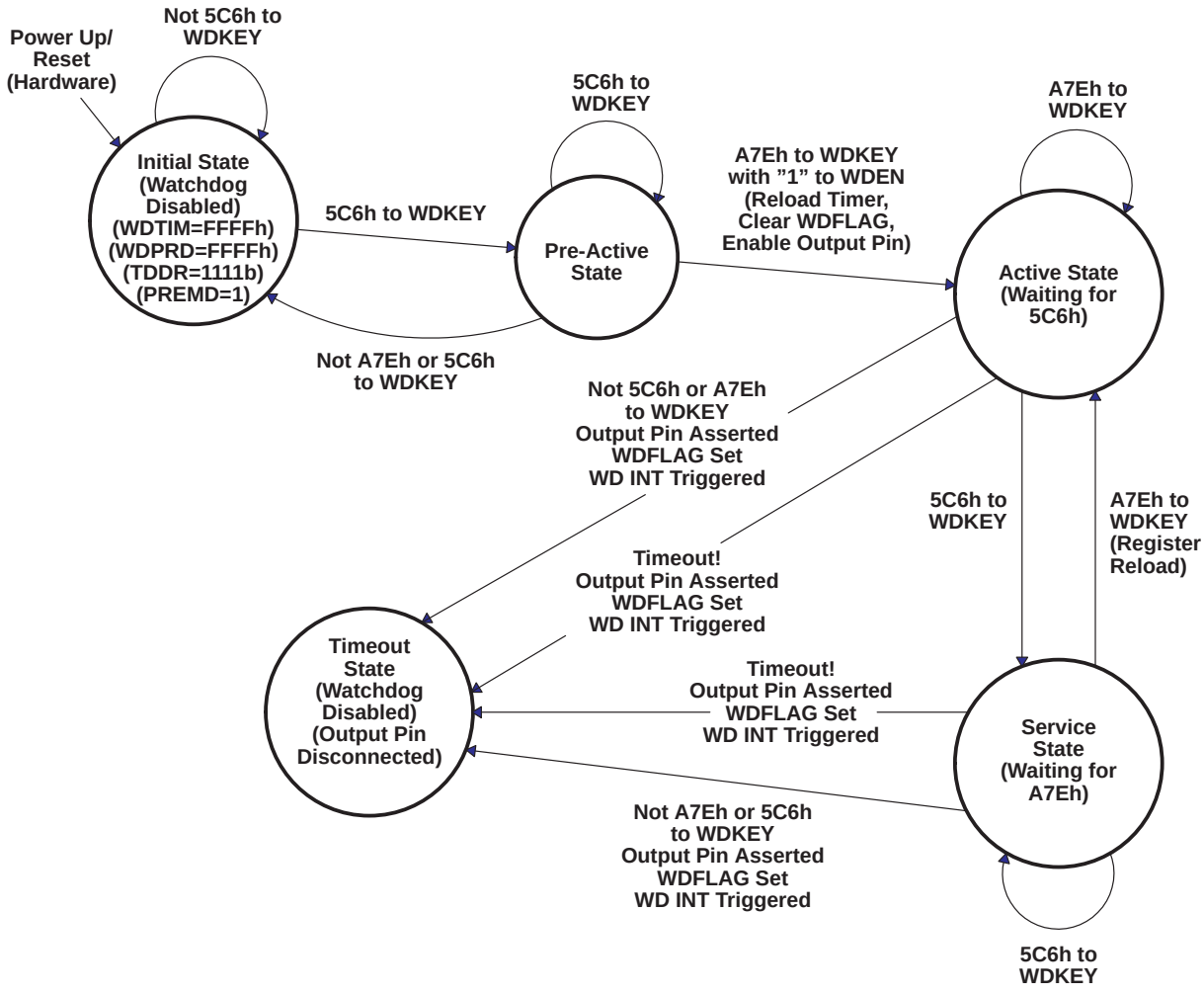
The watchdog timer consists of a 16-bit counter and a prescaler, and supports up to a 32-bit dynamic range. Out of reset, the watchdog timer is disabled by default, this allows a flexible period of time for code to be loaded into the on-chip memory. However, during this time, the counter starts to run with the default values in the counters and prescalers, and the watchdog timer output is disconnected from the watchdog time-out event. The counter reaches zero when the counter value and prescaler value are exhausted, consequently, the watchdog timer interrupt is triggered and the WDFLAG bit is set to 1. The counters and prescalers are reloaded automatically and start to run continuously. Once the watchdog timer is enabled by setting the WDEN bit to 1, the watchdog timer output is connected to the watchdog time-out event and the counters and prescalers are reloaded accordingly. To enable the watchdog timer, a certain sequence of events shall be followed as shown in the watchdog timer operation state diagram (Figure 15–1).

As shown in Figure 15–1, before the watchdog timer enters the Active state, the watchdog timer is disabled. The watchdog timer outputs (INT3, NMI, or RESET) are active only during Active, Service, and Timeout states.

Once the watchdog timer is enabled, it cannot be disabled by software but can be disabled by a watchdog time-out event or hardware reset. A special key sequence is provided to prevent the watchdog timer from being accidentally serviced while the software is trapped in a dead loop or in some other software failures.

The watchdog timer’s clock is part of the clock generator domain; thus, the watchdog timer is constantly clocked during an IDLE instruction as long as the CLKI bit of the IDLE configuration register (ICR) is set.

Figure 15–1. Watchdog Timer Operation State Diagram



15.2 Watchdog Timer Registers

Once the watchdog timer is enabled, the registers are under write protection. Writes to WDTIM, WDPRD, and WDTCR has no effect. Writes to the WDFLAG, WDEN, and PREMD bits in WDTCR2 has no effect. However, incorrect key (not 5C6h or A7Eh) sequence to the WDKEY bit results in a watchdog timer time-out event.

The watchdog timer memory-mapped registers are listed in Table 15–1.

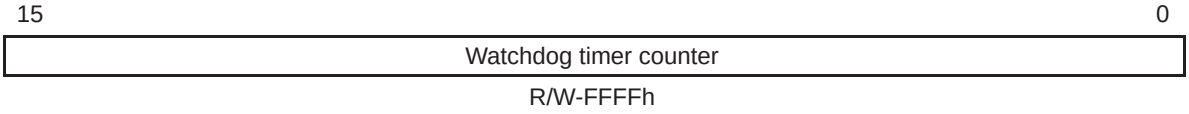
Table 15–1. Watchdog Timer Memory-Mapped Registers

Address (Hex)	Name	Description
4000	WDTIM	Watchdog Timer Counter Register
4001	WDPRD	Watchdog Timer Period Register
4002	WDTCR	Watchdog Timer Control Register
4003	WDTCR2	Watchdog Timer Control Register 2

15.2.1 Watchdog Timer Counter Register (WDTIM)

The WDTIM is loaded with the 16-bit value in the watchdog period register (WDPRD). When the PSC bit (in the WDTCR) is decremented past 0 or the watchdog timer is reset, the WDTIM value is decremented.

Figure 15–2. Watchdog Timer Counter Register (WDTIM)



Note: R/W-x = Read/Write-Reset value

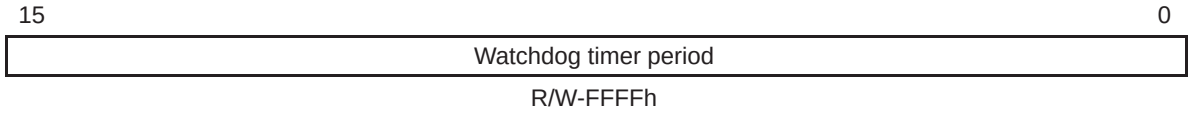
Table 15–2. Watchdog Timer Counter Register (WDTIM) Field Values

Bit	*field*	*symval*	Value	Description
15–0		OF(*value*)	0–FFFFh	This 16-bit value is loaded with the watchdog timer period register (WDPRD) value and decremented.

15.2.2 Watchdog Timer Period Register (WDPRD)

The WDPRD is used to reload the watchdog timer counter register (WDTIM).

Figure 15–3. Watchdog Timer Period Register (WDPRD)



Note: R/W-x = Read/Write-Reset value

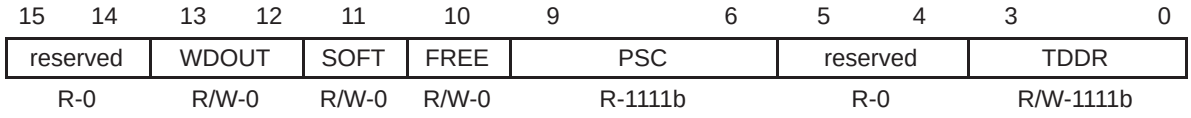
Table 15–3. Watchdog Timer Period Register (WDPRD) Field Values

Bit	*field*	*symval*	Value	Description
15–0		OF(*value*)	0–FFFFh	This 16-bit value is used to reload the watchdog timer counter register (WDTIM).

15.2.3 Watchdog Timer Control Register (WDTCR)

The WDTCR provides the control and status information for the watchdog timer.

Figure 15–4. Watchdog Timer Control Register (WDTCR)



Note: R/W-x = Read/Write-Reset value

Table 15–4. Watchdog Timer Control Register (WDTCR) Field Values

Bit	*field*	*symval*	Value	Description
15–14	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
13–12	WDOUT			Watchdog timer output bits control the 4-output mux to determine where the watchdog timer output is connected.
			00	Output is internally connected to timer interrupt (INT3).
			01	Output is internally connected to NMI.
			10	Output is internally connected to RESET.
			11	Output is not connected.

Table 15–4. Watchdog Timer Control Register (WDTCR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
11	SOFT			Used in conjunction with FREE bit to determine the state of the watchdog timer when a breakpoint is encountered in the HLL debugger. When FREE bit is cleared, SOFT bit selects the watchdog timer mode.
		BRKPTNOW	0	The watchdog timer stops immediately.
		WAITZERO	1	The watchdog timer stops when the watchdog timer counter register (WDTIM) decrements to 0.
10	FREE			Used in conjunction with SOFT bit to determine the state of the watchdog timer when a breakpoint is encountered in the HLL debugger. When FREE bit is cleared, SOFT bit selects the watchdog timer mode.
		WITHSOFT	0	SOFT bit selects the watchdog timer mode.
		NOSOFT	1	The watchdog timer runs free regardless of SOFT bit status.
9–6	PSC			Watchdog timer prescaler counter bits. This read-only bit specifies the count for the watchdog timer when in direct mode (PREMD bit is cleared in the WDTCR2). When the PSC bit is decremented past 0 or the watchdog timer is reset, the PSC bit is loaded with the contents of the TDDR bit and the WDTIM value is decremented.
5–4	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
3–0	TDDR			The watchdog timer prescaler bits.
				In prescaler direct mode (PREMD = 0 in WDTCR2):
		OF(*value*)	0–15	This value specifies a direct prescaler count, up to 15, for the watchdog timer. When the PSC bit is decremented past 0, the PSC bit is loaded with contents of this TDDR bit.
				In prescaler indirect mode (PREMD = 1 in WDTCR2):
		OF(*value*)		This value specifies an indirect prescaler count, up to 65535, for the watchdog timer. When the PSC bit is decremented past 0, the PSC bit is loaded with this prescaler value.
			0000	Prescaler value: 0001h
			0001	Prescaler value: 0003h
			0010	Prescaler value: 0007h
			0011	Prescaler value: 000Fh

Table 15–4. Watchdog Timer Control Register (WDTCR) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
			0100	Prescaler value: 001Fh
			0101	Prescaler value: 003Fh
			0110	Prescaler value: 007Fh
			0111	Prescaler value: 00FFh
			1000	Prescaler value: 01FFh
			1001	Prescaler value: 03FFh
			1010	Prescaler value: 07FFh
			1011	Prescaler value: 0FFFh
			1100	Prescaler value: 1FFFh
			1101	Prescaler value: 3FFFh
			1110	Prescaler value: 7FFFh
			1111	Prescaler value: FFFFh

15.2.4 Watchdog Timer Control Register 2 (WDTCR2)

The WDTCR2 contains bits to indicate watchdog flag, to enable watchdog, to set general timer or watchdog mode, to set prescaler mode as well as provides the 12 bit WDKEY for watchdog service.

Figure 15–5. Watchdog Timer Control Register 2 (WDTCR2)

15	14	13	12	11 ... 0
WDFLAG	WDEN	reserved	PREMD	WDKEY
R/W-0	R/W-0	R-0	R/W-1	R/W-0

Note: R/W-x = Read/Write-Reset value

Table 15–5. Watchdog Timer Control Register 2 (WDTCR2) Field Values

Bit	*field*	*symval*	Value	Description
15	WDFLAG			Watchdog timer flag bit. This bit can be cleared by enabling the watchdog timer, by a device reset, or by being written with a 1.
			0	No watchdog timer time-out event occurred.
			1	Watchdog timer time-out event occurred.

Table 15–5. Watchdog Timer Control Register 2 (WDTCR2) Field Values (Continued)

Bit	*field*	*symval*	Value	Description
14	WDEN			Watchdog timer enable bit.
			0	Watchdog timer is disabled. Watchdog timer output is disconnected from the watchdog timer time-out event and the counter starts to run.
			1	Watchdog timer is enabled. Watchdog timer output is connected to the watchdog timer time-out event. Watchdog timer can be disabled by a watchdog timer time-out event or by a device reset.
13	reserved			Reserved. The reserved bit location is always read as zero. A value written to this field has no effect.
12	PREMD			Prescaler mode select bit.
			0	Direct mode. When PSC bit in WDTCR is decremented past 0, PSC bit is loaded with TDDR content in WDTCR.
			1	Indirect mode. When PSC bit in WDTCR is decremented past 0, PSC bit is loaded with the prescaler value associated with TDDR bit in WDTCR.
11–0	WDKEY			Watchdog timer reset key bits. A 12-bit value that before a watchdog timer time-out event occurs, only a write sequence of a 5C6h followed by an A7Eh services the watchdog timer. Any other writes triggers a watchdog timer time-out event immediately.

15.3 Watchdog Timer Servicing

The watchdog timer has to be serviced periodically such that a 5C6h is written followed by an A7Eh to the WDKEY bit, in WDTCR2, before a watchdog timer time-out event occurs. Both 5C6h and A7Eh are allowed to be written to the WDKEY bit. Only the write sequence of a 5C6h followed by an A7Eh to the WDKEY bit services the watchdog timer. Any other writes to the WDKEY bit triggers the watchdog timer time-out event immediately, and consequently:

- ❏ the WDFLAG bit, in WDTCR2, is set to 1
- ❏ the internal maskable watchdog timer interrupt, nonmaskable interrupt (NMI), or RESET is triggered

However, reads from WDTCR2 do not cause a time-out event.

When the watchdog timer is in a time-out state, the watchdog timer is disabled and the WDEN bit, in WDTCR2, is cleared. The watchdog timer output is disconnected from the watchdog timer time-out event, the timer is reloaded and continues to run.

Out of reset, the watchdog timer is disabled and reads or writes to the watchdog timer registers are allowed. However, once a 5C6h is written to the WDKEY bit and the watchdog timer enters the Pre-Active state, writes to WDTCR2 are allowed only when the write comes with the correct key (5C6h or A7Eh) to the WDKEY bit. In the Pre-Active state, an A7Eh written to the WDKEY bit with the WDEN bit set to 1 enables the watchdog timer (Active state). Once the watchdog timer is enabled, it cannot be disabled by software but can be disabled by a watchdog time-out event or hardware reset. The watchdog timer service is secured by the WDKEY bit.

The WDTIM, WDPRD, WDTCR, and the PREMD bit in WDTCR2 shall be configured as desired before the watchdog timer enters the Active state. You can set the WDEN bit to 1 and configure the PREMD bit at the same time an A7Eh is written to the WDKEY bit, during the Pre-Active state. By default, WDTIM = FFFFh, WDPRD = FFFFh, the PREMD bit = 1, and the TDDR bit = 1111b. Every time the watchdog timer is serviced, the watchdog timer counters and prescalers are automatically reloaded accordingly.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y