TMS320C6000 Peripherals Reference Guide

Literature Number: SPRU190C
April 1999

TEXAS INSTRUMENTS

Preface

Read This First

About This Manual

This reference guide describes the on-chip peripherals of the TMS320C6000 digital signal processors (DSPs). Main topics are the program memory, the data memory, the direct memory access (DMA) controller, the enhanced DMA controller (EDMA), the host-port interface (HPI), the exansion bus, the external memory interface (EMIF), the boot configuration, the multichannel buffered serial ports (McBSPs), the timers, the interrupt selector and external interrupts, and the power-down modes.

The TMS320C62x (’C62x) and the TMS320C67x (’C67x) generations of digital signal processors make up the TMS320C6000 platform of the TMS320 family of digital signal processors. The ’C62x devices are fixed-point DSPs, and the ’C67x devices are floating-point DSPs. The TMS320C6000 (’C6000) is the first DSP to use the VelociTI™ architecture, a high-performance, advanced VLIW (very long instruction word) architecture. The VelocTI architechure makes the ’C6x an excellent choice for multichannel, multifunction, and high data rate applications.

Notational Conventions

This document uses the following conventions:

- Program listings, program examples, names are shown in a `special font`. Here is a sample program listing:

  ```
  LDW .D1    *A0,A1
  ADD .L1    A1,A2,A3
  NOP        3
  MPY .M1    A1,A4,A5
  ```

- Throughout this book MSB means *most significant bit,* and LSB means *least significant bit*.

Registers are described throughout this book in register diagrams. Each diagram shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its name inside, its beginning and ending bit numbers above, and its properties below. A legend explains the notation used for the properties. For example:

31 25	24 23	22 21	20 18	17 16
FIELDA	FIELDB	FIELDC	R, +1	RW, +0
RW, +0	RC, +x	R, +0	R, +1	HRW, +0

Note: R = Readable by the CPU, W = Writeable by the CPU, +x = Value undefined after reset, +0 = Value is 0 after reset, +1 = Value is 1 after reset, C = Clearable by the CPU, H = reads/writes performed by the host

Related Documentation From Texas Instruments

The following documents describe the TMS320C6x family 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.

TMS320C6000 Technical Brief (literature number SPRU197) gives an introduction to the 'C6000 platform of digital signal processors, development tools, and third-party support.

TMS320C6000 CPU and Instruction Set Reference Guide (literature number SPRU189) describes the 'C6000 CPU architecture, instruction set, pipeline, and interrupts for these digital signal processors.

TMS320C6000 Programmer's Guide (literature number SPRU198) describes ways to optimize C and assembly code for the TMS320C6000 DSPs and includes application program examples.

TMS320C6000 Assembly Language Tools User's Guide (literature number SPRU186) 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 the 'C6000 generation of devices.

TMS320C6000 Optimizing C Compiler User's Guide (literature number SPRU187) describes the 'C6000 C compiler and the assembly optimizer. This C compiler accepts ANSI standard C source code and produces assembly language source code for the 'C6000 generation of devices. The assembly optimizer helps you optimize your assembly code.

TMS320C6x C Source Debugger User's Guide (literature number SPRU188) tells you how to invoke the 'C6x simulator and emulator versions of the C source debugger interface. This book discusses various aspects of the debugger, including command entry, code execution, data management, breakpoints, profiling, and analysis.

TMS320C6201, TMS320C6201B Digital Signal Processors Data Sheet (literature number SPRS051) describes the features of the TMS320C6201 and TMS320C6201B fixed-point DSPs and provides pinouts, electrical specifications, and timings for the devices.

TMS320C6202 Digital Signal Processor Data Sheet (literature number SPRS072) describes the features of the TMS320C6202 fixed-point DSP and provides pinouts, electrical specifications, and timings for the device.

TMS320C6701 Digital Signal Processor Data Sheet (literature number SPRS067) describes the features of the TMS320C6701 floating-point DSP and provides pinouts, electrical specifications, and timings for the device.

TMS320C6211 Digital Signal Processor Data Sheet (literature number SPRS073) describes the features of the TMS320C6211 fixed-point DSP and provides pinouts, electrical specifications, and timings for the device.

TMS320C6711 Digital Signal Processor Data Sheet (literature number SPRS088) describes the features of the TMS320C6711 fixed-point DSP and provides pinouts, electrical specifications, and timings for the device.

Trademarks

320 Hotline On-line, VelociTI, and XDS510 are trademarks of Texas Instruments Incorporated.

PC is a trademark of International Business Machines Corporation.

Solaris and SunOS are trademarks of Sun Microsystems, Inc.

SPI is a trademark of Motorola, Inc.

ST-BUS is a trademark of Mitel.

Windows and Windows NT are registered trademarks of Microsoft Corporation.

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
DSP Solutions	http://www.ti.com/dsps
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580	
TI Literature Response Center U.S.A.	(800) 477-8924	
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285	
U.S. Technical Training Organization	(972) 644-5580	
DSP Hotline		Email: dsph@ti.com

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32

Email: epic@ti.com

Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type "Go TIASP"	

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: dsph@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Contents

Figures

Tables

Chapter 1

Introduction

The TMS320C6000 (‘C6000) platform of devices consists of the first off-the-shelf digital signal processors (DSPs) to use advanced very long instruction word (VLIW) to achieve high performance through increased instruction-level parallelism. The VelociTI™ advanced very long instruction word (VLIW) architecture uses multiple execution units operating in parallel to execute multiple instructions during a single clock cycle. Parallelism is the key to extremely high performance, taking these DSPs well beyond the performance capabilities of traditional designs.

This chapter introduces the TMS320 family of DSPs and the ’C6000 platform of this family, and it describes the features, memory, and peripherals of the ’C6000 devices.

Topic **Page**

1.1 TMS320 Family Overview

The TMS320 family consists of fixed-point, floating-point, and multiprocessor digital signal processors (DSPs). TMS320 DSPs are specifically designed for real-time signal processing.

1.1.1 History of TMS320 DSPs

In 1982, Texas Instruments introduced the TMS32010—the first fixed-point DSP in the TMS320 family. Before the end of the year, *Electronic Products* magazine awarded the TMS32010 the title “Product of the Year”. Today, the TMS320 family consists of these generations: ’C1x, ’C2x, ’C27x, ’C5x, and ’C54x, ’C55x fixed-point DSPs; ’C3x and ’C4x floating-point DSPs; and ’C8x multiprocessor DSPs. Now there is a new generation of DSPs, the TMS320C6000 platform, with performance and features that are reflective of Texas Instruments’ commitment to lead the world in DSP solutions.

1.1.2 Typical Applications for the TMS320 Family

Table 1-1 lists some typical applications for the TMS320 family of DSPs. The TMS320 DSPs offer adaptable approaches to traditional signal-processing problems. They also support complex applications that often require multiple operations to be performed simultaneously.

Table 1–1. Typical Applications for the TMS320 DSPs

Automotive	**Consumer**	**Control**
Adaptive ride control	Digital radios/TVs	Disk drive control
Antiskid brakes	Educational toys	Engine control
Cellular telephones	Music synthesizers	Laser printer control
Digital radios	Pagers	Motor control
Engine control	Power tools	Robotics control
Global positioning	Radar detectors	Servo control
Navigation	Solid-state answering machines	
Vibration analysis		
Voice commands		
General Purpose	**Graphics/Imaging**	**Industrial**
Adaptive filtering	3-D computing	Numeric control
Convolution	Animation/digital maps	Power-line monitoring
Correlation	Homomorphic processing	Robotics
Digital filtering	Image compression/transmission	Security access
Fast Fourier transforms	Image enhancement	
Hilbert transforms	Pattern recognition	
Waveform generation	Robot vision	
Windowing	Workstations	
Instrumentation	**Medical**	**Military**
Digital filtering	Diagnostic equipment	Image processing
Function generation	Fetal monitoring	Missile guidance
Pattern matching	Hearing aids	Navigation
Phase-locked loops	Patient monitoring	Radar processing
Seismic processing	Prosthetics	Radio frequency modems
Spectrum analysis	Ultrasound equipment	Secure communications
Transient analysis		Sonar processing
Telecommunications		**Voice/Speech**
1200- to 56 600-bps modems	Faxing	Speaker verification
Adaptive equalizers	Future terminals	Speech enhancement
ADPCM transcoders	Line repeaters	Speech recognition
Base stations	Personal communications	Speech synthesis
Cellular telephones		Speech vocoding
Channel multiplexing	systems (PCS)	Text-to-speech
Data encryption	Personal digital assistants (PDA)	Voice mail
Digital PBXs	Speaker phones	
Digital speech interpolation (DSI)	Spread spectrum communications	
DTMF encoding/decoding	Digital subscriber loop (xDSL)	
Echo cancellation	Video conferencing	
	X.25 packet switching	

1.2 Overview of the TMS320C6000 Platform of DSPs

With a performance of up to 2000 million instructions per second (MIPS) and an efficient C compiler, the TMS320C6000 DSPs give system architects unlimited possibilities to differentiate their products from others. High performance, ease of use, and affordable pricing make the TMS320C6000 platform the ideal solution for multichannel, multifunction applications, such as:

- Pooled modems
- Wireless local loop base stations
- Remote access servers (RAS)
- Digital subscriber loop (DSL) systems
- Cable modems
- Multichannel telephony systems

The TMS320C6000 platform is also an ideal solution for exciting new applications, for example:

- Personalized home security with face and hand/fingerprint recognition
- Advanced cruise control with GPS navigation and accident avoidance
- Remote medical diagnostics
- Beam-forming base stations
- Virtual reality 3-D graphics
- Speech recognition
- Audio
- Radar
- Atmospheric modeling
- Finite element analysis
- Imaging (for example, fingerprint recognition, ultrasound, and MRI)

1.3 Features and Options of the TMS320C6000 Devices

The ’C6000 devices execute up to eight 32-bit instructions per cycle. The device’s core CPU consists of 32 general-purpose registers of 32-bit-word length and eight functional units:

- Two multipliers
- Six arithmetic logic units (ALUs)

The ’C6000 generation has a complete set of optimized development tools, including an efficient C compiler, an assembly optimizer for simplified assembly-language programming and scheduling, and a Windows™ based debugger interface for visibility of source code execution characteristics.

Features of the ’C6000 devices include:

- Advanced VLIW CPU with eight functional units, including two multipliers and six arithmetic units
  - Executes up to eight instructions per cycle for up to ten times the performance of other DSPs
  - Allows designers to develop highly effective RISC-like code for rapid development
- Instruction packing
  - Gives code size equivalence for eight instructions executed serially or in parallel
  - Reduces code size, program fetches, and power consumption
- Conditional execution of all instructions
  - Reduces costly branching
  - Increases parallelism for higher sustained performance
- Efficient code execution on independent functional units
  - Industry’s most efficient C compiler on DSP benchmark suite
  - Industry’s first assembly optimizer for fast development and improved parallelism

- 8/16/32-bit data support, providing efficient memory support for a variety of applications
- 40-bit arithmetic options, which add extra precision for vocoders and other computationally intensive applications
- Saturation and normalization, which provide support for key arithmetic operations
- Field manipulation and instruction extract, set, clear, and bit counting, which support common operations found in control and data manipulation applications.
- Hardware support for IEEE single-precision and double-precision instructions. (’C6701 only)
- Pin-compatible fixed-point and floating-point DSPs.

For more information on features and options of the TMS320C6000, see the *TMS320C6000 CPU and Instruction Set Reference Guide*.

1.4 Overview of TMS320C6000 Memory

The internal memory configuration varies between the different ’C6000 devices. All devices include:

- Internal data/program memory
- Internal peripherals
- External memory accessed through the external memory interface (EMIF)

TMS320C6201/C6202/C6701: The ‘C6201, ‘C6202, and ‘C6701 each have separate data and program memories. The internal program memory can be mapped into the CPU address space or operated as a program cache. A 256-bit-wide path is provided from to the CPU to allow a continuous stream of eight 32-bit instructions for maximum performance.

Data memory is accessed through the data memory controller, which controls the following functions:

- The CPU and the direct memory access (DMA) controller accesses to the internal data memory, and performs the necessary arbitration.
- The CPU data access to the EMIF
- The CPU access to on-chip peripherals

The internal data memory is divided into 16-bit-wide banks. The data memory controller performs arbitration between the CPU and the DMA controller independently for each bank, allowing both sides of the CPU and the DMA to access different memory locations simultaneously without contention. The data memory controller supports configurable endianness. The LENDIAN pin on the device selects the endianness of the device.

TMS320C6211/C6711: The ‘C6211/C6711 is a cache-based architecture, with separate level-one program and data caches. These cache spaces are not included in the memory map and are enabled at all times. The level-one caches are only accessible by the CPU.

The level-one program cache (L1P) controller interfaces the CPU to the L1P. A 256-bit wide path is provided from to the CPU to allow a continuous stream of 8 32-bit instructions for maximum performance.

The level-one data cache (L1D) controller provides the interface between the CPU and the L1D. The L1D is a dual-ported memory, which allows simultaneous access by both sides of the CPU.

On a miss to either L1D or L1P, the request is passed to the L2 controller. The L2 controller facilitates:

- The CPU and the enhanced direct memory access (EDMA) controller accesses to the internal memory, and performs the necessary arbitration
- The CPU data access to the EMIF
- The CPU accesses to on-chip peripherals
- Sends request to EMIF for an L2 data miss

The internal SRAM of the ‘C6211/C6711 is a unified program and data memory space. The L2 memory space may be configured as all memory-mapped SRAM, all cache, or a combination of the two.

1.5 Overview of TMS320C6000 Peripherals

Peripherals available on the TMS320C6000 devices are shown in Table 1-2.

Table 1–2. TMS320C6000 Peripherals

Peripheral	C6201	C6202	C6211	C6701	C6711
Direct memory access (DMA) controller	Y	Y	N	Y	N
Enhanced direct memory access (EDMA) controller	N	N	Y	N	Y
Host-port interface (HPI)	Y	N	Y	Y	Y
Expansion bus	N	Y	N	N	N
External memory interface (EMIF)	Y	Y	Y	Y	Y
Boot configuration	Y	Y	Y	Y	Y
Multichannel buffered serial ports (McBSPs)	2	3	2	2	2
Interrupt selector	Y	Y	Y	Y	Y
32-bit timers	2	2	2	2	2
Power-down logic	Y	Y	Y	Y	Y

The user-accessible peripherals are configured via a set of memory-mapped control registers. The peripheral bus controller performs the arbitration for accesses of on-chip peripherals. The Boot Configuration logic is interfaced through external signals only, and the Power-down logic is accessed directly by the CPU.

Figure 1-1 shows the peripherals in the block diagram for the TMS320C6201, ‘C6202, and ‘C6701 devices. Figure 1-2 shows a block diagram for the TMS320C6211 and ’C6711 devices.

Figure 1–1. TMS320C6201/C6202/C6701 Block Diagram

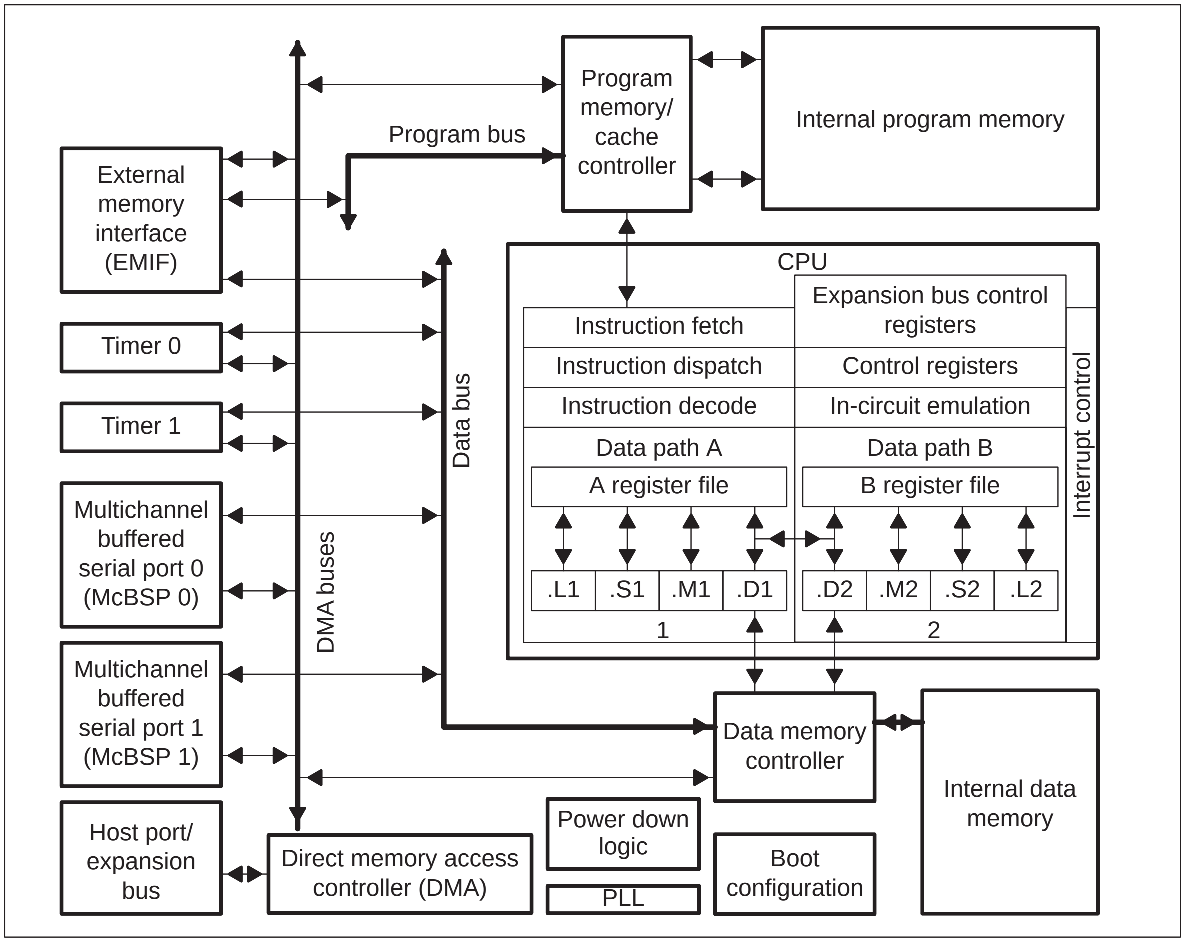
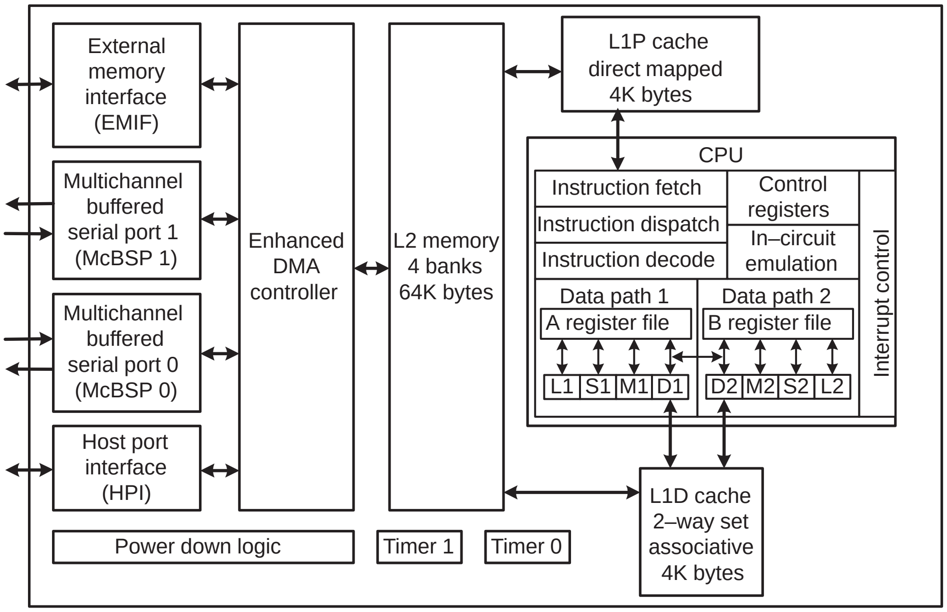


Figure 1–2. TMS320C6211/C6711 Block Diagram


External memory interface (EMIF)
Multichannel buffered serial port 1 (McBSP 1)
Multichannel buffered serial port 0 (McBSP 0)
Host port interface (HPI)
Enhanced DMA controller
L2 memory 4 banks 64K bytes
L1P cache direct mapped 4K bytes
CPU
Instruction fetch
Instruction dispatch
Instruction decode
Control registers
In–circuit emulation
Interrupt control
Data path 1
A register file
L1
S1
M1
D1
Data path 2
B register file
D2
M2
S2
L2
L1D cache 2–way set associative 4K bytes
Power down logic
Timer 1
Timer 0

DMA Controller: The DMA controller transfers data between address ranges in the memory map without intervention by the CPU. The DMA controller has four programmable channels and a fifth auxiliary channel.

EDMA Controller: The EDMA controller performs the same functions as the DMA controller. The EDMA has sixteen programmable channels, as well as a RAM space to hold multiple configurations for future transfers.

HPI: The HPI is a parallel port through which a host processor can directly access the CPU's memory space. The host device has ease of access because it is the master of the interface. The host and the CPU can exchange information via internal or external memory. In addition, the host has direct access to memory-mapped peripherals.

Expansion Bus: The expansion bus is a replacement for the HPI, as well as an expansion of the EMIF. The expansion provides two distinct areas of functionality, (host port and I/O port) which can co-exist in a system. The host port of the expansion bus can operate in either asynchronous slave mode, similar to the HPI, or in synchronous master/slave mode. This allows the device to interface to a variety of host bus protocols. Synchronous FIFOs and asynchronous peripheral I/O devices may interface to the expansion bus.

EMIF: The EMIF supports a glueless interface to several external devices, including:

❑ Synchronous burst SRAM (SBSRAM)
❑ Synchronous DRAM (SDRAM)
❑ Asynchronous devices, including SRAM, ROM, and FIFOs
❑ An external shared-memory device

Boot Configuration: The TMS320C62x and TMS320C67x provide a variety of boot configurations that determine what actions the DSP performs after device reset to prepare for initialization. These include loading in code from an external ROM space on the EMIF and loading code through the HPI/expansion bus from an external host.

McBSP: The multichannel buffered serial port (McBSP) is based on the standard serial port interface found on the TMS320C2000 and ’C5000 platform devices. In addition, the port can buffer serial samples in memory automatically with the aid of the DMA/EDMA controller. It also has multichannel capability compatible with the T1, E1, SCSA, and MVIP networking standards. Like its predecessors, it provides:

❑ Full-duplex communication

❑ Double-buffered data registers that allow a continuous data stream

❑ Independent framing and clocking for receive and transmit

❑ Direct interface to industry-standard codecs, analog interface chips (AICs), and other serially connected A/D and D/A devices

In addition, the McBSP has the following capabilities:

❑ Direct interface to:
  ■ T1/E1 framers
  ■ ST-BUS™ compliant devices
  ■ IOM-2 compliant devices
  ■ AC97 compliant devices
  ■ IIS compliant devices
  ■ SPI™ devices
❑ Multichannel transmission and reception of up to 128 channels
❑ A wider selection of data sizes including 8-, 12-, 16-, 20-, 24-, and 32-bits
❑ μ-law and A-law companding
❑ 8-bit data transfers with LSB or MSB first
❑ Programmable polarity for both frame synchronization and data clocks
❑ Highly programmable internal clock and frame generation

Timer: The ’C6000 devices have two 32-bit general-purpose timers that are used to:

❑ Time events
❑ Count events
❑ Generate pulses
❑ Interrupt the CPU
❑ Send synchronization events to the DMA/EDMA controller

Interrupt Selector: The ’C6000 peripheral set produces 14–16 interrupt sources. The CPU has 12 interrupts available. The interrupt selector allows you to choose which 12 interrupts your system needs. The interrupt selector also allows you to change the polarity of external interrupt inputs.

Power-down: The power-down logic allows reduced clocking to reduce power consumption. Most of the operating power of CMOS logic dissipates during circuit switching from one logic state to another. By preventing some or all of the chip’s logic from switching, you can realize significant power savings without losing any data or operational context.

Chapter 2

TMS320C6201/C6701 Program and Data Memory

This chapter describes the program memory organization, the program memory and cache modes, and access of program memory through the DMA controller for the TMS320C6201/C6701.

Topic **Page**

2.1 Program Memory Controller

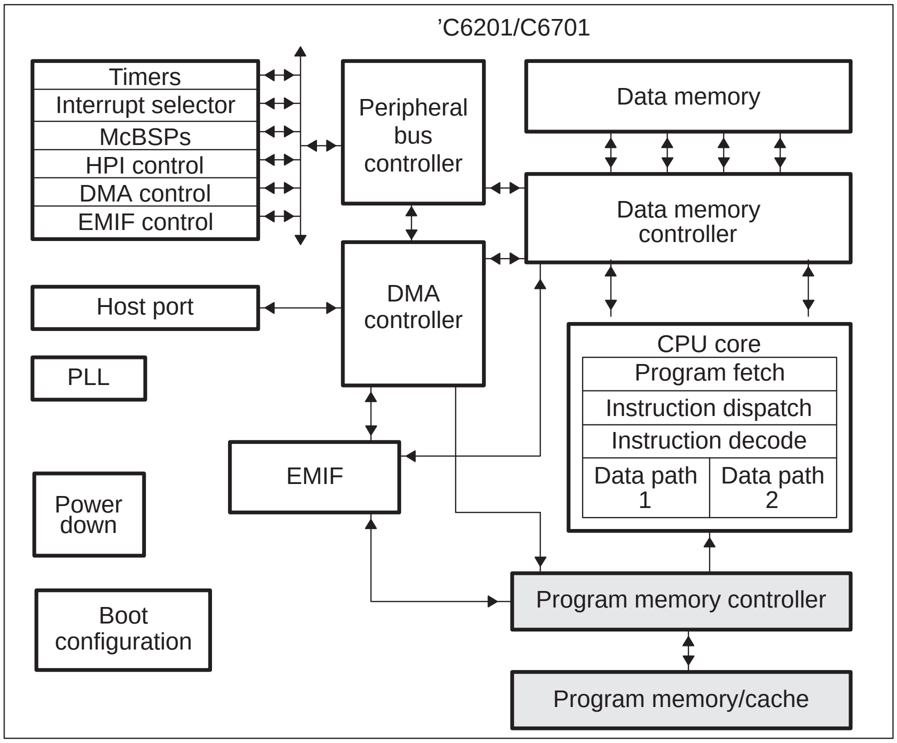
The program memory controller, shown in Figure 2–1 performs the following tasks:

❑ Performs CPU and DMA requests to internal program memory and the necessary arbitration

❑ Performs CPU requests to external memory through the external memory interface (EMIF)

❑ Manages the internal program memory when it is configured as cache.

Figure 2–1. TMS320C6201/C6701 Program Memory Controller in the Block Diagram



2.2 Internal Program Memory

The internal program memory contains 64K bytes of RAM or, equivalently, 2K 256-bit fetch packets or 16K 32-bit instructions. The CPU, through the program memory controller, has a single-cycle throughput, 256-bit-wide connection to internal program memory.

2.2.1 Internal Program Memory Modes

The internal program memory can be used in any of four modes which are selected by the program cache control (PCC) field (bits 7–5) in the CPU control and status register (CSR) as shown in Table 2–1. The modes are:

- **Mapped:** Depending on the memory map selected, the program memory is located at one of these addresses:

  - 0000 0000h–0000 FFFFh for map 1
  - 0140 0000h–0140 FFFFh for map 0

  In mapped mode, program fetches from the internal program memory address return the fetch packet at that address. In the other modes, CPU accesses to this address range return undefined data. Mapped mode is the default state of the internal program memory at reset. The CPU cannot access internal program memory through the data memory controller. (See Chapter 7, *Boot Configuration, Reset, and Memory Map*, for information about how to select the memory map.)

- **Cache enabled:** In cache enabled mode, any initial program fetch at an address causes a cache miss. In a cache miss, the fetch packet is loaded from the external memory interface (EMIF) and stored in the internal cache memory, one 32-bit instruction at a time. While the fetch packet is being loaded, the CPU is halted. The number of wait states incurred depends on the type of external memory used, the state of that memory, and any contention for the EMIF with other requests, such as the DMA controller or a CPU data access. Any subsequent read from a cached address causes a cache hit, and that fetch packet is sent to the CPU from the internal program memory without any wait states. Changing from program memory mode to cache enabled mode flushes the program cache. This mode transition is the only means to flush the cache.

- **Cache freeze:** During a cache freeze, the cache retains its current state. A program read of a frozen cache is identical to a read of an enabled cache except that on a cache miss the data read from the external memory interface is not stored in the cache. A subsequent read of the same address causes a cache miss, and the data is again fetched from external memory.

Cache freeze ensures that critical program data is not overwritten in the cache.

- **Cache bypass:** When the cache is bypassed, any program read fetches data from external memory. The data is not stored in the cache memory. As in cache freeze, the cache retains its state in cache bypass. This mode ensures that external program data is being fetched.

Table 2–1. Internal Program Memory Mode Summary

Internal Program Memory Mode	PCC Value	Description
Mapped	000	Cache disabled (default state at reset)
Cache enabled	010	Cache accessed and updated on reads
Cache freeze	011	Cache accessed but not updated on reads
Cache bypass	100	Cache not accessed or updated on reads
	Other	Reserved

Note:

If you change the operation mode of the PMEMC, you should use the following assembly routine to ensure correct operation of the PMEMC. This routine enables the cache. To change the PMEMC operation mode to a state other than cache enable, you should modify line four of the routine to correspond the the value of PCC that you want moved into B5. For example, to put the cache into mapped mode 0000h should be moved into B5. The CPU registers used in this example have no significance. Any of the registers A0–A15 or B0–B15 can be used in the program.

```
   .align    32
   MVC .S2    CSR,B5    ;copy control status register
|| MVK .S1    0xff1f,A5
   AND .L1x   A5,B5,A5  ;clear PCC field of CSR value
|| MVK  S2    0x0040,B5 ;set cache enable mask
   OR  .L2x   A5,B5,B5  ;set cache enable bit
   MVC .S2    B5,CSR    ;update CSR to enable cache
   NOP 4
   NOP
```

2.2.2 Cache Architecture

The architecture of the cache is directly mapped. The 64K byte cache contains 2K fetch packets, thus, 2K frames. The width of the cache (the frame size) is 256 bits. Each frame in the cache is one fetch packet.

2.2.2.1 Cache Usage of CPU Address

Figure 2–2 shows how the cache uses the fetch packet address from the CPU:

- ❑ 5-bit fetch packet alignment: The five LSBs of the address are assumed to be 0 because all program fetch requests are aligned on fetch packet boundaries (eight words or 32 bytes).
- ❑ 11-bit tag block offset: Because the cache is directly mapped, any external address maps to only one of the 2K frames. Any two fetch packets that are separated by an integer multiple of 64K bytes map to the same frame. Thus, bits 15–5 of the CPU address create the 11-bit block offset that determines which of the 2K frames any particular fetch packet maps to.
- ❑ 10-bit tag: The cache assumes a maximum external address space of 64M bytes (from 0000 0000h–03FF FFFFh). Thus, bits 25–16 of the address correspond to the tag that determines the original location of the fetch packet in external memory space. The cache also has a separate 2K $\times$ 11 tag RAM that holds all the tags. Each address location in this RAM contains a 10-bit tag plus a valid bit that is used to record frame validity information.

Figure 2–2. Logical Mapping of Cache Address

31 26	25 16	15 5	4 0
Outside external range. assumed to be 0	Tag	Block offset	Fetch packet alignment. assumed 0

2.2.2.2 Cache Flush

A dedicated valid bit in each address location of the tag RAM indicates whether the contents of the corresponding cache frame is valid data. During a cache flush, all of the valid bits are cleared to indicate that no cache frames have valid data. Cache flushes occur only at the transition of the internal program memory from mapped mode to cache enabled mode. You initiate this transition by setting the cache enable pattern in the PCC field of the CPU control and status register.

2.2.2.3 Frame Replacement

A cache miss is detected when the tag corresponding to the block offset of the fetch packet address requested by the CPU does not correspond to bits 25–16 of the fetch packet address or if the valid bit at the block offset location is clear. If enabled, the cache loads the fetch packet into the corresponding frame, sets the valid bit, sets the tag to bits 25–16 of the requested address, and delivers this fetch packet to the CPU after all eight instructions are available.

2.3 DMA Controller Access to Program Memory

The DMA controller can read and write to internal program memory when the memory is configured in mapped mode. The CPU always has priority over the DMA controller for access to internal program memory regardless of the value of the PRI bit for that DMA channel. DMA controller accesses are postponed until the CPU stops making requests. To avoid losing future requests that occur after arbitration and while a DMA controller access is in progress, the CPU incurs one wait state per DMA controller access. The maximum throughput to the DMA is one access every other cycle. In a cache mode, a DMA controller write is ignored by the program memory controller, and a read returns an undefined value. For both DMA reads and writes in cache modes, the DMA controller is signaled that its request has finished. At reset, the program memory system is in mapped mode, allowing the DMA controller to boot load code into the internal program memory.

See Chapter 7, *TMS320C6000 Boot Modes,* for more information on bootloading code.

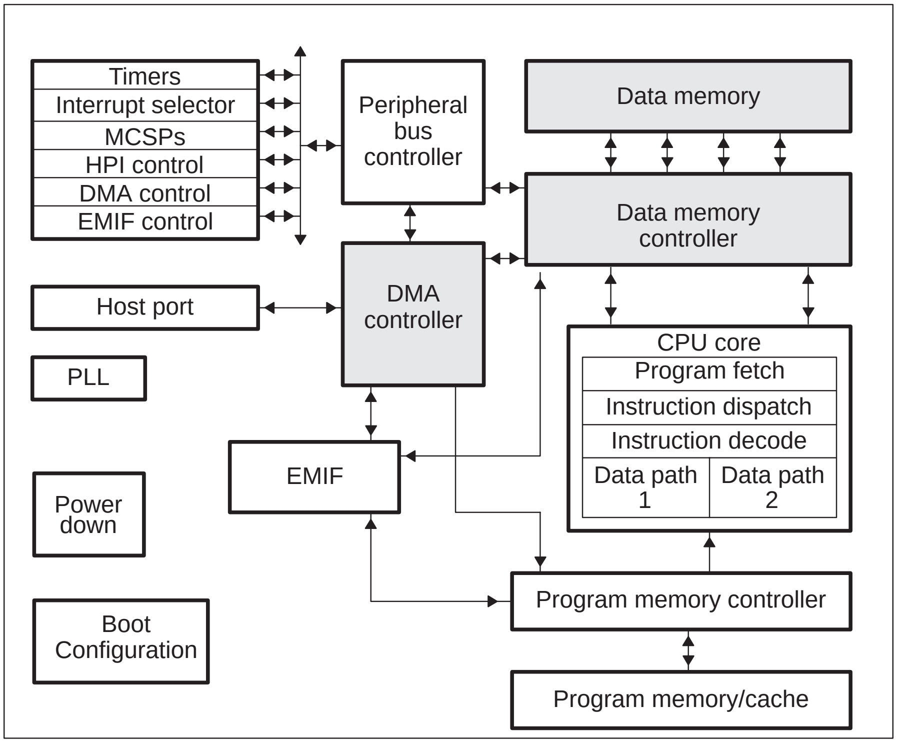
2.4 Data Memory Controller

As shown inFigure 2–3, the data memory controller connects:

- ❑ The CPU and direct memory access (DMA) controller to internal data memory and performs the necessary arbitration.
- ❑ CPU to the external memory interface (EMIF).
- ❑ The CPU to the on chip peripherals through the peripheral bus controller.

The peripheral bus controller performs arbitration between the CPU and DMA for the on-chip peripherals.

Figure 2–3. TMS320C6x Block Diagram



2.5 Data Memory Access

The data memory controller services all CPU and DMA controller data requests to internal data memory. Figure 2–4, Figure 2–5, and Figure 2–6 show the directions of data flow and the master (requester) and slave (resource) relationships between the modules:

- The CPU requests data reads and writes to:
  - Internal data memory
  - On-chip peripherals through the peripheral bus controller
  - EMIF
- The DMA controller requests reads and writes to internal data memory.
- The CPU cannot access internal program memory through the data memory controller.

The CPU sends requests to the data memory controller through the two address buses (DA1 and DA2). Store data is transmitted through the CPU data store buses (ST1 and ST2). Load data is received through the CPU data load buses (LD1 and LD2). The CPU data requests are mapped, based on address, to either the internal data memory, internal peripheral space (through the peripheral bus controller), or the external memory interface. The data memory controller also connects the DMA controller to the internal data memory and performs arbitration between the CPU and DMA controller.

2.6 Internal Data Memory Organization

The following sections describe the memory organization of each device in the ’C6x generation of DSPs ’C6201 and ’C6701 devices.

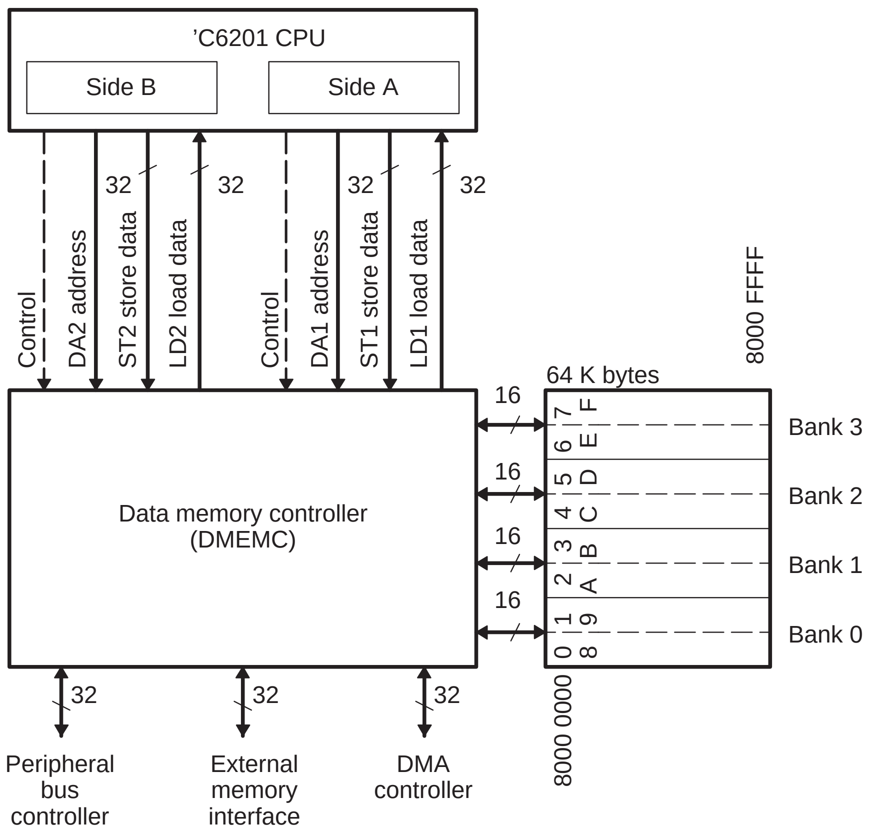
2.6.1 TMS320C6201 Revision 2

The 64K bytes of internal data RAM are organized as one block of 64K bytes located from address 8000 0000h to 8000 FFFFh. This block is organized as four 8K banks of 16-bit halfwords. Both the CPU and the DMA controller can simultaneously access data that resides in different banks. This organization allows the two CPU data ports, A and B, to simultaneously access neighboring 16-bit data elements inside the block without a resource conflict.

Table 2–2. Data Memory Organization (TMS320C6201 Revision 2)

	Bank 0		**Bank 1**		**Bank 2**		**Bank 3**	
First address	80000000	80000001	80000002	80000003	80000004	80000005	80000006	80000007
	80000008	80000009	8000000A	8000000B	8000000C	8000000D	8000000E	8000000F
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	8000FFF0	8000FFF1	8000FFF2	8000FFF3	8000FFF4	8000FFF5	8000FFF6	8000FFF7
Last address	8000FFF8	8000FFF9	8000FFFA	8000FFFB	8000FFFC	8000FFFD	8000FFFE	8000FFFF

Figure 2–4. Data Memory Controller Interconnect to Other Banks (TMS320C6201 Revision 2)



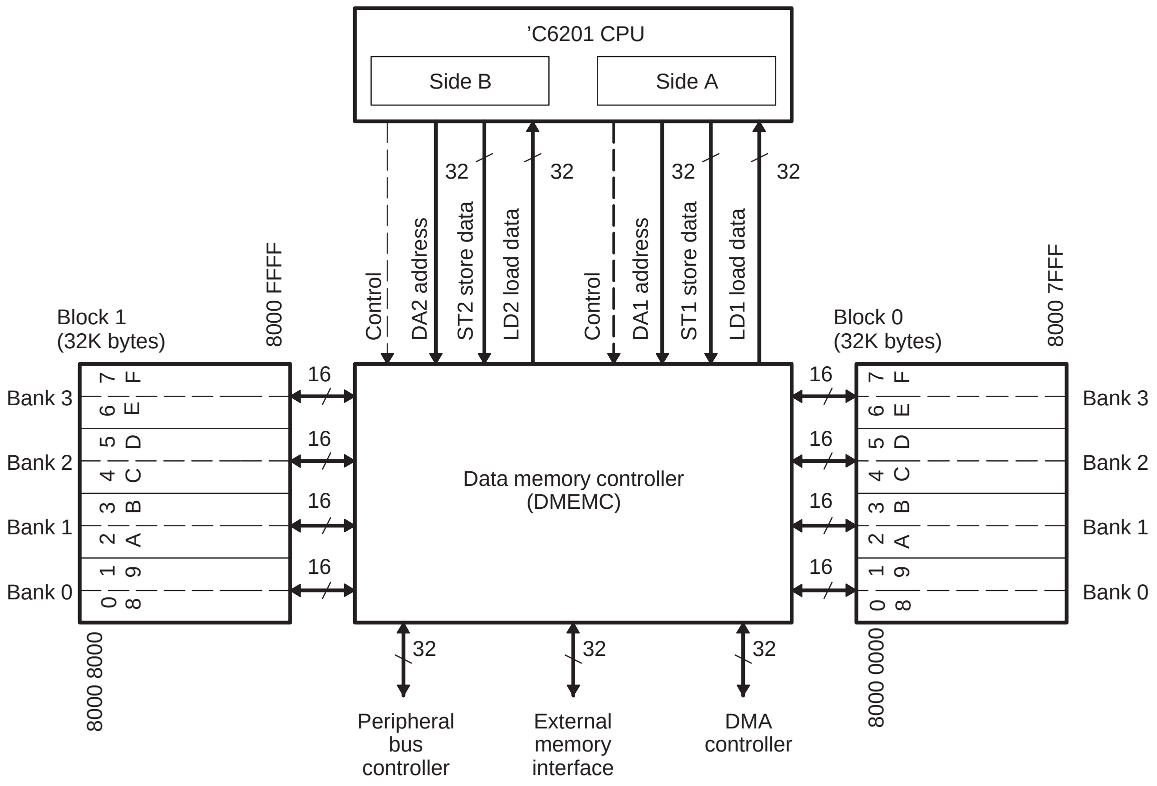
2.6.2 TMS320C6201 Revision 3

The 64K bytes of internal data RAM are organized as two blocks of 32K bytes located from address 8000 0000h to 8000 7FFFh and 8000 8000h to 8000 FFFFh. The DMA controller or side A and side B of the CPU can simultaneously access any portion of the internal memory without conflict, when using different blocks. Both blocks are organized as four 4K banks of 16-bit halfwords. Therefore you do not have to consider the address within a block if simultaneous accesses occur to different blocks. Accesses to different blocks never cause performance penalties. Both CPU and DMA can still simultaneously access data that resides in different banks within the same block without a performance penalty. To avoid performance penalties, you have to pay attention to address LSBs when the two accesses involve data in the same block. This organization also allows the two CPU data ports, A and B, to simultaneously access neighboring 16-bit data elements inside the block without a resource conflict.

Table 2–3. Data Memory Organization (TMS320C6201 Revision 3)

	Bank 0		**Bank 1**		**Bank 2**		**Bank 3**	
First address	80000000	80000001	80000002	80000003	80000004	80000005	80000006	80000007
(Block 0)	80000008	80000009	8000000A	8000000B	8000000C	8000000D	8000000E	8000000F
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	80007FF0	80007FF1	80007FF2	80007FF3	80007FF4	80007FF5	80007FF6	80007FF7
Last address	80007FF8	80007FF9	80007FFA	80007FFB	80007FFC	80007FFD	80007FFE	80007FFF
(Block 0)								
First address	80008000	80008001	80008002	80008003	80008004	80008005	80008006	80008007
(Block 1)	80008008	80008009	8000800A	8000800B	8000800C	8000800D	8000800E	8000800F
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	8000FFF0	8000FFF1	8000FFF2	8000FFF3	8000FFF4	8000FFF5	8000FFF6	8000FFF7
Last address	8000FFF8	8000FFF9	8000FFFA	8000FFFB	8000FFFC	8000FFFD	8000FFFE	8000FFFF
(Block 1)								

Figure 2–5. Data Memory Controller Interconnect to Other Banks (TMS320C6201 Revision 3)



2.6.3 TMS320C6701

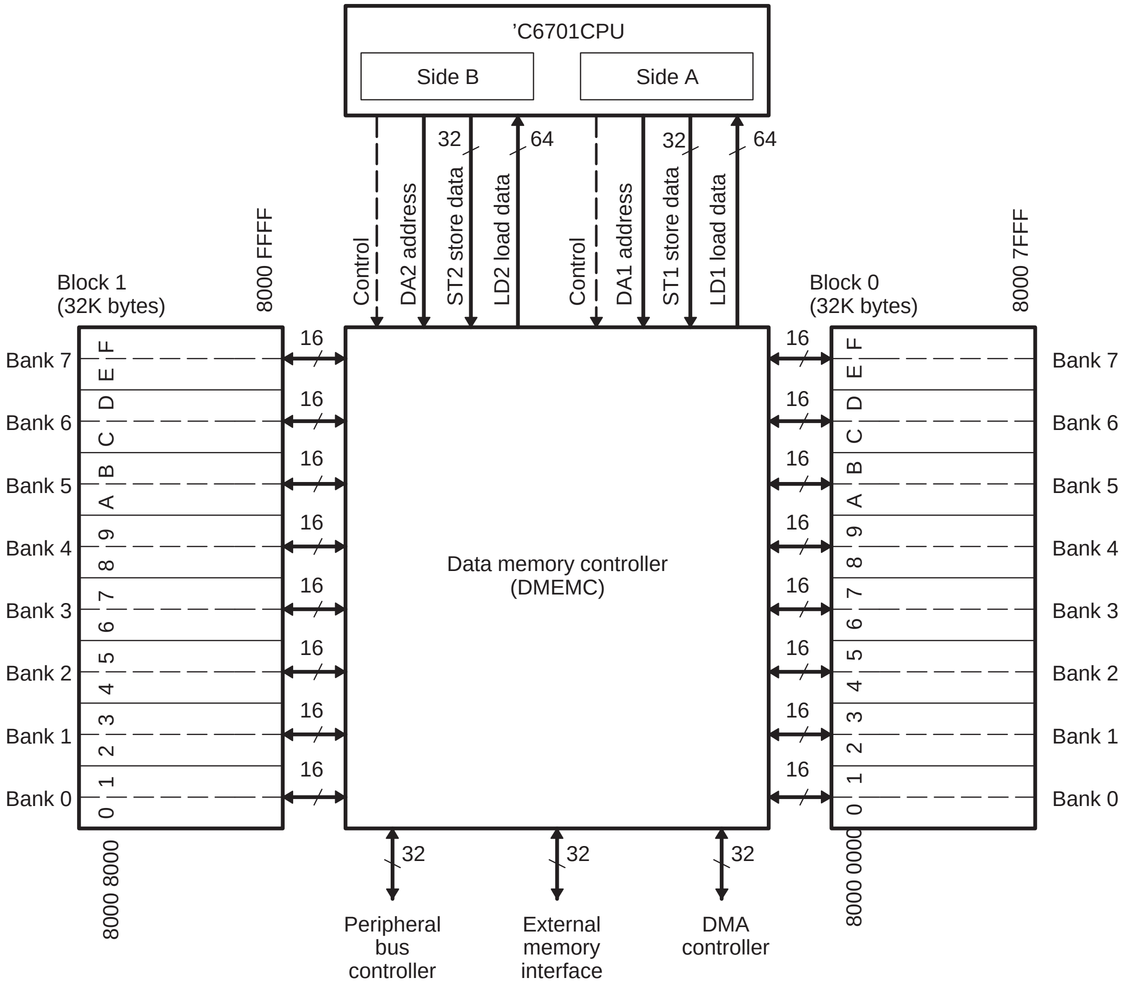
The 64K bytes of internal data RAM are organized as two blocks of 32K bytes located from address 8000 0000h to 8000 7FFFh and 8000 8000h to 8000 FFFFh. Side A and side B of the CPU or the DMA Controller can simultaneously access any portion of the internal data memory without conflict, when using different blocks. Therefore, you do not have to consider the address within a block if simultaneous accesses occur to different blocks. Accesses to different blocks never cause performance penalties. You only have to pay attention to the address when the two accesses occur in different blocks. Both blocks are organized as eight 2K banks of 16-bit halfwords. Both the CPU and DMA controller can still simultaneously access data that resides in different banks within the same block without performance penalty. To avoid performance penalties, you have to pay attention to address LSBs when two accesses involve data in the same block. This organization also allows the two CPU data ports, A and B, to simultaneously access neighboring 16-bit data elements inside the same block without a resource conflict.

Table 2–4. Data Memory Organization

	Bank 0		**Bank 1**		**Bank 2**		**Bank 3**	
First address (Block 0)	80000000	80000001	80000002	80000003	80000004	80000005	80000006	80000007
Last address (Block 0)	80007FF0	80007FF1	80007FF2	80007FF3	80007FF4	80007FF5	80007FF6	80007FF7
	Bank 4		**Bank 5**		**Bank 6**		**Bank 7**	
First address (Block 0)	80000008	80000009	8000000A	8000000B	8000000C	8000000D	8000000E	8000000F
Last address (Block 0)	80007FF8	80007FF9	80007FFA	80007FFB	80007FFC	80007FFD	80007FFE	80007FFF

	Bank 0		**Bank 1**		**Bank 2**		**Bank 3**	
First address (Block 1)	80008000	80008001	80008002	80008003	80008004	80008005	80008006	80008007
Last address (Block 1)	8000FFF0	8000FFF1	8000FFF2	8000FFF3	8000FFF4	8000FFF5	8000FFF6	8000FFF7
	Bank 4		**Bank 5**		**Bank 6**		**Bank 7**	
First address (Block 1)	80008008	80008009	8000800A	8000800B	8000800C	8000800D	8000800E	8000800F
Last address (Block 1)	8000FFF8	8000FFF9	8000FFFA	8000FFFB	8000FFFC	8000FFFD	8000FFFE	8000FFFF

Figure 2–6. Data Memory Controller Interconnect to Other Blocks (TMS320C6701)



2.6.4 Data Alignment

The following data alignment restrictions apply:

Doublewords: (’C6701 only) Doublewords are aligned on even 8-byte (doubleword) boundaries, and always start at a byte address where the three LSBs are 0. Doublewords are used only on loads triggered by the LDDW instruction. Store operations do not use doublewords.

Words: Words are aligned on even 4-byte (word) boundaries, and always start at a byte address where the two LSBs are 0. A word access requires two adjacent 16-bit-wide banks.

Halfwords: Halfwords are aligned on even 2-byte (halfword) boundaries, and always start at byte addresses where the LSB is 0. Halfword accesses require the entire 16-bit-wide bank.

Bytes: There are no alignment restrictions on byte accesses.

2.6.5 Dual CPU Accesses to Internal Memory

Both the CPU and DMA can read and write 8-bit bytes, 16-bit halfwords, and 32-bit words. The data memory controller performs arbitration individually for each 16-bit bank. Although arbitration is performed on 16-bit-wide banks, the banks have byte enables to support byte-wide accesses. However, a byte access prevents the entire 16 bits containing the byte from simultaneously being used by another access.

As long as multiple requesters access data in separate banks, all accesses are performed simultaneously with no penalty. Also, when two memory accesses involve separate 32K byte memory blocks, there are no memory conflicts, regardless of the address. For multiple data accesses within the same block, the memory organization also allows simultaneous multiple memory accesses as long as they involve different banks. In one CPU cycle, two simultaneous accesses to two different internal memory banks occur without wait states. Two simultaneous accesses to the same internal memory bank stall the entire CPU pipeline for one CPU clock, providing two accesses in two CPU clocks. These rules apply regardless of whether the accesses are loads or stores.

Loads and stores from the same execute packet are seen by the data memory controller during the same CPU cycle. Loads and stores from future or previous CPU cycles do not cause wait states for the internal data memory accesses in the current cycle. Thus, internal data memory access causes a wait state only when a conflict occurs between instructions in the same fetch packet accessing the same 16-bit wide bank. This conflict is an internal memory conflict. The data memory controller stalls the CPU for one CPU clock, serializes the accesses, and performs each access separately. In prioritizing the two accesses, any load occurs before any store access. A load in parallel with a store always has priority over the store. If both the load and the store access the same resource (for example, the EMIF, or peripheral bus, internal memory block), the load always occurs before the store. If both accesses are stores, the access from DA1 takes precedence over the access from DA2. If both accesses are loads, the access from DA2 takes precedence over the access from DA1. Figure 3–3 shows what access conditions cause internal memory conflicts when the CPU makes two data accesses (on DA1 and DA2).

Figure 2–7. Conflicting Internal Memory Accesses to the Same Block (TMS320C6201 Revisions 2 and 3)

	DA1	**Byte**								**Halfword**				**Word**	
DA2	2:0	000	001	010	011	100	101	110	111	000	010	100	110	000	100
Byte	000	■	■							■				■	
	001	■	■							■				■	
	010			■	■						■			■	
	011			■	■						■			■	
	100					■	■					■			■
	101					■	■					■			■
	110							■	■				■		■
	111							■	■				■		■
Halfword	000	■	■							■				■	
	010			■	■						■			■	
	100					■	■					■			■
	110							■	■				■		■
Word	000	■	■	■	■					■	■			■	
	100					■	■	■	■			■	■		■

Note: Conflicts shown in shaded areas.

Figure 2–8. Conflicting Internal Memory Accesses to the Same Block (TMS320C6701)

	DA1	**Byte**																**Halfword**								**Word**				**Double -word**	
DA2	**3–0**	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0010	0100	0110	1000	1010	1100	1110	0000	0100	1000	1100	0000	1000
Byte	0000	■	■															■								■				■	
	0001	■	■															■								■				■	
	0010			■	■														■							■				■	
	0011			■	■														■							■				■	
	0100					■	■													■							■			■	
	0101					■	■													■							■			■	
	0110							■	■												■						■			■	
	0111							■	■												■						■			■	
	1000									■	■											■						■			■
	1001									■	■											■						■			■
	1010											■	■										■					■			■
	1011											■	■										■					■			■
	1100													■	■									■					■		■
	1101													■	■									■					■		■
	1110															■	■								■				■		■
	1111															■	■								■				■		■
Halfword	0000	■	■															■								■				■	
	0010			■	■														■							■				■	
	0100					■	■													■							■			■	
	0110							■	■												■						■			■	
	1000									■	■											■						■			■
	1010											■	■										■					■			■
	1100													■	■									■					■		■
	1110															■	■								■				■		■
Word	0000	■	■	■	■													■	■							■				■	
	0100					■	■	■	■											■	■						■			■	
	1000									■	■	■	■									■	■					■			■
	1100													■	■	■	■							■	■				■		■
DW	0000	■	■	■	■	■	■	■	■									■	■	■	■					■	■			■	
	1000									■	■	■	■	■	■	■	■					■	■	■	■			■	■		■

Note: Conflicts shown in shaded areas.

2.6.6 DMA Accesses to Internal Memory

The DMA controller can accesss any portion of one block of internal data memory while the CPU is simultaneously accessing any portion of another block. If both the CPU and the DMA controller are accessing the same block, and portions of both accesses are to the same 16-bit bank, the DMA operation can take place first or last, depending on the CPU/DMA priority settings. You can use Figure 3–3 to determine DMA versus CPU conflicts. Assume that one axis represents the DMA access and the other represents the CPU access from one CPU data port. Then, perform this analysis again for the other data port. If both comparisons yield no conflict, then there is no CPU/DMA internal memory conflict. If either comparison yields a conflict, then there is a CPU/DMA internal memory conflict. In this case, the priority is resolved by the PRI bit of the DMA channel as described in Chapter 4, *TMS320C6211 Two-Level Internal Memory*. If the DMA channel is configured as higher priority than the CPU (PRI = 1), any CPU accesses are postponed until the DMA accesses finish and the CPU incurs a 1-CPU-clock wait state. If both CPU ports and the DMA access the same memory block, the number of wait states increases to two. If the DMA has multiple consecutive requests to the block required by the CPU, the CPU is held off until all DMA accesses to the necessary blocks finish. In contrast, if the CPU has higher priority (PRI = 0), then the DMA access is postponed until the both CPU data ports stop accessing that bank. In this configuration, a DMA access request never causes a wait state.

2.6.7 Data Endianness

Two standards for data ordering in byte-addressable microprocessors exist:

- ❑ Little-endian ordering, in which bytes are ordered from right to left, the most significant byte having the highest address
- ❑ Big-endian ordering, in which bytes are ordered from left to right, the most significant byte having the lowest address

Both the CPU and the DMA controller support a programmable endianness. This endianness is selected by the LENDIAN pin on the device. LENDIAN = 1 selects little endian, and LENDIAN big. Byte ordering within word and half word data resident in memory is identical for little-endian and big-endian data. Table 2–5 shows which bits of a data word in memory are loaded into which bits of a destination register for all possible CPU data loads from big- or little-endian data. The data in memory is assumed to be the same data that is in the register results from the LDW instruction in the first row. Table 2–7 and Table 2–8 show which bits of a register are stored in which bits of a destination memory word for all possible CPU data stores from big- and little-endian data. The data in the source register is assumed to be the same data that is in the memory results from the STW instruction in the first row.

Table 2–5. Register Contents After Little-Endian or Big-Endian Data Loads (TMS320C6201 and TMS320C6701)

Instruction	Address Bits (1:0)	Big-Endian Register Result	Little-Endian Register Result
LDW	00	BA98 7654h	BA98 7654h
LDH	00	FFFF BA98h	0000 7654h
LDHU	00	0000 BA98h	0000 7654h
LDH	10	0000 7654h	FFFF BA98h
LDHU	10	0000 7654h	0000 BA98h
LDB	00	FFFF FFBAh	0000 0054h
LDBU	00	0000 00BAh	0000 0054h
LDB	01	FFFF FF98h	0000 0076h
LDBU	01	0000 0098h	0000 0076h
LDB	10	0000 0076h	FFFF FF98h
LDBU	10	0000 0076h	0000 0098h
LDB	11	0000 0054h	FFFF FFBAh
LDBU	11	0000 0054h	00000 0BAh

Note: The contents of the word in data memory at location xxxx xx00 is BA98 7654h.

Table 2–6. Register Contents After Little-Endian or Big-Endian Data Loads (TMS320C6701 only)

Instruction	**Address Bits (2:0)**	**Big-Endian Memory Result**	**Little-Endian Memory Result**
LDDW ('C6701 only)	000	FEDC BA98 7654 3210h	FEDC BA98 7654 3210h
LDW	000	FEDC BA98h	7654 3210h
LDW	100	7654 3210h	FEDC BA98h

Note: The contents of the doubleword in data memory at location xxxx x000 before the ST instruction executes is FEDC BA98 7654 3210h.

Table 2–7. Memory Contents After Little-Endian or Big-Endian Data Stores (TMS320C6201/C6701)

Instruction	**Address Bits (1:0)**	**Big-Endian Memory Result**	**Little-Endian Memory Result**
STW	00	BA98 7654h	BA98 7654h
STH	00	7654 1970h	0112 7654h
STH	10	0112 7654h	7654 1970h
STB	00	5412 1970h	0112 1954h
STB	01	0154 1970h	0112 5470h
STB	10	0112 5470h	0154 1970h
STB	11	0112 1954h	5412 1970h

Note: The contents of the word in data memory at location xxxx xx00 before the ST instruction executes is 0112 1970h. The contents of the source register is BA98 7654h.

2.7 Peripheral Bus

The peripherals are controlled by the CPU and the DMA controller through accesses of control registers. The CPU and the DMA controller access these registers through the peripheral data bus. The DMA controller directly accesses the peripheral bus controller, whereas the CPU accesses it through the data memory controller.

2.7.1 Byte and Halfword Access

The peripheral bus controller converts all peripheral bus accesses to word accesses. However, on read accesses both the CPU and the DMA controller can extract the correct portions of the word to perform byte and halfword accesses properly. Any side-effects caused by a peripheral control register read occur regardless of which bytes are read. In contrast, for byte or halfword writes, the values the CPU and the DMA controller only provide correct values in the enabled bytes. The values that are always correct are shown in Table 2–8. Undefined results are written to the nonenabled bytes. If you are not concerned about the values in the disabled bytes, this is acceptable. Otherwise, access the peripheral registers only via word accesses.

Table 2–8. Memory Contents After Little-Endian or Big-Endian Data Stores

Access Type	**Address Bits (1:0)**	**Big-Endian Register**	**Little-Endian Memory Result**
Word	00	XXXXXXXX	XXXXXXXX
Halfword	00	XXXX????	????XXXX
Halfword	10	????XXXX	XXXX????
Byte	00	XX??????	??????XX
Byte	01	??XX????	????XX??
Byte	10	????XX??	??XX????
Byte	11	??????XX	XX??????

Note: X indicates nybbles correctly written, ? indicates nybbles with undefined value after write

2.7.2 CPU Wait States

Isolated peripheral bus controller accesses from the CPU causes six CPU wait states. These wait states are inserted to allow pipeline registers to break up the paths between traversing the on-chip distances between the CPU and peripherals as well as for arbitration time.

2.7.3 Arbitration Between the CPU and the DMA Controller

As shown in Figure 2–5 and Figure 2–6, the peripheral bus controller performs arbitration between the CPU and the DMA controller for the peripheral bus. Like internal data access, the PRI bits in the DMA controller determine the priority between the CPU and the DMA controller. If a conflict occurs between the CPU (via the data memory controller) the lower priority requester is held off until the higher priority requester completes all accesses to the peripheral bus controller. The peripheral bus is arbitrated as a single resource, so the lower priority resource is blocked from accessing all peripherals, not just the one accessed by the higher priority requester.

Chapter 3

TMS320C6202 Program and Data Memory

This chapter describes the TMS320C6202 program memory and data memory controller. Program memory modes including cache operation and bootload operation are discussed.

Topic **Page**

3.1 TMS320C6202 Program Memory Controller

The TMS320C6202 program memory controller (PMEMC) provides all of the functionality available in the TMS320C6201 revision 3. The PMEMC operates as either a 128K byte memory or direct-mapped cache. In addition to the memory/cache, the C6202 provides 128K bytes of memory that operates as a memory-mapped block. To achieve this functionality, the block of program memory has been expanded to 128K bytes. A second 128K byte block of program memory has been added. These two blocks can be accessed independently, allowing for program fetch from one block by the CPU to occur in parallel and without interfering with a DMA transfer with the other block of program memory. Table 3–1 and Table 3–2 compare the internal memory and cache configurations available on the current TMS320C6000 devices. Figure 3–1 shows a block diagram of the connections between the C6202 CPU, PMEMC, and memory blocks. The addresses shown in Figure 3–1 are for operation in memory map mode 1.

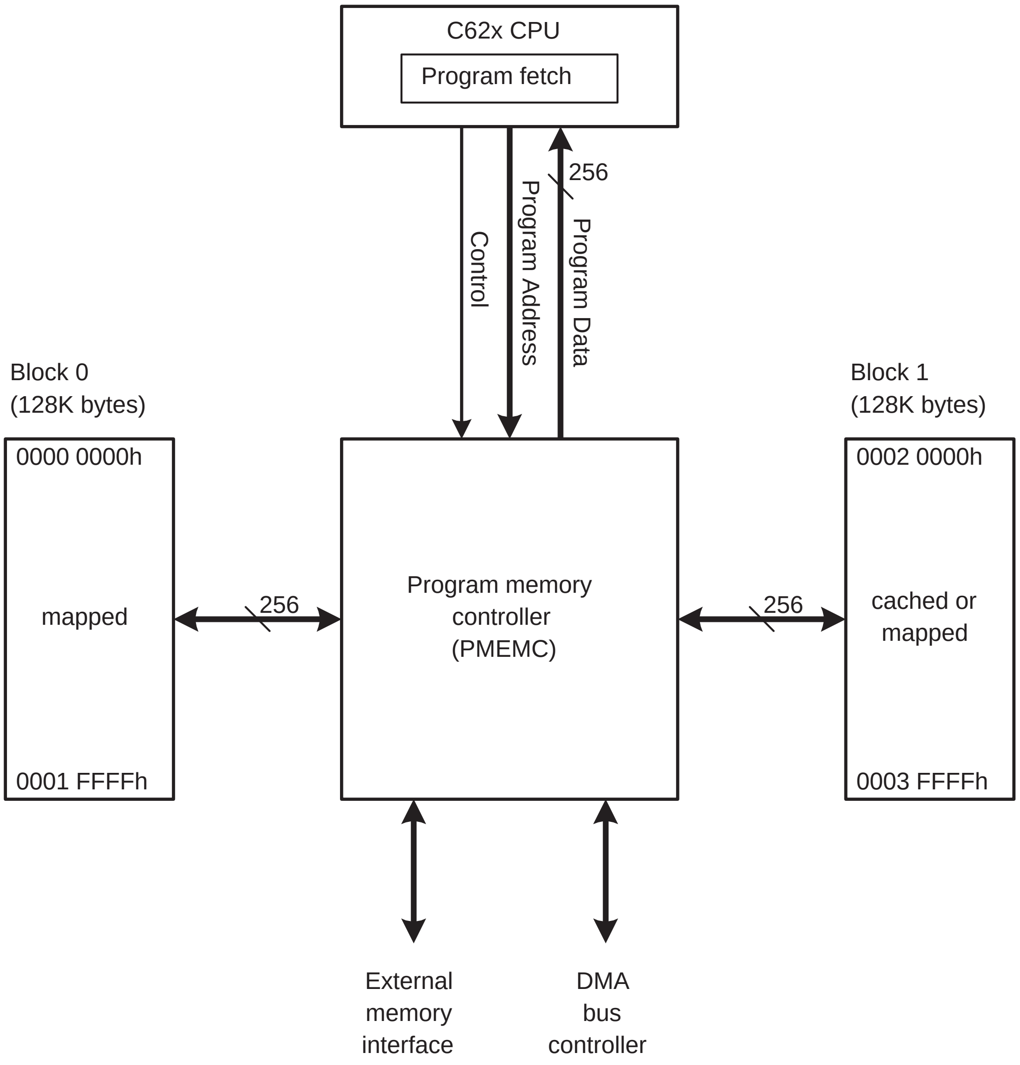
Table 3–1. TMS320C6201/C6701/C6202 Internal Memory Configurations

Device	CPU	Internal Memory Architecture	Total Memory (Bytes)	Program Memory (Bytes)	Data Memory (Bytes)
’C6201	6200	Harvard	128K	64K (map/cache)	64K (map)
’C6701	6700	Harvard	128K	64K (map/cache)	64K (map)
’C6202	6200	Harvard	384K	128K (map) 128K (map/cache)	128K (map)

Table 3–2. TMS320C6201/C6701/C6202 Cache Architectures

Cache Space	Size (Bytes)	Associativity	Line Size (Bytes)
’C6201 program	64K	Direct mapped	32
’C6701 program	64K	Direct mapped	32
’C6202 program	128K	Direct mapped	32

Figure 3–1. TMS320C6202 Program Memory Controller Block Diagram



3.2 Memory Mapped Operation

When the PCC field of the CPU control status register is programmed for Mapped mode, both blocks of internal program RAM are mapped into internal program space. Table 3–3 shows the address space for both blocks of RAM for the map mode selected at device reset.

Table 3–3. Internal Program RAM Address Mapping in Memory Mapped Mode

	Map 0	Map 1
Block 0	0140 0000h – 0141 FFFFh	0000 0000h – 0x0001 FFFFh
Block 1	0142 0000h – 0143 FFFFh	0002 0000h – 0x0003 FFFFh

In mapped mode, both the CPU and the DMA can access all locations in both blocks of RAM. Any access outside of the address space that the internal RAM is mapped to is forwarded to the EMIF. The DMA can only access one of the two blocks of RAM at a time. The CPU and DMA can access the internal RAM without interference as long as each accesses a different block. If the CPU and DMA attempt to access the same block of RAM at the same time, then the DMA is stalled until the CPU completes its accesses to that block. After the CPU access is complete, the DMA is allowed to access the RAM. The DMA cannot cross between Block 0 and Block 1 in a single transfer. You must use separate DMA transfers to cross block boundaries.

3.3 Cache Operation

When the PCC field of the CPU Control Status Register is programmed for one of the Cache modes, block 1 operates as a cache while block 0 remains mapped into internal program space. Table 3–4 shows the addresses occupied by the RAM that is not used for cache, for each Map Mode.

Table 3–4. Internal Program RAM Address Mapping in Cache Mode

	Map 0	Map 1
Block 0	0140 0000h – 0141 FFFFh	0000 0000h – 0001 FFFFh

The cache on the C6202 operates identically to the C6201 cache. Any CPU or DMA access to the memory range that was occupied by the cache RAM returns undefined results. As in mapped mode, simultaneous accesses to block 0 by the CPU and DMA stalls the DMA until the CPU has completed its access. A DMA access to block 0 while the cache is flushed continues without stalling. The CPU is halted during a cache flush. You must ensure that all DMA accesses to block 1 have completed before the cache is enabled.

Note:

If you change the operation mode of the PMEMC, you should use the following assembly routine to ensure correct operation of the PMEMC. This routine enables the cache. To change the PMEMC operation mode to a state other than cache enable, you should modify line four of the routine to correspond the the value of PCC that you want moved into B5. For example, to put the cache into mapped mode 0000h should be moved into B5. The CPU registers used in this example have no significance. Any of the registers A0–A15 or B0–B15 can be used in the program.

```
   .align    32
   MVC .S2    CSR,B5    ;copy control status register
|| MVK .S1    0xff1f,A5
   AND .L1x   A5,B5,A5  ;clear PCC field of CSR value
|| MVK  S2    0x0040,B5 ;set cache enable mask
   OR  .L2x   A5,B5,B5  ;set cache enable bit
   MVC .S2    B5,CSR    ;update CSR to enable cache
   NOP 4
   NOP
```

3.4 Bootload Operation

The ’C6202 bootload operates identically to the C6201 revision 3. During ROM bootload, a 64K byte block of data is transferred from the beginning of CE1 to memory at address 0. During HPI bootload, the host can read or write any internal or external memory location, including the entire internal program space.

3.5 TMS320C6202 Data Memory Controller

The TMS320C6202 data memory controller (DMEMC) provides all of the functionality available in the TMS320C6201 revision 3. The C6202 DMEMC contains 128K bytes of RAM organized in two blocks of four banks each. Each bank is 16 bits wide. The DMEMC for the C6202 operates identically to the C6201 DMEMC, the DMA controller or side A or side B of the CPU can simultaneously access two different banks without conflict. Figure 3–2 shows a block diagram of the connections between the C6202 CPU, DMEMC, and memory blocks. Table 3–5 shows the memory range occupied by each block of internal data RAM.

Figure 3–2. TMS320C6202 Data Memory Controller Block Diagram

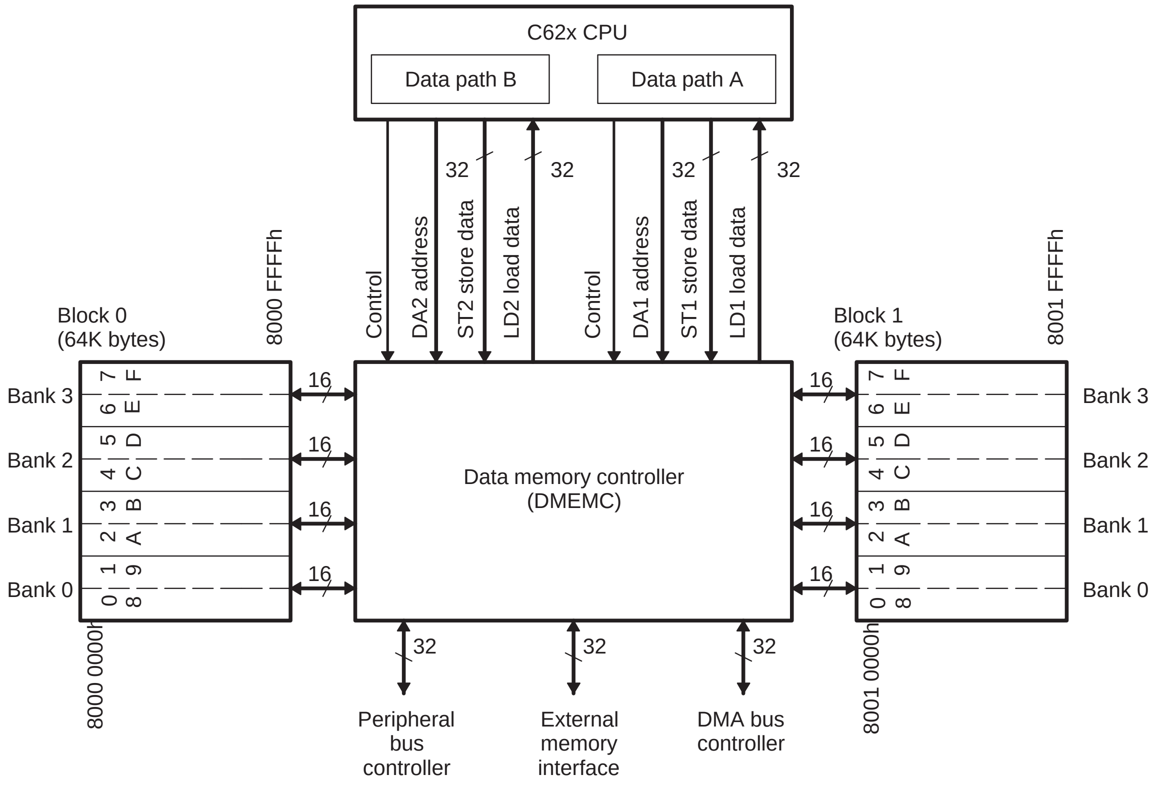


Table 3–5. Internal Data RAM Address Mapping

Block 0	8000 0000h – 8000 FFFFh
Block 1	8001 0000h – 8001 FFFFh

Chapter 4

TMS320C6211/C6711 Two-Level Internal Memory

The TMS320C6211/C6711 provides a two level memory architecture for the internal program and data busses. The first level memory for both the internal program and data bus is a 4K byte cache, designated L1P for the program cache and L1D for the data cache. The second level memory is a 64K byte memory block that is shared by both the program and data memory buses, designated L2.

Topic **Page**

4.1 Overview

Figure 4–1 illustrates how the L1P, L1D, and L2 are arranged in the TMS320C6211/C6711. Figure 4–2 illustrates the bus connections between the CPU, internal memories, and the enhanced DMA for the ’C6211, and.

Figure 4–1. TMS320C6211/C6711 Block Diagram

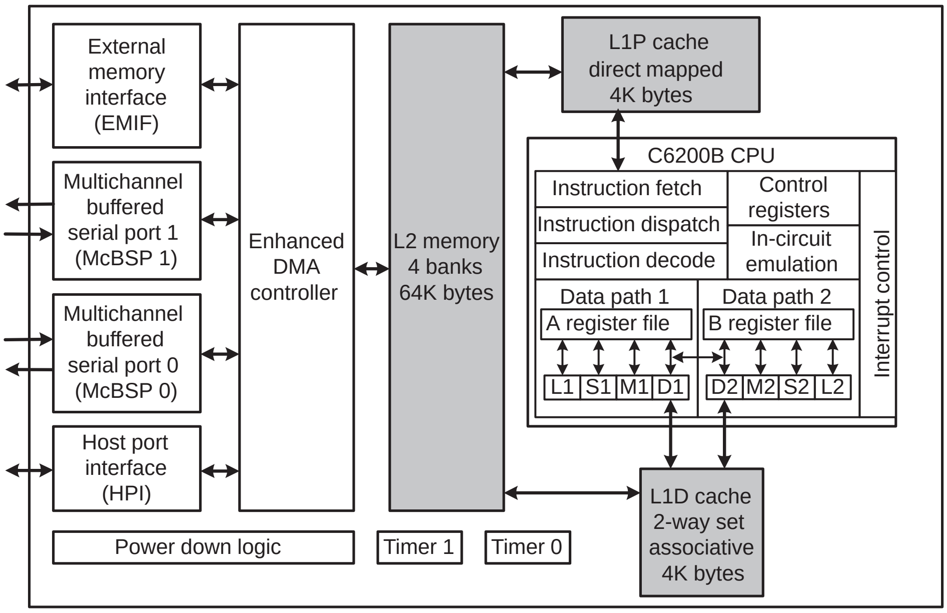

External memory interface (EMIF)
Multichannel buffered serial port 1 (McBSP 1)
Multichannel buffered serial port 0 (McBSP 0)
Host port interface (HPI)
Enhanced DMA controller
L2 memory 4 banks 64K bytes
L1P cache direct mapped 4K bytes
C6200B CPU
Instruction fetch
Instruction dispatch
Instruction decode
Control registers
In-circuit emulation
Interrupt control
Data path 1
A register file
L1 S1 M1 D1
Data path 2
B register file
D2 M2 S2 L2
L1D cache 2-way set associative 4K bytes
Power down logic
Timer 1
Timer 0

Table 4–1. TMS320C6211/C6711 Internal Memory Configurations

Device	CPU	Internal Memory Architecture	Total Memory (Bytes)	Program Memory (Bytes)	Data Memory (Bytes)	Unified Memory (Bytes)
’C6211/ C6711	6200	Harvard (L1) Unified (L2)	72K	4K (cache)	4K (cache)	64K (map/cache)

Table 4–2. TMS320C6211/C6711 Cache Architectures

Cache Space	Size (Bytes)	Associativity	Line Size (Bytes)
L1P	4K	Direct mapped	64
L1D	4K	2-way	32
L2	64K	1- to 4-way	128

Figure 4–2. TMS320C6211 Internal Memory Block Diagram

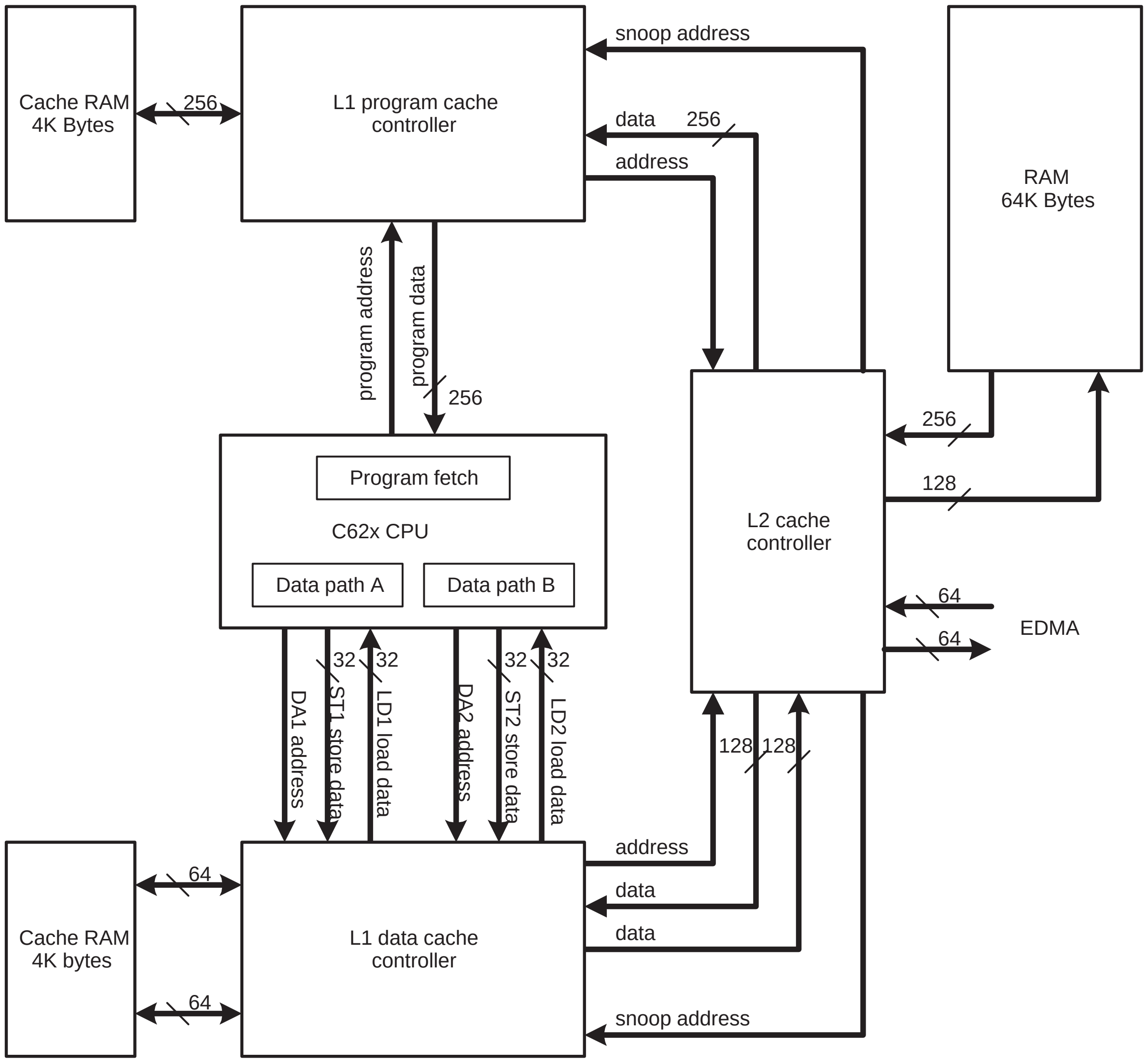
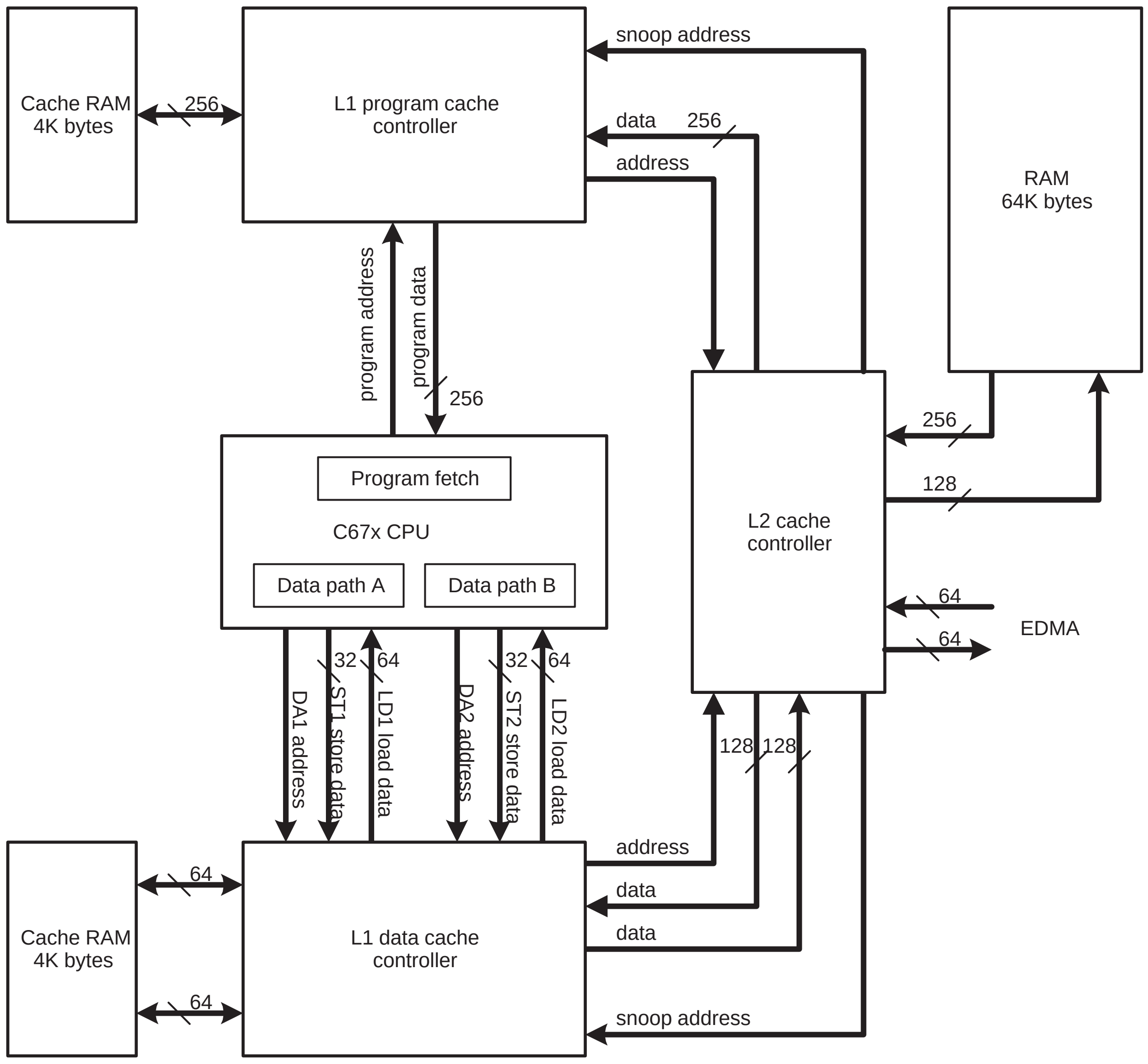


Figure 4–3. TMS320C6711 Internal Memory Block Diagram



4.2 Internal Memory Control Registers

The L1P, L1D, and L2 are controlled by a set of memory configuration registers. The CPU can read and write to the internal memory control registers. The EDMA (and thus the HPI) can only read these registers. Table 4–3 lists these control registers and their associated addresses. You should initialize a memory attribute register by setting the appropriate MAR bits, and then read that memory attribute register before continuing execution to insure proper operation.

Table 4–3. Internal Memory Control Register Addresses

Register Address (byte)	Register Mnemonic	Register Name
0184 0000h	CCFG	Cache configuration register
0184 4000h	L2FBAR	L2 flush base address register
0184 4004h	L2FWC	L2 flush word count register
0184 4010h	L2CBAR	L2 clean base address register
0184 4014h	L2CWC	L2 clean word count register
0184 4020h	L1PFBAR	L1P flush base address register
0184 4024h	L1PFWC	L1P flush word count register
0184 4030h	L1DFBAR	L1D flush base address register
0184 4034h	L1DFWC	L1D flush word count register
0184 5000h	L2FLUSH	L2 flush register
0184 5004h	L2CLEAN	L2 clean register
0184 8200h	MAR0	Memory attribute register – Region 0
0184 8204h	MAR1	Memory attribute register – Region 1
0184 8208h	MAR2	Memory attribute register – Region 2
0184 820Ch	MAR3	Memory attribute register – Region 3
0184 8240h	MAR4	Memory attribute register – Region 4
0184 8244h	MAR5	Memory attribute register – Region 5
0184 8248h	MAR6	Memory attribute register – Region 6
0184 824Ch	MAR7	Memory attribute register – Region 7
0184 8280h	MAR8	Memory attribute register – Region 8
0184 8284h	MAR9	Memory attribute register – Region 9
0184 8288h	MAR10	Memory attribute register – Region 10
0184 828Ch	MAR11	Memory attribute register – Region 11
0184 82C0h	MAR12	Memory attribute register – Region 12
0184 82C4h	MAR13	Memory attribute register – Region 13
0184 82C8h	MAR14	Memory attribute register – Region 14
0184 82CCh	MAR15	Memory attribute register – Region 15

4.3 L1P Description

The L1P is organized as a 64 line direct mapped cache with a 64 byte (2 fetch packet) line size. The L1P data request size is one line, thus the six least significant bits of a requested address are ignored. The next six bits of the address are used to reference the set within the cache that the addressed data maps to. The remaining bits of the address are used as a unique tag for the requested data. Figure 4–4 illustrates how a 32 bit address is allocated to provide the set index and tag data for the L1P.

Figure 4–4. L1P Address Allocation

31 12	11 6	5 0
Tag	Set	Offset

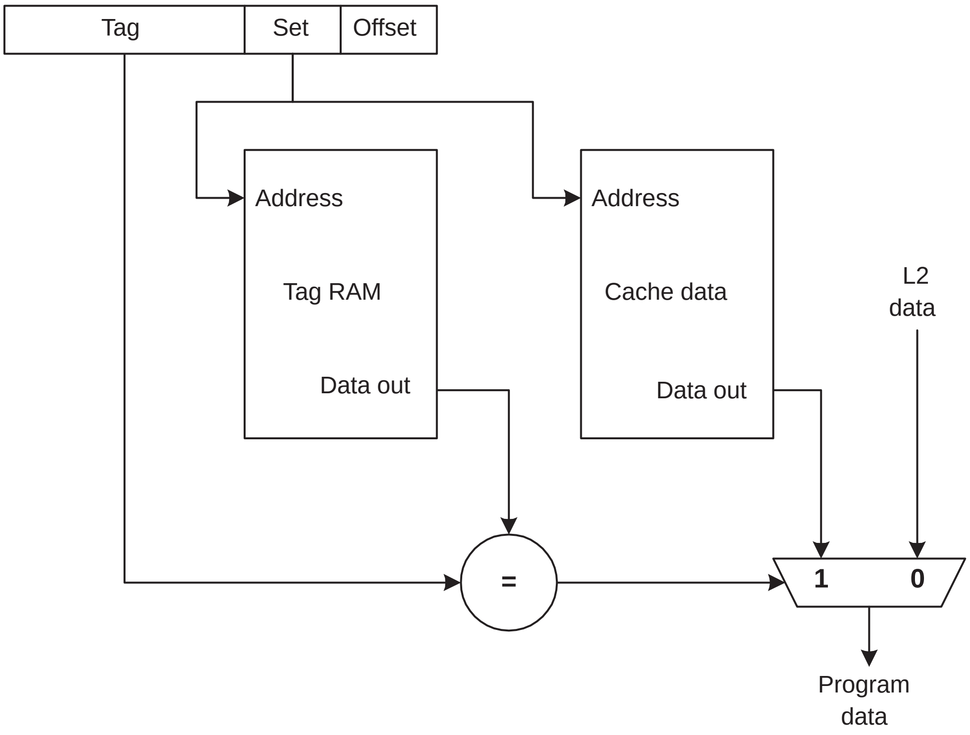
A cache hit returns data to the CPU in a single cycle. Unlike the TMS320C6201, the L1P only operates as a cache and cannot be memory mapped. The L1P does not support freeze or bypass modes. The only values allowed for the program cache control (PCC) field in the CPU control and status register (CSR) are 000b and 010b. All other values for PCC are reserved, as shown in Table 4–4.

Table 4–4. Level 1 Program Cache Mode Settings

Cache Mode	PCC value	Description
Cache enable	010b	Direct mapped cache
Cache enable	000b	Direct mapped cache
	other	Reserved

Any initial program fetch of an address causes a cache miss to occur. The data is requested from the L2 and stored in the internal cache memory. Any subsequent read from a cached address causes a cache hit and that data is loaded from the L1P memory. Figure 4–5 illustrates the organization of a direct mapped cache.

Figure 4–5. L1P Direct Mapped Cache Diagram



There are two methods for user-controlled invalidation of data in the L1P. Writing a 1 to the IP bit of the cache configuration register (CCFG) invalidates all of the cache tags in the L1P tag RAM. This is a write-only bit, a read of this bit will always return a 0. Any CPU access to the L1P while invalidation is being processed stalls the CPU until the invalidation has completed and the CPU request has been fetched. Figure 4–12 shows the format for the CCFG register. Table 4–6 describes the operation of this register.

The second method for invalidating the L1P requires the L1PFBAR and L1PFWC registers. This is useful for invalidating a block of data in the L1P. You must first write a word–aligned address into the L1PFBAR. This value is the starting address for the invalidation. The number of words to be invalidated will be equal to the value written into the L1PFWC register. The L1P searches for and invalidates all lines whose external memory address falls within the range from L1PFBAR to L1PFBAR+L1PFWC–4. If L1PFBAR or L1PFWC are not aligned to the L1P line size (16 words), all lines which contain any address in the specified range are invalidated. Using this block invalidation will not stall any pending CPU accesses. The block invalidation begins when the L1PFWC is written, therefore you should take care to ensure that the L1PFBAR register is set up correctly prior to writing the L1PFWC. Figure 4–6 and Figure 4–7 show the format for the L1PFBAR and L1PFWC.

Figure 4–6. L1P Flush Base Address Register Fields (L1PFBAR)

31 0
L1P flush base address
RW,+x

Figure 4–7. L1P Flush Word Count Register Fields (L1PFWC)

31 16	15 0
rsvd	L1P flush word count
R,+x	RW,+x

4.4 L1D Description

The L1D is organized as a 64 set 2–way set associative cache with a 32 byte line size. The two least significant bits of a requested address are ignored by the L1D since the smallest access size is for a word. The next bit of the address is used to address the correct word. Bits four and three select one of the four 8 byte sublines in the addressed set. The next six bits select the set within the cache that the addressed data maps to. The remaining bits of the address are used as a unique tag for the requested data. Figure 4–8 illustrates how a 32 bit address is allocated to provide the word index, subline index, set index and tag data for the L1D.

Figure 4–8. L1D Address Allocation

31 11	10 5	4 3	2	1 0
Tag	Set	Subline	Word	Offset

A cache hit returns data to the CPU in a single cycle. Operation on a cache miss depends on the direction of the access. On a read miss, the L1D sends a read request to the L2 to fetch the data. When the data is returned from the L2, the L1D analyzes the set that the addressed data maps to in each way. The L1D controller stores the new data into the set that was least recently used (LRU). If the data in that set has been modified but the corresponding address has not be updated (the cache line is dirty), that data is written out to the L2. In this way, cached data that has been modified will not be discarded before it is updated in its original address. If two read misses occur in the same cycle, they are serialized by the L1D so that only one request is presented to the L2 at a time. On a write miss, the L1D sends the write request to the L2. The data is not stored in the L1D. Write requests from the L1D to the L2 are buffered. If a write request is still pending from the L1D when a read miss occurs, this buffer is allowed to empty before the read request is sent to the L2.

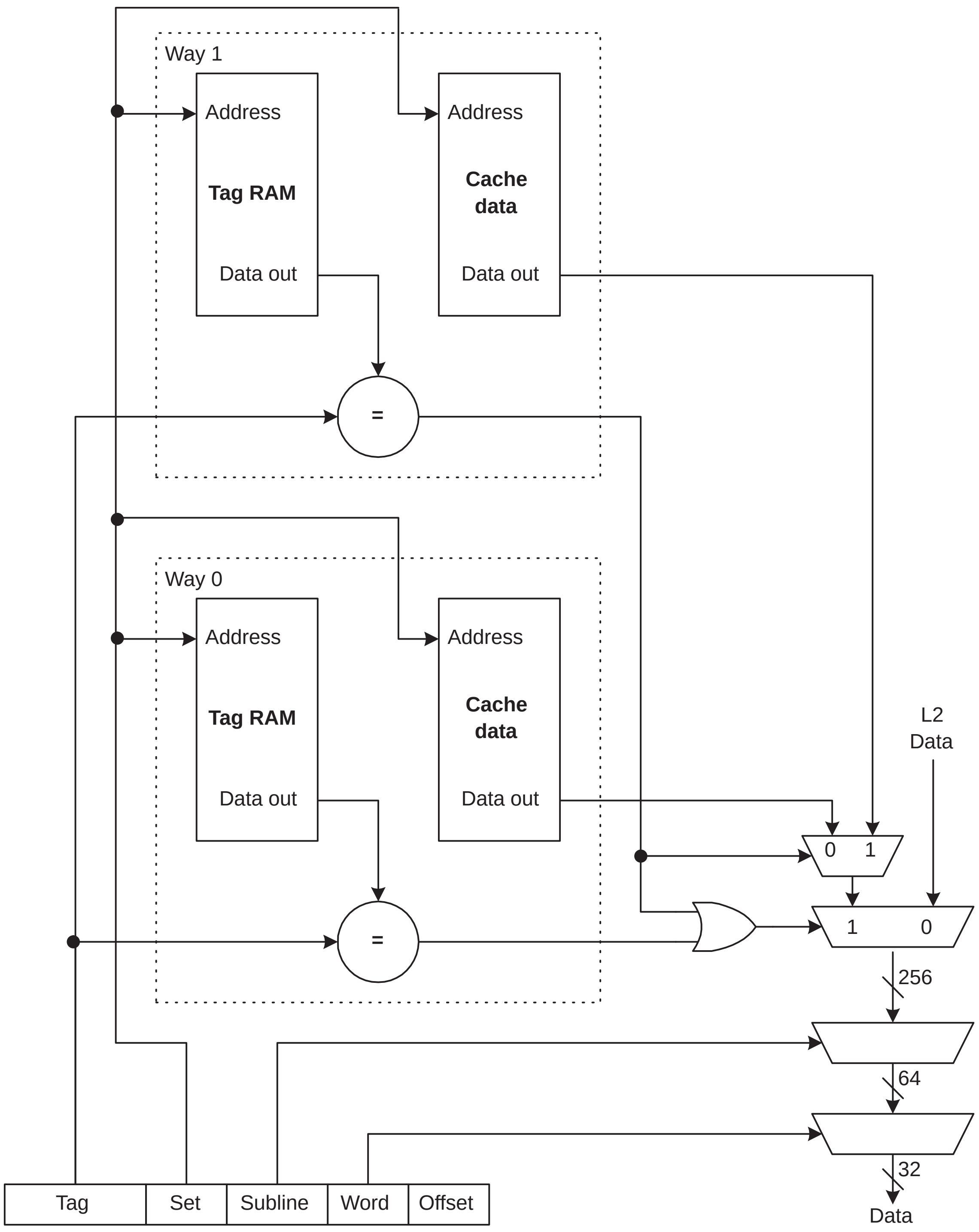
The L1D only operates as a cache and cannot be memory mapped. The L1D does not support freeze or bypass modes. The only values allowed for the data cache control (DCC) field in the CPU control and status register (CSR) are 000b and 010b. All other values for DCC are reserved, as shown in Table 4–5.

Table 4–5. Level 1 Data Cache Mode Settings

Cache Mode	DCC value	Description
Cache enable	000b	2-way cache
Cache enable	010b	2-way cache
	Other	Reserved

Any initial load of an address causes a cache miss to occur. The data is loaded and stored in the internal cache memory. Any subsequent read from a cached address will cause a cache hit and that data will be loaded from the internal cache memory. Figure 4–9 illustrates the organization for a 2-way set associative cache.

Figure 4–9. L1D 2–Way Set Associative Cache Diagram.



There are two methods for user-controlled invalidation of data in the L1D. Writing a 1 to the ID bit of the cache configuration register (CCFG) invalidates all the cache tags in the L1D tag RAM. This is a write-only bit, a read of this bit returns a 0. Any CPU access to the L1D while invalidation is being processed stalls until the invalidation has completed and the CPU request has been fetched.

The second method for invalidating the L1D requires the L1DFBAR and L1DFWC registers. This is useful for invalidating a block of data in the L1D. You must first write a word-aligned address into the L1DFBAR. This value is used as the starting address for the invalidation. The number of words invalidated equals the value written into the L1DFWC register. The L1D searches for and invalidate all lines whose external memory address falls within the range from L1DFBAR to L1DFBAR+L1DFWC–4. The data in these lines is sent to the L2 to be stored in the original memory location. In this way, the L2 and external memory will remain coherent with the data that is invalidated. If L1DFBAR or L1DFWC are not aligned to the L1D line size (8 words) all lines which contain data in the address range specified are invalidated. However only those words that are contained in the range from L1DFBAR to L1DFBAR+L1DFWC–4 will be saved to the L2. This block invalidation will occur in the background and not stall any pending CPU accesses. The block invalidation begins when the L1DFWC is written, therefore you should take care to ensure that the L1DFBAR register is set up correctly prior to writing the L1DFWC. This is the preferred method for writing data that has been cached in the L1D to the external memory space. Figure 4–10 and Figure 4–11 show the format for the L1DFBAR and L1DFWC.

Figure 4–10. L1D Flush Base Address Register Fields (L1DFBAR)

31–0
L1D flush base address
RW,+x

Figure 4–11. L1D Flush Word Count Register Fields (L1DFWC)

31–16	15–0
rsvd	L1D flush word count
R,+x	RW,+x

4.5 L2 Description

The L2 is accessible from both the L1P and the L1D. On a cache miss from the L1P or L1D, the request is first sent to the L2 to be serviced. How the L2 services the request depends on the selected operation mode of the L2. Table 4–6 shows the supported operation modes for the L2. Figure 4–13 illustrates the division of the L2 memory space according to the L2 Mode.

Writing to the L2MODE field of the cache configuration register (CCFG) sets the L2 mode. Figure 4–12 shows the format for the CCFG register. Table 4–6 describes the operation of this register.

Figure 4–12. Cache Configuration Register Fields (CCFG)

31	30 10	9	8	7 3	2 0
P	rsvd	IP	ID	rsvd	L2MODE
RW,+0	R,+x	W,+0	W,+0	R,+0 0000	RW,+000

Table 4–6. Cache Configuration Register Field Description

Field	Description

L2MODE	L2 Operation Mode
	L2MODE = 000b: 64K bytes SRAM
	L2MODE = 001b: 16K bytes 1-way cache / 48 Kbytes mapped RAM
	L2MODE = 010b: 32K bytes 2-way cache / 32 Kbytes mapped RAM
	L2MODE = 011b: 48K bytes 3-way cache / 16 Kbytes mapped RAM
	L2MODE = 111b: 64K bytes 4-way cache
	L2MODE = other: Reserved

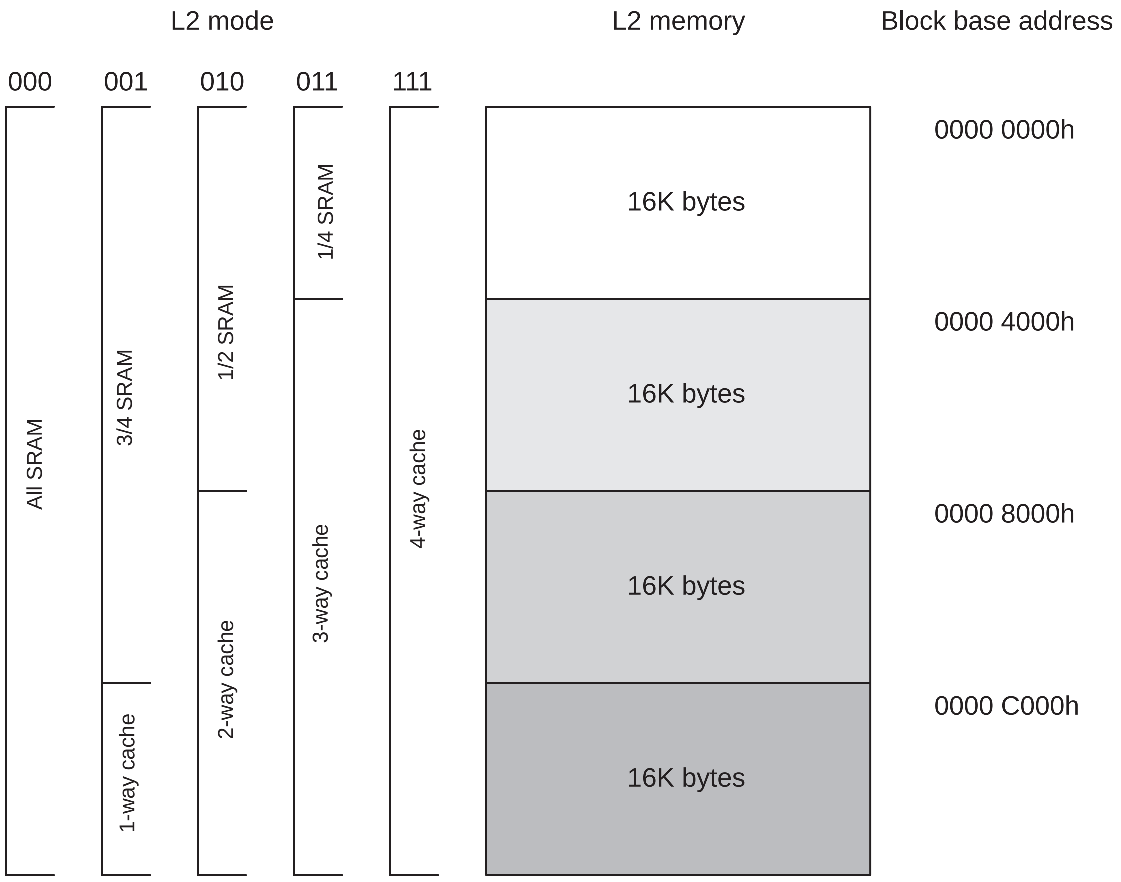
ID	Invalidate L1D
	ID = 0: Normal L1D operation
	ID = 1: All L1D lines invalidated

IP	Invalidate LIP
	IP = 0: Normal L1P operation
	IP = 1: All L1P lines invalidated

P	L2 Requestor Priority P = 0: CPU accesses prioritized over enhanced DMA accesses P = 1: Enhanced DMA accesses prioritized over CPU accesses

The reset value of the L2MODE field is 000b, thus the L2 RAM is configured as 64K bytes of mapped memory at reset to support bootloading. Any L2 RAM that is configured as cache is no longer in the memory map. For example, in L2 Mode 010b, the address space from 0000 8000h to 0000 FFFFh is no longer mapped. The associativity of the L2 cache RAM is a function of the L2 Mode. Each 16K byte block of RAM included in the cache adds one way to the associativity. The line size for the L2 cache is 128 bytes. Figure 4–13 shows the cache associativity for each L2 Mode.

Figure 4–13. L2 Memory Configuration



4.5.1 L2 Interfaces

The L2 Controller services requests from three different requestors – the L1P, the L1D, and the Enhanced DMA. Since the L1P only sends read requests, a single 256 bit wide data bus transfers data from the L2 to the L1P. The L1D to L2 interface consists of a 128 bit read bus from the L2 to the L1D and a 128 bit write bus from the L1D to the L2. The L2 transfers data to and from the EDMA through a 64 bit read and a 64 bit write bus.

4.5.2 L2 Operation

Each L1D access to the L2 memories takes two cycles. Since the line size of L1D cache is twice the width of the bus between the cache and the L2, a miss to the L2 requires two accesses. Therefore, a miss from L1D to the L2 takes four cycles to complete if the data is available in the L2. A miss from the L1P to the L2 completes in five cycles.

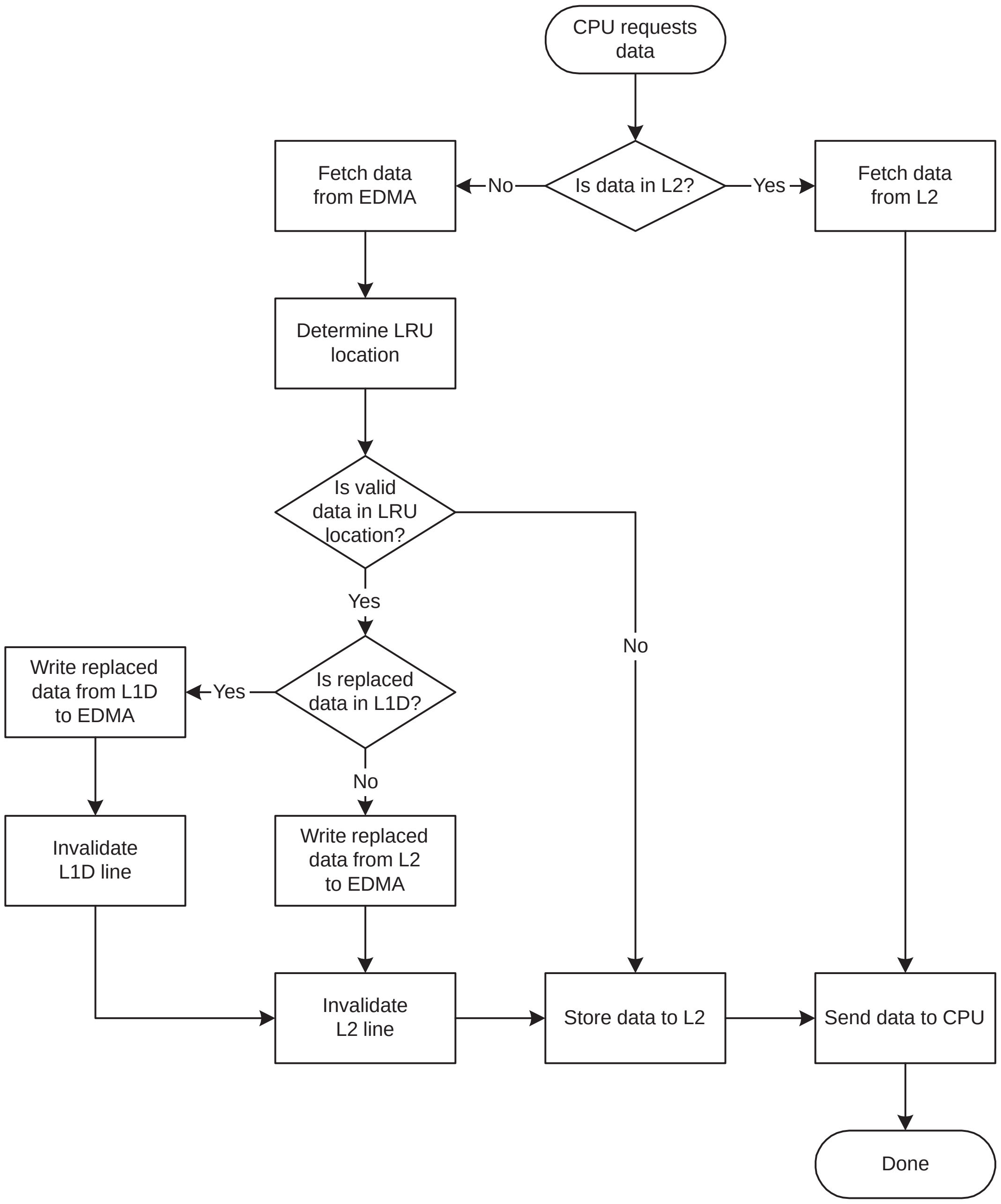
The L2 memories are organized as four 64 bit wide banks. Two accesses can be serviced at the same time if the two accesses do not use the same bank. Since the L1P data bus is 256 bits wide, any L1P request that occurs at the same time as an L1D or EDMA request will cause a bank collision and therefore a stall. Concurrent accesses between the L1D and EDMA busses to different banks can be serviced without stalling.

The priority bit (P) in the Cache Configuration Register (CCFG) determines the priority when a bank collision occurs between requestors. If the P bit is set to 0, CPU accesses (L1P and L1D) are given priority over an EDMA request. Thus, any pending CPU request will complete before the EDMA request is serviced. If this bit is set to 1, EDMA requests are prioritized over CPU accesses. When an L1P and L1D access collide, the L1P request is always given priority.

When an L2 location is operating as mapped RAM, an access to that location operates like a standard RAM. A read request reads the value stored in that location and a write request updates that location with the new data. When an L2 location is enabled as a cache, the operation is similar to the L1D cache. If a read request is made to the L2, the tag RAM for each of the cached blocks is searched for that address. If a tag hit occurs, that data is sent to the requestor. If the data is not in the L2 the requestor is stalled and the data is requested from the Enhanced DMA. To fulfill an L1P request, the L2 controller must make eight 64 bit requests to the EDMA. Similarly, four requests to the EDMA are required to service an L1D request.

The L2 uses a least recently used (LRU) replacement strategy to replace old cached data with new data. To determine which cache lines to replace, the address for the new data is used to calculate the set which that address maps to. Each external address maps to one set in each cache way. Then, that set in each way is interrogated to determine which way contains the least recently used (LRU) data. The new data is stored at that location. If the cache location to be replaced contains valid data, the previous data is evicted. An eviction occurs as follows. The L2 first polls the L1D to determine if the evicted address is also cached in the L1D. This is referred to as snooping the L1D. If data is returned from the L1D, it is written out to the EDMA. Then, both the L1D and L2 lines are invalidated. If the L1D does not cache the evicted address, the data in the L2 to written out to the EDMA. In this case, only the L2 line is invalidated. Finally, the requested data is stored in the L2 and sent to the requestor. This mechanism ensures that the CPU does not access stale data and that no data is lost. Figure 4–14 diagrams the decision process used by the L2 controller to service a data request from the CPU when the L2 is operating as a cache. The L2 performs evictions for read and write requests.

Figure 4–14. L2 Cache Data Request Flow Chart



The memory attribute registers (MARs) can be programmed to turn on caching of each of the external chip enable (CE) spaces. In this way, you can perform single word reads to external mapped devices. Without this feature any external read would always read an entire L2 line of data. Each of the four CE spaces is divided into four ranges, each of which maps the least significant bit of an MAR register. If an MAR register is set, the corresponding address range is cached by the L2. At reset, the MAR registers are set to 0. To begin caching data in the L2, you must initialize the appropriate MAR register to 1. The MAR registers define cacheability for the EMIF only. Addresses accessed by the EMIF which are not defined by the MAR registers are always cacheable. Figure 4–15 shows the format for the MARs. Table 4–3 illustrates which address range each MAR bit enables for caching.

Figure 4–15. L2 CE Space Allocation Register Fields

MAR0

31 1	0
rsvd	CE 0.0
R,+x	RW,+0

MAR1

31 1	0
rsvd	CE 0.1
R,+x	RW,+0

MAR2

31 1	0
rsvd	CE 0.2
R,+x	RW,+0

MAR3

31 1	0
rsvd	CE 0.3
R,+x	RW,+0

MAR4

31 1	0
rsvd	CE 1.0
R,+x	RW,+0

MAR5

31 1	0
rsvd	CE 1.1
R,+x	RW,+0

Figure 4–15. L2 CE Space Allocation Register Fields (Continued)

MAR6

31	1	0
rsvd		CE 1.2
R,+x		RW,+0

MAR7

31	1	0
rsvd		CE 1.3
R,+x		RW,+0

MAR8

31	1	0
rsvd		CE 2.0
R,+x		RW,+0

MAR9

31	1	0
rsvd		CE 2.1
R,+x		RW,+0

MAR10

31	1	0
rsvd		CE 2.2
R,+x		RW,+0

MAR11

31	1	0
rsvd		CE 2.3
R,+x		RW,+0

MAR12

31	1	0
rsvd		CE 3.0
R,+x		RW,+0

MAR12

31	1	0
rsvd		CE 3.1
R,+x		RW,+0

Figure 4–15. L2 CE Space Allocation Register Fields (Continued)

MAR14

31 ... 1	0
rsvd	CE 3.2
R,+x	RW,+0

MAR15

31 ... 1	0
rsvd	CE 3.3
R,+x	RW,+0

Table 4–7. Memory Attribute Register Functions

MAR	Address Range Enabled	CE Space
15	B300 0000h – B3FF FFFFh	CE3
14	B200 0000h – B2FF FFFFh	CE3
13	B100 0000h – B1FF FFFFh	CE3
12	B000 0000h – B0FF FFFFh	CE3
11	A300 0000h – A3FF FFFFh	CE2
10	A200 0000h – A2FF FFFFh	CE2
9	A100 0000h – A1FF FFFFh	CE2
8	A000 0000h – A0FF FFFFh	CE2
7	9300 0000h – 93FF FFFFh	CE1
6	9200 0000h – 92FF FFFFh	CE1
5	9100 0000h – 91FF FFFFh	CE1
4	9000 0000h – 90FF FFFFh	CE1
3	8300 0000h – 83FF FFFFh	CE0
2	8200 0000h – 82FF FFFFh	CE0
1	8100 0000h – 81FF FFFFh	CE0
0	8000 0000h – 80FF FFFFh	CE0

4.5.3 L2 EDMA Service

EDMA accesses are only allowed to L2 space that is configured as mapped RAM. When the EDMA makes a read request to the L2, the L2 snoops the data from the L1D and stalls the EDMA until a response is returned. If data that must be updated is returned, that data is placed in the L2 and the EDMA request proceeds. In this case, the L1D line is invalidated to maintain coherency. If the L1D does not return data to the L2 then the data is read from the L2. The L2 does not snoop the L1P for data when a EDMA read request is received because the CPU cannot modify data in L1P so it’s data will not be incoherent.

When the EDMA makes a write request to the L2, both the L1P and the L1D are snooped for the data. Both the L1P and the L1D must be notified of the write because the L2 has no knowledge of the type of data being written by the EDMA, whether program or data. If the L1P responds that it is caching the addressed data, then that line is invalidated and the data is written into L2. Similarly, if the L1D is caching that address, then that line in the L1D is invalidated and the data is written to L2. By invalidating the lines in the L1P or the L1D, the correct data will be fetched from the L2 on the next CPU request of that data.

4.5.4 L2 Invalidation

The method for user controlled invalidation of data in the L2 is similar to those for the L1P and the L1D. For the L2, however, there are two types of invalidation. The first type of invalidation is an L2 flush. During a flush, the contents of the L2 are copied out through the enhanced DMA. Like an EDMA read or L2 data eviction, the L1D is snooped for any modified (dirty) data that is being copied out by the flush. The second type of L2 invalidation is a clean. The clean operation copies data from the L2 through the EDMA to the external memory space and snoops data from the L1D. In addition, the clean operation invalidates any line in the L1P, L1D, or L2 that caches data that is copied to the external memory space.

To initiate an L2 flush of the entire L2 cache space, write a 1 to the F bit of the L2FLUSH register. This bit remains set to 1 until the flush is complete at which time the register is cleared to 0 by the L2 controller. Figure 4–16 shows the fields of the L2FLUSH register. Table 4–8 describes the operation of the L2FLUSH register. Similarly, to initiate an L2 clean of the entire L2 cache space set the C bit of the L2CLEAN register to 1. This bit remains set to 1 until the clean is complete at which time the register is cleared to 0. Figure 4–17 shows the fields of the L2CLEAN register. Table 4–9 describes the operation of the L2CLEAN register.

Figure 4–16. L2 Flush Register Fields (L2FLUSH)

31 … 1	0
rsvd	F
R,+x	RW,+0

Table 4–8. L2 Flush Register Fields Description

Field	Description

F	Flush L2
	F = 0: Normal L2 operation
	F = 1: All L2 lines flushed

Figure 4–17. L2 Clean Register Fields (L2CLEAN)

31 … 1	0
rsvd	C
R,+x	RW,+0

Table 4–9. L2 Clean Register Fields Description

Field	Description

C	Clean L2
	C = 0: Normal L2 operation
	C = 1: All L2 lines cleaned

It is also possible to flush and clean a range of addresses from the L2. To flush a range of address from the L2, write the word–aligned address for the start of the flush into the L2FBAR. The number of words to be flushed is equal to the value written into the L2FWC register. The L2 controller then searches all L2 cache blocks for all lines whose external memory address falls within the range from L2FBAR to L2FBAR+L2FWC–4 and copies that data through the EDMA to the external memory space. The L1D is snooped to ensure that the correct data is stored in the original memory location. The L2 flush occurs in the background and does not stall any pending CPU accesses. The flush begins when the L2FWC is written, therefore you should take care to ensure that the L2FBAR register is set up correctly prior to writing the L2FWC. Figure 4–18 shows the fields in the L2FBAR register. Figure 4–19 shows the fields in the L2FWC register.

Figure 4–18. L2 Flush Base Address Register Fields (L2FBAR)

31 … 0
L2 Flush Base Address
RW,+x

Figure 4–19. L2 Flush Word Count Register Fields (L2FWC)

31 — 16	15 — 0
rsvd	L2 Flush Word Count
R,+x	R,+x

To clean a range of address from the L2, write the word-aligned address for the start of the clean into the L2CBAR. The number of words to clean is equal to the value written into the L2CWC register. The L2 controller then searches all L2 cache blocks for all lines whose external memory address falls within the range from L2CBAR to L2CBAR+L2CWC–4 and copies that data through the EDMA to external memory space. The L1D is snooped to ensure that the correct data is stored in the original memory location. In addition to snooping data from the L1D, any L1P or L1D lines that cache a cleaned address are invalidated. The L2 clean occurs in the background and does not stall any pending CPU accesses. The clean begins when the L2CWC is written, therefore you should take care to ensure that the L2CBAR register is set up correctly prior to writing the L2CWC. If L2CBAR or L2CWC are not aligned to the L2 line size (32 words), all lines which contain the words specified are invalidated. However only those words that are contained in the range from L2CBAR to L2CBAR + L2CWC–4 are saved to the external memory space. Figure 4–20 shows the fields in the L2CBAR register. Figure 4–21 shows the fields in the L2CWC register.

Figure 4–20. L2 Clean Base Address Register Fields (L2CBAR)

31 — 0
L2 Clean Base Address
RW,+x

Figure 4–21. L2 Clean Word Count Register Fields (L2CWC)

31 — 16	15 — 0
rsvd	L2 Clean Word Count
R,+x	R,+x

If more than one block invalidation, block flush, or block clean is requested at one time, the CPU is stalled until all are completed. For example, if an L1P invalidate is being processed and you set up an L2 clean by writing to the L2CWC register, the CPU is stalled until both the L1P invalidate and L2 clean are complete.

Chapter 5

Direct Memory Access (DMA) Controller

This chapter describes the direct memory access channels and registers available for the TMS320C6201/C6202/C6701 devices.

Topic **Page**

5.1 Overview

The direct memory access (DMA) controller transfers data between regions in the memory map without intervention by the CPU. The DMA controller allows movement of data to and from internal memory, internal peripherals, or external devices to occur in the background of CPU operation. The DMA controller has four independent programmable channels, allowing four different contexts for DMA operation. In addition, a fifth (auxiliary) channel allows the DMA controller to service requests from the host port interface (HPI). In discussing DMA operations, several terms are important:

- Read transfer: The DMA controller reads a data element from a source location in memory.
- Write transfer: The DMA controller writes the data element that was read during a read transfer to its destination in memory.
- Element transfer: This form refers to the combined read and write transfer for a single data element.
- Frame transfer: Each DMA channel has an independently programmable number of elements per frame. In completing a frame transfer, the DMA controller moves all elements in a single frame.
- Block transfer: Each DMA channel also has an independently programmable number of frames per block. In completing a block transfer, the DMA controller moves all frames that it has been programmed to move.
- Transmit element transfer: In split mode, data elements are read from the source address, and writing it to the split destination address. See section 5.8 for details.
- Receive element transfer: In split mode, data elements are read from the split source address, and writing it to the destination address. See section 5.8 for details.

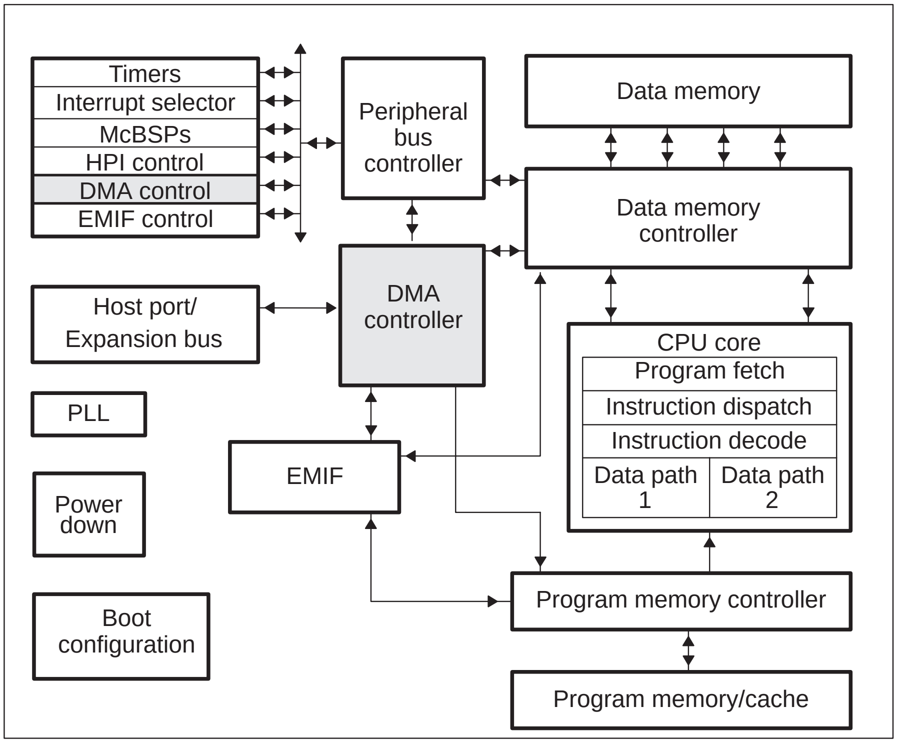
The DMA controller has the following features:

- Background operation: The DMA controller operates independently of the CPU.
- High throughput: Elements can be transferred at the CPU clock rate. See section 5.11, *Structure*, on page 5-35 for more information.
- Four channels: The DMA controller can keep track of the contexts of four independent block transfers. See section 5.2, *DMA Registers*, on page 5-5 for more information about saving the contents of multiple block transfers.

- Auxiliary channel: This channel allows the host port to make requests into the CPU's memory space. The auxiliary channel requests may be prioritized relative to other channels and the CPU.
- Split-channel operation: A single channel can be used to perform both the receive and transmit element transfers from or to a peripheral simultaneously, effectively acting like two DMA channels. See section 5.8 on page 5-28 for more information.
- Multiframe transfer: Each block transfer can consist of multiple frames of a programmable size. See Section 5.5, *Transfer Counting*.
- Programmable priority: Each channel has independently programmable priorities versus the CPU.
- Programmable address generation: Each channel's source and destination address registers can have configurable indexes for each read and write transfer. The address can remain constant, increment, decrement, or be adjusted by a programmable value. The programmable value allows an index for the last transfer in a frame distinct from that used for the preceding transfers. See section 5.7.1 on page 5-22 for more information.
- Full 32-bit address range: The DMA controller can access any region in the memory map:
  - On-chip data memory
  - On-chip program memory when it is mapped into memory space rather than being used as cache
  - On-chip peripherals
  - External memory via the EMIF
  - Expansion memory via the expansion bus
- Programmable width transfers: Each channel can be independently configured to transfer either bytes, 16-bit halfwords, or 32-bit words. See section 5.7.3 on page 5-23 for more information.
- Autoinitialization: Once a block transfer is complete, a DMA channel can automatically reinitialize itself for the next block transfer. See section 5.4.1 on page 5-13 for more information.
- Event synchronization: Each read, write, or frame transfer may be initiated by selected events. See Section 5.6 on page 5-17 for more information.
- Interrupt generation: On completion of each frame transfer or block transfer, as well as on various error conditions, each DMA channel can send an interrupt to the CPU. See section 5.10 on page 5-33 for more information.

Figure 5–1 shows the ’C6000 block diagram with the DMA-related components shaded.

Figure 5–1. DMA Controller Interconnect to TMS320C6201/C6202/C6701 Memory-Mapped Modules



5.2 DMA Registers

The DMA registers configure the operation of the DMA controller. Table 5–1 and Table 5–2 show how the DMA control registers are mapped in memory. These registers include the DMA global data, count reload, index, and address registers, as well as independent control registers for each channel.

Table 5–1. DMA Control Registers by Address

Hex Byte Address	Name	Described in Section
0184 0000	DMA channel 0 primary control	5.2.1
0184 0004	DMA channel 2 primary control	5.2.1
0184 0008	DMA channel 0 secondary control	5.10
0184 000C	DMA channel 2 secondary control	5.10
0184 0010	DMA channel 0 source address	5.7
0184 0014	DMA channel 2 source address	5.7
0184 0018	DMA channel 0 destination address	5.7
0184 001C	DMA channel 2 destination address	5.7
0184 0020	DMA channel 0 transfer counter	5.5
0184 0024	DMA channel 2 transfer counter	5.5
0184 0028	DMA global count reload register A	5.5
0184 002C	DMA global count reload register B	5.5
0184 0030	DMA global index register A	5.7.2
0184 0034	DMA global index register B	5.7.2
0184 0038	DMA global address register A	5.8
0184 003C	DMA global address register B	5.8
0184 0040	DMA channel 1 primary control	5.2.1
0184 0044	DMA channel 3 primary control	5.2.1
0184 0048	DMA channel 1 secondary control	5.10
0184 004C	DMA channel 3 secondary control	5.10
0184 0050	DMA channel 1 source address	5.7
0184 0054	DMA channel 3 source address	5.7
0184 0058	DMA channel 1 destination address	5.7
0184 005C	DMA channel 3 destination address	5.7
0184 0060	DMA channel 1 transfer counter	5.5
0184 0064	DMA channel 3 transfer counter	5.5
0184 0068	DMA global address register C	5.8
0184 006C	DMA global address register D	5.8
0184 0070	DMA auxiliary control register	5.9.1

Table 5–2. DMA Control Registers by Register Name

Name	Hex Byte Address	Described in Section
DMA auxiliary control register	0184 0070	5.9.1
DMA channel 0 destination address	0184 0018	5.7
DMA channel 0 primary control	0184 0000	5.2.1
DMA channel 0 secondary control	0184 0008	5.10
DMA channel 0 source address	0184 0010	5.7
DMA channel 0 transfer counter	0184 0020	5.5
DMA channel 1 destination address	0184 0058	5.7
DMA channel 1 primary control	0184 0040	5.2.1
DMA channel 1 secondary control	0184 0048	5.10
DMA channel 1 source address	0184 0050	5.7
DMA channel 1 transfer counter	0184 0060	5.5
DMA channel 2 destination address	0184 001C	5.7
DMA channel 2 primary control	0184 0004	5.2.1
DMA channel 2 secondary control	0184 000C	5.10
DMA channel 2 source address	01840014	5.7
DMA channel 2 transfer counter	0184 0024	5.5
DMA channel 3 destination address	0184 005C	5.7
DMA channel 3 primary control	0184 0044	5.2.1
DMA channel 3 secondary control	0184 004C	5.10
DMA channel 3 source address	0184 0054	5.7
DMA channel 3 transfer counter	0184 0064	5.5
DMA global address register A	0184 0038	5.8
DMA global address register B	0184 003C	5.8
DMA global address register C	0184 0068	5.8
DMA global address register D	0184 006C	5.8
DMA global count reload register A	0184 0028	5.5
DMA global count reload register B	0184 002C	5.5
DMA global index register A	0184 0030	5.7.2
DMA global index register B	0184 0034	5.7.2

5.2.1 DMA Channel Control Registers

The DMA channel primary and secondary control registers (Figure 5–2 and Figure 5–3) contain-fields that control each DMA channel independently. These fields are summarized in Table 5–3 and Table 5–4.

Figure 5–2. DMA Channel Primary Control Register

31 30	29 28	27	26	25	24	23 19	18 16
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC
RW, +0	RW, +0	RW,+0	RW,+0	RW, +0	RW, +0	RW, +0	RW, +0

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
RSYNC	INDEX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	R, +0	RW, +0

Table 5–3. DMA Channel Primary Control Register Field Descriptions

Field	Description	Section
DST RELOAD, SRC RELOAD	Source/destination address reload for autoinitialization SRC/DST RELOAD = 00b: do not reload during autoinitialization SRC/DST RELOAD = 01b: use DMA global address register B as reload SRC/DST RELOAD = 10b: use DMA global address register C as reload SRC/DST RELOAD = 11b: use DMA global address register D as reload	5.4.1.1
EMOD	Emulation mode EMOD = 0: DMA channel keeps running during an emulation halt EMOD = 1: DMA channel pauses during an emulation halt	5.13
FS	Frame synchronization FS = 0: disable FS = 1: RSYNC event used to synchronize entire frame	5.6

TCINT	Transfer controller interrupt
	
	TCINT = 0: interrupt disabled
	TCINT = 1: interrupt enabled
		5.10

PRI	Priority mode: DMA versus CPU
	
	PRI = 0: CPU priority
	PRI = 1: DMA priority
		5.9

WSYNC, RSYNC	Read transfer/write transfer synchronization (R/W)SYNC = 00000b: no synchronization (R/W)SYNC = other: sets synchronization event	5.6

Table 5–3. DMA Channel Primary Control Register Field Descriptions (Continued)

Field	Description	Section

INDEX	Selects the DMA global data register to use as a programmable index
	
	INDEX = 0: use DMA global index register A
	INDEX = 1: use DMA global index register B
		5.7.2

CNT RELOAD	Transfer counter reload for autoinitialization and multiframe transfers CNT RELOAD = 0: reload with DMA global count reload register A CNT RELOAD = 1: reload with DMA global count reload register B	5.4.1.1

SPLIT	Split channel mode
	
	SPLIT = 00b: split-channel mode disabled
	SPLIT = 01b: split-channel mode enabled; use DMA global address register A as split address
	SPLIT = 10b: split-channel mode enabled; use DMA global address register B as split address
	SPLIT = 11b: split-channel mode enabled; use DMA global address register C as split address
		5.8

ESIZE	Element size
	
	ESIZE = 00b: 32-bit
	ESIZE = 01b: 16-bit
	ESIZE = 10b: 8-bit
	ESIZE = 11b: reserved
		5.7.3

DST DIR, SRC DIR	Source/destination address modification after element transfers SRC/DST DIR = 00b: no modification SRC/DST DIR = 01b: increment by element size in bytes SRC/DST DIR = 10b: decrement by element size in bytes SRC/DST DIR = 11b: adjust using DMA global index register selected by INDEX	5.7.1, 5.7.2

STATUS	STATUS = 00b: stopped
	STATUS = 01b: running without autoinitialization
	STATUS = 10b: paused
	STATUS = 11b: running with autoinitialization
		5.4

START	START = 00b: stop
	START = 01b: start without autoinitialization
	START = 10b: pause
	START = 11b: start with autoinitialization
		5.4

Figure 5–3. DMA Channel Secondary Control Register

31–19	18–16
Reserved	DMAC EN
R, +0000 0000 0000 0	RW, +000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WSYNC CLR	WSYNC STAT	RSYNC CLR	RSYNC STAT	WDROP IE	WDROP COND	RDROP IE	RDROP COND	BLOCK IE	BLOCK COND	LAST IE	LAST COND	FRAME IE	FRAME COND	SX IE	SX COND
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +1	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0

Table 5–4. DMA Channel Secondary Control Register Field Descriptions

Field	Description	Section
SX COND FRAME COND LAST COND BLOCK COND RDROP COND WDROP COND	DMA condition. Each bit indicates a separate condition. A0 value indicates that the condition is not detected. A1 value indicates that the condition is detected.	5.10
SX IE FRAME IE LAST IE BLOCK IE RDROP IE WDROP IE	DMA condition interrupt enable IE = 0: associated condition does not enable DMA channel interrupt IE = 1: associated condition enables DMA channel interrupt	5.10.1
RSYNC STAT WSYNC STAT	Read or write synchronization status STAT = 0: synchronization is not received STAT = 1: synchronization is received	5.6.1
DMAC EN	DMAC pin control DMAC EN = 000b: DMAC pin is held low. DMAC EN = 001b: DMAC pin is held high. DMAC EN = 010b: DMAC reflects RSYNC STAT. DMAC EN = 011b: DMAC reflects WSYNC STAT. DMAC EN = 100b: DMAC reflects FRAME COND. DMAC EN = 101b: DMAC reflects BLOCK COND. DMAC EN = other: reserved	5.12
RSYNC CLR WSYNC CLR	Read or write synchronization status clear Read as 0 write 1 to clear associated status	5.6.1

The DMA channel secondary control register of the ‘C6202 has been expanded to include three new fields: WSPOL, RSPOL, and FSIG. This field is used to add control to a frame-synchronized data transfer. The ‘C6202 secondary control register is shown in Figure 5–4; the new field is shown in gray. Table 5–5 describes the possible configurations of the new field.

Figure 5–4. TMS320C6202 Secondary Control Register

31–22	21	20	19	18–16
Reserved	WSPOL	RSPOL	FSIG	DMAC
R, +0	RW, +0	RW, +0	RW, +0	RW, +000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WSYNC CLR	WSYNC STAT	RSYNC CLR	RSYNC STAT	WDROP IE	WDROP COND	RDROP IE	RDROP COND	BLOCK IE	BLOCK COND	LAST IE	LAST COND	FRAME IE	FRAME COND	SX IE	SX COND
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +1	RW, +0	RW,+0	RW,+0	RW, +0	RW, +0	RW,+0	RW,+0

Table 5–5. Synchronization Configuration Options

Field	Description	Section
WSPOL/ RSPOL	Synchronization event polarity. Selects the polarity of an external sync event: 1 = active low, 0 = active high This field is valid only if EXT_INTx is selected.	5.6.3
FSIG	Frame sync ignore. Setting FSIG = 1 causes the DMA channel to ignore any event transitions during a current burst. Synchronization is level triggered instead of edge triggered.	5.6.3

5.3 Memory Map

The DMA controller assumes the device memory map shown in Chapter 10, *Boot Configuration, Reset, and Memory Maps*. Requests are sent to one of five resources:

- ❑ Expansion bus
- ❑ External memory interface
- ❑ Internal program memory
- ❑ Internal peripheral bus
- ❑ Internal data memory

The source address is assumed to point to one of these four spaces throughout a block transfer. This constraint also applies to the destination address.

5.4 Initiating a Block Transfer

Each DMA channel can be started independently, either manually through direct CPU access or automatically through autoinitialization. Each DMA channel can be stopped or paused independently through direct CPU access. The status of a DMA channel can be observed by reading the STATUS field in the DMA channel's primary control register.

Manual start operation: To start DMA operation for a particular channel, once the desired values are written to all other DMA control registers, the desired value should be written to the DMA control register with START = 01b. Writing this value to a DMA channel that has already been started has no effect. Once started, the value on STATUS is 01b.

Pause operation: Once started, a DMA channel can be paused by writing START = 10b. When paused, the DMA channel completes any write transfers whose read transfer requests have completed. Also, if the DMA channel has all of the necessary read synchronizations, one additional element transfer is allowed to finish. Once paused, the value on STATUS becomes 10b after the DMA has completed all pending write transfers.

Stop operation: The DMA controller can be stopped by writing START = 00b. Stop operation is identical to pause operation. Once a DMA transfer is completed, unless autoinitialization is enabled, the DMA channel returns to the stopped state and STATUS becomes 00b after the DMA has completed all pending write transfers.

5.4.1 DMA Autoinitialization

The DMA controller can automatically reinitialize itself after completion of a block transfer. Some of the DMA control registers can be preloaded for the next block transfer through selected DMA global data registers. Using this capability, some of the parameters of the DMA channel can be set well in advance of the next block transfer. Autoinitialization allows:

Continuous operation: The CPU is given a long slack time during which it can reconfigure the DMA controller for a subsequent transfer. Normally, the CPU would have to reinitialize the DMA controller immediately after completion of the last write transfer in the current block transfer and before the first read synchronization for the next block transfer. With the reload registers, it can reinitialize these values for the next block transfer anytime after the current block transfer begins.

Repetitive operation: This operation is a special case of continuous operation. Once a block transfer finishes, the DMA controller repeats the previous block transfer. In this case, the CPU does not preload the reload registers with new values for each block transfer. Instead, the CPU loads the registers only before the first block transfer.

Enabling autoinitialization: By writing START = 11b in the channel's primary control register, you enable autoinitialization. In this case, after completion of a block transfer, the selected DMA channel registers are reloaded and the DMA channel is restarted . If you are restarting after a pause, START must be rewritten as 11b for autoinitialization to be enabled.

5.4.1.1 DMA Channel Reload Registers

For autoinitialization, the successive block transfers are assumed to be similar. Thus, the reload values are selectable only for those registers that are modified during a block transfer: the transfer counter and address registers. Thus, the DMA channel transfer counter as well as the DMA channel source and destination address registers have associated reload registers, as selected by the associated RELOAD fields in the DMA channel primary control register (see Figure 5–2).

It is possible to not reload the source or destination address register in autoinitialization mode. This capability allows a register to maintain its value during a block transfer. Thus, you do not have to dedicate a DMA global data register to a value that was static during a block transfer. A single channel can use the same value for multiple channel registers. For example, in split-channel mode, the source and destination address can be the same. On the other hand, multiple channels can use the same reload registers. For example, two channels can have the same transfer count reload register.

Upon completion of a block transfer, the channel registers are reloaded with the value from the associated reload register value. In the case of the DMA channel transfer counter register, reload occurs after the end of each frame transfer, not just after the end of the entire block transfer. The reload value for the DMA channel transfer counter is necessary whenever multiframe transfers are configured, not just when autoinitialization is enabled.

As discussed in section 5.11.2, the DMA controller can allow read transfers to get ahead of write transfers, and it provides the necessary buffering to facilitate this capability. To support this, the reload that's necessary at the end of blocks and frames occurs independently for the read (source) and write (destination) portions of the DMA channel. Similarly, in the case of split-channel operation described in section 5.8, the source and destination addresses are independently reloaded when the associated transmit or receive element transfers are completed.

You can rewrite the DMA channel transfer counter reload only after the next-to-last frame in the current block transfer it completed. Otherwise, the new reload values would affect subsequent frame boundaries in the current block transfer. However, if the frame size is the same for the current and next block transfers, this restriction is not relevant. See section 5.5 for more explanation of the DMA channel transfer counter.

5.5 Transfer Counting

The DMA channel transfer counter register, shown in Figure 5–5 contains fields that represent the number of frames and the number of elements per frame to be transferred. Figure 5–6 shows the DMA global count reload register.

FRAME COUNT: The 16-bit unsigned value in this field sets the total number of frames in the block transfer. The maximum number of frames per block transfer is 65 535. This counter is decremented upon the completion of the last read transfer in a frame transfer. Once the last frame is transferred, the entire counter is reloaded with the DMA controller global count reload register selected by the CNT RELOAD field in the DMA channel primary control register (see section 5.4.1.1). Initial values of 0 and 1 in FRAME COUNT have the same effect of transferring a single frame.

ELEMENT COUNT: The 16-bit unsigned value in this field sets the number of elements per frame. This counter is decremented after the read transfer of each element. The maximum number of elements per frame transfer is 65 535. Once the last element in each frame is reached, ELEMENT COUNT is reloaded with the 16 LSBs of the DMA controller global count reload register selected by the CNT RELOAD field in the DMA controller channel primary control register. This reloading is unaffected by autoinitialization mode. Before a block transfer begins, the counter and selected DMA controller global count reload register must be loaded with the same 16 LSBs to assure that the first and remaining frames have the same number of elements per frame. In any multiframe transfer, a reload value must always be specified, not just when autoinitialization is enabled. If the element count is initialized as 0, operation is undefined.

Figure 5–5. DMA Channel Transfer Counter Register

31 16	15 0
FRAME COUNT	ELEMENT COUNT
RW, +0	RW, +0

Figure 5–6. DMA Global Count Reload Register Used As Transfer Counter Reload

31 16	15 0
FRAME COUNT RELOAD	ELEMENT COUNT RELOAD
RW, +0	RW, +0

5.6 Synchronization: Triggering DMA Transfers

Synchronization allows DMA transfers to be triggered by events such as interrupts from internal peripherals or external pins. Three types of synchronization can be enabled for each channel:

- ❑ Read synchronization: Each read transfer waits for the selected event to occur before proceeding.
- ❑ Write synchronization: Each write transfer waits for the selected event to occur before proceeding.
- ❑ Frame synchronization: Each frame transfer waits for the selected event to occur before proceeding.

Selection of Synchronization Events: The events are selected by the RSYNC and WSYNC fields in the DMA channel primary control register. If FS = 1 in this register, then the event selected by RSYNC enables an entire frame, and WSNYC must be set to 00000b. If a channel is set up to operate in split mode (SPLIT $\neq$ 00b), RSYNC and WSYNC must be set to non-zero values. Up to 31 events are available. If the value of these fields is set to 00000b, no synchronization is necessary. In this case, the read, write, or frame transfers occur as soon as the resource is available to that channel. The association between values in these fields and events is shown in Table 5–6. This is similar to the fields in the interrupt selector (see section 13.4, *Configuring the Interrupt Selector*). The differences are that the McBSP generates separate interrupts and DMA synchronization events and that the DSPINT is located differently in the encoding.

Table 5–6. Synchronization Events

Event Number (Binary)	Event Acronym	Event Description
00000	None	No synchronization
00001	TINT0	Timer 0 interrupt
00010	TINT1	Timer 1 interrupt
00011	SD_INT	EMIF SDRAM timer interrupt
00100	EXT_INT4	External interrupt pin 4
00101	EXT_INT5	External interrupt pin 5
00110	EXT_INT6	External interrupt pin 6
00111	EXT_INT7	External interrupt pin 7
01000	DMA_INT0	DMA channel 0 interrupt
01001	DMA_INT1	DMA channel 1 interrupt

Table 5–6. Synchronization Events (Continued)

Event Number (Binary)	Event Acronym	Event Description
01010	DMA_INT2	DMA channel 2 interrupt
01011	DMA_INT3	DMA channel 3 interrupt
01100	XEVT0	McBSP 0 transmit event
01101	REVT0	McBSP 0 receive event
01110	XEVT1	McBSP 1 transmit event
01111	REVT1	McBSP 1 receive event
10000	DSPINT	Host processor to DSP interrupt
10001	XEVT2	McBSP 2 transmit event
10010	REVT2	McBSP 2 receive event
Other	Reserved	

5.6.1 Latching of DMA Channel Event Flags

The DMA channel secondary control register (described in Table 5–4) contains STAT and CLR fields for read and write synchronization (RSYNC and WSYNC) events.

Latching of DMA Synchronization Events: A low-to-high transition (or high-to-low transition when selected by WSPOL or RSPOL) of the selected event is latched by each DMA channel. The occurrence of this transition causes the associated STAT field to be set in the DMA channel secondary control register. If no synchronization is selected, the STAT bit is always read as 1. A single event can trigger multiple actions.

User Clearing and Setting of Events: By clearing pending events before starting a block transfer, you can force the DMA channel to wait for the next event. Conversely, by setting events before starting a block transfer, you can force the synchronization events necessary for the first element transfer. You can clear or set events (and thus the related STAT bit) by writing 1 to the corresponding CLR or STAT field, respectively. Writing a 0 to either of these bits has no effect. Also, the CLR bits are always read as 0 and have no associated storage. Separate bits for setting or clearing are provided to allow clearing of some bits without setting others and vice versa. Your user manipulation of events has priority over any simultaneous automated setting or clearing of events.

5.6.2 Automated Event Clearing

The latched STAT for each synchronizing event is automatically cleared only when any action associated with that event is completed. Events are cleared as quickly as possible to reduce the minimum time between synchronizing events. This capability effectively increases the rate at which events can be recognized. This is described for each type of synchronization:

- ❑ Clearing read synchronization condition: The latched condition for read synchronization is cleared when the DMA completes the request for the associated read transfer.
- ❑ Clearing write synchronization condition: The latched condition for write synchronization is cleared when the DMA completes the request for the associated write transfer.
- ❑ Clearing frame synchronization condition: Frame synchronization clears the RSYNC STAT field when the DMA completes the request for the first read transfer in the new frame.

5.6.3 Synchronization Control

The DMA of the ’C6202 allows for more flexible control over how external synchronization events are recognized. The polarity of external events can be inverted to an active-low by setting WSPOL and/or RSPOL to 1. WSPOL affects write-synchronized transfers, while RSPOL affects read- and frame-synchronized transfers.

During a frame-synchronized transfer, the DMA channel may be configured (by setting FSIG = 1) to not recognize an external interrupt as a synchronization event while performing a burst. The channel will internally monitor its burst status, and will latch its synchronization event only when a frame transfer is not in progress.

Figure 5–7 shows the scenario to produce the desired synchronizing event. The figure illustrates both active-high and active-low operation, but the following explanation pertains to active-low operation.

1) The transition of EXT_INTx from high-to-low while a burst is not in progress triggers a synchronizing event.

2) The synchronizing event triggers a frame transfer, which gates off the DMA sync event. During the sync event, transitions on EXT_INTx are ignored.

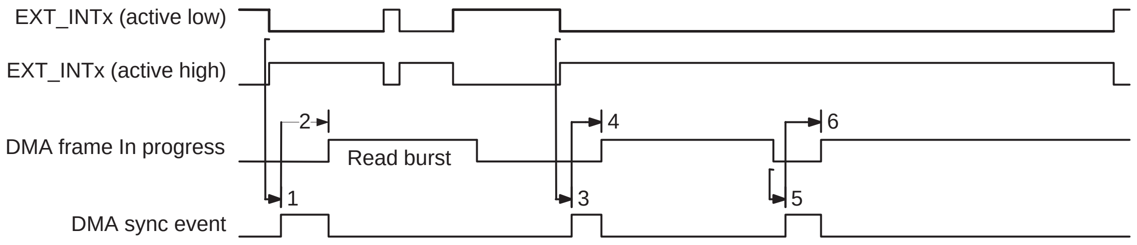
3) Same as 1

4) Same as 2

5) If EXT_INTx is still active after the burst, then the high-to-low transition on the internal frame-in-progress signal causes a DMA sync event.

6) The new DMA sync event triggers another burst.

Figure 5–7. Synchronization Flags



The new synchronization modes are available to better interface to an external FIFO that is serving as a data buffer. Since a synchronization event is often triggered off of a flag indicating the amount of data currently inside the FIFO, there is a high likelihood that a race-condition could occur. If the DMA were to read from the FIFO (clearing the flag that generated the synchronization event), and a new element were written to the FIFO immediately after, then the flag could be reset and a new frame would be synchronized to start immediately following the current burst. By setting the DMA to ignore events during a current burst, this situation is avoided.

Another feature of this is that if the synchronization event stays active throughout a burst, then it will be latched again following the burst. This, too, was done for a more robust FIFO interface. This is due to the fact that the transition from active to inactive of the FLAG can only occur during a burst. For example, if the ‘C6202 is the reader from a FIFO, the only way for the FIFO to go from half-full (/HF active) to less than half-full (/HF inactive) is by reading from the FIFO. If the flag were to stay active throughout the burst, then it is known that the data source was able to provide another set of data to the FIFO before the ‘C6202 was able to read the frame.

These new features are only used by the DMA when WSPOL, RSPOL, or FSIG are properly configured. If all fields are left as 0 (default) the ‘C6202 DMA functions identically to the ‘C6201 DMA.

5.7 Address Generation

For each channel, the DMA controller performs address computation for each read transfer and write transfer. The DMA controller allows creation of a variety of data structures. For example, the DMA controller can traverse an array incrementing through every nth element. Also, you can program it to effectively treat the various elements in a frame as coming from separate sources and group each source's data together.

The DMA channel source address and destination address registers (shown in Figure 5–8 and Figure 5–9, respectively) hold the addresses for the next read transfer and write transfer, respectively.

Figure 5–8. DMA Channel Source Address Register

31	0
SOURCE ADDRESS	
RW, +x	

Figure 5–9. DMA Channel Destination Address Register

31	0
DESTINATION ADDRESS	
RW, +x	

5.7.1 Basic Address Adjustment

As indicated in Table 5–3, the SRC DIR and DST DIR fields can set the index to increment by element size, decrement by element size, use a global index value, or not affect the DMA channel source and destination address registers, respectively. By default, these values are set to 00b to disable address modification. If incrementing or decrementing is selected, the amount of the address adjustment is determined by the size of the element size in bytes. For example, if the source address is set to increment and 16-bit halfwords are being transferred, then the address is incremented by 2 after each read transfer.

5.7.2 Address Adjustment With the Global Index Registers

The particular DMA global index register shown in Figure 5–10 is selected via the INDEX field in the DMA channel primary control register. Unlike basic address adjustment, this mode allows different adjustment amounts depending on whether the element transfer is the last in the current frame. The normal adjustment value (ELEMENT INDEX) is contained in the 16 LSBs of the selected DMA global index register. The adjustment value for the end of the frame (FRAME INDEX) is determined by the 16 MSBs of the selected DMA global index register. Both of these fields contain signed 16-bit values. Thus, the index amounts can range from –32 768 to 32 767.

Figure 5–10. DMA Global Index Register

31 16	15 0
FRAME INDEX	ELEMENT INDEX
RW, +0	RW, +0

These fields affect address adjustment as follows.

❑ ELEMENT INDEX: For element transfers except the last one in a frame, ELEMENT INDEX determines the amount to be added to the DMA channel source or the destination address register as selected by the SRC DIR or DST DIR field after each read or write transfer, respectively.

❑ FRAME INDEX: If the read or write transfer is the last in a frame, FRAME INDEX (and not ELEMENT INDEX) is used for address adjustment. This adjustment occurs in both single frame and multiframe transfers, including transfers after the last frame in a block.

5.7.3 Element Size, Alignment, and Endianness

By using the ESIZE field in the DMA channel primary control register, you can configure the DMA to transfer 8-bit bytes, 16-bit halfwords, or 32-bit words on each transfer. The following registers and fields must be loaded with properly aligned values:

❑ DMA channel source and destination address registers and any associated reload registers

❑ ELEMENT INDEX

❑ FRAME INDEX

In the case of word transfers, these registers must contain values that are multiples of 4 and thus aligned on a word address. In the case of halfword transfers, the values must be multiples of 2 and thus aligned on a halfword address. If unaligned values are loaded, operation is undefined. There is no alignment restriction for byte transfers. All accesses to program memory must be 32 bits in width. Also, you must be aware of the endianness when trying to read a particular 8-bit or 16-bit field within a 32-bit register. For example, in little endian mode, an address ending in 00b selects the least significant byte, whereas 11b selects the least significant byte in big-endian mode.

5.7.4 Using a Frame Index to Reload Addresses

In an autoinitialized, single-frame block transfer, the FRAME INDEX can be used in place of a reload register to recompute the next address. If the following fields contain the values listed, a single frame transfer moves the ten bytes from a static external address to alternating locations (skipping one byte between each two bytes):

- ❑ SRC DIR = 00b, the static source address
- ❑ DST DIR = 11b, the programmable index value
- ❑ ELEMENT INDEX = 10b, the 2-byte destination stride
- ❑ FRAME INDEX = –(9 × 2) = –18 = FFEEh, restart destination for the transfer at the same location by moving 18 bytes.

5.7.5 Transferring a Large Single Block

ELEMENT COUNT can be used in conjunction with FRAME COUNT to allow single-frame block transfers of more than 65 535 bytes. The product of ELEMENT COUNT and FRAME COUNT can form a larger effective element count. The following must be performed:

- ❑ If the address is to be adjusted using a programmable value (DIR = 11b), FRAME INDEX must equal ELEMENT INDEX if the address adjustment is determined by a DMA global index register. This applies to both source and destination addresses. If the address is not to be adjusted by a programmable value, this constraint does not apply, because the same address adjustment occurs by default at element and frame boundaries.
- ❑ Frame synchronization must be disabled (that is, FS must be set to 0 in the DMA channel primary control register). This prevents requirements for synchronization in the middle of the large block.
- ❑ The number of elements in the first frame is Ei. The number of elements in successive frames is ((F – 1) × Er). The effective element count is ((F – 1) × Er) + Ei

  where:

F	=	Initial value of FRAME COUNT
Er	=	ELEMENT COUNT reload value
Ei	=	Initial value of ELEMENT COUNT

  Thus, to transfer 128K + 1 elements, you could set F to 5, Er to 32K, and Ei to 1.

5.7.6 Sorting

The following procedure is used to locate transfers in memory by ordinal location within a frame (i.e., the first transfer of the first frame followed by the first transfer of the second frame):

- ❑ ELEMENT INDEX is set to F × S.
- ❑ FRAME INDEX is set to –(((E – 1) × F) – 1) × S

where:

E =	Initial value of ELEMENT COUNT (the number of elements per frame) initial value of the ELEMENT COUNT RELOAD
F =	Initial value of FRAME COUNT (the total number of frames)
S =	Element size in bytes

Consider a transfer with three frames (F = 3) of four halfword elements each (E = 4, S = 2). This corresponds to ELEMENT INDEX = 3 × 2 = 6 and FRAME INDEX = –(((4 – 1) × 3) – 1) × 2 = FFF0h. Assume that the source address is not modified and the destination increments starting at 8000 0000h. Table 5–7 shows the data in the order in which it is transferred, and Table 5–8 shows how the data appears in memory after transfers are finished.

Table 5–7. Sorting Example in Order of DMA Transfers

Frame	Element	Address (Hex)	Postadjustment
0	0	8000 0000	+6
0	1	8000 0006	+6
0	2	8000 000C	+6
0	3	8000 0012	–16
1	0	8000 0002	+6
1	1	8000 0008	+6
1	2	8000 000E	+6
1	3	8000 0014	–16
2	0	8000 0004	+6
2	1	8000 000A	+6
2	2	8000 0010	+6
2	3	8000 0016	–16

Table 5–8. Sorting in Order of First by Address

Address (Hex)	Frame	Element
8000 0000	0	0
8000 0002	1	0
8000 0004	2	0
8000 0006	0	1
8000 0008	1	1
8000 000A	2	1
8000 000C	0	2
8000 000E	1	2
8000 0010	2	2
8000 0012	0	3
8000 0014	1	3
8000 0016	2	3

5.8 Split-Channel Operation

Split-channel operation allows a single DMA channel to service both the input (receive) and output (transmit) streams from an external or internal peripheral with a fixed address.

5.8.1 Split DMA Operation

Split-channel operation consists of transmit element transfers and receive element transfers. In turn, these transfers each consist of a read and a write transfer:

- ❑ Transmit element transfer
  - ■ Transmit read transfer: Data is read from the DMA channel source address. The source address is then adjusted as configured. The transfer count is then decremented. This event is not synchronized.
  - ■ Transmit write transfer: Data from the transmit read transfer is written to the split destination address. This event is synchronized as indicated by the WSYNC field. The DMA channel keeps track internally of the number of pending receive transfers.
- ❑ Receive element transfer
  - ■ Receive read transfer: Data is read from the split source address. This event is synchronized as indicated by the RSYNC field.
  - ■ Receive write transfer: Data from the receive read transfer is written to the destination address. The destination address is then adjusted as configured. This event is not synchronized.

Because only a single element count and frame count exists per channel, the element count and the frame count are the same for both the received and the transmitted data. For split-channel operation to work properly, both the RSYNC and WSYNC fields must be set to non-zero synchronization events. Also, frame synchronization must be disabled in split-channel operation.

The above sequence is maintained for all transfers. However, the transmit transfers do not have to wait for all previous receive element transfers to finish before proceeding. Therefore, it is possible for the transmit stream to get ahead of the receive stream. The DMA channel transfer counter decrements (or reinitialize) after the associated transmit transfer finishes. However, re-initialization of the source address register occurs after all transmit element transfers finish. This configuration works as long as transmit transfers do not eight or more transfers ahead of the receive transfers. If the transmit transfers do get ahead of the receive transfers, transmit element transfers are stopped, possibly causing synchronization events to be missed. For cases in which receive or transmit element transfers are within seven or less transfers of the other, the DMA channel maintains this information as internal status.

5.8.2 Split Address Generation

The DMA global address register selected by the SPLIT field in the DMA primary control register determines the address of the peripheral that is to be accessed for split transfer:

- ❑ Split source address: This address is the source for the input stream to the ’C6000. The selected DMA global address register contains this split source address.
- ❑ Split destination address: This address is the destination for the output data stream from the ’C6000. The split destination address is assumed to be one word address (four byte addresses) greater than the split source address.

Figure 5–11. DMA Global Address Register Used for Split Address

31 – 3	2 – 0
SPLIT ADDRESS	Reserved
RW, +0	R, +0

The two LSBs are fixed at 0 to force alignment at a word address. The third LSB is 0 because the split source address is assumed to be on an even word boundary. Thus, the split destination address is assumed to be on an odd word boundary. These relationships hold regardless of the width of the transfer. For external peripherals, you must design address decoding appropriately to adhere to this convention.

5.9 Resource Arbitration and Priority Configuration

Priority decides which of competing requesters have control of a resource with multiple requests. The requesters include:

- ❑ The DMA channels
- ❑ The CPU's program and data accesses

The resources include:

- ❑ Internal data memory
- ❑ Internal program memory
- ❑ The internal peripheral registers, which are accessed through the peripheral bus
- ❑ External memory, accessed through the external memory interface (EMIF)
- ❑ Expansion memory, accessed through the expansion bus

Two aspects of priority are programmable:

- ❑ DMA versus CPU priority: Each DMA channel can be independently configured in high-priority mode by setting the PRI bit in the associated DMA channel primary control register. The AUXPRI field in the DMA auxiliary control register allows the same feature for the auxiliary channel. When in high-priority mode, the associated channel's requests are sent to the appropriate resource with a signal indicating the high priority status. By default, all these fields are 0, disabling the high-priority mode. Each resource can use this signal in its own priority scheme for resolving conflicts. See to resource specific documentation for information how a particular resource uses this signal.
- ❑ Priority between DMA channels: The DMA controller has a fixed priority scheme, with channel 0 having highest priority and channel 3 having lowest priority. The auxiliary channel can be given a priority anywhere within this hierarchy.

5.9.1 DMA Auxiliary Control Register and Priority Between Channels

The fields in the DMA auxiliary control register affect the auxiliary channel. The fields in this register are shown in Figure 5–12 and are summarized Table 5–9.

Figure 5–12. DMA Auxiliary Control Register

31 5	4	3 0
Reserved	AUXPRI	CH PRI
R, +0	RW, +0	RW, +0

Table 5–9. DMA Auxiliary Control Register Field Descriptions

Field	Description
CH PRI	DMA channel priority CH PRI = 0000b: fixed channel priority mode auxiliary channel highest priority CH PRI = 0001b: fixed channel priority mode auxiliary channel 2nd-highest priority CH PRI = 0010b: fixed channel priority mode auxiliary channel 3rd-highest priority CH PRI = 0011b: fixed channel priority mode auxiliary channel 4th-highest priority CH PRI = 0100b: fixed channel priority mode auxiliary channel lowest priority CH PRI = other, reserved

AUXPRI	Auxiliary channel priority mode
	
	AUXPRI = 0: CPU priority
	AUXPRI = 1: DMA priority

The priority assigned to the DMA channels determines which DMA channel performs a read or write transfer first, given that two or more channels are ready to perform transfers.

The priority of the auxiliary channel is configurable by programming the CH PRI field in the DMA auxiliary control register. By default, CH PRI contains the value 0000b at reset. This value sets the auxiliary channel as highest priority, followed by channel 0, followed by channel 1, followed by channel 2, with channel 3 having lowest priority.

Arbitration between channels occurs independently every CPU clock cycle for read and write transfers. Any channel that is in the process of waiting for synchronization of any kind can lose control of the DMA controller to a lower priority channel. Once that synchronization is received, that channel can regain control of the DMA controller from a lower priority channel. This rule is applied independently to the transmit and receive portions of a split mode transfer. The transmit portion has higher priority than the receive portion.

If multiple DMA channels and the CPU are contending for the same resource, the arbitration between DMA channels occurs first. Then, arbitration between the highest priority DMA channel and the CPU occurs. Normally, if a channel has lower priority than the CPU, all lower priority channels should also are lower priority than the CPU. Similarly, if a channel has a higher priority than the CPU, all higher priority channels should also be higher priority than the CPU.

The arbitration between the DMA controller and the CPU is performed by the resource for which they are contending. For more information, see resource-specific documentation. Note that a channel's PRI field should be modified only when that channel is paused or stopped.

5.9.2 Switching Channels

A higher priority channel gains control of the DMA controller from a lower priority channel once it has received the necessary read synchronization. In switching channels, the current channel allows all data from requested reads to be completed. The DMA controller determines which higher priority channel gains control of the DMA controller read operation. That channel then starts its read operation. Simultaneously, write transfers from the previous channel are allowed to finish.

5.10 DMA Channel Condition Determination

Several condition status flags are available to inform you of significant events or potential problems in DMA channel operation. These flags reside in the DMA channel secondary control register.

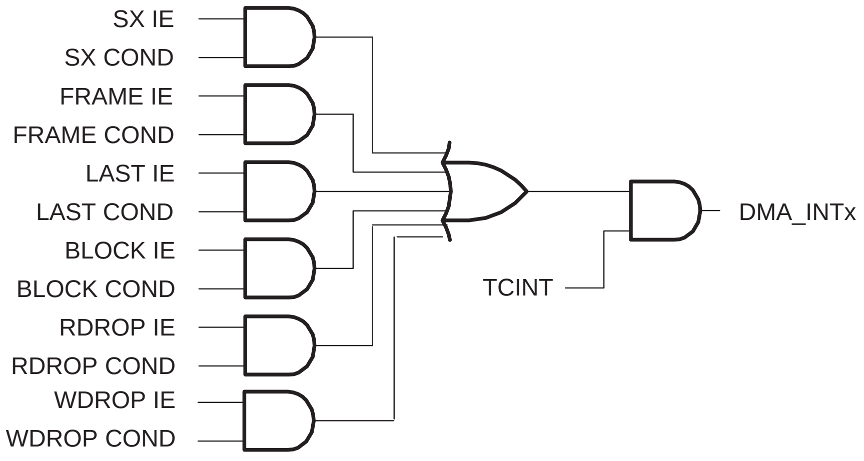
These registers also provide the means to enable the DMA channels to interrupt the CPU through their corresponding interrupt enable (IE) fields. If a condition flag and its corresponding IE bit are set, that condition is enabled to contribute to the status of the interrupt signal from the associated DMA channel to the CPU. If the TCINT bit in the DMA channel *x* primary control register is set, the logical OR of all enabled conditions forms the DMA_INT*x* signal. Otherwise, the DMA_INT*x* remains inactive. This logic is shown in Figure 5–13. If selected by the interrupt selector, a low-to-high transition on that DMA_INT causes an interrupt condition to be latched by the CPU.

The SX COND, WDROP COND, and RDROP COND bits in the DMA channel secondary control register are treated as warning conditions. If these conditions are enabled and active, they move the DMA channel from the running to the pause state, regardless of the value of the TCINT bit.

If a condition bit's associated IE bit is set, that condition bit can be cleared only by you writing a 0 to it. Otherwise, that condition bit can be cleared automatically. Writing a 1 to a COND bit has no effect. Thus, you cannot manually force one of the conditions.

Most bits in this register are cleared at reset. The exception is the interrupt enable for the block transfer complete event (BLOCK IE), which is set at reset. Thus, by default, the block transfer complete condition is the only condition that can contribute to the CPU interrupt. Other conditions can be enabled by setting the associated IE bit.

Figure 5–13. Generation of DMA Interrupt for Channel x From Conditions



5.10.1 Definition of Channel Conditions

Table 5–10 describes each of the condition flags in the DMA channel secondary control register.

Depending on the system application, these conditions can represent errors. The last frame condition can be used to change the reload register values for autoinitialization. The frame index and element count reload are used every frame. Thus, you must wait to change these values until all but the last frame transfer in a block transfer finishes. Otherwise, the current block transfer is affected.

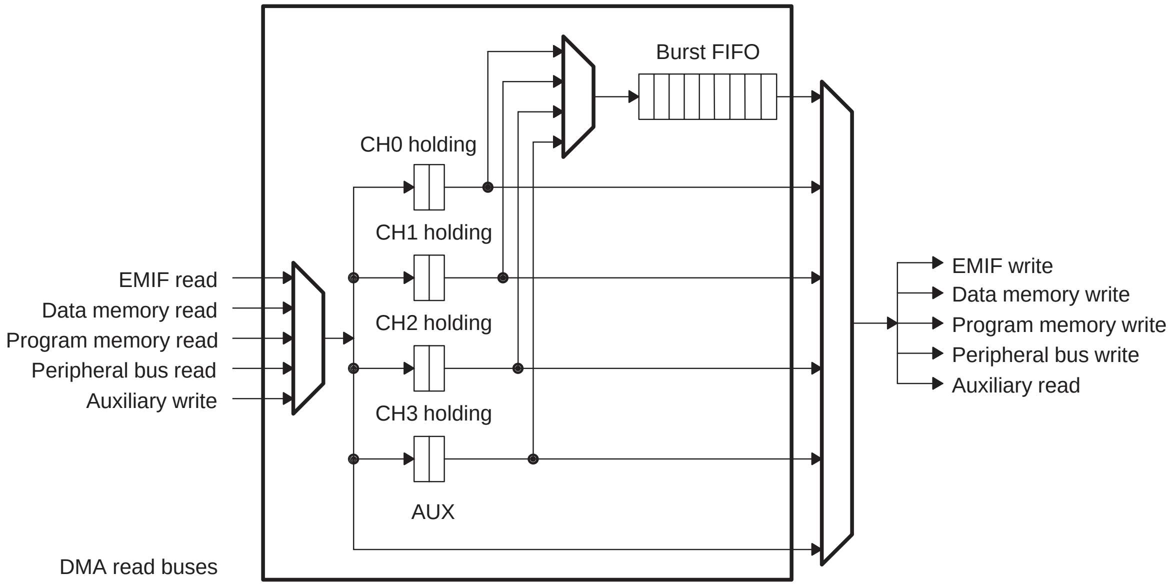
Table 5–10. DMA Channel Condition Descriptions

			COND Cleared By	
Bitfield	**Event**	**Occurs if…**	**If IE Enabled**	**Otherwise**
SX	Split transmit overrun receive	The split operation is enabled and transmit element transfers get seven or more element transfers ahead of receive element transfers	A user write of 0 to COND	
FRAME	Frame complete	After the last write transfer in each frame is written to memory	A user write of 0 to COND	Two CPU clocks later
LAST	Last frame	After all counter adjustments for the next-to-last frame in a block transfer finish	A user write of 0 to COND	Two CPU clocks later
WDROP RDROP	Dropped read/write synchronization	A subsequent synchronization event occurs before the last one is cleared	A user write of 0 to COND	
BLOCK	Block transfer finished	After the last write transfer in a block transfer is written to memory	A user write of 0 to COND	Two CPU clocks later

5.11 DMA Controller Structure

Figure 5–14 shows the internal data movement paths of the DMA controller, including data buses and internal holding registers.

Figure 5–14. DMA Controller Data Bus Block Diagram



5.11.1 Read and Write Buses

Each DMA channel can independently select one of four sources and destinations:

- ❑ EMIF
- ❑ Internal program memory
- ❑ Internal data memory
- ❑ Internal peripheral bus

Read and write buses from each source interface to the DMA controller.

The auxiliary channel also has read and write buses. However, since the auxiliary channel provides address generation for the DMA, the naming convention of its buses differs. For example, data writes from the auxiliary channel through the DMA controller are performed through the auxiliary write bus. Similarly, data reads from the auxiliary channel through the DMA controller are performed through the auxiliary read bus.

5.11.2 DMA FIFO

A 9-level DMA FIFO holding path facilitates bursting to high-performance memories, such as internal program and data memory, as well as external synchronous DRAM (SDRAM) or synchronous burst SRAM (SBSRAM). When combined with a channel’s holding registers this path effectively becomes an 11-level FIFO. Only one channel controls the FIFO at any given time. For a channel to gain control of the FIFO, all of the following conditions must be met:

- ❑ The channel does not have read or write synchronization enabled. Since split-channel mode requires read and write synchronization, the FIFO is not used by a channel in that mode. If only frame synchronization is enabled, the FIFO can still be used by that channel.
- ❑ The channel is running.
- ❑ The FIFO is void of data from any other channel.
- ❑ The channel is the highest priority channel of those that meet the preceding three conditions.

The third restriction minimizes head-of-line blocking. Head-of-line blocking occurs when a DMA request of higher priority waits for a series of lower priority requests to come in before issuing its first request. If a higher priority channel requests control of the DMA controller from a lower priority channel, only the last request of the previous channel must finish. After that, the higher priority channel completes its requests through its holding registers. The holding registers do not allow as high of a throughput through the DMA controller. The lower priority channel begins no more read transfers but flushes the FIFO by completing its write transfers in the gaps. Because the higher priority channel is not yet in control of the FIFO, there are gaps in its access where the lower priority channel can drain its transfer from the FIFO. Once the FIFO is clear, if the higher priority channel has not stopped, it gains control of the FIFO.

The DMA FIFO has two purposes:

- ❑ Increasing performance
- ❑ Decreasing arbitration latency

For increased performance the FIFO allows read transfers to get ahead of write transfers. This feature minimizes penalties for variations in available transfer bandwidth at either end of the element transfer. Thus, the DMA can capitalize on separate windows of opportunity at the read and write portion of an element transfer. If the requesting DMA channel is using the FIFO, the resources are capable of sustaining read or write accesses at the CPU clock cycle rate. However, there may be some latency in performing the first access. The handshaking between a resource and the DMA controller controls the rate of consecutive requests and the latency of received read transfer data.

The other function of the DMA FIFO is capturing read data from any pending requests for a particular resource. For example, consider the situation in which the DMA controller is reading data from pipelined external memory such as SDRAM or SBSRAM into internal data memory. Assume that the CPU is given higher priority over the DMA channel making requests and that it makes a competing program fetch request to the EMIF. Assume that simultaneously the CPU is accessing all banks of internal memory, blocking out the DMA controller. In this case, the FIFO allows the pending DMAs to finish and the program fetch to proceed. Due to the pipelined request structure of the DMA controller, at any time the DMA controller can have pending read transfer requests whose data has not yet arrived. Once enough requests to fill the empty spots in the FIFO are outstanding, the DMA controller stops making further read transfer requests.

5.11.3 Internal Holding Registers

Each channel has dedicated internal holding registers. If a DMA channel is transferring data through its holding registers rather than the internal FIFO, read transfers are issued consecutively. Depending on whether the DMA controller is in split mode or not, additional restrictions can apply:

In split mode, the two registers serve as separate transmit and receive data stream holding registers for split mode. For both the transmit and receive read transfer, no subsequent read transfer request is issued until the associated write transfer request completes.

In nonsplit mode, once the data arrives a subsequent read transfer can be issued without waiting for the associated write transfer to finish. However, because there are two holding registers, read transfers can get only one transfer ahead of write transfers.

5.11.4 DMA Performance

The DMA controller can perform element transfers with single-cycle throughput if it accesses separate resources for the read transfer and write transfer and both these resources have single-cycle throughput. An example is an unsynchronized block transfer from single-cycle external SBSRAM to internal data memory without any competition from any other channels or the CPU. The DMA controller performance can be limited by:

- ❑ The throughput and latency of the resources it requests
- ❑ Waiting for read, write, or frame synchronization
- ❑ Interruptions by higher priority channels
- ❑ Contention with the CPU for resources

5.12 DMA Action Complete Pins

The DMA action complete pins (DMAC0–DMAC3) provide a method of feedback to external logic by generating an event for each channel. If it is specified by the DMAC EN field in the DMA channel secondary control register, the DMAC pin can reflect the status of RSYNC STAT, WSYNC STAT, BLOCK COND, or FRAME COND or be treated as a high or low general purpose output. If the DMAC pin reflects RSYNC STAT or WSYNC STAT externally, then once a synchronization event has been recognized, DMAC transitions from low-to-high. Once that event has been serviced as indicated by the status bit being cleared, DMAC changes from high-to-low. Before being sent off chip, the DMAC signals are synchronized by CLKOUT1. The active period of these signals is a minimum of two CLKOUT1 periods wide.

5.13 Emulation

When you are using the emulator for debugging, you can halt the CPU on an execute packet boundary for single-stepping, benchmarking, profiling, or other debugging purposes. You can configure the DMA controller pause during this time or to continue running. This configuration is accompanied by setting the EMOD bit in the DMA primary control register to 0 or 1. If the DMA controller is paused, the STATUS field reflects the paused state of the channel. The auxiliary channel continues running during an emulation halt. This emulation closely simulates single-stepping DMA transfers. DMA channels with EMOD = 1 can couple multiple transfers between single steps; a successful step can require multiple outstanding transfers to finish first.

Chapter 6

EDMA Controller

This chapter describes the new enhanced DMA controller for the TMS320C6211/C6711. EDMA transfer parameters, types and performance are discussed. This chapter also describes the new quick DMA for fast data requests.

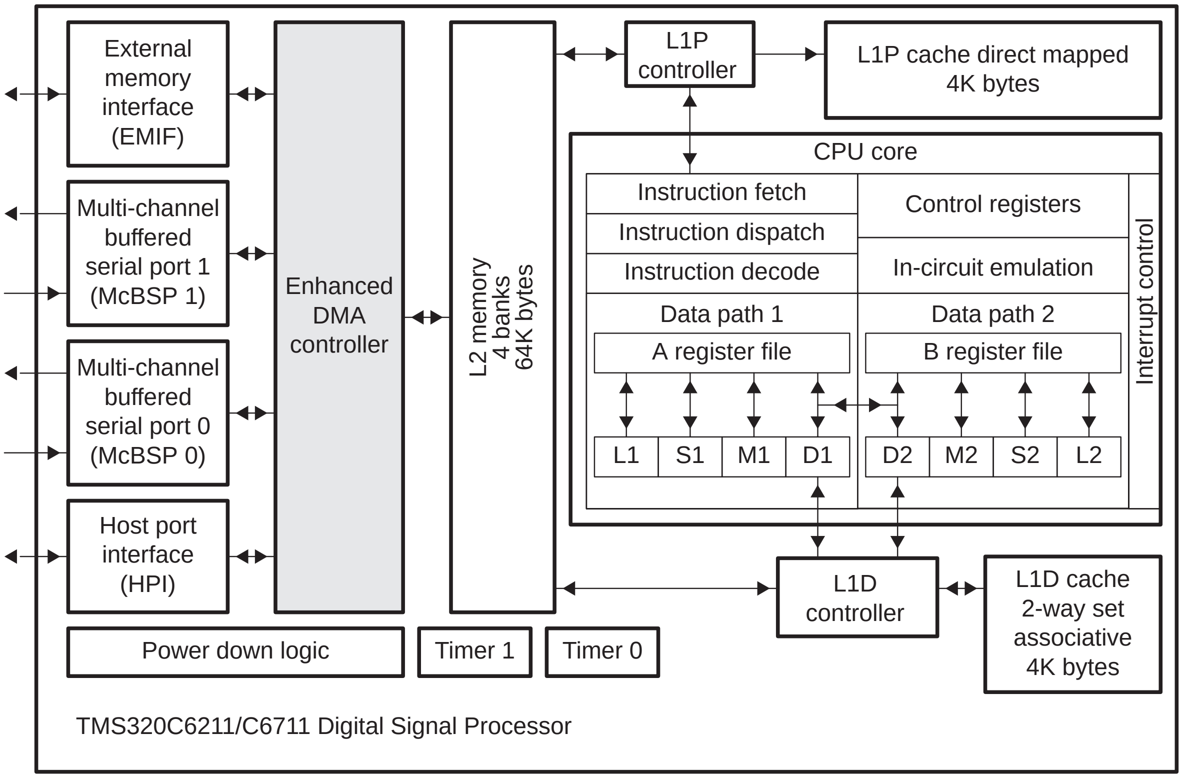
Topic **Page**

6.1 Overview

The TMS320C6211/C6711 device performs data transfers between on-chip and/or off-chip locations using either the CPU or the enhanced direct memory access (EDMA) controller. Typically, block data transfers and transfer requests from peripherals are performed by the EDMA thus relieving the CPU to do performance-intensive operations.

The EDMA controller in the ’C6211/C6711 is different in architecture to the previous TMS320C6000 devices. The EDMA includes several enhancements to the ‘C6201/’C6701 DMA in that it provides 16 channels with programmable priority, and the ability to link data transfers. The EDMA allows movement of data to/from internal memory (L2 SRAM), peripherals, and between external memory spaces.

Figure 6–1. TMS320C6211/C6711 Block Diagram

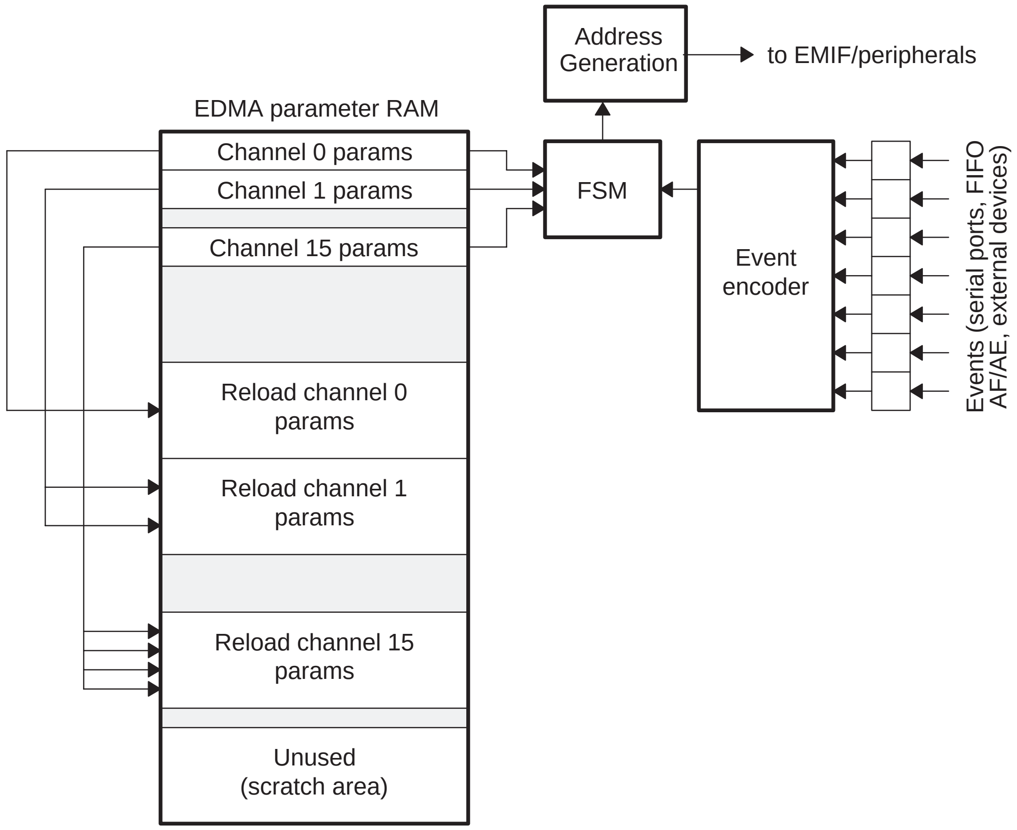

External memory interface (EMIF)
Multi-channel buffered serial port 1 (McBSP 1)
Multi-channel buffered serial port 0 (McBSP 0)
Host port interface (HPI)
Enhanced DMA controller
L2 memory 4 banks 64K bytes
L1P controller
L1P cache direct mapped 4K bytes
CPU core
Instruction fetch
Instruction dispatch
Instruction decode
Control registers
In-circuit emulation
Interrupt control
Data path 1
Data path 2
A register file
B register file
L1
S1
M1
D1
D2
M2
S2
L2
L1D controller
L1D cache 2-way set associative 4K bytes
Power down logic
Timer 1
Timer 0
TMS320C6211/C6711 Digital Signal Processor

The EDMA controller comprises:

- ❑ Event and interrupt processing registers
- ❑ Event encoder
- ❑ Parameter RAM, and
- ❑ Address generation hardware

A block diagram of the EDMA controller is shown in Figure 6–2.

Figure 6–2. EDMA Controller



EDMA events are captured in the event register. An event is a synchronization signal that triggers an EDMA channel to start a transfer. If events occur simultaneously, they are resolved by way of the event encoder. The transfer parameters corresponding to this event which is stored in the EDMA parameter RAM, are passed onto the address generation hardware, which address the EMIF and/or peripherals to perform the necessary read and write transactions.

The quick DMA (QDMA) is a new feature in the ’C6211/C6711 device that provides a fast and efficient way to transfer data. QDMA functions similarly to the EDMA. QDMA is best suited for applications that require quick data transfers, such as data requests in a tight loop algorithm. See section 6.16 for more details.

6.2 EDMA Terminology

The following definitions help in understanding some of the terms used in this chapter:

- **Element transfer:** The transfer of a single data element from source to destination. Each element can be transferred based on a sync event if required.
- **Frame:** A group of elements comprise a frame. The elements in a frame can be staggered or can be contiguous. A frame can be transferred with or without a synchronizing event. The term ‘frame’ is used in context with non-2D transfer. Non-2D transfer is defined below.
- **Array:** A group of contiguous elements comprise an array. Therefore, the elements in an array cannot be spaced by an element index. An array can be transferred with or without a synchronizing event. The term ‘array’ is used in context with 2-Dimensional transfers. 2D transfer is defined below.
- **Block:** A group of arrays or frames form a block. Synchronized and unsynchronized block transfers are supported.
- **2-dimensional (2D) transfer:** A group of arrays comprise a 2D block transfer. The first dimension is the contiguous elements in an array, and the second dimension is the number of such arrays. The number of arrays (frame/array count, FC) in a block can range from 1 to 65536 (corresponding to an FC value range of 0 to 65535).
- **Non-2D transfer:** A group of frames comprise a non-2D block transfer. The number of frames (frame/array count, FC) in a block can be between 1 and 65536 (corresponding to an FC value range of 0 to 65535).

6.3 Event Processing and EDMA Control Registers

Each of the 16 channels in the EDMA have specific events associated with them. These events trigger the data transfer associated with that channel. The list of control registers that perform various processing of events is shown in Table 6–1.

An event is signaled to the EDMA controller by way of a low-to-high transition on one of its 16 event inputs. All events are captured in the event register (ER), even when the events are disabled. The 32-bit ER shown in Figure 6–3 contains one bit for each event, or a total of 16 bits. Section 6.7.1 describes the type of synchronization events and the EDMA channels associated with each of them.

Table 6–1. EDMA Control Registers

Byte Address	Acronym	Register Name	Section
01A0 FFE0h	PQSR	Priority queue status register	6.14
01A0 FFE4h	CIPR	Channel interrupt pending register	6.13
01A0 FFE8h	CIER	Channel interrupt enable register	6.13
01A0 FFECh	CCER	Channel chain enable register	6.13.2
01A0 FFF0h	ER	Event register	6.3
01A0 FFF4h	EER	Event enable register	6.3
01A0 FFF8h	ECR	Event clear register	6.3
01A0 FFFCh	ESR	Event set register	6.3

In addition to the event register, the EDMA controller also provides the user the option of enabling/disabling events. Any of the 16 event bits in the 32-bit event enable register (EER) shown in Figure 6–4 can be set to ‘1’ to enable that event. Note that all events that are captured by the EDMA are latched in the ER even if that event is disabled. This is analogous to an interrupt enable and interrupt-pending register for interrupt processing. This ensures that no events are dropped by the EDMA. Thus, re-enabling an event with a pending event signaled in the ER forces the EDMA controller to process that event according to its priority. Writing a ‘0’ to the corresponding bit in the EER disables an event.

Once an event has been posted in the ER, the event can be cleared in two ways. If the event is enabled in the event enable register (EER), the corresponding event bit in the ER is cleared as soon as the EDMA submits a transfer request for that event. Alternatively, if the event is disabled in the EER, the CPU can clear the event by way of the event clear register (ECR), shown in Figure 6–5. Writing a ‘1’ to any of the bits clears the corresponding event; writing a ’0’ has no effect. This feature allows the CPU to release a lock-up or error condition. Therefore, once an event bit is set in the ER, it remains set until the EDMA submits a transfer request for that event or the CPU clears the event by setting the relevant bit in the ECR.

The CPU can also set events by way of the event set register (ESR) shown in Figure 6–6. Writing a ‘1’ to one of the 16 event bits causes the corresponding bit to be set in the event register. The event does not have to be enabled in this case. This provides a good debugging tool and also allows the CPU to submit EDMA requests in the system. Note that such CPU-initiated EDMA transfers are basically unsynchronized transfers. In other words, an EDMA transfer occurs when the relevant ER bit is set and is not triggered by any event as such.

Figure 6–3. Event Register (ER)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVT15	EVT14	EVT13	EVT12	EVT11	EVT10	EVT9	EVT8	EVT7	EVT6	EVT5	EVT4	EVT3	EVT2	EVT1	EVT0
R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0	R,+0

Figure 6–4. Event Enable Register (EER)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EE15	EE14	EE13	EE12	EE11	EE10	EE9	EE8	EE7	EE6	EE5	EE4	EE3	EE2	EE1	EE0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Figure 6–5. Event Clear Register (ECR)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EC15	EC14	EC13	EC12	EC11	EC10	EC9	EC8	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Figure 6–6. Event Set Register (ESR)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ES15	ES14	ES13	ES12	ES11	ES10	ES9	ES8	ES7	ES6	ES5	ES4	ES3	ES2	ES1	ES0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

6.4 Event Encoder

Up to 16 events can be captured by the EDMA controller's event register. Hence, it is quite possible that events occur simultaneously on the EDMA event inputs. For such cases, the order of processing is resolved by the event encoder. This mechanism only sorts simultaneous events and has nothing to do with the actual priority of the event. The actual priority of the event is determined by its EDMA parameters stored in the parameter RAM of the EDMA controller. Parameter RAM is discussed in the next section.

6.5 Parameter RAM (PaRAM)

Unlike the existing ‘C6201 DMA controller which is a register-based architecture, the enhanced DMA controller is a RAM-based architecture. The parameter RAM as the name indicates is used to store the parameters that define a particular EDMA transfer. The 2K byte parameter RAM holds transfer parameters (or entries) for all the 16 events. Parameter entries can also be linked to one another to provide for processing of complex streams, circular buffering, and sorting functions. The link entries are also specified in the parameter RAM.

Once an event is captured, its parameters are read from one of the top 16 entries in the PaRAM as shown in Table 6–2. These parameters are then sent to the address generation hardware.

The contents of the 2K byte parameter RAM shown in Table 6–2 comprises:

❑ 16 transfer parameter entries for the 16 EDMA events. Each entry is six words or 24 bytes totaling 384 bytes. Address range is 01A0 0000h to 01A0 017Fh.

❑ 69 transfer parameter sets that can be used for linking events. Each set or entry is 24 bytes totaling 1656 bytes. Address range is 01A0 0180h to 01A0 07F7h.

❑ 8 bytes of unused RAM that can be used as scratch pad area. Address range is 01A0 07F8h to 01A0 07FFh. Note that a part or entire EDMA RAM can be used as a scratch pad RAM provided this area corresponding to an event(s) is disabled. It is the user’s responsibility to provide the transfer parameters when the event is eventually enabled.

Table 6–2. EDMA Parameter RAM Contents

Address	Event Parameters	
01A0 0000h	Event 0, options	
01A0 0004h	Event 0, SRC address	
01A0 0008h	Event 0, array/frame count	Event 0, element count
01A0 000Ch	Event 0, DST address	
01A0 0010h	Event array/frame index 0,	Event 0, element index
01A0 0014h	Event 0, element count reload	Event 0, link address
01A0 0018h to 01A0 002Fh	Parameters for event 1 (6 words)	
01A0 0030h to 01A0 0047h	Parameters for event 2 (6 words)	
01A0 0048h to 01A0 005Fh	Parameters for event 3 (6 words)	
01A0 0060h to 01A0 0077h	Parameters for event 4 (6 words)	
01A0 0078h to 01A0 008Fh	Parameters for event 5 (6 words)	
01A0 0090h to 01A0 00A7h	Parameters for event 6 (6 words)	
01A0 00A8h to 01A0 00BFh	Parameters for event 7 (6 words)	
01A0 00C0h to 01A0 00D7h	Parameters for event 8 (6 words)	
01A0 00D8h to 01A0 00EFh	Parameters for event 9 (6 words)	
01A0 00F0h to 01A0 0107h	Parameters for event 10 (6 words)	
01A0 0108h to 01A0 011Fh	Parameters for event 11 (6 words)	

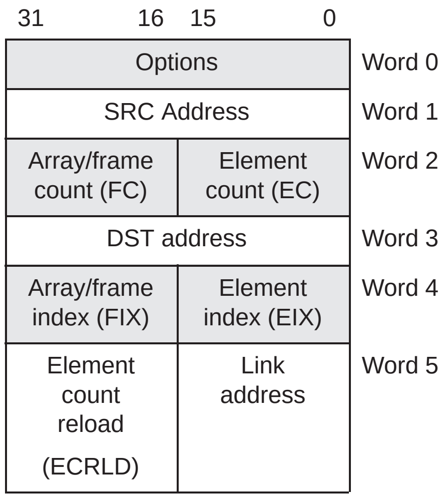
Table 6–2. EDMA Parameter RAM Contents (Continued)

Address	Event Parameters	
01A0 0120h to 01A0 0134h	Parameters for event 12 (6 words)	
01A0 0138h to 01A0 014Ch	Parameters for event 13 (6 words)	
01A0 0150h to 01A0 0164h	Parameters for event 14 (6 words)	
01A0 0168h to 01A0 017Ch	Parameters for event 15 (6 words)	
Address	**Reload/Link Parameters**	
01A0 0180h	Event N, options	
01A0 0184h	Event N, SRC address	
01A0 0188h	Event N, array/frame count	Event N, element count
01A0 018Ch	Event N, DST address	
01A0 0190h	Event N, array/frame index	Event N, element index
01A0 0194h	Event N, element count reload	Event N, link address
…	…	
…	…	
01A0 07E0h to 01A0 07F7h	Reload parameters for event Z (6 words)	
	Unused RAM	
01A0 07F8h to 01A0 07FFh	Scratch pad area (2 words)	

6.5.1 EDMA Transfer Parameter Entry

Each parameter entry of an EDMA event is organized in six 32-bit words or 192 bits as shown in Figure 6–7. Access to the EDMA parameter RAM is provided only via the 32-bit peripheral bus.

Figure 6–7. Parameter Storage for an EDMA Event



6.6 EDMA Transfer Parameters

Depending on the parameter options associated with a transfer, the source/destination address, element/array/frame count can be updated by the EDMA. The following sections describe the various parameters shown in Table 6–3.

6.6.1 Options Parameter

The options parameter in the EDMA channel/event entry is a 32-bit field as shown in Figure 6–8.

Figure 6–8. Options Bit-Fields

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	rsvd	LINK	FS
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	R,+0	RW,+0	RW,+0

Table 6–3. EDMA Channel Options Field Description

Field	Description	Section
FS	Frame synchronization FS=0; Frame sync is not needed to start a frame transfer. FS=1; Frame synchronization enabled. The relevant event for a given EDMA channel is used to synchronize a frame.	6.7
LINK	Linking events LINK=0; Linking of event parameters disabled LINK=1; Linking of event parameters enabled. Allows reloading of event parameters from the parameter RAM. Link address must be aligned on a 24-byte boundary.	6.6.7 and 6.9
TCC	Transfer complete code TCC=0000b to 1111b; 4-bit code is used to set the relevant bit in CIPR (i.e. CIPR[TCC] bit) provided TCINT=1.	6.13

Table 6–3. EDMA Channel Options Field Description (Continued)

Field	Description	Section
TCINT	Transfer complete interrupt TCINT=0; Transfer complete indication disabled. CIPR bits are not set upon completion of a transfer. TCINT=1; The relevant CIPR bit is set on channel transfer completion. The bit (position) set in the CIPR is the TCC value specified.	6.13
2DD/2DS	2-dimensional destination or source transfer 2DD/2DS = 0; Not a 2-D transfer. 2DD/2DS = 1; 2-Dimensional transfer enabled.	6.8 and 6.12
DUM/SUM	Destination / source (address) update mode DUM/SUM = 00b; No address modification DUM/SUM = 01b; Address increment depends on 2DD/2DS, and FS bit-fields DUM/SUM = 10b; Address decrement depends on 2DD/2DS, and FS bit-fields DUM/SUM = 11b; Address modified by the element index/frame index depending on 2DD/2DS, and FS bits.	6.12

ESIZE	Element size
	ESIZE=00b; 32-bit word
	ESIZE=01b; 16-bit half-word
	ESIZE=10b; 8-bit byte
	ESIZE=11b; reserved
		6.10

PRI	Priority levels for EDMA events
	PRI=000b; Reserved; Urgent priority level reserved ONLY for L2 requests. Not valid for EDMA transfer requests.
	PRI=001b; High priority EDMA transfer
	PRI=010b; Low priority EDMA transfer
	PRI=011b to 111b; reserved
		6.14

6.6.2 SRC/DST Address

The 32-bit source/destination address fields in the EDMA parameters specifies the starting byte address of the source and destination. The src/dst addresses can be modified using the SUM/DUM field in the options parameter. See details in section 6.12.

6.6.3 Element Count

Element count is a 16-bit unsigned value that specifies the number of elements in a frame (non-2D) or an array (for 2D transfers). Valid values for the element count can be anywhere between 1 and 65535. Therefore, the maximum number of elements in a frame is 65535. Operation is undefined if element count is zero. Details in section 6.11.

6.6.4 Frame/Array Count

Frame count is also a 16-bit unsigned value and it specifies the number of frames in a non-2D block transfer or number of arrays in a 2D block transfer. The maximum number of frames in a block is 65536. Therefore a frame/array count of 0 is actually one frame/array and frame count of 1 corresponds to 2 frames/arrays. Details in section 6.11.

6.6.5 Element/(Frame/Array) Index

The 16-bit signed value specified in the element and frame index fields are used for address modification. These fields are used by the EDMA for address updates depending on the type of transfer chosen (1D or 2D), FS, and SUM/DUM fields. The src/dst address is modified by an index whose range is between –32768 and 32767.

Element index provides an address offset to the next element in a frame. Element index is used *only* for non-2D transfers. This is because 2D transfers do not allow spacing between elements, and hence the term ‘array’ is used to define a group of contiguous elements. Frame index provides an offset to the next frame in a block.

6.6.6 Element Count Reload

The 16-bit unsigned element count reload value is used to reload the element count field once the last element in a frame is transferred. This field is used only for a non-2D read/write sync (FS=0) transfer since the EDMA has to keep track of the next element address using the element count. This is necessary for multi-frame EDMA transfers where frame count value is greater than 0. More details in section 6.11.1.

6.6.7 Link Address

The EDMA controller provides a mechanism to link EDMA transfers. This is analogous to the auto-initialization feature in the DMA. The 16-bit link address specified in the EDMA parameter RAM specifies the lower 16-bit address in the parameter RAM from which the EDMA loads/reloads the parameters of the next event in the chain. Since the entire EDMA parameter RAM is located in the 01A0 xxxxh area, only the lower 16-bit address matters.

The reload parameters are specified in the address range 01A0 0180h to 01A0 07F7h. It is the user's responsibility to ensure that the link address is on a 24-byte boundary. Operation is undefined if the rule is violated. This is discussed in section 6.9.

6.7 Initiating an EDMA Transfer

There are two ways to initiate data transfer using the EDMA. One is CPU-initiated EDMA and the other is an event-triggered EDMA. The latter is a more typical usage of the EDMA. Each EDMA channel can be started independently. The CPU can also disable an EDMA channel by disabling the event associated with that channel.

❑ **CPU-initiated EDMA or unsynchronized EDMA:** The CPU can write to the event set register, ESR (described in section 6.3) in order to start an EDMA transfer. Writing a ‘1’ to the corresponding event in the ESR triggers an EDMA event. Just as with a normal event, the transfer parameters in the EDMA parameter RAM corresponding to this event are passed to the address generation hardware, which performs the requested access of the EMIF, L2 memory or peripherals, as appropriate. CPU-initiated EDMA transfers are unsynchronized data transfers. The event’s enable bit does not have to be set in the EER for CPU-initiated EDMA transfers. This is because a CPU write to the ESR is treated as a real-time event.

❑ **Event-triggered EDMA:** As the name suggests, an event that is latched in the event register, ER, via the event encoder (see section 6.4) causes its transfer parameters to be passed on to the address generation hardware, which performs the requested accesses. Although the event causes this transfer, it is very important that the event itself be enabled by the CPU. Writing a ‘1’ to the corresponding bit in EER enables an event. Alternatively, an event is still latched in the ER even if its corresponding enable bit in EER is ‘0’ (disabled). The EDMA transfer related to this event occurs as soon as it is enabled in EER.

6.7.1 Synchronization of EDMA Transfers

Synchronization allows EDMA transfers to be triggered by events either from peripherals, interrupts from external devices, or an EDMA channel completion event. The 16 EDMA channels can start a transfer depending on the type of synchronization event associated with that channel. Table 6–4 shows the 16 events that initiate the 16 EDMA channels to start a transfer.

The association of an event to a channel is fixed. Unlike the existing ‘C6201-type DMA, each of the 16 EDMA channels have one specific type of event associated with it. For example, if bit 4 (event 4) in EER is set, then an external interrupt on EXT_INT4 pin initiates a transfer on EDMA channel 4.

Events originate from a peripheral such as the McBSP (R/XEVT), or an external device in the form of an external interrupt (say, EXT_INT4). The source of these events is listed in Table 6–4. The event is specific to a channel, the priority of each event can be specified independently in the transfer parameters stored in the EDMA parameter RAM.

Table 6–4. EDMA Channel Association with Sync Events

EDMA Channel Number	Event Acronym	Event Description
0	DSPINT	Host port host to DSP interrupt
1	TINT0	Timer 0 interrupt
2	TINT1	Timer 1 interrupt
3	SD_INT	EMIF SDRAM timer interrupt
4	EXT_INT4	External interrupt pin 4
5	EXT_INT5	External interrupt pin 5
6	EXT_INT6	External interrupt pin 6
7	EXT_INT7	External interrupt pin 7
8	EDMA_TCC8	EDMA transfer complete code 1000b interrupt
9	EDMA_TCC9	EDMA TCC 1001b interrupt
10	EDMA_TCC10	EDMA TCC 1010b interrupt
11	EDMA_TCC11	EDMA TCC 1011b interrupt
12	XEVT0	McBSP0 transmit event
13	REVT0	McBSP0 receive event
14	XEVT1	McBSP1 transmit event
15	REVT1	McBSP1 receive event

There are two types of synchronization that can be used to synchronize transfers on each channel. They are:

- **Read/write synchronization (R/WSYNC, FS=0):** For non-2D transfers, each EDMA channel performs a source to destination element transfer only after receiving a read/write sync event. A read/write sync event can originate from a peripheral or an external interrupt which is specific for a given EDMA channel. In the case of 2-D transfers, this EDMA channel R/WSYNC event is used to transfer an array from the source to the destination.

- **Frame (block) synchronization (FS=1):** Setting FS=1 in the options field of an EDMA channel's transfer parameters causes the channel's frame transfers to be synchronized as per the event shown in Table 6–4. For the case of non-2D transfers, each frame is transferred when the sync event is detected. For 2-D transfers, frame sync causes an entire block (group of arrays) to be transferred. Each frame transfer waits for the selected frame sync event to occur to start the transfer.

6.8 Types of EDMA Transfers

The EDMA provides for two types of data transfers, namely non-2-dimensional (non-2D) and 2-dimensional (2D) transfers. This is selected by setting the 2DD and 2DS bits in the event's options field. 2DD when set to 1 represents two-dimensional transfer on the destination. Similarly, a 2-D transfer on the source is performed when 2DS is equal to1. Various combinations of 2DS and 2DD are supported.

6.8.1 Non-2Dimensional Transfers

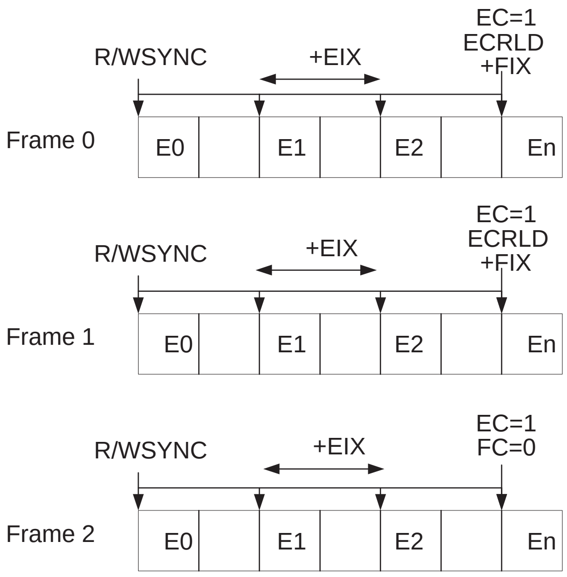
For non-2D transfers, a group of elements equal to element count constitute a frame. Each element transfer in a frame can be driven by the R/WSYNC event (FS=0). In addition, the elements can be contiguous or spaced by an element index amount. Once a complete frame is transferred, the element count reaches zero. Therefore for multi-frame transfers, the element count has to be reloaded by the element count reload field in the transfer entry. Frame count is the number of frames in a non-2D transfer. The start of a frame transfer can be triggered by a frame sync (FS=1) wherein the channel-specific event is used to synchronize the entire frame.

6.8.1.1 R/WSYNC Non-2D Transfer (FS=0)

Figure 6–9 shows the concept of a non-2D EDMA transfer with 'n' elements in each frame and frame count is 2, for a total of three frames. Each element in a frame is transferred from its source to destination address upon receiving the channel-specific sync event. After the channel receives a sync event, it sends off a transfer request for DMA service. The EDMA controller then decrements the element count (EC) by 1 in the parameter RAM. When a channel sync event occurs and EC = 1 (indicating the last element in a frame), the EDMA controller first sends off the transfer request triggered by the event. Afterward, element count reload occurs and frame count (FC) decrements by 1. User-specified element index (EIX) is used to compute the address of the next element in a frame. Similarly, frame index (FIX) is added to the last element address in a frame to derive the next frame start address. The address modification and count modification depends on the type of update modes chosen. They are mentioned here only for an understanding of a non-2D transfer. Specific updates are described in sections 6.11 and 6.12.

If linking is enabled (LINK=1, see section 6.9), the complete transfer parameters get reloaded (from the parameter reload space in EDMA parameter RAM) after sending the last transfer request to the address generation hardware. This sets up a new set of parameters in advance for the next occurrence of the event.

Figure 6–9. Non-2D R/W Sync EDMA Transfer Without Frame Sync

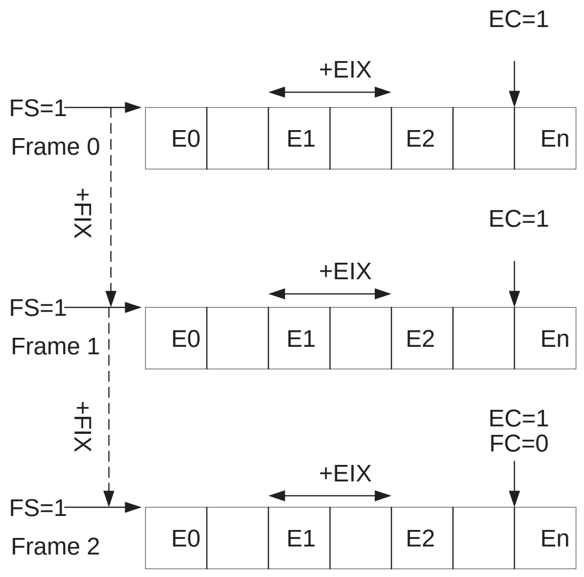


6.8.1.2 Frame Synchronized Non-2D Transfer (FS=1)

Figure 6–10 shows the concept of a non-2D EDMA transfer with frame synchronization. Here, the element transfer in each frame is not synchronized, but instead each frame transfer is synchronized by the channel event. FS bit (in options field) should be set to ‘1’ to enable frame-synchronized transfer. User-specified element index (EIX) can be used to stagger elements in a frame. Frame index (FIX) can be added to the start element address in a frame to derive the next frame start address. The address modification and count modification depends on the type of update modes chosen. They are mentioned here only for an understanding of a non-2D transfer. Specific updates are described in sections 6.11 and 6.12.

If linking is enabled (LINK=1, see section 6.9), the complete transfer parameters get reloaded (from the parameter reload space in EDMA parameter RAM) after sending the last transfer request to the address generation hardware.

Figure 6–10. Non-2D EDMA Transfer With Frame Sync



6.8.2 2-Dimensional Transfers

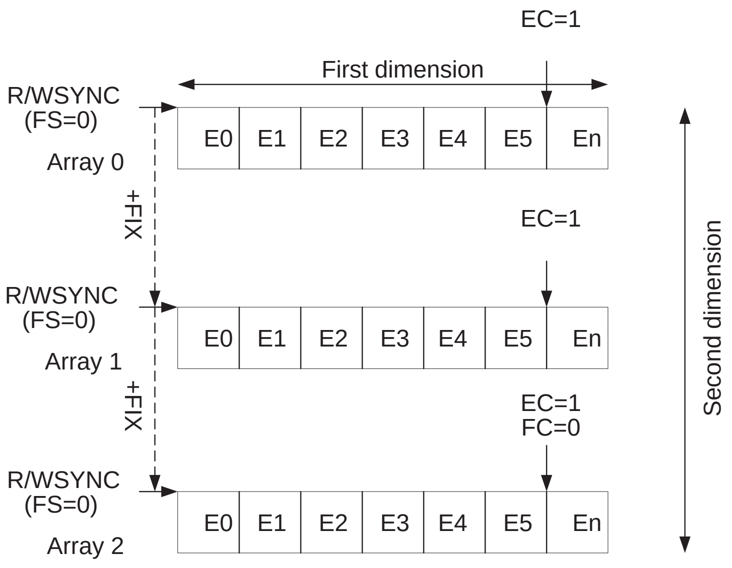
2-dimensional transfers are useful for imaging applications where contiguous set of elements (referred to as array) has to be transferred on receiving a sync event. This means there is no spacing or indexing between elements in an array and hence, EIX is not used in 2D transfers. The number of elements in an array makes up for the first dimension of the transfer. A group of arrays forms the second dimension and is called a block.

6.8.2.1 R/WSYNC 2D Transfer (FS = 0)

A conceptual diagram in Figure 6–11 shows a 2-dimensional, read/write synchronized transfer without frame synchronization. Since this 2-D transfer is not frame synchronized, the R/WSYNC is the sync event on which every array or contiguous group of elements is transferred. The example shows ‘n’ elements in an array and number of arrays to be transferred as 3 (frame count = 2). Frame count (FC) decrements after each array is transferred. Frame index is added to an array’s start address to derive the next array’s start address depending on the address update mode chosen (SUM/DUM).

When FC reaches zero and if linking is enabled (LINK = 1, see section 6.9), the complete transfer parameters get reloaded (from the parameter reload space in EDMA parameter RAM) after sending the last transfer request to the address generation hardware.

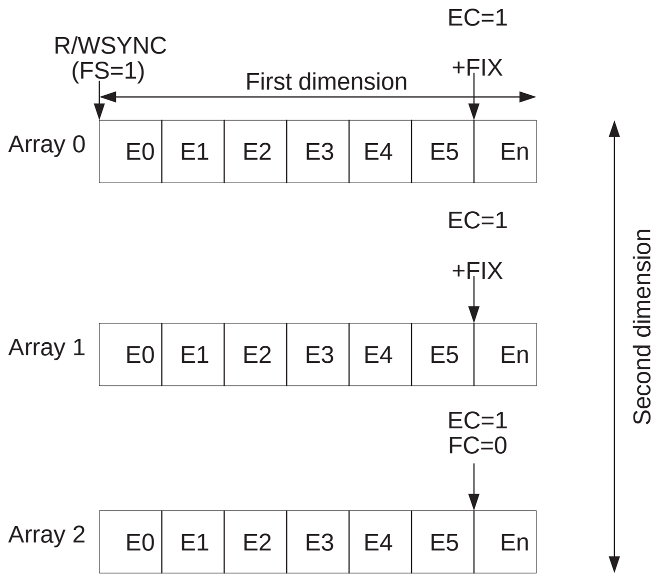
Figure 6–11. Read/Write Synchronized 2-D Transfer (No Frame Sync)



6.8.2.2 Frame Synchronized 2D Transfer (FS=1)

An example 2-dimensional block transfer with frame sync is shown in Figure 6–12. Again, the contiguous group of elements (element index, EIX=0) form an array and the group of arrays form a 2D-block (frame).

Figure 6–12. Frame Synchronized 2-D Transfer



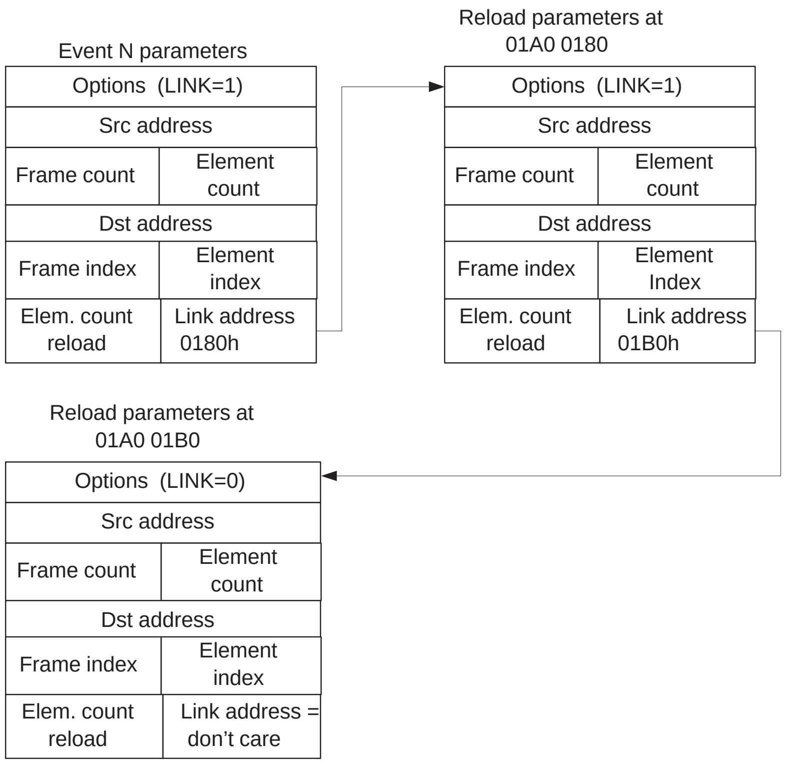
The complete block gets transferred when the channel's event occurs and FS=1. Note that the frame index (FIX) is added to the last element address in an array to derive the next array start address. This address update is transparent to the user and does not reflect in the parameter RAM.

If linking is enabled (LINK=1), the next EDMA block transfer in the link (as specified by the link address) is performed as soon as the next frame sync arrives.

6.9 Linking EDMA Transfers

The EDMA controller provides a mechanism known as ‘linking’, which allows multiple EDMA transfers to be linked. The completion of one transfer links the next transfer in a link causing its event parameters to be loaded from a location within the parameter RAM. This feature is especially useful for complex sorting, circular buffering type of applications. The 16-bit link address field in the EDMA parameter RAM and the LINK bit in the options field is used for this purpose. The link address points to the next transfer entry location in the linked list. The entire EDMA parameter RAM is located in the 01A0 xxxxh area. Therefore the 16-bit link address, which corresponds to the lower 16-bit physical address, is sufficient to specify the location of the next transfer entry. The link address must be aligned on a 24-byte boundary. An example of a linked EDMA transfer is shown in Figure 6–13.

Figure 6–13. Linked EDMA Transfer



The link address is evaluated only if LINK is equal to 1 *and* only after the event parameters have been exhausted. An event's parameters are exhausted when the EDMA controller has completed the transfer associated with the request. Table 6–5 shows the conditions when the linking of parameters is performed. The link conditions for a 2-D transfer is different from a non-2D transfer and it also depends on the type of synchronization chosen. Since the EDMA parameter RAM is 2048 bytes, it allows up to 69 reload entries in addition to the 16 EDMA event parameter entries (see section 6.5). There is virtually no limit to the length of linked transfers. However, the last transfer parameter entry should have its LINK = 0 so that the linked transfer stops after the last transfer.

Table 6–5. Link Conditions

LINK = 1	**Non-2D Transfers**	**2D Transfers**
Read/write sync (FS = 0)	Frame count == 0 && Element count == 1	Frame count == 0
Frame sync (FS = 1)	Frame count == 0	Always

Once the link conditions are met for an event, the transfer parameters located at the link address are loaded into one of the 16 EDMA channel/event parameter space for the corresponding event. Now, the EDMA is ready to start the next transfer. To eliminate possible timing windows posed during this parameter reload mechanism, the EDMA controller does not evaluate the event register during this time. However, events are still captured in the ER, and will be processed after the parameter reload is complete.

6.10 Element Size and Alignment

The ESIZE field in the options of an event parameter entry allows the user to specify the element size that the EDMA should use for a transfer. The EDMA controller can transfer 32-bit words, 16-bit half-words, or 8-bit bytes in a transfer.

The addresses must be aligned on the element size boundary. Word and half-word accesses must be aligned on a word (multiple of 4) and half-word (multiple of 2) boundary respectively. Unaligned values can result in undefined operation.

6.11 Element and Frame/Array Count Updates

The EDMA parameter RAM has 16-bit unsigned values of element count (EC) and frame count (FC) each. Additionally, it also holds 16-bit signed values each for the element index (EIX) and frame index (FIX). The maximum number of elements in a frame or an array (for 2D transfers) is 65535. The maximum number of frames in a block is 65536.

The element count and frame count are updated in the corresponding event's transfer entry depending on the type of transfer (2D or non-2D) and the synchronization type as shown in Table 6–6.

Table 6–6. EDMA Element and Frame/Array Count Updates

Synchronization	Transfer Mode	Element Count Update	Frame/Array Count Update†
Read/write (FS=0)	Non-2D; (2DS&2DD=0)	–1 (reload if EC = 1) See section 6.11.1	–1 (if element count = 1)
Read/write (FS=0)	2D; (2DS|2DD=1)	None	–1
Frame (FS=1)	Non-2D; (2DS&2DD=0)	None	–1
Frame (FS=1)	2D; (2DS|2DD=1)	None	None

† No frame/array count update occurs if the frame/array count is zero (FC = 0).

6.11.1 Element Count Reload (ECRLD)

There is a special condition for reloading the element count for read/write synchronized (FS = 0), non-2D transfers. In this case the address is updated by element size or element/frame index depending on SUM/DUM fields. See the first row in Table 6–7. Therefore, the EDMA controller keeps track of the element count to update the address. When a read/write sync event occurs at the end of a frame (EC = 1), the EDMA controller sends off the transfer request, and reloads the EC from the element count reload field in the parameter RAM. This element count reload occurs when element count is one, and the frame count is non-zero. For all other types of transfers, the 16-bit element count reload field is not used because the address generation hardware tracks the address directly.

6.12 Src/Dst Address Updates

Depending on the SUM/DUM fields in the options word of EDMA transfer parameters, the source and/or destination addresses can be modified. The EDMA controller performs the necessary address computation. The various address update modes listed in Table 6–3 provide for a variety of data structures that can be created. The source and/or destination address is updated depending on whether frame sync is enabled or not, or 2D transfer is selected or not. All address updates should occur after the current transfer request is sent. Therefore, these updates are used to set the EMDA parameters for the next event.

The update of the source or destination address depends on the transfer type chosen for both the source and destination. For example, a transfer from non-2D source to a 2D destination requires that the source be updated on a frame basis (not on element basis) to provide 2D type data to the destination. Table 6–7 shows the amount by which the source address is modified for each of the combinations of FS, 2DD/2DS, and SUM parameters. Table 6–8 shows the destination address updates that are possible.

Note that when either the source or the destination is a 2D transfer *and* the transfer is frame synchronized, it means that the complete block of data is transferred on a frame sync event. Therefore, address updates are not applicable in this case. If LINK = 1 and the link conditions outlined in Table 6–5 are met, no address updates occur. Instead, the link parameters are copied directly to the event parameter.

Table 6–7. EDMA SRC Address Parameter Updates

		Source Update Mode (SUM)			
Frame Sync	**Transfer-Type (2DS:2DD)**	**00**	**01**	**10**	**11**
FS = 0	00	None	+ESIZE; Increment by element size	–ESIZE; Decrement by element size	+EIX or +FIX if EC=1; Add signed EIX to each element in a frame except the last. Add signed FIX to the last element in a frame when EC = 1.
	01	None	+(EC x ESIZE bytes); Add EC scaled by element size to the start address of previous frame	–(EC x ESIZE bytes); Subtract EC scaled by element size from the start address of previous frame	Reserved
	10	None	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in increasing order.	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in decreasing order.	Reserved
	11	None	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in increasing order.	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in decreasing order.	Reserved
FS = 1	00	None	+(EC x ESIZE bytes); Add EC scaled by element size to the start address of previous frame	–(EC x ESIZE bytes); Subtract EC scaled by element size from the start address of previous frame	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame spaced by EIX.
	01	None	None	None	None
	10	None	None	None	None
	11	None	None	None	None

Note: EC: Element count
EIX: 16-bit signed element index value
FC: Frame/array count
FIX: 16-bit signed frame index value

Table 6–8. EDMA DST Address Parameter Updates

		Destination Update Mode (DUM)			
Frame Sync	**Transfer Type (2DS:2DD)**	**00**	**01**	**10**	**11**
FS = 0	00	None	+ESIZE; Increment by element size.	–ESIZE; Decrement by element size.	+EIX or +FIX if EC = 1 Add signed EIX to each element in a frame except the last. Add signed FIX to the last element in a frame when EC = 1.
	01	None	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in increasing order.	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in decreasing order.	Reserved
	10	None	+(EC x ESIZE bytes); Add EC scaled by element size to the start address of previous frame	–(EC x ESIZE bytes); Subtract EC scaled by element size from the start address of previous frame	Reserved
	11	None	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in increasing order.	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame are in decreasing order.	Reserved
FS = 1	00	None	+(EC x ESIZE bytes); Add EC scaled by element size to the start address of previous frame	–(EC x ESIZE bytes); Subtract EC scaled by element size from the start address of previous frame	+FIX; Add signed FIX to the first element in a frame. Element addresses in a frame spaced by EIX.
	01	None	None	None	None
	10	None	None	None	None
	11	None	None	None	None

Note: EC: Element count
EIX: 16-bit signed element index value
FC: Frame/array count
FIX: 16-bit signed frame index value

6.13 EDMA Interrupt Generation

The EDMA controller is responsible for generating channel-complete interrupts to the CPU. Unlike the ’C6201 DMA controller which has individual interrupts for each DMA channel, the EDMA generates a single interrupt (EDMA_INT) to the CPU on behalf of all 16 channels. The various control registers and bit fields facilitate EDMA interrupt generation.

When TCINT bit in options entry is set to ‘1’ for a EDMA channel and a specific transfer complete code (TCC) is provided, the EDMA controller sets a bit in the channel interrupt pending register (CIPR) shown in Figure 6–14. The CIPR bit number that gets set is dictated by the TCC value programmed. Lastly, the important action is to generate the EDMA_INT to the CPU. To do this, the corresponding interrupt enable bit should be set in the channel interrupt enable register (CIER) shown in Figure 6–15.

Therefore for a channel completion event to generate an interrupt to the CPU, the TCINT and the relevant CIER bit should be enabled. CIPR is equivalent to an interrupt pending register whose sources are the transfer complete codes and CIER is similar to an interrupt enable register. Note that if the CIER bit is disabled, the channel completion event is still registered in the CIPR if its TCINT=1. Once the CIER bit is enabled, the corresponding channel interrupt is sent to the CPU. If the CPU interrupt (defaults to CPU_INT8) is enabled, its ISR is executed.

Figure 6–14. Channel Interrupt Pending Register (CIPR)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CIP15	CIP14	CIP13	CIP12	CIP11	CIP10	CIP9	CIP8	CIP7	CIP6	CIP5	CIP4	CIP3	CIP2	CIP1	CIP0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Figure 6–15. Channel Interrupt Enable Register (CIER)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CIE15	CIE14	CIE13	CIE12	CIE11	CIE10	CIE9	CIE8	CIE7	CIE6	CIE5	CIE4	CIE3	CIE2	CIE1	CIE0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

The TCC field can have values between 0000b to 1111b. These are directly mapped to the CIPR bits as shown in Table 6–9. For example, if TCC = 1100b, CIPR[12] is set to 1 after the transfer is complete, and this generates a CPU interrupt only if CIER[12] = 1. The user can program the TCC value to be anything between 0000b to 1111b for any EDMA channel. In other words, there need not necessarily be a direct relation between the channel number and the TCC value. This allows multiple channels having the same TCC value to cause the CPU to execute the same ISR (for different channels).

Table 6–9. Transfer Complete Code (TCC) to DMA Interrupt Mapping

TCC in Options (TCINT=1)	**CIPR[15:0] Bits Set**
0000b	CIPR[0]
0001b	CIPR[1]
0010b	CIPR[2]
0011b	CIPR[3]
0100b	CIPR[4]
0101b	CIPR[5]
0110b	CIPR[6]
0111b	CIPR[7]
1000b	CIPR[8]
1001b	CIPR[9]
1010b	CIPR[10]
1011b	CIPR[11]
1100b	CIPR[12]
1101b	CIPR[13]
1110b	CIPR[14]
1111b	CIPR[15]

6.13.1 EDMA Interrupt Servicing by the CPU

Since the EDMA controller is aware when the EDMA channel transfer is complete, it sets the appropriate bit in the CIPR as per the TCC specified by the user. The CPU ISR should read the CIPR and determine what, if any events/ channels have completed and perform the operations necessary. The ISR should clear the bit in CIPR upon servicing the interrupt, therefore enabling recognition of further interrupts. Writing a ‘1’ to the relevant bit can clear CIPR bits, writing a ‘0’ has no effect.

By the time one interrupt is serviced, many others could have occurred and relevant bits set in CIPR. Each of these bits in CIPR would probably need different types of service, and therefore the ISR continues until all the posted interrupts are serviced.

6.13.2 Chaining EDMA Channels by an Event

Four of the user-specified 4-bit transfer complete codes (TCC values 8, 9, 10, and 11) can be used to trigger another EDMA channel transfer. The purpose of these events triggering an EDMA transfer is to provide the user the ability to chain several EDMA channels from one event that is driven by a peripheral or external device (see Table 6–4).

To enable the EDMA controller to chain channels by way of a single event, the TCINT bit must be set to ‘1’. Additionally, the relevant bit in the channel chain enable register (CCER) in Figure 6–16 should be set to trigger off the next channel transfer specified by TCC. Since events 8 to 11 are the only EDMA channels that support chaining, only these bits are implemented in CCER. Reading unused bits returns a ‘0’ and writing to them has no effect. Therefore, one can still specify a TCC value between 8 and 11, and need not necessarily initiate the transfer on channels 8-11. However, the event is still captured in the ER[11:8] even if the corresponding bit in CCER is disabled. This allows selective enabling and disabling of these 4 specific events.

Figure 6–16. Channel Chain Enable Register (CCER)

31 12	11	10	9	8	7 0
rsvd	CCE11	CCE10	CCE9	CCE8	rsvd
R, +0	RW, +0	RW, +0	RW, +0	RW, +0	R, +0

For example, if TCC = 1000b and CCER[8] = 1 is specified for EDMA channel 4, an external interrupt on EXT_INT4 initiates the EDMA transfer. Once channel 4 transfer is complete, the EDMA controller initiates (TCINT = 1) the next transfer specified by EDMA channel 8. This is because TCC = 1000b (channel 4 transfer completion code) is the sync event for EDMA channel 8. The corresponding CIPR bit 8 is set after channel 4 completes and generates an EDMA_INT (provided CIER[8] = 1) to the CPU. If the CPU interrupt is not desired, the corresponding interrupt enable bit, CIER[8] must be set to ‘0’. If channel 8 transfer is not desired, CCER[8] must be set to ‘0’.

6.14 Resource Arbitration and Priority Processing

The 16 EDMA channels can have programmable priority in the two lower levels. The PRI bit in options specifies the two priority levels: level1 (high priority, PRI = 001b) and level 2 (low priority, PRI = 010b). The highest priority available in the system is level 0 or the urgent priority, which is dedicated to L2 requests. L2 requests comprise of data and program requests from the CPU, L1 and L2 controllers. The EDMA controller and the host port interface (HPI) can submit requests with either of the two lower priority levels.

Table 6–10. Programmable Priority Levels for Data Requests

PRI(31:29)	Priority Level	Requesters
000b	Level0; urgent priority	L2 controller
001b	Level1; high priority	EDMA and/or HPI
010b	Level2; low priority	EDMA and/or HPI
011b – 111b	Reserved	Reserved

The user should take care in not over-burdening the system by not submitting all requests in high priority. Oversubscribing requests in one priority level can cause EDMA stalls. This can be alleviated by balanced bandwidth distribution in the two levels of priority.

The requesters in the ‘C6211/C6711 device include the L2 controller, the EDMA, and the HPI. The HPI and L2 controller have direct ties to the address generation hardware, so no EDMA parameter RAM is required for access requests from these sources. The resources for the various requesters include the L2 SRAM space, the various peripheral registers, and the external memory space managed by the EMIF. Due to the number of requesters and resources, there are situations where one or more requesters contend for the same resource. An example would be the EDMA and CPU requesting data from the same bank in L2 SRAM. The L2 controller resolves this contention by examining the user-specified ‘P’ bit in L2CFG register (see Section 4.5 in TMS320C6211/C6711 Internal Memory). The L2 controller prioritizes and serializes the requests from these modules. If P is equal to 1, the EDMA request gets priority over the CPU.

The priority queue status register (PQSR) shown in Figure 6–17 indicates if the transfer request queue is empty on the three priority levels (0 – urgent, 1 – high, and 2 – low). EDMA transfers can be submitted only with priority level one or two. The urgent priority level ’0’ is reserved for L2 requests. Status bits PQ[2:0] in the PQSR provide the status of the three queues. The three LSBs in this register, PQ[2:0], if set to ’1’ indicate that there are no requests pending in the respective priority level. If PQSR[0] is ’1’, this means all L2 requests for data movement have been completed and there are no requests pending.

Figure 6–17. Priority Queue Status Register(PQSR)

31 3	2	1	0
rsvd	PQ2	PQ1	PQ0
R, +0	R, +1	R, +1	R, +1

The three priority queue bits are mainly used for emulation, context switching for multitasking applications, and submitting requests with a higher priority – when possible. For the emulation case, the PQ0 bit is used to ensure that all cache requests via L2 are completed before updating any memory windows for the emulation halt. Another use is to determine the right time to do a task switch. For example, allocating L2 SRAM to a new task after ensuring that there are no EDMA transfer requests in progress which might write to L2 SRAM. Lastly, the PQ bits in PQSR can be used to allocate or submit requests judiciously on the lower two priority levels (by the EDMA or HPI) depending on which priority queue is empty. Therefore a low-priority request can be upgraded to a high priority if required. This helps prevent all requests from being queued under the same priority level which could lead to EDMA stalls.

6.15 EDMA Performance

The EDMA can perform element transfers with single cycle throughput provided the source and destination are two different resources that provide a single-cycle throughput. The performance can be limited by:

- ❑ EDMA stalls: When there are multiple transfer requests on the same priority level
- ❑ EDMA accesses to L2 SRAM with lower priority than CPU

6.16 Quick DMA (QDMA)

QDMA, or quick DMA, provides one of the most efficient ways to move data around in the 'C6211 architecture. Quick DMA supports nearly all of the same transfer modes of the EDMA. However, as the name implies, QDMA submits transfer requests more quickly than the EDMA. In a typical system, the user will use the EDMA for periodic real-time peripheral servicing, such as providing the McBSP with transmit data at a regular rate. For some applications, however, data must be moved in blocks under direct control of the code running on the CPU. For these applications, the QDMA is ideally suited.

6.16.1 QDMA Registers

The QDMA is supported through two sets of memory-mapped registers. The first set of five memory-mapped registers contains parameters that define a QDMA transfer, similar to the EDMA transfer parameters. The second set of five memory-mapped registers is a pseudo-mapping of the registers in the first set. The pseudo-mapping registers optimize the QDMA performance. Figure 6–18 shows the five memory mapped registers, and Figure 6–19 shows the five pseudo-mapping registers.

Figure 6–18. QDMA Memory-Mapped Registers

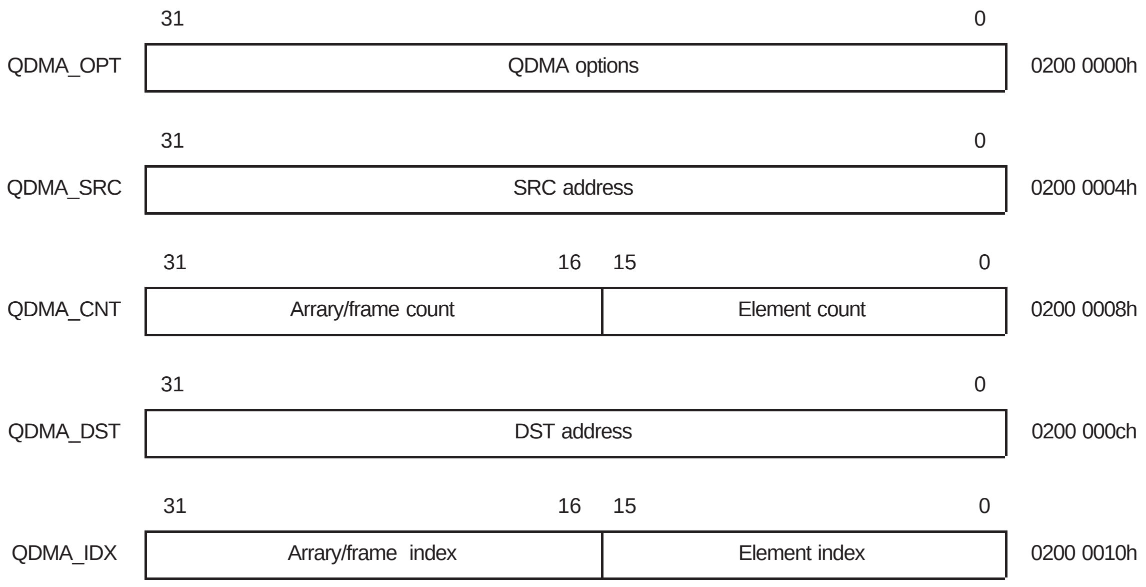


Figure 6–19. QDMA Pseudo Registers

	31	16	15	0	
QDMA_S_OPT	QDMA options				0200 0020h
QDMA_S_SRC	SRC address				0200 0024h
QDMA_S_CNT	Arrary/frame count		Element count		0200 0028h
QDMA_S_DST	DST address				0200 002Ch
QDMA_S_IDX	Arrary/frame index		Element index		0200 0030h

The QDMA options register shown in Figure 6–20 is similar to the options parameter in the EDMA parameter RAM, which is shown in Figure 6–8 and described in Table 6–3. All fields in the registers function identically to the corresponding EDMA transfer parameters. Refer to section 6.6 and the corresponding sections for details. However, the QDMA does not support parameter updates (e.g. element count reload), or transfer event linking. Since the QDMA does not support transfer event linking, bit 1 of the QDMA_OPT register is reserved, as opposed to the LINK bit-field in the EDMA options parameter.

Figure 6–20. QDMA Options Register (QDMA_OPT, QDMA_S_OPT)

31 29	28 27	26	25 24	23	22 21	20	19 16	15 1	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	FS
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	R,+0	RW,+0

Although the QDMA mechanism does not support event linking, it supports completion interrupts, as well as QDMA chaining with EDMA events. The QDMA_OPT and the QDMA_S_OPT registers include the same TCINT and TCC fields as the EDMA options parameter. QDMA completion interrupts are enabled and set in the same way as EDMA completion interrupts. To set a QDMA completion interrupt, the user needs to set TCINT to ’1’ and program the transfer completion code field (TCC) to specify the interrupt desired. The

TCC field can have values between 0000b to 1111b, just as the TCC field in the EDMA options parameter. Please refer to section 6.13 for detail. Similar to the EDMA, the QDMA completion event is captured in the EDMA channel interrupt pending register (CIPR). If the corresponding interrupt bit (specified by the TCC field) in the EDMA channel interrupt enable register (CIER) is set, the QDMA completion event will generate an interrupt to the CPU via the EDMA interrupt signal, EDMA_INT. If you specify a TCC value in the QDMA_OPT register to be between 8 and 11, chaining of the QDMA transfer to an EDMA channel transfer is possible, provided the relevant bit in the channel chain enable register (CCER) is set. Refer to section 6.16.3 for detail.

6.16.2 QDMA Register Access

Each of the QDMA registers is considered write only. Reads of the QDMA registers will return invalid data. Access to each of the above registers is limited to 32-bits only. Halfword and byte writes to the QDMA registers will write the entire register, and thus should be avoided.

6.16.3 Pseudo Mappings

The five physical QDMA registers are shadowed by five pseudo-mappings of the same registers. The pseudo-mappings serve as the mechanism for actually submitting the transfer request for DMA service. Writes to the physical QDMA registers (i.e. addresses 0200 0000h – 0200 0010h) are performed as normal store operations. A write to any one of the pseudo-registers will perform a write to the corresponding physical register, and also submit a transfer request for DMA service using the values stored in the physical registers. Thus, a typical submission sequence might look like the following:

```
QDMA_SRC = SOME_SRC_ADDRESS;
QDMA_DST = SOME_DST_ADDRESS;
QDMA_CNT = LINE_CNT<<16 | NUM_ELEMENTS & 0xFFFF;// Array Frame Count
QDMA_IDX = 0x00000000;        // no indexing specified
QDMA_S_OPT = 0x21B80001;      // frame synchronized 1D–SRC to 2D-DST,send
                              // completion code 8 when finished
                              // and submit transfer
```

6.16.4 QDMA Performance

The QDMA mechanism is extremely efficient at submitting DMA requests. Stores to the QDMA registers are passed to L2 cache as regular writes rather than peripheral writes. Because the QDMA registers are decoded to a special address region (0200 xxxxh), a fast decode is performed and writes may proceed to the QDMA registers on every cycle. Consequently, it is physically possible to submit the first QDMA request in as little as five cycles (one cycle write for each of the five QDMA registers), as opposed to 36 cycles for the first EDMA transfer request (6-cycle store for each of the six EDMA transfer parameters). Therefore, the QDMA registers can be used within the context of tight loop algorithms if desired.

Furthermore, the QDMA registers retain their value even after submitting the DMA transfer request. Hence, all five of the registers need not be programmed for each DMA submitted, provided that other application code has not modified these registers since the last DMA transfer request. As a result, subsequent QDMA requests can be processed in as little as one cycle per request-where the user only modifies the ONE corresponding pseudo-mapping register.

6.16.5 QDMA Stalls and Priority

The QDMA has several stalling conditions. Once a write has been performed to one of the pseudo-registers (resulting in a pending QDMA transfer request), future writes to the QDMA registers are stalled until the transfer request is sent. Normally this will occur for 2 cycles, as this is how long it takes to submit a transfer. The L2 controller includes a four-entry write buffer, so that stalls are not generally seen by the CPU.

Because the QDMA and the L2 cache controller share the same transfer request node, cache activity requiring the use of this transfer request node may delay submission of the QDMA transfer request. The L2 controller is given priority during this sort of arbitration, as in general it is assumed the cache requests have a greater likelihood of eventually stalling the CPU. The L2 write buffer typically keeps the CPU from being affected by this stall condition.

Similar to the EDMA channels, QDMA can have programmable priority in the two lower levels as described in section 6.14. The PRI bit-field in the QDMA_OPT register specifies the priority level of the QDMA. Once again, level 0 (urgent priority) is reserved for L2 cache accesses. QDMA request priority needs to be set to either level 1 (PRI = 001b) or level 2 (PRI = 010b). QDMA requests with any other level not equal to 1 or 2 will be discarded.

In the case when an EDMA request and a QDMA request happen simultaneously, the QDMA request will get submitted first. However, this only applies to the order of request submission. The PRI field determines the actual priority of the request. An EDMA request with level 1 priority has higher priority than a QDMA request with level 2 priority, even if the two events happen simultaneously and the QDMA request gets submitted first. Therefore, it is very important that the user programs the PRI field to specify the priority of an EDMA/ QDMA request, rather than relying on the order of the requests.

Chapter 7

Host-Port Interface

This chapter describes the host-port interface that external processors use to access the memory space. The host port control registers and signals are described.

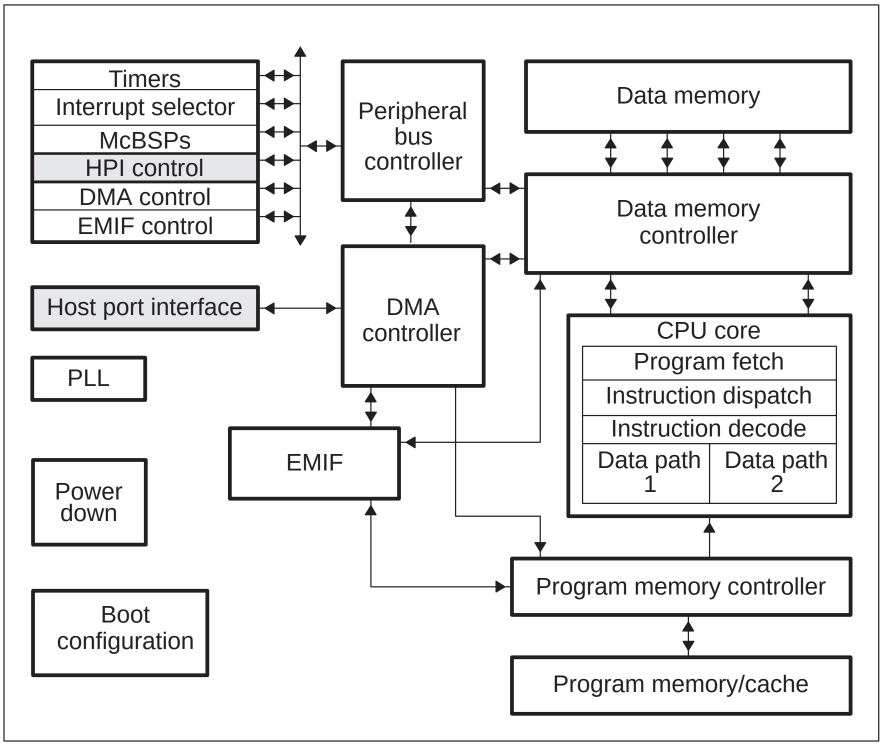
Topic **Page**

7.1 Overview

The host-port interface (HPI) is a 16-bit-wide parallel port through which a host processor can directly access the CPU's memory space. The host device functions as a master to the interface, which increases ease of access. The host and CPU can exchange information via internal or external memory. The host also has direct access to memory-mapped peripherals. Connectivity to the CPU's memory space is provided through the DMA controller. Dedicated address and data registers not accessible to the CPU connect the HPI to the DMA auxiliary channel, which connects the HPI to the CPU's memory space. Both the host and the CPU can access the HPI control register (HPIC). The host can access the HPI address register (HPIA), the HPI data register (HPID), and the HPIC by using the external data and interface control signals.

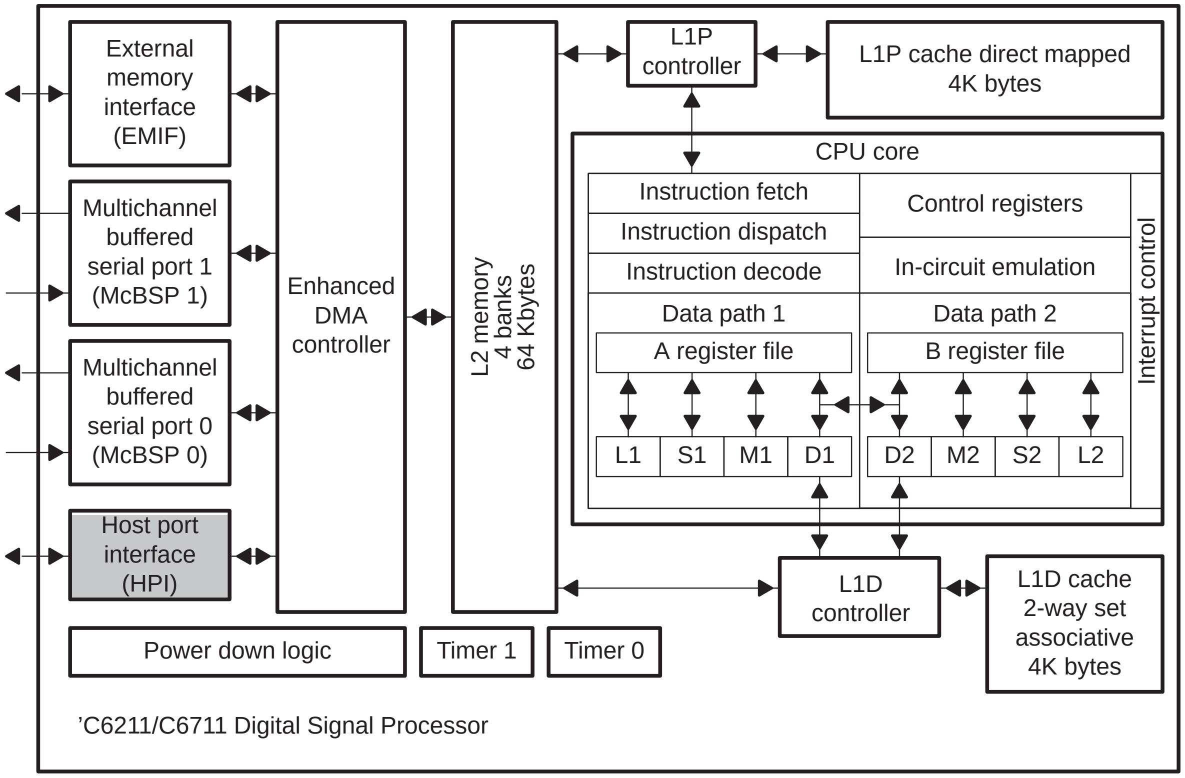
Figure 7–1 shows the host-port components in the block diagram of the on-chip peripherals.

Figure 7–1. TMS320C6201/C6701 Block Diagram



As with the ‘C6201/’C6701 HPI, the ’C6211/C6711 HPI allows an external host processor to perform read and write accesses from/to the ‘C6211/C6711 address space. Unlike the ‘C6201 HPI interface which uses the DMA auxiliary channel to perform accesses, the ’C6211/C6711 the HPI ties directly into internal address generation hardware. No specific EDMA channel is used for performing ’C6211/C6711 HPI accesses. Instead, the internal address generation hardware handles the read/write requests and accesses.

Figure 7–2. TMS320C6211/C6711 Block Diagram



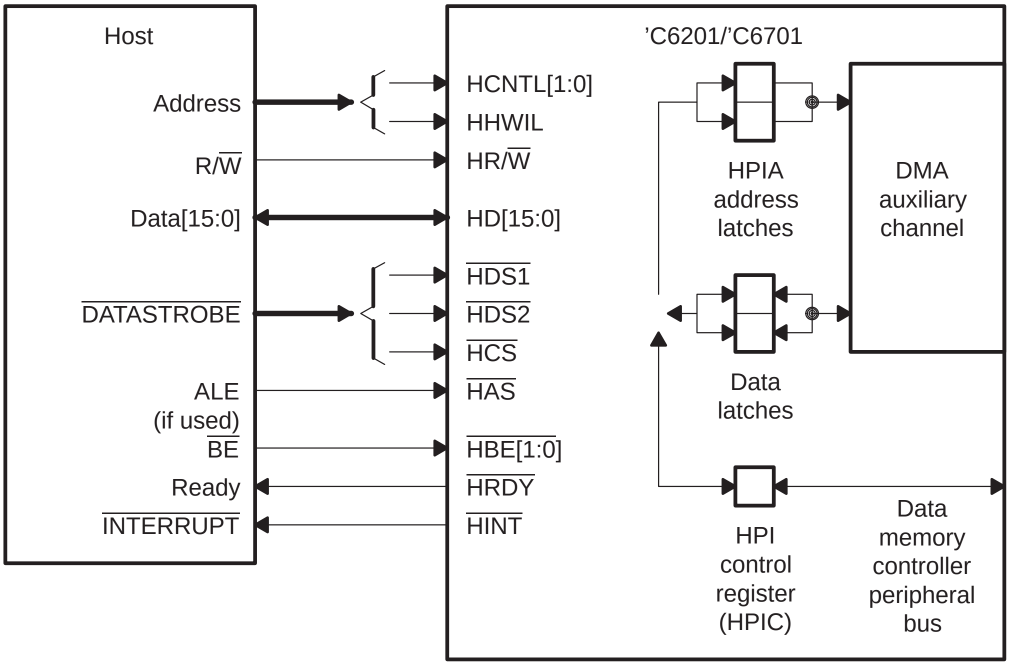
External memory interface (EMIF)
Multichannel buffered serial port 1 (McBSP 1)
Multichannel buffered serial port 0 (McBSP 0)
Host port interface (HPI)
Enhanced DMA controller
L2 memory 4 banks 64 Kbytes
L1P controller
L1P cache direct mapped 4K bytes
CPU core
Instruction fetch
Instruction dispatch
Instruction decode
Control registers
In-circuit emulation
Interrupt control
Data path 1
A register file
Data path 2
B register file
L1
S1
M1
D1
D2
M2
S2
L2
L1D controller
L1D cache 2-way set associative 4K bytes
Power down logic
Timer 1
Timer 0
’C6211/C6711 Digital Signal Processor

Figure 7–3 is a simplified diagram of the ’C6201/’C6701 HPI.

The HPI provides 32-bit data to the CPU with an economical 16-bit external interface by automatically combining successive 16-bit transfers. When the host device transfers data through HPID, the DMA auxiliary channel accesses the CPU’s address space.

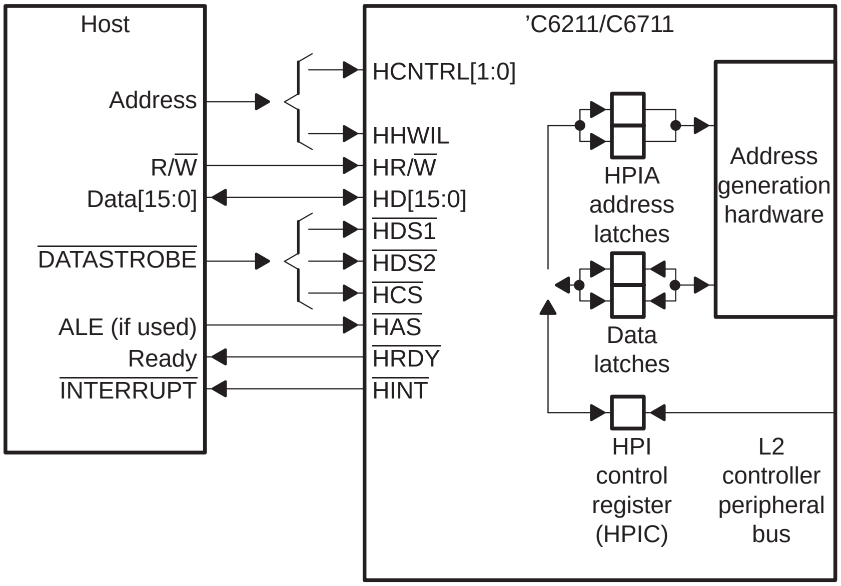
The 16-bit data bus, HD[15:0], exchanges information with the host. Because of the 32-bit-word structure of the chip architecture, all transfers with a host consist of two consecutive 16-bit halfwords. On HPI data (HPID) write accesses, the $\overline{\text{HBE[1:0]}}$ byte enables select the bytes to be written. For HPIA, HPIC, and HPID read accesses, the byte enables are not used. The dedicated HHWIL pin indicates whether the first or second halfword is being transferred. An internal control register bit determines whether the first or second halfword is placed into the most significant halfword of a word. For a full word access, the host must not break the first halfword/second halfword (HHWIL low/high) sequence of an ongoing HPI access.

Figure 7–3. HPI Block Diagram



The pin interface is similar to the ‘C6201 HPI interface as shown in Figure 7–4 except for the byte enables (/HBE[1:0] in ’C6201/’C6701) which are not supported. All accesses through the 16-bit data bus HD[15:0] have to be in pairs.

Figure 7–4. HPI Block Diagram of TMS320C6211/C6711



The two data strobes ($\overline{HDS1}$ and $\overline{HDS2}$), the read/write select (HR/$\overline{W}$), and the address strobe ($\overline{HAS}$) enable the HPI to interface to a variety of industry-standard host devices with little or no additional logic. The HPI can easily interface to hosts with a multiplexed or dedicated address/data bus, a data strobe and a read/write strobe, or two separate strobes for read and write.

The HCNTL[1:0] control inputs indicate which HPI register is accessed. Using these inputs, the host can specify an access to the HPIA (which serves as the pointer into the source or destination space), HPIC, or HPID. These inputs, along with HHWIL, are commonly driven directly by host address bus bits or a function of these bits. The host can interrupt the CPU by writing to the HPIC; the CPU can activate the $\overline{HINT}$ output to interrupt the host.

The host can access HPID with an optional automatic address increment of HPIA. This feature facilitates reading and writing to sequential word locations. In addition, during an HPID read with autoincrement, data is prefetched from the autoincremented address to reduce latency on the subsequent host read request.

The HPI ready pin ($\overline{\text{HRDY}}$) allows insertion of host wait states. Wait states may be necessary, depending on latency to the point in the memory map accessed via the HPI, as well as on the rate of host access. The rate of host access can force not-ready conditions if the host attempts to access the host port before any previous HPID write access or prefetched HPID read access finishes. In this case, the HPI simply holds off the host via $\overline{\text{HRDY}}$. $\overline{\text{HRDY}}$ provides a convenient way to automatically adjust the host access rate to the rate of data delivery from the DMA auxiliary channel (no software handshake is needed). In the cases of hardware systems that cannot take advantage of the $\overline{\text{HRDY}}$ pin, an HRDY bit in the HPIC is available for use as a software handshake.

7.2 HPI Signal Descriptions

The external HPI interface signals implement a flexible interface to a variety of host devices. Table 7–1 lists the HPI pins and their functions. The remainder of this section discusses the pins in detail.

Table 7–1. HPI External Interface Signals

Signal Name	Signal Type†	Signal Count	Host Connection	Signal Function
HD[15:0]	I/O/Z	16	Data bus	
HCNTL[1:0]	I	2	Address or control lines	HPI access type control
HHWIL	I	1	Address or control lines	Halfword identification input
$\overline{\text{HAS}}$	I	1	Address latch enable (ALE), address strobe, or unused (tied high)	Differentiation between address 2nd data values on multiplexed address/data host
$\overline{\text{HBE[1:0]}}$	I	2	Byte enables	Data write byte enables
HR/$\overline{\text{W}}$	I	1	Read/write strobe, address line, or multiplexed address/ data	Read/write select
$\overline{\text{HCS}}$	I	1	Address or control lines	Data strobe inputs
$\overline{\text{HDS[1:2]}}$	I	1 1	Read strobe and write strobe or data strobe	Data strobe inputs
$\overline{\text{HRDY}}$	O	1	Asynchronous ready	Ready status of current HPI access
$\overline{\text{HINT}}$	O	1	Host interrupt input	Interrupt signal to host

† I = input, O = output, Z = high impedance

7.2.1 Data Bus: HD[15:0]

HD[15:0] is a parallel, bidirectional, 3-state data bus. HD is placed in the high-impedance state when it is not performing an HPI read access.

7.2.2 Access Control Select: HCNTL[1:0]

HCNTL[1:0] indicate which internal HPI register is being accessed. The states of these two pins select access to the HPI address (HPIA), HPI data (HPID), or HPI control (HPIC) registers. Additionally, the HPID register can be accessed with an optional automatic address increment. Table 7–2 describes the HCNTL[1:0] bit functions.

Table 7–2. HPI Input Control Signals Function Selection Descriptions

HCNTL1	HCNTL0	Description
0	0	Host reads from or writes to the HPI control register (HPIC).
0	1	Host reads from or writes to the HPI address register (HPIA).
1	0	Host reads or writes to the HPI data register (HPID). The HPI address register (HPIA) is postincremented by a word address (four byte addresses).
1	1	Host reads or writes to the HPI data register (HPID). HPI address register (HPIA) is not affected.

7.2.3 Halfword Identification Select: HHWIL

HHWIL identifies the first or second halfword of a transfer, but not the most significant or least significant halfword. The status of the HWOB bit of the HPIC register, described later in this chapter, determines which halfword is least significant or most significant. HHWIL is low for the first halfword and high for the second halfword.

Since byte enable pins are removed from the ’C6211/C6711 HPI, HHWIL in combination with HWOB specify the half-word position in the data register, HPID. This is shown in Table 7–3 along with the LSB address bits depending on endianness.

Table 7–3. HPI Data Write Access

Data-Type Little-Endian (LE)/ Big-Endian (BE)	HWOB	First Write (HHWIL=0) / Logical LSB Address Bits	Second Write (HHWIL=1) / Logical LSB Address Bits

Half-word:
Little endian (LE)
Big endian (BE)
	0	MS half-word
		LE = 10
		BE = 00
			LS half-word
			LE = 00
			BE = 10

Half-word:
Little endian (LE)
Big endian (BE)
	1	LS half-word
		LE = 00
		BE = 10
			MS half-word
			LE = 10
			BE = 00

Word:
Little endian (LE)
Big endian (BE)
	0	MS half-word
		LE = 00
		BE = 00
			LS half-word
			LE = 00
			BE = 00

Word:
Little endian (LE)
Big endian (BE)
	1	LS half-word
		LE = 00
		BE = 00
			MS half-word
			LE = 00
			BE = 00

7.2.4 Byte Enables: $\overline{\text{HBE[1:0]}}$

On HPID writes, the value of $\overline{\text{HBE[1:0]}}$ indicates which bytes of the 32-bit word are written. The value of $\overline{\text{HBE[1:0]}}$ is not important on HPIA or HPIC accesses or on HPID reads. On HPID writes, $\overline{\text{HBE0}}$ enables the least significant byte in the halfword and $\overline{\text{HBE1}}$ enables the most significant byte in the halfword. Table 7–4 lists the valid combinations of byte enables. For byte writes, only one $\overline{\text{HBE}}$ in either of the halfword accesses can be enabled. For halfword data writes, both the $\overline{\text{HBE}}$s must be held active(low) in either (but not both) halfword access. For word accesses, both $\overline{\text{HBE}}$s must be held active (low) in both half-word accesses. No other combinations are valid. The selection of byte enables and the endianness of the CPU (selected via the LENDIAN pin) determine the logical address implied by the access.

Table 7–4. Byte Enables for HPI Data Write Access

		$\overline{\text{HBE}[1:0]}$		Effective Logical Address LSBs (Binary)	
	HWOB = 0	**First Write HHWIL = 0**	**Second Write HHWIL = 1**		
Data Write Type	**HWOB = 1**	**Second Write HHWIL = 1**	**First Write HHWIL = 0**	**Little Endian**	**Big Endian**
Byte		11	10	00	11
Byte		11	01	01	10
Byte		10	11	10	01
Byte		01	11	11	00
Halfword		11	00	00	10
Halfword		00	11	10	00
Word		00	00	00	00

7.2.5 Read/Write Select: HR/$\overline{\text{W}}$

HR/$\overline{\text{W}}$ is the host read/write select input. The host must drive HR/$\overline{\text{W}}$ high to read and low to write HPI. A host without either a read/write select output or a read or write strobe can use an address line for this function.

7.2.6 Ready: $\overline{\text{HRDY}}$

When active (low), $\overline{\text{HRDY}}$ indicates that the HPI is ready for a transfer to be performed. When inactive, $\overline{\text{HRDY}}$ indicates that the HPI is busy completing the internal portion of a current read access or a previous HPID read prefetch or write access. $\overline{\text{HCS}}$ enables $\overline{\text{HRDY}}$; $\overline{\text{HRDY}}$ is always low when $\overline{\text{HCS}}$ is high.

7.2.7 Strobes: $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, $\overline{\text{HDS2}}$

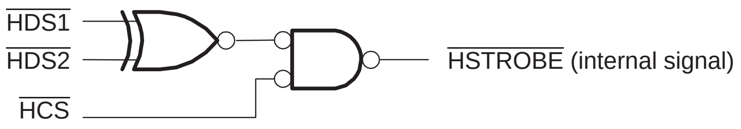
$\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, and $\overline{\text{HDS2}}$ allow connection to a host that has either:

❑ A single strobe output with read/write select

❑ Separate read and write strobe outputs. In this case, read or write select can be done by using different addresses.

Figure 7–5 shows the equivalent circuit of the $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, and $\overline{\text{HDS2}}$ inputs.

Figure 7–5. Select Input Logic



Used together, $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, and $\overline{\text{HDS2}}$ generate an active (low) internal $\overline{\text{HSTROBE}}$ signal. $\overline{\text{HSTROBE}}$ is active (low) only when both $\overline{\text{HCS}}$ is active and either (but not both) $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$ is active. The falling edge of $\overline{\text{HSTROBE}}$ when $\overline{\text{HAS}}$ is tied inactive (high) samples HCNTL[1:0], HHWIL, and HR/$\overline{\text{W}}$. Therefore, the latest of $\overline{\text{HDS1}}$, $\overline{\text{HDS2}}$, or $\overline{\text{HCS}}$ controls the sampling time. $\overline{\text{HCS}}$ serves as the enable input for the HPI and must be low during an access. However, because the $\overline{\text{HSTROBE}}$ signal determines the actual boundaries between accesses, $\overline{\text{HCS}}$ can stay low between successive accesses as long as both $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ transition appropriately.

Hosts with separate read and write strobes connect these strobes to either $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$. Hosts with a single data strobe connect it to either $\overline{\text{HDS1}}$ or a $\overline{\text{HDS2}}$, tying the unused pin high. Regardless of $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ connections, HR/$\overline{\text{W}}$ is required to determine the direction of transfer. Because $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ are internally exclusive-NORed, hosts with a high true data strobe can connect this strobe to either $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$ with the other signal tied low.

$\overline{\text{HSTROBE}}$ is used for four purposes:

- On a read, the falling edge of $\overline{\text{HSTROBE}}$ initiates HPI read accesses for all access types.
- On a write, the rising edge of $\overline{\text{HSTROBE}}$ initiates HPI write accesses for all access types.
- The falling edge latches the HPI control inputs, including HHWIL, HR/$\overline{\text{W}}$, and HCNTL[1:0]. $\overline{\text{HAS}}$ also affects latching of control inputs. See section 7.2.8 for a description of $\overline{\text{HAS}}$.
- The rising edge of $\overline{\text{HSTROBE}}$ latches the $\overline{\text{HBE[1:0]}}$ input as well as the data to be written.

$\overline{\text{HCS}}$ gates the $\overline{\text{HRDY}}$ output. In other words, a not-ready condition is indicated by the $\overline{\text{HRDY}}$ pin being driven high only if $\overline{\text{HCS}}$ is active (low). Otherwise $\overline{\text{HRDY}}$ is active (low).

7.2.8 Address Strobe Input: $\overline{\text{HAS}}$

$\overline{\text{HAS}}$ allows HCNTL[1:0], HR/$\overline{\text{W}}$, and HHWIL to be removed earlier in an access cycle, which allows more time to switch bus states from address to data information. This feature facilitates interface to multiplexed address and data buses. In this type of system, an address latch enable (ALE) signal is often provided and is normally the signal connected to $\overline{\text{HAS}}$.

Hosts with a multiplexed address and data bus connect $\overline{\text{HAS}}$ to their ALE pin or an equivalent pin. HHWIL, HCNTL[1:0], and HR/$\overline{\text{W}}$ are latched on the falling edge of $\overline{\text{HAS}}$. When used, $\overline{\text{HAS}}$ must precede the latest of $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, or $\overline{\text{HDS2}}$. Hosts with separate address and data buses can tie $\overline{\text{HAS}}$ high. In this case, HHWIL, HCNTL[1:0], and HR/$\overline{\text{W}}$ are latched by the latest falling edge of $\overline{\text{HDS1}}$, $\overline{\text{HDS2}}$, or $\overline{\text{HCS}}$ while $\overline{\text{HAS}}$ stays inactive (high).

7.2.9 Interrupt to Host: $\overline{\text{HINT}}$

$\overline{\text{HINT}}$ is the host interrupt output that is controlled by the HINT bit in the HPIC. This bit is set to 0 when the chip is being reset. This signal is described in more detail in section 7.3.4. Thus, the $\overline{\text{HINT}}$ pin is high at reset.

7.2.10 HPI Bus Access

Figure 7–6 and Figure 7–8 show HPI access timing for cases in which HAS is not used. Figure 7–7 and Figure 7–9 show HPI access timing for cases in which $\overline{\text{HAS}}$ is used. $\overline{\text{HSTROBE}}$ represents the internally generated strobe described in Figure 7–5. Control signals: HCNTL[1:0], HR/$\overline{\text{W}}$, HHWIL, and $\overline{\text{HBE[1:0]}}$ are typically driven by the host. HCNTL[1:0] and HR/$\overline{\text{W}}$ should have the same values for both halfword accesses. HHWIL is shown separately to indicate that it must be low for the first halfword transfer and high for the second. If $\overline{\text{HAS}}$ is not used (if it is tied high as shown in Figure 7–6), the falling edge of $\overline{\text{HSTROBE}}$ latches these signals. If $\overline{\text{HAS}}$ is used as shown in Figure 7–7 and Figure 7–9, the falling edge of $\overline{\text{HAS}}$ latches these values. In this case, the falling edge of $\overline{\text{HAS}}$ must precede the falling edge of $\overline{\text{HSTROBE}}$. On a read, data is valid at some time after the falling edge of $\overline{\text{HSTROBE}}$. If valid data is not already present in the HPID, the data is set up at the falling edge of $\overline{\text{HRDY}}$ and held until the rising edge of $\overline{\text{HSTROBE}}$. On a write, the host must set up data and $\overline{\text{HBE[1:0]}}$ on the rising edge of $\overline{\text{HSTROBE}}$. The HPI provides 32-bit data to the CPU through a 16-bit external interface. This is accomplished by automatically combining two successive halfword transfers.

When low, $\overline{\text{HRDY}}$ indicates that the HPI is ready for a transfer to be performed. $\overline{\text{HCS}}$ enables $\overline{\text{HRDY}}$, and $\overline{\text{HRDY}}$ is always low when $\overline{\text{HCS}}$ is high. Case 1 in Figure 7–6 and Figure 7–7, where $\overline{\text{HRDY}}$ goes high when $\overline{\text{HCS}}$ falls, indicates that the HPI is busy completing a previous HPID write or read with autoincrement.

When the host performs a read access from HPID without autoincrement, the HPI sends the read request to the DMA auxiliary channel, and $\overline{\text{HRDY}}$ becomes high. This event occurs with the first falling edge of $\overline{\text{HSTROBE}}$. $\overline{\text{HRDY}}$ remains high until the DMA auxiliary channel loads the requested data into HPID. At the beginning of the second read access, the data is already present in HPID (the DMA auxiliary channel performs word reads). Thus, the second halfword HPID read never encounters a not-ready condition, and HRDY remains low.

In the case of HPID read access with autoincrement, the data pointed to by the next address is fetched immediately after the completion of the current read. Therefore, after the second halfword transfer of the current read (with the second rising edge of $\overline{\text{HSTROBE}}$), $\overline{\text{HRDY}}$ becomes high again, indicating that HPI is busy pre–fetching data. During an HPID write access, two halfword portions of the HPID are transferred from the host. At the end of this write access, $\overline{\text{HRDY}}$ becomes high (with the second rising edge of $\overline{\text{HSTROBE}}$), and the contents of HPID are transferred as a 32-bit word to the address specified by HPIA. Reading or writing to HPIC or HPIA does not affect the $\overline{\text{HRDY}}$ signal.

Figure 7–6. HPI Read Timing ($\overline{\text{HAS}}$ Not Used, Tied High)

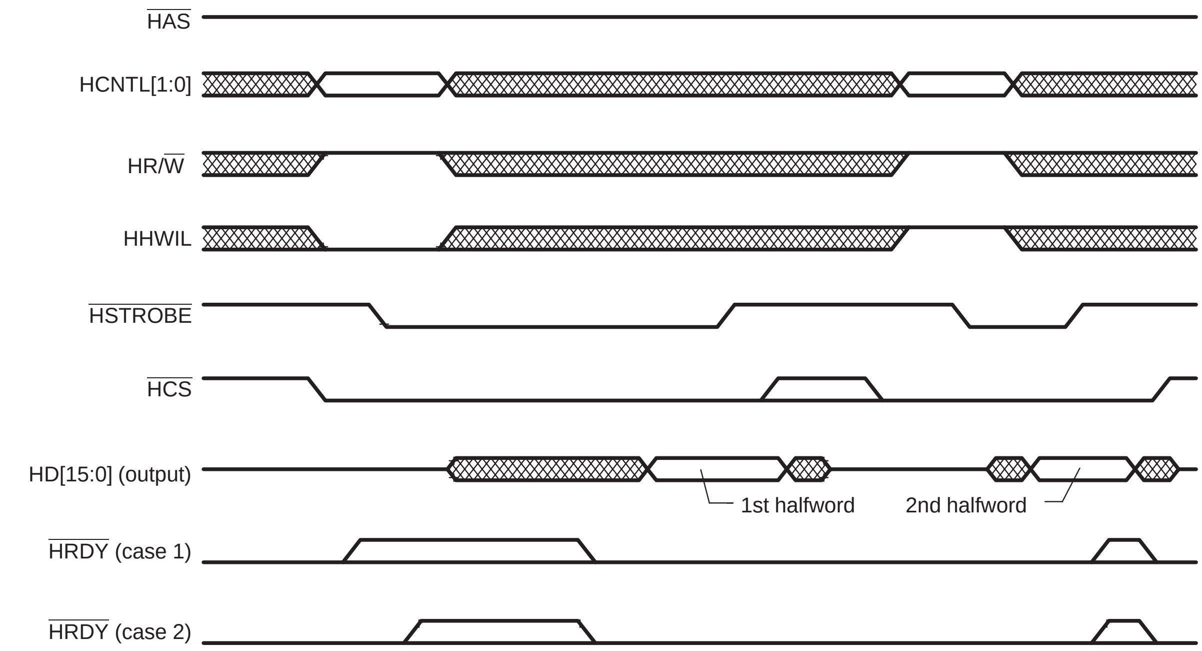


Figure 7–7. HPI Read Timing ($\overline{\text{HAS}}$ Used)

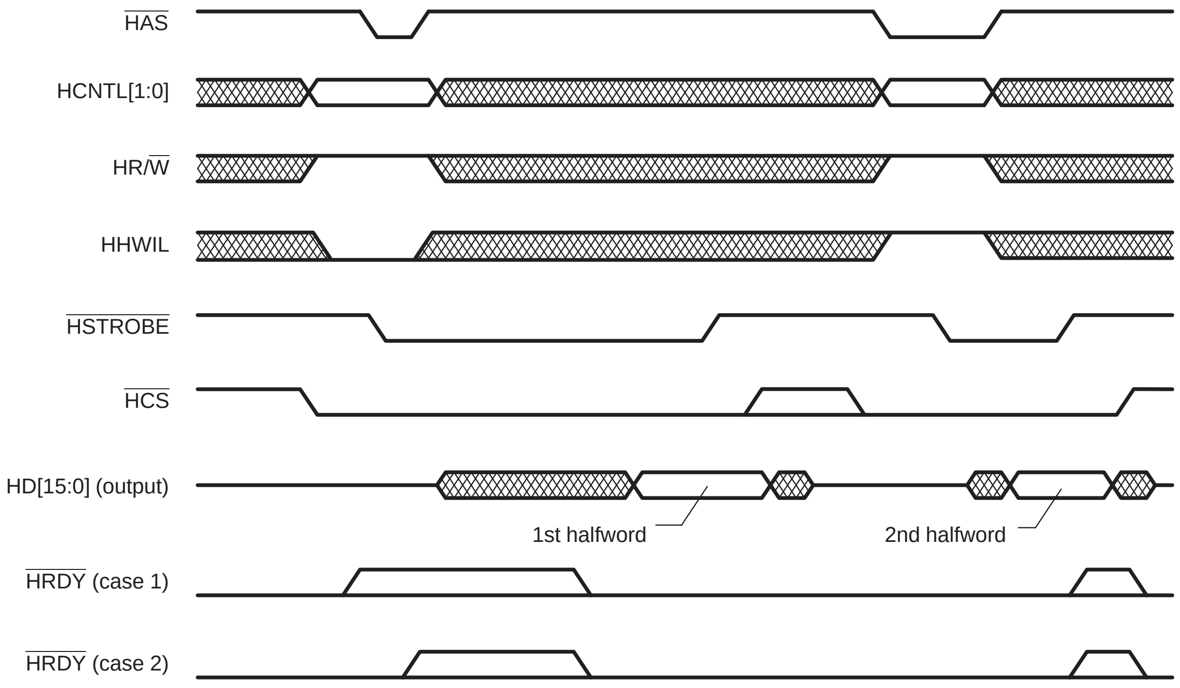


Figure 7–8. HPI Write Timing ($\overline{\text{HAS}}$ Not Used, Tied High)

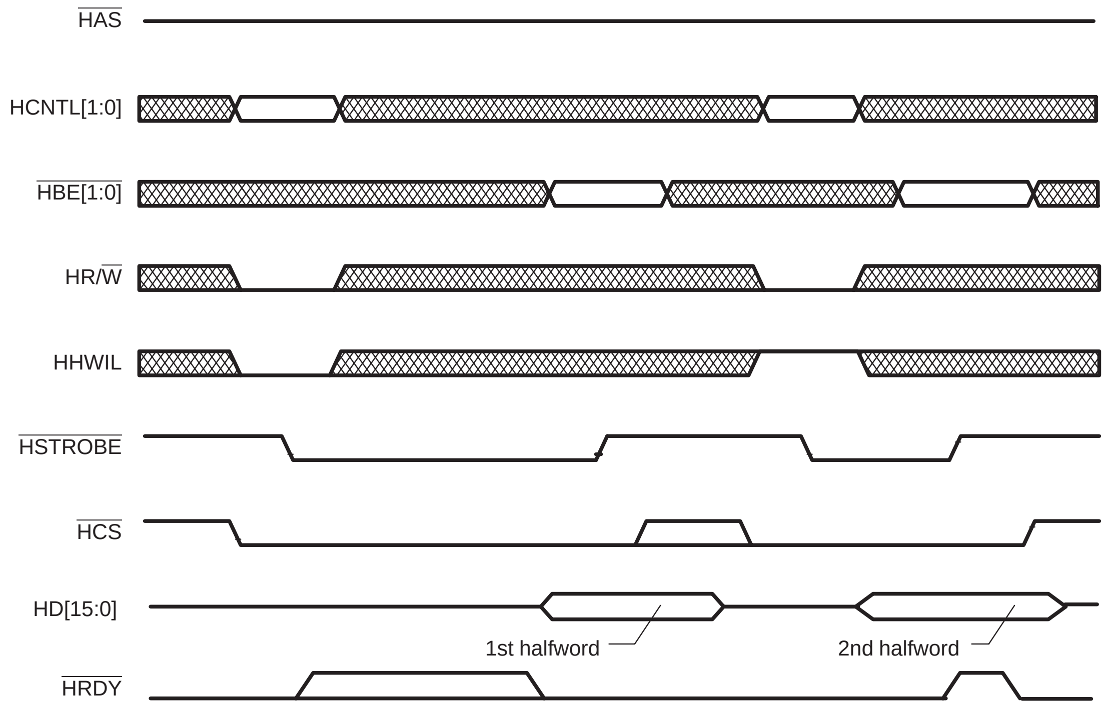
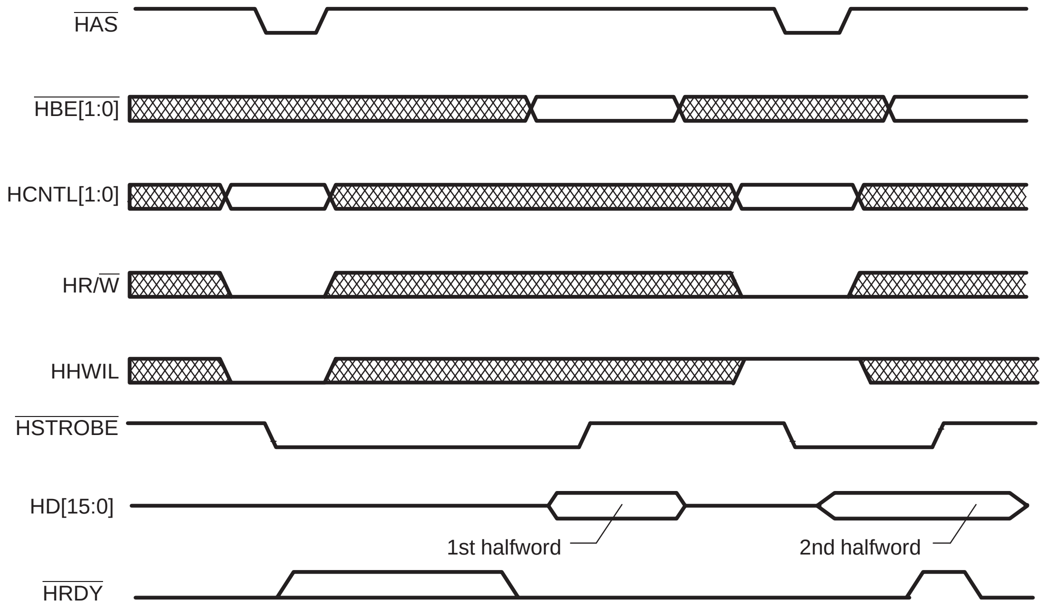


Figure 7–9. HPI Write Timing ($\overline{\text{HAS}}$ Used)



7.3 HPI Registers

Table 7–5 summarizes the three registers that the HPI uses for communication between the host device and the CPU. HPID contains the data that was read from the memory accessed by the HPI if the current access is a read or the data that is written to the memory if the current access is a write. HPIA contains the address of the memory accessed by the HPI at which the current access occurs. This address as a 30-bit-word address, so the two LSBs are unaffected by HPIA writes and are always read as 0.

Table 7–5. HPI Registers

Register Abbreviation	Register Name	Host Read/Write Access	CPU Read/Write Access	CPU Read/Write (Hex Byte Address)
HPID	HPI data	RW	–	–
HPIA	HPI address	RW	–	–
HPIC	HPI control	RW	RW	0188 0000h

7.3.1 HPI Control Register (HPIC)

The HPIC, shown in Figure 7–10 and summarized in Table 7–6, is normally the first register accessed to set configuration bits and initialize the interface. The HPIC is organized as a 32-bit register whose high halfword and low halfword contents are the same. On a host write, both halfwords must be identical. The low halfword and the high halfword are actually the same storage locations. No storage is allocated for the read-only reserved values. Only CPU writes to the lower halfword affect HPIC values and HPI operation.

Figure 7–10. HPIC Register

31–21	20	19	18	17	16
rsvd		HRDY	HINT	DSPINT	HWOB
HR,CR,+0	HRW,CR,+0	HR,CR,+0	HRW,CR,+0	HRW,CR,+0	HRW,CR,+0

15–5	4	3	2	1	0
rsvd	FETCH	HRDY	HINT	DSPINT	HWOB
HR,CR,+0	HRW,CR,+0	HR,CR,+1	HRW,CRW,+0	HRW,CRW,+0	HRW,CR,+0

Table 7–6. HPI Control Register (HPIC) Bit Descriptions

Bit	Description	Section
HWOB	Halfword ordering bit If HWOB = 1, the first halfword is least significant. If HWOB = 0, the first halfword is most significant. HWOB affects both data and address transfers. Only the host can modify this bit. HWOB must be initialized before the first data or address register access.	7.4
DSPINT	The host processor-to-CPU/DMA interrupt	7.3.3
HINT	DSP-to-host interrupt. The inverted value of this bit determines the state of the CPU $\overline{\text{HINT}}$ output.	7.3.4
HRDY	Ready signal to host. Not masked by $\overline{\text{HCS}}$ (as the $\overline{\text{HRDY}}$ pin is). If HRDY = 0, the internal bus is waiting for an HPI data access request to finish.	7.3.2
FETCH	Host fetch request The value read by the host or CPU from this register field is always 0. The host writes a 1 to this bit to request a fetch into HPID of the word at the address pointed to by HPIA. The 1 is never actually written to this bit, however.	7.3.2

7.3.2 Software Handshaking Using $\overline{\text{HRDY}}$ and FETCH

As described previously, the $\overline{\text{HRDY}}$ pin can indicate to a host that an HPID access has not finished. For example, the current HPID access can be waiting for a previous HPID access write to finish or for a previous HPID prefetched read to finish. Also, the current HPID read access can be waiting for its requested data to arrive. The HRDY and FETCH bits in the HPIC register allow for a software handshake that allows an HPI connection in systems in which a hardware ready control is not desired. The FETCH and HRDY bits can be used to perform a read transfer as follows:

1) The host polls the HPIC register for HRDY = 1.
2) The host writes the desired HPIA value. This step is skipped if HPIA is already set to the desired value.
3) The host writes a 1 to the FETCH bit.
4) The host polls again for HRDY = 1.
5) The host performs an HPID read operation. In this case, the HPI is already in the ready state (HRDY = 1).
6) If this was a read with postincrement, go to step 4. For a read from the same location, go to step 3.For a read to a different address, go to step 2.

The HRDY bit can be used alone for write operations as follows:

1) The host polls for HRDY = 1.
2) The host writes the desired HPIA value. (This step is skipped if HPIA is already set to the desired value.)
3) The host performs an HPID write operation. For another write operation, go to step 1.

7.3.3 Host Device Using DSPINT to Interrupt the CPU

The host can interrupt the CPU by writing to the DSPINT bits in the HPIC. The DSPINT bit is tied directly to the internal DSPINT signal. By writing DSPINT = 1 when DSPINT = 0, the host causes a low-to-high transition on the DSPINT signal. If you program the selection of the DSPINT interrupt with interrupt selector, the transition of DSPINT is detected as an interrupt condition by the CPU. The CPU can clear the DSPINT bits by writing a 1 to DSPINT. Neither a host nor a CPU HPIC write with DSPINT = 0 affects the DSPINT bit or signal.

7.3.4 CPU Using $\overline{\text{HINT}}$ to Interrupt the Host

The CPU can send an active interrupt condition on the $\overline{\text{HINT}}$ signal by writing to the HINT bit in the HPIC. The HINT bit is inverted and tied directly to the $\overline{\text{HINT}}$ pin. The CPU can set $\overline{\text{HINT}}$ active by writing HINT = 1. The host can clear the $\overline{\text{HINT}}$ to inactive by writing a 1 to HINT. Neither a host nor a CPU write to HPIC with HINT = 0 affects either the HINT bit or the $\overline{\text{HINT}}$ signal.

The HINT bit is read twice on the host interface side. The first and second halfword reads by the host can yield different data if the CPU changes the state of this bit between the two read operations.

7.4 Host Access Sequences

The host begins HPI accesses by performing the following tasks in the order giving:

1) Initializing the HPIC register
2) Initializing the HPIA register
3) Writing data to or reading data from HPID register

Reading from or writing to HPID initiates an internal cycle that transfers the desired data between the HPID register and the DMA auxiliary channel. Host access of any HPI register requires two halfword accesses on the HPI bus: the first with HHWIL low and the second with HHWIL high. Typically, the host does not break the first halfword/second halfword (HHWIL low/high) sequence. If this sequence is broken, data can be lost, and undesired operation can result. The first halfword access may have to wait for a previous HPI request to finish. Previous requests include HPID writes and prefetched HPID reads. Thus, the HPI deasserts $\overline{\text{HRDY}}$ (drives $\overline{\text{HRDY}}$ high) until the HPI can begin this request. The second halfword access always has $\overline{\text{HRDY}}$ active because all previous accesses have been completed for the first halfword access.

7.4.1 Host Initialization of HPIC and HPIA

Before accessing data, the host must first initialize the HWOB bit of the HPIC register and then HPIA (in this order, because HWOB affects the HPIA access). After initializing HWOB, the host can write to HPIA with the correct halfword alignment. Table 7–7 and Table 7–8 summarize the initialization sequence for HWOB = 1 and HWOB = 0, respectively. In these examples, HPIA is set to 80001234h. In all these accesses, the HRDY bits in the HPIC register are set. A question mark in these tables indicates that the value is unknown.

Table 7–7. Initialization of HWOB = 1 and HPIA

	Value During Access					Value After Access		
Event	**HD**	**$\overline{\text{HBE}}$[1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**HHWIL**	**HPIC**	**HPIA**	**HPID**
Host writes HPIC 1st halfword	0001	xx	0	00	0	00090009	????????	????????
Host writes HPIC 2nd halfword	0001	xx	0	00	1	00090009	????????	????????
Host writes HPIA 1st halfword	1234	xx	0	01	0	00090009	????1234	????????
Host writes HPIA 2nd halfword	8000	xx	0	01	1	00090009	80001234	????????

Note: A ? in this table indicates the value is unknown.

Table 7–8. Initialization of HWOB = 0 and HPIA

	Value During Access					**Value After Access**		
Event	**HD**	**$\overline{\text{HBE}}$[1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**HHWIL**	**HPIC**	**HPIA**	**HPID**
Host writes HPIC 1st halfword	0000	xx	0	00	0	00080008	????????	????????
Host writes HPIC 2nd halfword	0000	xx	0	00	1	00080008	????????	????????
Host writes HPIA 1st halfword	8000	xx	0	01	0	00080008	8000????	????????
Host writes HPIA 2nd halfword	1234	xx	0	01	1	00080008	80001234	????????

Note: A ? in this table indicates the value is unknown.

7.4.2 HPID Read Access Without Autoincrement

Assume that once the HPI is initialized, the host wishes to perform a read access to an address without an autoincrement. Assume that the host wants to read the word at address 80001234h and that the word value at that location is 789ABCDEh. Table 7–9 and Table 7–10 summarize this access for HWOB = 1 and HWOB = 0, respectively. On the first halfword access, the HPI waits for any previous requests to finish. During this time, $\overline{\text{HRDY}}$ pin is held high. Then, the HPI sends the read request to the DMA auxiliary channel. If no previous requests are pending, this read request occurs on the falling edge of $\overline{\text{HSTROBE}}$. $\overline{\text{HRDY}}$ pin remains high until the DMA auxiliary channel loads the requested data into HPID. Because all DMA auxiliary channel reads are word reads, at the beginning of the second read access, the data is already present in HPID. Thus, the second halfword HPID read never encounters a not-ready condition, and $\overline{\text{HRDY}}$ pin remains active. The byte enables are not important in this instance, because the HPI performs only word reads.

Table 7–9. Data Read Access to HPI Without Autoincrement: HWOB = 1

	Value During Access						Value After Access		
Event	**HD**	**$\overline{\text{HBE}}$[1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**
Host reads HPID 1st halfword Data not ready	????	xx	1	11	1	0	00010001	80001234	????????
Host reads HPID 1st halfword Data ready	BCDE	xx	1	11	0	0	00090009	80001234	789ABCDE
Host reads 2nd halfword	789A	xx	1	11	0	1	00090009	80001234	789ABCDE

Note: A ? in this table indicates the value is unknown.

Table 7–10. Data Read Access to HPI Without Autoincrement: HWOB = 0

	Value During Access						Value After Access		
Event	**HD**	**$\overline{\text{HBE}}$[1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**
Host reads HPID1st halfword Data not ready	????	xx	1	11	1	0	00000000	80001234	????????
Host reads HPID 1st halfword Data ready	789A	xx	1	11	0	0	00080008	80001234	789ABCDE
Host reads HPID 2nd halfword	BCDE	xx	1	11	0	1	00080008	80001234	789ABCDE

Note: A ? in this table indicates the value is unknown.

7.4.3 HPID Read Access With Autoincrement

The autoincrement feature results in efficient sequential host accesses. For both HPID read and write accesses, this removes the need for the host to load incremented addresses into HPIA. For read accesses, the data pointed to by the next address is fetched immediately after the completion of the current read. Because the intervals between successive reads are used to prefetch data, the latency for the next access is reduced. Prefetching also occurs after a host write of FETCH = 1 to the HPIC register. If the next HPI access is an HPID read, then the data is not refetched and the prefetched data is sent to the host. Otherwise, the HPI must wait for the prefetch to finish.

Table 7–11 summarizes a read access with autoincrement. After the first half-word access is complete (with the rising edge of the first $\overline{\text{HSTROBE}}$), the address increments to the next word, or 80001238h in this example. Assume that the data at that location is 87654321h. This data is prefetched and loaded into HPID. Prefetching begins on the rising edge of $\overline{\text{HSTROBE}}$ in the second half-word read.

Table 7–11. Read Access to HPI With Autoincrement: HWOB = 1

	Value During Access						Value After Access		
Event	**HD**	**$\overline{\text{HBE}}$[1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**
Host reads HPID 1st halfword Data not ready	????	xx	1	10	1	0	00010001	80001234	????????
Host reads HPID 1st halfword Data ready	BCDE	xx	1	10	0	0	00090009	80001234	789ABCDE
Host reads HPID 2nd halfword	789A	xx	1	10	0	1	00090009	80001234	789ABCDE
Prefetch Data not ready	????	xx	x	xx	1	x	00010001	80001238	789ABCDE
Prefetch Data ready	????	xx	x	xx	0	x	00090009	80001238	87654321

Note: A ? in this table indicates the value is unknown.

Table 7–12. Read Access to HPI With Autoincrement: HWOB = 0

	Value During Access						Value After Access		
Event	**HD**	**$\overline{\text{HBE}}$[1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**
Host reads 1st halfword Data not ready	????	xx	1	10	1	0	00000000	80001234	????????
Host reads 1st halfword Data ready	789A	xx	1	10	0	0	00080008	80001234	789ABCDE
Host reads 2nd halfword	BCDE	xx	1	10	0	1	00080008	80001234	789ABCDE
Prefetch Data not ready	????	xx	x	xx	1	x	00000000	80001238	789ABCDE
Prefetch Data ready	????	xx	x	xx	0	x	00080008	80001238	87654321

Note: A ? in this table indicates the value is unknown.

7.4.4 Host Data Write Access Without Autoincrement

During a write access to the HPI, the first halfword portion of HPID (the least significant halfword or most significant halfword, as selected by HWOB) is overwritten by the data coming from the host, and the first $\overline{\text{HBE}}$[1:0] pair is latched while the HHWIL pin is low. The second halfword portion of HPID is overwritten by the data coming from the host, and the second $\overline{\text{HBE}}$[1:0] pair is latched on the rising edge of $\overline{\text{HSTROBE}}$ while the HHWIL pin is high. At the end of this write access (with the second rising edge of $\overline{\text{HSTROBE}}$), HPID is transferred as a 32-bit word to the address specified by HPIA with the four related byte enables.

Table 7–13 and Table 7–14 summarize an HPID write access with HWOB = 1 and HWOB = 0, respectively. The host writes 5566h to the 16 LSBs of location 80001234h, which is already pointed to by HPIA. This location is assumed to start with the value 0. The HPI delays the host until any previous transfers are completed by setting $\overline{\text{HRDY}}$ high. If there are no pending writes waiting in HPID, then write accesses normally proceed without a not-ready time. The $\overline{\text{HBE}}$[1:0] pair is enabled only for the transfer of the 16 LSBs.

Table 7–13. Data Write Access to HPI Without Autoincrement: HWOB = 1†

	Value During Access						Value After Access			
Event	**HD**	**$\overline{\text{HBE}[1:0]}$**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**	**Location 80001234**
Host writes HPID 1st halfword Waiting for previous access to complete	5566	00	0	11	1	0	00010001	80001234	????????	00000000
Host writes HPID 1st halfword	5566	00	0	11	0	0	00090009	80001234	????5566	00000000
Host writes HPID 2nd halfword	wxyz	11	0	11	0	1	00090009	80001234	wxyz5566	00000000
Waiting for access to complete	????	??	?	??	1	?	00010001	80001234	wxyz5566	00005566

Note: A ? in this table indicates the value is unknown.

Table 7–14. Data Write Access to HPI Without Autoincrement: HWOB = 0†

	Value During Access						Value After Access			
Event	**HD**	**$\overline{\text{HBE}[1:0]}$**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**	**Location 80001234**
Host writes HPID 1st halfword Waiting for previous access to complete	wxyz	11	0	11	1	0	00000000	80001234	????????	00000000
Host writes HPID 1st halfword	wxyz	11	0	11	0	0	00080008	80001234	wxyz????	00000000
Host writes HPID 2nd halfword	5566	00	0	11	0	1	00080008	80001234	wxyz5566	00000000
Waiting for access to complete	????	??	?	??	1	?	00080008	80001234	wxyz5566	00005566

† wxyz represents a don’t care value on the HD pins.

Note: A ? in this table indicates the value is unknown.

7.4.5 HPID Write Access With Autoincrement

Table 7–15 and Table 7–16 summarize a host data write with autoincrement for HWOB = 1 and HWOB = 0, respectively. These examples are identical to the ones in section 7.4.4, except for the HCNTL[1:0] value and a subsequent write of 33h to the most significant byte of the word at address 8000 1238h. The increment occurs on the rising edge of $\overline{\text{HSTROBE}}$ on the next HPID write access. If the next access is an HPIA or HPIC access or an HPID read, the autoincrement does not occur.

Table 7–15. Write Access to HPI With Autoincrement: HWOB = 1[†]

	Value During Access						Value After Access				
Event	**HD**	**$\overline{\text{HBE}}$ [1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL [1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**	**Location 80001234**	**Location 80001238**
Host writes HPID 1st halfword Waiting for previous access to complete	5566	00	0	10	1	0	00010001	80001234	????????	00000000	00000000
Host writes HPID 1st halfword Ready	5566	00	0	10	0	0	00090009	80001234	????5566	00000000	00000000
Host writes HPID 2nd halfword	wxyz	11	0	10	0	1	00090009	80001234	wxyz5566	00000000	00000000
Host writes HPID 1st halfword Waiting for previous access to complete	nopq	11	0	10	1	0	00010001	80001234	wxyz5566	00005566	00000000
Host writes HPID 1st halfword	nopq	11	0	10	0	0	00090009	80001238	wxyznopq	00005566	00000000
Host writes HPID 2nd halfword	33rs	01	0	10	0	1	00090009	80001238	33rsnopq	00005566	00000000
Waiting for access to complete	????	??	?	??	1	?	00010001	80001238	33rsnopq	00005566	33000000

† wxyz, rs, and nopq represent don't care values on HPID.

Note: A ? in this table indicates the value is unknown.

Table 7–16. Write Access to HPI With Autoincrement: HWOB = 0[†]

	Value During Access						Value After Access				
Event	HD	$\overline{\text{HBE}}$ [1:0]	HR/$\overline{\text{W}}$	HCNTL [1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID	Location 80001234	Location 80001238
Host writes HPID 1st halfword Waiting for previous access to complete	wxyz	11	0	10	1	0	00000000	80001234	????????	00000000	00000000
Host writes HPID 1st halfword Ready	wxyz	11	0	10	0	0	00080008	80001234	wxyz???	00000000	00000000
Host writes HPID 2nd halfword	5566	00	0	10	0	1	00080008	80001234	wxyz5566	00000000	00000000
Host writes HPID 1st halfword Waiting for previous access to complete	33rs	01	0	10	1	0	00000000	80001234	wxyz5566	00005566	00000000
Host writes HPID 1st halfword	33rs	01	0	10	0	0	00080008	80001238	33rs5566	00005566	00000000
Host writes HPID 2nd halfword	nopq	11	0	10	0	1	00080008	80001238	33rsnopq	00005566	00000000
Waiting for access to complete	????	??	?	??	1	?	00000000	80001238	33rsnopq	00005566	33000000

[†] wxyz, rs, and nopq represent don't care values on HPID.

Note: A ? in this table indicates the value is unknown.

7.4.6 Single Halfword Cycles

In normal operation, every transfer must consist of two halfword accesses. However, to speed operation you can perform single halfword accesses. These can be useful in performing the following tasks:

❑ Writes to and reads from HPIC: In Table 7–7, the entire HPIC was written to correctly after the first write. When writing the HPIC, the host does not have to be concerned about HHWIL, nor does it have to perform two consecutive writes to both halfwords. Similarly, the host can choose to read the HPIC only once, because both halves contain the same value.

- Writes to and reads from HPIA: In Table 7–7, the portion of HPIA accesses selected by HHWIL and HWOB is updated automatically after each halfword access. Thus, to change either the upper or the lower 16 bits of HPIA, the host must select the half to modify through a combination of the HHWIL and HWOB bits. The host can also choose to read only half of HPIA.

- HPID read accesses: Read accesses are actually triggered by the first halfword access (HHWIL low). Thus, if on reads the host is interested only in the first halfword (the least or most significant halfword, as selected by HWOB), the host does not need to request the second address. However, prefetching does not occur unless the second halfword is also read. A subsequent read of the first halfword (HHWIL low) or a write of a new value to HPIA overrides any previous prefetch request. On the other hand, a read of just the second halfword (HHWIL high) is not allowed and results in undefined operation.

- Write accesses: Write accesses are triggered by the second halfword access (HHWIL word high). Thus, if the host desires to change only the portion of HPID selected by HHWIL high (and the associated byte enables) during consecutive write accesses, only a single cycle is needed. This technique's primary use is for memory fills: the host writes both halfwords of the first write access with $\overline{\text{HBE[1:0]}}$ = 00. On subsequent write accesses, the host writes the same value to the portion of HPID selected by HHWIL as the first write access did. In this case, the host performs autoincrementing writes (HCNTL[1:0] = 10) on all write accesses.

7.5 Memory Access Through the HPI During Reset

During reset, when $\overline{\text{HCS}}$ is active low, $\overline{\text{HRDY}}$ is inactive high, and when $\overline{\text{HCS}}$ is inactive, $\overline{\text{HRDY}}$ is active. The HPI cannot be used while the chip is in reset. However, certain boot modes can allow the host to write to the CPU's memory space (including configuring EMIF configuration registers to define external memory before accessing it). Although the device is not in reset during these boot modes, the CPU itself is in reset until the boot completes. See Chapter 10, *Boot Configuration, Reset, and Memory Maps*, for more details.

Chapter 8

Expansion Bus

This section describes the expansion bus used by CPU to access off-chip peripherals, FIFOs and PCI interface chips.

Topic **Page**

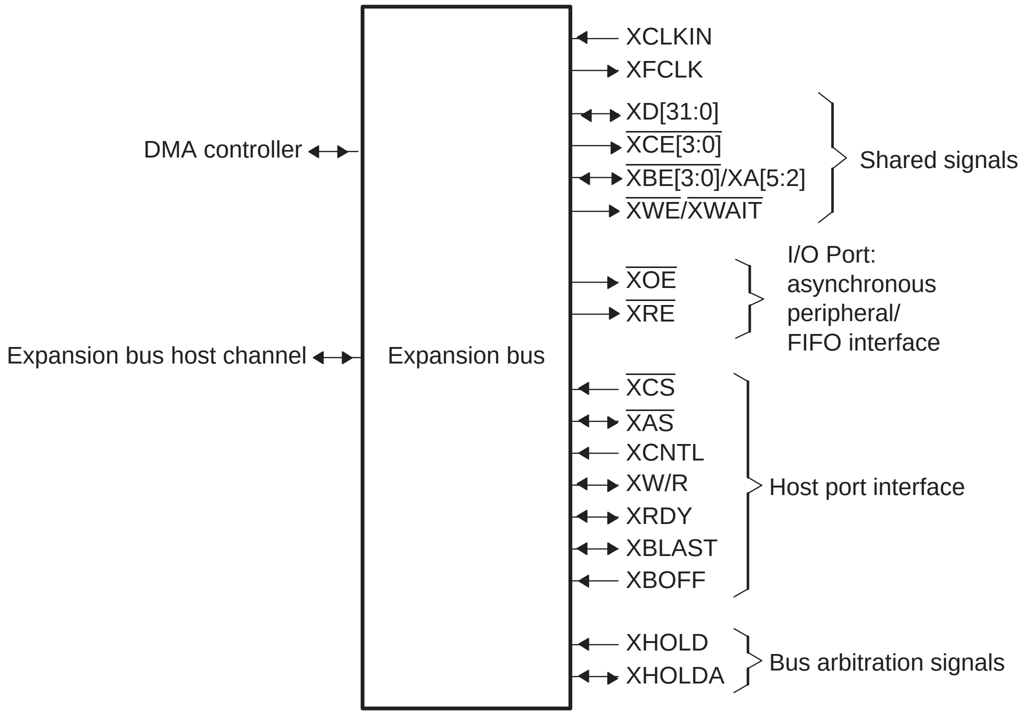
8.1 Overview

The expansion bus is a 32-bit wide bus that supports interfaces to a variety of asynchronous peripherals, asynchronous or synchronous FIFOs, PCI bridge chips, and other external masters.

The expansion bus offers a flexible bus arbitration scheme, implemented with two signals, XHOLD and XHOLDA. The expansion bus can operate with the Internal arbiter enabled, in which case any external hosts must request the bus from the DSP. For increased flexibility, the internal arbiter can be disabled, and the DSP requests the bus from an external arbiter.

The expansion bus has two major sub blocks—the I/O port and host port interface. A block diagram of the expansion bus is shown in Figure 8–1.

Figure 8–1. Expansion Bus Block Diagram

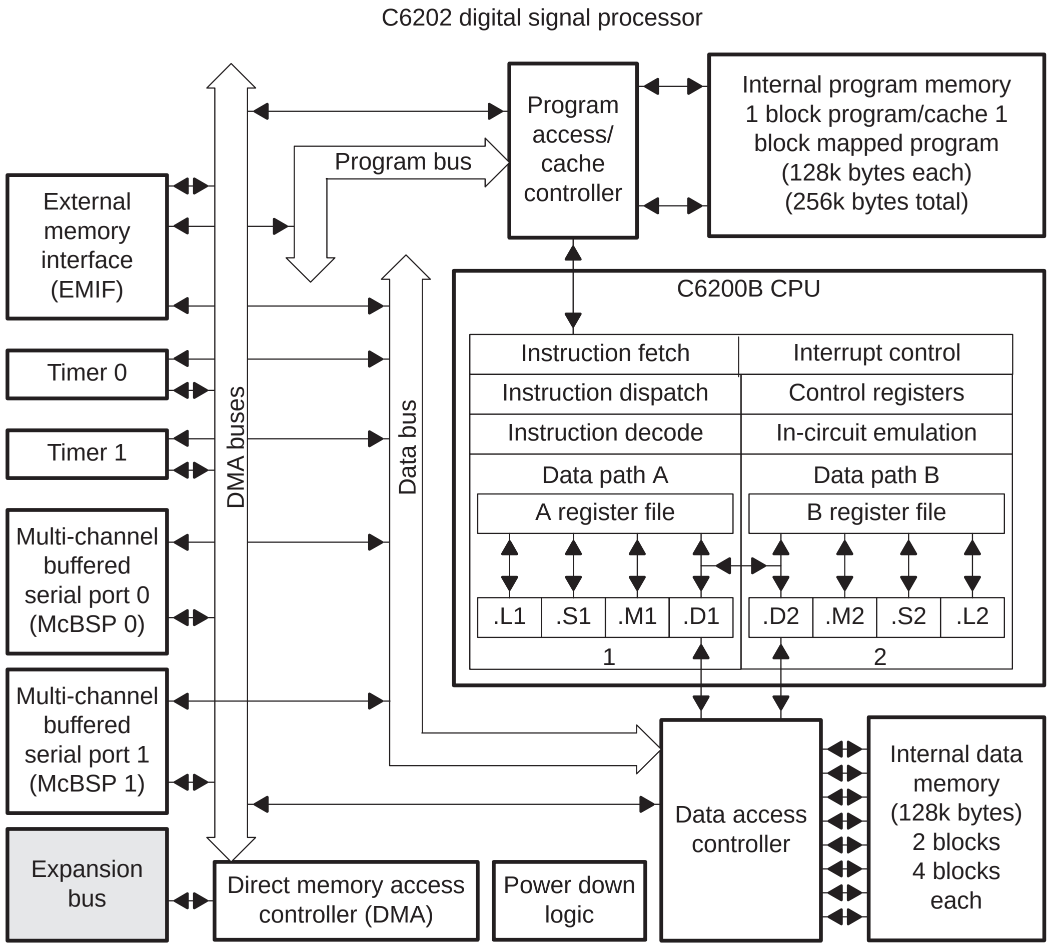


The I/O port has two modes of operation, which can coexist in a single system: asynchronous I/O mode and synchronous FIFO mode. These modes are selectable for each of expansion bus's four XCE spaces. The first mode (asynchronous I/O mode) provides output strobes, which are highly programmable like the asynchronous signals of the external memory interface (EMIF). The expansion bus interface provides four output address signals in this mode, with external decode this provides for up to 16 devices per XCE space. The FIFO mode provides a glueless interface to a single synchronous read FIFO, or up to four synchronous write FIFOs. With a minimal amount of glue, this can be extended to up to 16 read and 16 write FIFOs per XCE space. Connectivity of the expansion bus I/O port and DSP memory is provided through the DMA controller.

The second sub-block of the expansion bus consists of the host port interface. This interface can operate in one of two modes: synchronous and asynchronous. The synchronous mode offers master and slave functionality, and has multiplexed address and data signals. The asynchronous mode is slave only, and is similar to the HPI on the 'C6201/C6211/C6701/C6711, but is extended to a 32-bit data path. The asynchronous host port mode is used to interface to microprocessors which utilize an asynchronous bus.

Connectivity of the expansion bus host port interface and the DSP memory space is provided by the DMA auxiliary port. Dedicated address and data registers connect the host port interface to the expansion bus host channel. An external master accesses these registers using external data and interface control signals. Through a dedicated port the DMA provides connectivity between the processor and the expansion bus I/O port. To initiate transfers via the synchronous host port interface, the CPU has to configure a set of registers. Figure 8–2 shows a chip-level block diagram.

Figure 8–2. The Expansion Bus Interface in the TMS320C6202 Block Diagram


C6202 digital signal processor
Program access/ cache controller
Internal program memory 1 block program/cache 1 block mapped program (128k bytes each) (256k bytes total)
Program bus
External memory interface (EMIF)
Timer 0
Timer 1
Multi-channel buffered serial port 0 (McBSP 0)
Multi-channel buffered serial port 1 (McBSP 1)
Expansion bus
DMA buses
Data bus
C6200B CPU
Instruction fetch
Interrupt control
Instruction dispatch
Control registers
Instruction decode
In-circuit emulation
Data path A
Data path B
A register file
B register file
.L1
.S1
.M1
.D1
.D2
.M2
.S2
.L2
1
2
Data access controller
Internal data memory (128k bytes) 2 blocks 4 blocks each
Direct memory access controller (DMA)
Power down logic

8.2 Expansion Bus Signals

Table 8–1 lists the expansion bus signals and their functionality in each mode.

Table 8–1. Expansion Bus Signals

	I/O Port Mode (Non-Exclusive)				**Mutually Exclusive Host Port Modes**			
Expansion Bus Signal	**(I/O/Z)**	**Async Signal**	**(I/O/Z)**	**Sync FIFO Signal**	**(I/O/Z)**	**Sync Mode**	**(I/O/Z)**	**Async Mode**
XD[31:0]	I/O/Z	D[31:0]	I/O/Z	D[31:0]	I/O/Z	D[31:0]	I/O/Z	D[31:0]
XFCLK			O	XFCLK				
XCLKIN					I	CLK		
$\overline{\text{XCE0}}$	O	$\overline{\text{CS}}$	O	$\overline{\text{RE}}$/$\overline{\text{WE}}$/$\overline{\text{CS}}$				
$\overline{\text{XCE1}}$	O	$\overline{\text{CS}}$	O	$\overline{\text{RE}}$/$\overline{\text{WE}}$/$\overline{\text{CS}}$				
$\overline{\text{XCE2}}$	O	$\overline{\text{CS}}$	O	$\overline{\text{RE}}$/$\overline{\text{WE}}$/$\overline{\text{CS}}$				
$\overline{\text{XCE3}}$	O	$\overline{\text{CS}}$	O	$\overline{\text{RE}}$/$\overline{\text{WE}}$/$\overline{\text{CS}}$				
$\overline{\text{XBE0}}$/XA2	O/Z	XA2	O/Z	XA2	I/O/Z	BE0	I	BE0
$\overline{\text{XBE1}}$/XA3	O/Z	XA3	O/Z	XA3	I/O/Z	BE1	I	BE1
$\overline{\text{XBE2}}$/XA4	O/Z	XA4	O/Z	XA4	I/O/Z	BE2	I	BE2
$\overline{\text{XBE3}}$/XA5	O/Z	XA5	O/Z	XA5	I/O/Z	BE3	I	BE3
$\overline{\text{XOE}}$	O	$\overline{\text{OE}}$	O	$\overline{\text{OE}}$				
$\overline{\text{XRE}}$	O	$\overline{\text{RE}}$	O	$\overline{\text{RE}}$				
$\overline{\text{XWE}}$	O	$\overline{\text{WE}}$	O	$\overline{\text{WE}}$	O	$\overline{\text{WAIT}}$		
$\overline{\text{XAS}}$					I/O/Z	AS		
XRDY	I	XRDY			I/O/Z	READY	O/Z	READY
XW/R					I/O/Z	W/$\overline{\text{R}}$	I	W/$\overline{\text{R}}$
XBLAST					I/O/Z	BLAST		
XHOLD	I/O/Z	HOLD	I/O/Z	HOLD	I/O/Z	HOLD	I/O/Z	HOLD
XHOLDA	I/O/Z	HOLDA	I/O/Z	HOLDA	I/O/Z	HOLDA	I/O/Z	HOLDA
XCNTL					I	CNTL	I	CNTL
XBOFF					I	BOFF		
$\overline{\text{XCS}}$					I	CS	I	CS

8.3 Expansion Bus Registers

Control of the expansion bus and the peripheral interfaces is maintained through memory-mapped registers within the expansion bus. The memory-mapped registers are shown in Table 8–2.

Table 8–2. Expansion Bus Memory Mapped Registers

Byte Address	Name
0188 0000 h	Expansion Bus Global Control (XBGC) Register
0188 0004 h	XCE1 Space Control Register
0188 0008 h	XCE0 Space Control Register
0188 000c h	Expansion Bus Host Port Interface Control (XBHC) Register
0188 0010 h	XCE2 Space Control Register
0188 0014 h	XCE3 Space Control Register
0188 0018 h	Reserved
0188 001c h	Reserved
0188 0020 h	Expansion Bus Internal Master Address (XBIMA) Register
0188 0024 h	Expansion Bus External Address (XBEA) Register

8.3.1 Expansion Bus Host Port Registers

The external master on the expansion bus uses the XCNTL signal to select which internal register is being accessed. The state of this pin selects whether access is made to the expansion bus internal slave address (XBISA) register or, expansion bus data (XBD) register. In addition to that, the external master has access to the entire memory map of the DSP, including memory-mapped registers.

Table 8–3 summarizes the registers that the expansion bus host port uses for communication between the host device and the CPU.

Table 8–3. Expansion Bus Host Port Registers

Register Abbreviation	Register Name	Host Read/ Write Access	‘C6202 Read/ Write Access	Memory Mapped Address
XBHC	Expansion Bus Host Port Control	—	RW	0x0188 000C
XBEA	Expansion Bus External Address	—	RW	0x0188 0024
XBIMA	Expansion Bus Internal Master Address	—	RW	0x0188 0020
XBISA	Expansion Bus Internal Slave Address	RW	—	
XBD	Expansion Bus Data	RW	—	

8.3.2 Expansion Bus Global Control Register

The expansion bus global control register (shown in Figure 8–3, and described in Table 8–4) configures parameters of the expansion bus common to all interfaces.

Figure 8–3. Expansion Bus Global Control Register

31 16	15	14	13 12	11	10 0
Reserved	FMOD	XFCEN	XFRAT	XARB	Reserved
R, +0	R,+x	RW,+0	RW,+00	R,+x	RW,+x

Table 8–4. Expansion Bus Global Control Register Field Description

Field	**Description**
FMOD	FIFO mode set by boot-mode selection. FMOD = 0: Glue is used for FIFO read interface in all XCE spaces operating in FIFO mode FMOD = 1: Glueless read FIFO interface. If $\overline{\text{XCE3}}$ is selected for FIFO mode, then XOE acts as FIFO output enable and $\overline{\text{XCE3}}$ acts as FIFO read enable. *XOE is disabled in all other XCE spaces regardless of MType setting.*

XFCEN	FIFO clock enable
	XFCEN = 0: XFCLK held high
	XFCEN = 1: XFCLK enabled to clock.
	The FIFO clock enable cannot be changed while a DMA request to XCE space is active.

XFRAT	FIFO clock rate
	XFRAT = 00: XFCLK = 1/8 CPU clock rate
	XFRAT = 01: XFCLK = 1/6 CPU clock rate
	XFRAT = 10: XFCLK = 1/4 CPU clock rate
	XFRAT = 11: XFCLK = 1/2 CPU clock rate
	The FIFO clock setting cannot be changed while a DMA request to XCE space is active.

XARB	Arbitration mode, set by boot-mode selection
	XARB=0: internal arbiter disabled
	XARB=1: internal arbiter enabled

8.3.3 XCE Space Control Registers

The four XCE space control registers (shown in Figure 8–4 and described in Table 8–5) correspond to the four XCE memory spaces supported by the expansion bus.

Figure 8–4. Expansion Bus XCE(0/1/2/3) Space Control Register Diagram

31 28	27 22	21 20	19 16	15 14	13 8	7	6 4	3 2	1 0
WRITE SETUP	WRITE STROBE	WRITE HOLD	READ SETUP	rsvd	READ STROBE	rsvd	MTYPE	rsvd	READ HOLD
RW, +1111	RW, +111111	RW, +11	RW, +1111	R, +00	RW, +111111	R, +0	R,+xx	R, +00	RW, +11

Table 8–5. Expansion Bus XCE(0/1/2/3) Space Control Register Field Description

Field	Description
MTYPE	Memory type is configured during boot using pullup or pulldown resistors on the expansion bus.
	MTYPE=010b: 32-bit wide asynchronous interface.
	MTYPE=101b: 32-bit wide FIFO interface.
Others	Reserved

The remaining fields are defined in detail in chapter 6, external memory interface in the TMS320C6201/6701 Peripheral Reference Guide (SPRU190B). These fields are specific to the asynchronous interface and are functionally equivalent to the fields in the EMIF CE space control registers.

8.4 Expansion Bus I/O Port Operation

For external IO port accesses on the expansion bus, the $\overline{\text{XBE}}$ signals act as address signals XA[5:2]. You can use the address signals to address as many as 16 different R/W peripherals or 32 FIFOs in each XCE space. For the FIFO interface, 32 devices are possible since a separate Read and Write FIFO can be located at each address.

Access to the expansion bus I/O port can only be done through the DMA channels 0 through 3. The DMEMC does not have direct access to the expansion bus. Therefore, load and store (LD/ST) commands to the memory spaces of the expansion bus I/O port via the CPU are not allowed, and result in undefined operation. A DMA transfer cannot occur from one XCE space to another XCE space. Also, a host port transaction cannot access any of the XCE spaces.

For reads, care must be taken to ensure that contention on the data bus does not occur when switching from one peripheral to the next in the same XCE space. The DMA can accomplish this since inactive cycles occur when the DMA switches from one frame to the next. The DMA can be set up to read (or write) a frame from each of the peripherals or FIFOs in turn. For example, the element index can be set to 0 and the frame index can be set to a multiple of 4 (ensure word strides), thus incrementing to a different location after each frame has completed.

Although the expansion bus does not explicitly support memory widths of less than 32 bits, the DMA can be used to read/write to 8-bit or 16-bit peripherals or FIFOs by controlling the byte/half-word logical addressing. For example, if an 8-bit-wide FIFO is in XCE2, then the DMA ESIZE bit-field can specify 8-bit transfers. The lower two address bits in the DMA source or destination address register determines the byte lane used for accessing the I/O port. If the bottom two bits are 00b (word aligned), then only XD[7:0] is used for valid data. If A[1:0] = 01b, then XD[15:8] is used (see Figure 8–5 and Table 8–6).

Alternatively, if 16-bit (or 8-bit) peripherals are used, the DMA element index can be set up such that the stride value causes a read from alternating byte lanes during each read transfer. For example, the first access can be to address A[5:0] = xxxx00b, causing the lower half of the data bus to be driven by the peripheral. If the next address is A[5:0] = xxxx10b, the top half of the data bus is driven by the other peripheral (or FIFO) and no bus contention occurs. The only address signals which are externally provided are A[5:2]. If address decoding is required to address a specific peripheral or FIFO, these should be modified as necessary by the DMA to ensure that peripherals are only addressed when appropriate (see Figure 8–6 and Table 8–7).

Figure 8–5 illustrates how to interface four 8-bit FIFOs to the I/O port (memory map for this case is described in Table 8–7). Figure 8–6 is an example of interface between two 16-bit FIFOs and the I/O port.

Figure 8–5. Example of the Expansion Bus Interface to Four 8-Bit FIFOs

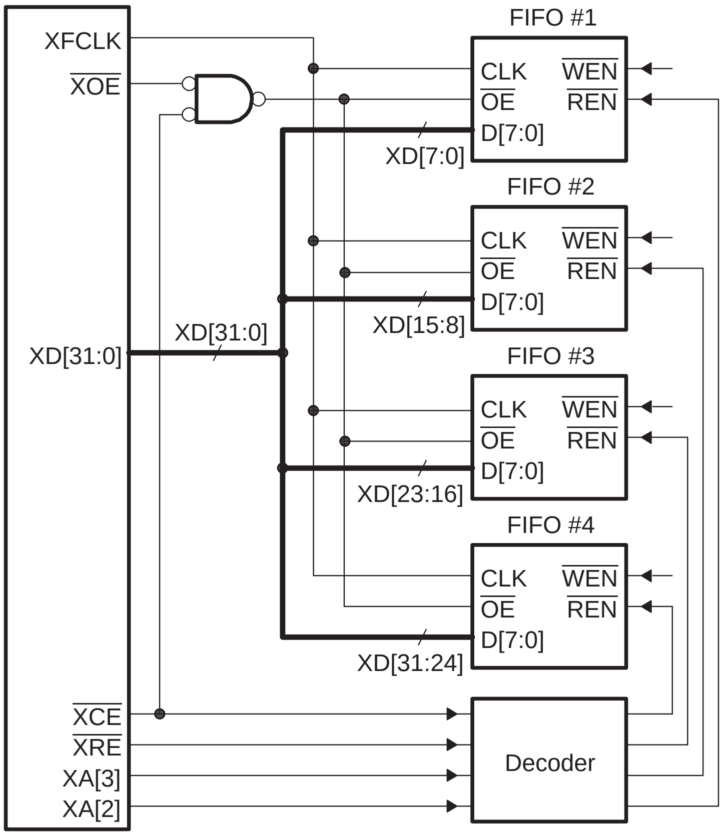


Table 8–6. Addressing Scheme – Case When Expansion Bus is Interfaced to Four 8-Bit FIFOs

Logical Address	**A[31:6]**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
FIFO #1 Address	X	X	X	0	0	0	0
FIFO #2 Address	X	X	X	0	1	0	1
FIFO #3 Address	X	X	X	1	0	1	0
FIFO #4 Address	X	X	X	1	1	1	1
Physical Address		XA5	XA4	XA3	XA2		

Figure 8–6. Example of the Expansion Bus Interface to Two 16-Bit FIFOs

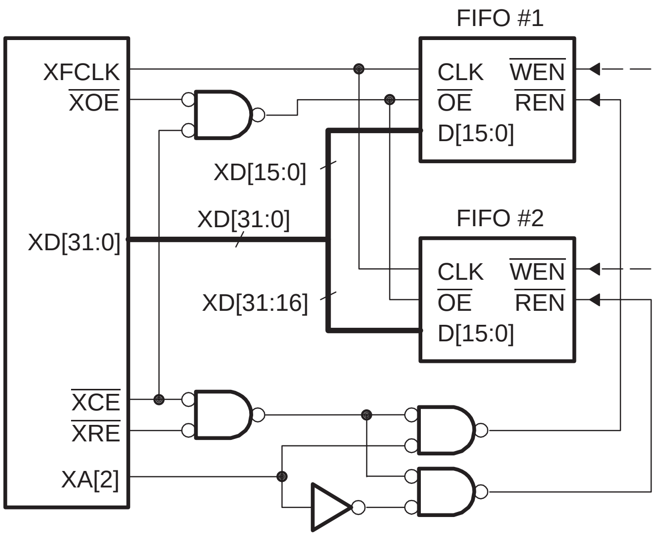


Table 8–7. Addressing Scheme – Case When the Expansion Bus is Interfaced to Two 16-Bit FIFOs

Logical Address	**A[31:6]**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
FIFO #1 Address	X	X	X	X	0	0	0
FIFO #2 Address	X	X	X	X	1	1	0
Physical Address		XA5	XA4	XA3	XA2		

8.4.1 Asynchronous Mode

The asynchronous cycles of the expansion bus are identical to the asynchronous cycles provided by the EMIF. During asynchronous peripheral accesses, XRDY acts as an active-high ready input and XBE[3:0]/XA[5:2] operate as address signals XA[5:2]. The remaining asynchronous peripheral signals operate exactly like their EMIF counterpart. For a complete description, see the External Memory Interface section. The following minimum values apply to the asynchronous parameters:

- SETUP + STROBE + HOLD ≥ 3
  - SETUP ≥ 1
  - STROBE ≥ 1
- If XRDY used to extend STROBE then HOLD ≥ 2.

Notes:

1) XRDY is active (low) during host-port accesses.
2) XBE[3:0]/XA[5:2] operate as XBE[3:0] during host-port accesses.

8.4.2 Synch FIFO Modes

The synchronous FIFO mode of the expansion bus offers a glueless and/or low glue interface to standard synchronous FIFOs.

The expansion bus can interface up to four write FIFOs without using glue logic (one per XCE space) or three write FIFOs and a single read FIFO (in $\overline{\text{XCE3}}$ only). However, with a minimal amount of glue, up to 16 read and write FIFOs can be used per XCE space.

The XOE, XRE, XWE, and $\overline{\text{XCEn}}$ signals are not tri-stated while the DSP releases control of the expansion bus.

Table 8–8. Synch FIFO Pin Description

Signal Name	(I/O/Z)	Signal Purpose	Signal Function	
			R/W Mode	**Read Mode**
XFCLK	O	FIFO clock output	Programmable to either 1/2, 1/4, 1/6, or 1/8 of the CPU clock frequency. If CPU clock = 250 MHz, then XFCLK = 125, 62.5, 41.7 or 31.25 MHz. The XFCLK continues to clock even when the DSP releases ownership of the XBUS.	
XD[31:0]	I/O/Z	Data	Data lines	
$\overline{\text{XCEx}}$	O	FIFO read enable/write enable/chip Select	Active for both read and write transactions. They should be logically OR-ed with output control signals externally to create dedicated controls for a FIFO. Also can be used directly as FIFO write enable signal for a single write FIFO per XCE space.	Acts as read enable signal(XCE3 only)
$\overline{\text{XWE}}$	O	FIFO write enable	Write-enable signal for FIFO. Must be logically OR-ed with corresponding XCE signal to ensure that only one FIFO is addressed at a time.	
$\overline{\text{XRE}}$	O	FIFO read enable	Read-enable signal for FIFO. Must be logically OR-ed with corresponding XCE signal to ensure that only one FIFO is addressed at a time.	
$\overline{\text{XOE}}$	O	FIFO output enable	Shared output enable signal. Must be logically OR-ed with corresponding XCE signal to ensure that only one FIFO is addressed at a time.	Dedicated output enable signal in XCE3 if FIFO read mode is selected. *If selected, this signal is disabled for all other modes*.
$\overline{\text{XBE[3:0]}}$/ XA[5:2]	O/Z	Expansion bus address	Operate as XA[5:2]. Can be decoded to specify up to 16 different addresses, enabling interface with glue to 16 Read FIFOs and 16 Write FIFOs in a single XCE space.	

8.4.2.1 Write Interface

During write accesses to a memory space configured for read/write FIFO mode, the XCE signal and XWE signal are both active for a single rising edge of XFCLK. So, depending on the specific system environment, the write interface can be accomplished either with glue or without glue.

The glueless interface can be used if only a single write FIFO is used in a given XCE space (see Figure 8–7), since the XCE signal is used as the write enable signal. If this is true, the XCE signal is tied directly to the write enable input of the FIFO. If a read FIFO is also used in the same XCE space, glue must be used, since the XCE signal also goes low for reads from the read FIFO.

Figure 8–8 shows an interface to a read FIFO and a write FIFO in the same XCE space. For this example, the XCE signal is used to gate the appropriate read/write strobes to the FIFOs. The FIFO write timing diagram for this interface is shown in Figure 8–9.

Several FIFOs can be accessed in a single XCE space if address decode logic is used to access each FIFO separately.

Figure 8–7. Glueless Write FIFO Interface

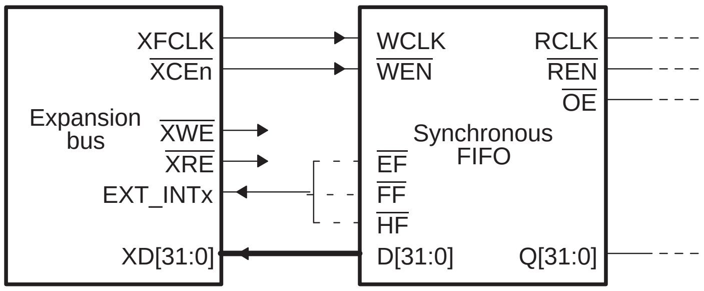


Figure 8–8. Read and Write FIFO Interface With Glue

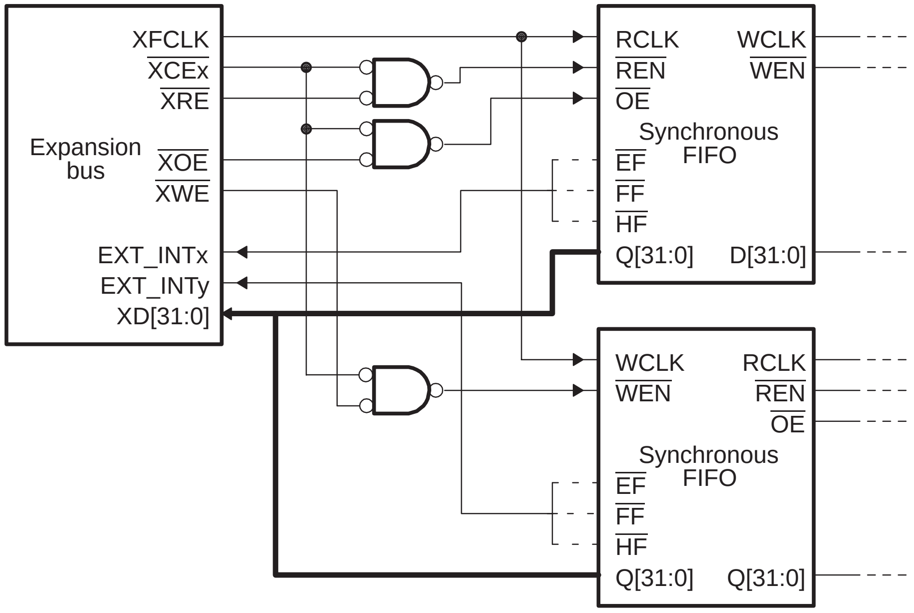
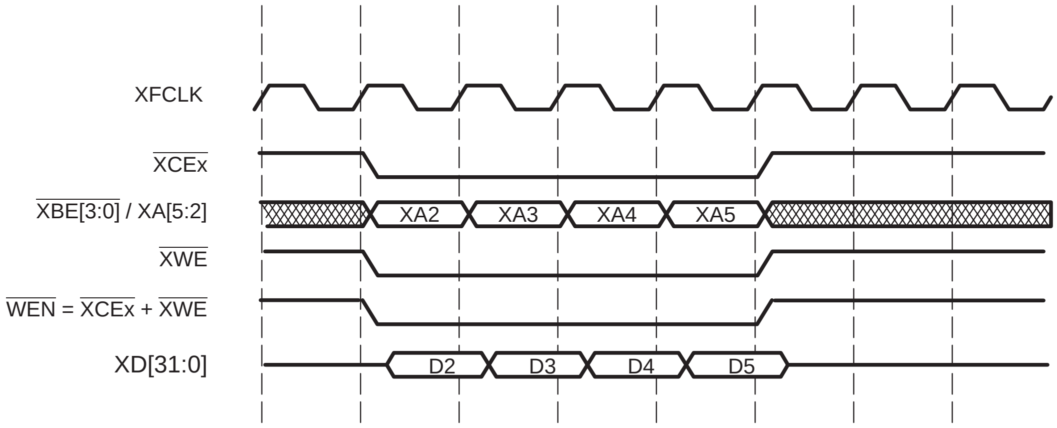


Figure 8–9. FIFO Write Cycles



8.4.2.2 Read FIFO Interface

The read FIFO interface can be accomplished gluelessly in XCE3 space or with a small amount of glue in any XCE space. If a glueless read FIFO interface is used (specified by boot configuration selection), the $\overline{\text{XOE}}$ signal is *only* enabled in XCE3 space, and is dedicated to use for the FIFO interface. If this mode is selected at boot, the *XOE signal is disabled in all other XCE spaces*. In this mode, $\overline{\text{XCE3}}$ is used as the read enable signal and $\overline{\text{XOE}}$ is used as the output enable signal of the FIFO. Figure 8–10 shows this interface (Figure 8–11 shows the timing diagram for this interface). If the glueless read FIFO mode is not chosen, then a minimal amount of glue can be used in any XCE space specified as a FIFO interface. Figure 8–8 shows the required glue. Figure 8–12 shows the timing diagram for the case when glue logic is used to read from FIFO.

Figure 8–10. Glueless Read FIFO Interface

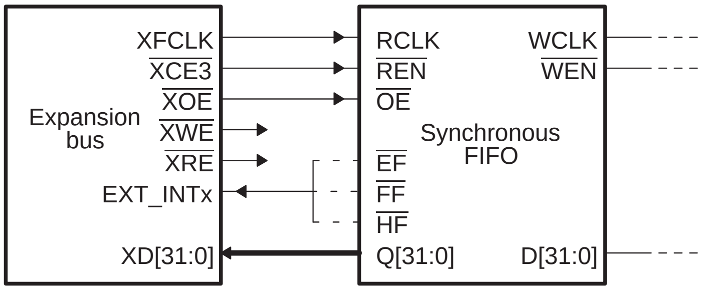


Figure 8–11. FIFO Read Mode – Read Timing (glueless case)

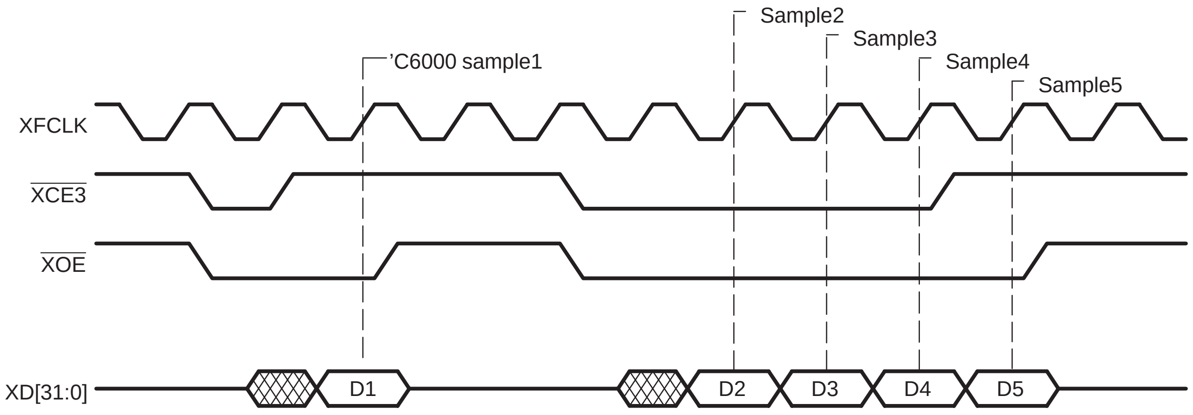
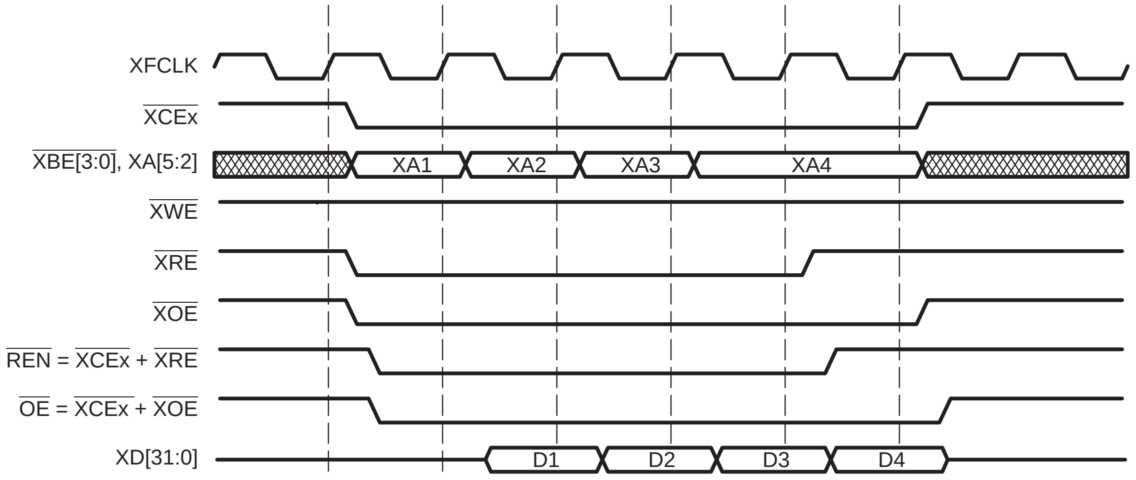


Figure 8–12. FIFO Read Mode – With Glue



8.4.2.3 Programming Offset Register

The programmable offset registers of the FIFO are used to hold the offset values for the flags that indicate the condition of the FIFO contents.

The programmable offset registers of the FIFO must be programmed in consecutive cycles and read in consecutive cycles. In addition, the reader cannot read from the FIFO until the writer has programmed the offset registers. This should not be a problem, since the FIFO is not read until it has been written to. The writer should not write to the FIFO until the offset registers have been programmed.

For programming (or reading) the offset registers, back-to-back accesses must be done. For example, the first XFCLK edge with the program input to the FIFO low programs the PAE register, and then the second XFCLK edge programs the PAF register. Also, for 9-bit or large 18-bit FIFOs, it is common to require two or three write cycles to fully program each register. The first write programs the LSB, the second write programs the middle bits and the third write programs the high bits.

A general-purpose output (DMACx or TOUTx) can be used to control whether FIFO reads/writes are done to the FIFO memory or to the programmable offset register of the memory. Or the XA[5:2] signals can be decoded to control when the FIFO offset register is accessed.

8.4.2.4 Flag Monitoring

To efficiently control bursts to and from the dedicated FIFO interfaces, the interrupt signals EXT_INT4, EXT_INT5, EXT_INT6, and EXT_INT7 are used as flags to control DMA transfers. The flag polarity used to start transfer can be programmed in the DMA secondary control register. The CPU EXT_INT and DMA EXT_INT polarity are controlled separately. For more details see the DMA section.

8.4.3 DMA Transfer Examples

8.4.3.1 Example 1 (single frame transfer)

Peripherals located on the I/O port of the expansion bus are accessible only via DMA transactions. This section gives a very simple example used to transfer a single frame of 256 words from a FIFO located in XCE0 into internal data memory at 8000 0000h. This example simply sets up the source and destination registers, and starts the DMA with incrementing destination address and a non-changing source address. The source address does not change, since the FIFO is located in a fixed memory location. The content of relevant registers and DMA channel primary control register are shown in Table 8–9 and Table 8–10.

Table 8–9. Content of Relevant Registers (single frame transfer)

Register	Contents
DMA primary control register	0000 0041h
DMA source	4000 0000h
DMA destination	8000 0000h
Transfer counter register	0000 0100h

Table 8–10. Content of DMA Channel Primary Control Register Fields

DST reload	SRC reload	EMOD	FS	TCINT	PRI	WSYNC	RSYNC	INDEX	CNT reload	SPLIT	ESZISE	DST DIR	SRC DIR	STATUS	START
00	00	0	0	0	0	00000	00000	0	0	00	00	01	00	00	01

8.4.3.2 Example 2 (transfer with frame synchronization)

In this example ten frames of 256 words from a FIFO located in XCE0 are transferred into internal data memory at 8000 0000h. This example simply sets up the source and destination registers, and starts the DMA with incrementing destination address and a non-changing source address. The source address does not change, since the FIFO is located in a fixed memory location. Active(high) EXT_INT4 is used for frame synchronization. The content of the relevant registers, and the content of the DMA channel primary and secondary control register fields are shown in Table 8–11, Table 8–12, and Table 8–13.

Table 8–11. Content of Relevant Registers (multiple frame transfer)

Register	Content
DMA Primary Control Register	0401 0041h
DMA Secondary Control Register	0008 0000h
DMA Source	4000 0000h
DMA Destination	8000 0000h
Transfer Counter Register	000A 0100h
Global Counter Reload Register A	0000 0100h

Table 8–12. Content of TMS320C6202 DMA Primary Control Register

DST reload	SRC reload	EMOD	FS	TCINT	PRI	WSYNC	RSYNC	INDEX	CNT reload	SPLIT	ESZISE	DST DIR	SRC DIR	STATUS	START
00	00	0	1	0	0	00000	00100	0	0	00	00	01	00	00	01

Table 8–13. Content of TMS320C6202 DMA Secondary Control Register

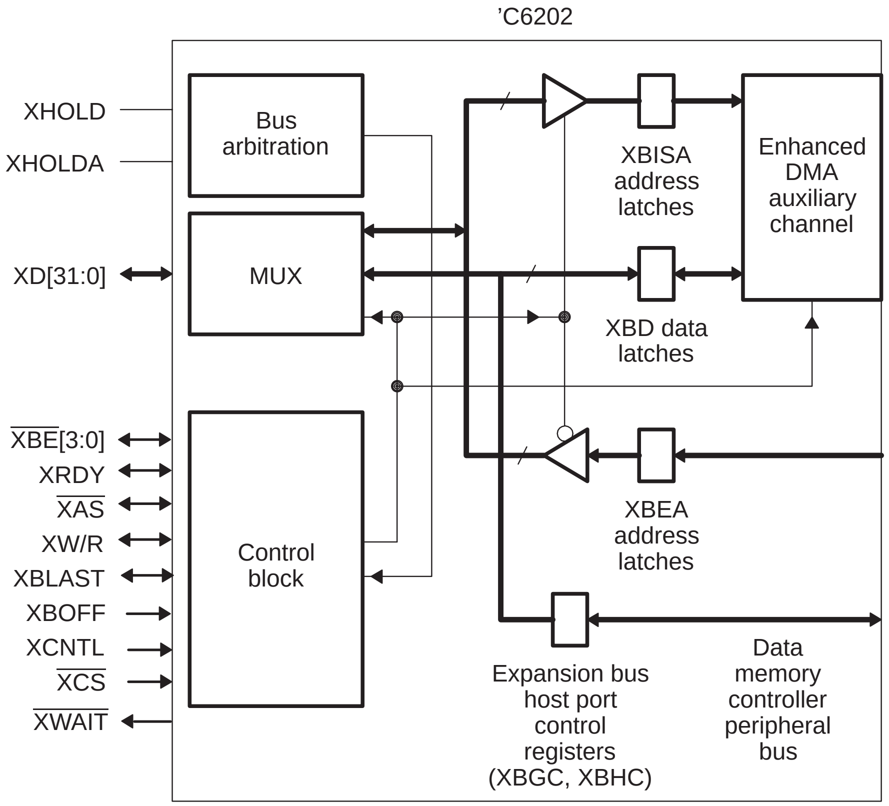
Reserved	SYNC CNTL	DMAC EN	WSYNC CLR/WSYNC STAT/RSYNC CLR/RSYNC STAT/WDROP IE/WDROP COND/WDROP COND/RDROP IE/RDROP COND/BLOCK IE/BLOCK COND/LAST IE/LAST COND/FRAME IE
0000 0000 00	001	0 00	0000 0000 0000 0000

8.5 Expansion Bus Host Port Operation

The expansion bus host port has two modes, which enable interfaces to external processors, PCI bridge chips, or other external peripherals. These are the synchronous host port mode and the asynchronous host port mode. The synchronous host port mode can interface with minimum glue to PCI bridge chips and many common microprocessors. The asynchronous host port mode enables interfacing to genuine asynchronous devices.

The expansion bus host port block diagram is shown is Figure 8–13.

Figure 8–13. Expansion Bus Host Port Interface Block Diagram



Using pull-up/down resistors on the data bus during reset sets the host port operational mode, the DSP bootmode, and endianness.

8.5.1 Expansion Bus Host Port Registers Description

8.5.1.1 Expansion Bus Data Register

The expansion bus data (XBD) register, shown in Figure 8–14, contains the data that was read from the memory accessed by the expansion bus host port if the current access is a read, or the data that is written to the memory if the current access is a write.

This register is used when expansion host port operates either in synchronous or asynchronous mode.

Figure 8–14. Expansion Bus Data Register

31 … 0
XBD
HRW,+0000 0000 0000 0000 0000 0000 0000 0000

8.5.1.2 Expansion Bus Internal Slave Address Register

The expansion bus internal slave address (XBISA) register is used when the external expansion bus master initiates data transfer. This register controls the memory location in the DSP memory map being accessed by the external mastering data transactions. This address is a 30-bit word address. The two LSB bits in this register are used by the host to enable or disable autoincrement of XBISA register, and to trigger the interrupt (by setting the DSPINT bit).The XBISA register is shown in Figure 8–15 and described in Table 8–14.

Figure 8–15. Expansion Bus Internal Slave Address Register (XBISA)

31 … 2	1	0
XBSA	AINC	DSPINT
HRW,+0000 0000 0000 0000 0000 0000 0000 00	HRW, +0	HRW, +0

Table 8–14. XBISA Register Description

Field	Description
DSPINT	The external master to DSP interrupt. Used to wake up the DSP from reset. This bit is cleared by corresponding bit in the XBHC.
AINC	Enable autoincrement. (Asynchronous mode only) AINC = 0: XBD register is accessed with autoincrement of XBSA field. AINC = 1: XBD register is accessed without autoincrement of XBSA field.
XBSA	30-bit word address. The XBSA bit-field controls memory location in the DSP memory map being accessed by the host.

This register is used when the host port operates either in synchronous or asynchronous mode. The ’C6202 does not have access to the XBISA register content. Burst transfers in the synchronous host-port mode are always expected to occur with autoincrement (AINC bit should be set to zero).

8.5.1.3 Expansion Bus Internal Master Address Register

The expansion bus internal master address (XBIMA) register, shown in Figure 8–16, specifies the source or destination address in the DSP memory map where the transaction starts. This register is set by the ‘C6202 when the DSP wants to initiate transfer on the expansion bus. Since all tranfers have a width of one word, the XBIMA register is incremented by four after each transfer.

This register is used when the host port operates in synchronous mode.

Figure 8–16. Expansion Bus Internal Master Address Register

31	0
XBIMA	
RW,+0000 0000 0000 0000 0000 0000 0000 0000	

8.5.1.4 Expansion Bus External Address Register

This register is set by the ‘C6202 when the DSP wants to initiate transfer on the expansion bus. The content of the XBEA register, shown in Figure 8–17, appears on the XD[31:0] lines during an address phase of the transfer initiated by the DSP. The expansion bus external address (XBEA) specifies where in the external slave memory map the data is accessed. Since all tranfers have a width of one word, the XBEA register is incremented by four after each transfer.

This register is used when the host port operates in synchronous mode.

Figure 8–17. Expansion Bus External Address Register

31	0
XBEA	
RW,+0000 0000 0000 0000 0000 0000 0000 0000	

8.5.1.5 Expansion Bus Host Port Interface Control Register

The expansion bus host port interface control (XBHC) register (shown in Figure 8–18 and described in Table 8–15) configures expansion bus host port parameters.

The START bit field in the XBHC register is not cleared to zero after a transfer is completed. Writing ’00’ to the the START field, when a transfer in progress is stalled by XRDY high, aborts the transfer. When a transfer is aborted the XBIMA and XBEA registers and the XFRCT transfer counter reflect the state of the aborted transfer. Using this state information, the transfer can be restarted. Writing other values than ’00’ to the START field is not recommended.

Figure 8–18. Expansion Bus Host Port Interface Control (XBHC) Register

31–16
XFRCT
RW,+0000 0000 0000 0000

15–6	5	4–3	2	1	0
Reserved	INTSRC	START	Reserved	DSPINT	Reserved
R,+0000 0000 00	RW, +0	RW, +00		RW, +0	

Note: R = Read, W = Write, +0 =Reset value

Table 8–15. XBHC Register Description

Field	Description
DSPINT	The external master to DSP interrupt (used to wake up the DSP from reset) is cleared when this bit is set.
START[1:0]	Start bus master transaction Start = 01: starts a write burst transaction from address pointed by XBIMA to address pointed by XBEA Start = 10: starts a read burst transaction from address pointed by XBEA to address pointed by XBIMA Writing ’00’ to the the START field, while an active transfer is stalled by XRDY high, aborts the transfer. When a transfer is aborted the expansion bus registers reflect the state of the aborted transfer. Using this state information, you can restart the transfer.
INTSRC	The expansion bus host port interrupt can be caused either by DSPINT bit or by XFRCT counter. The INTSRC selects interrupt source between DSPINT and XFRCT counter. INTSRC=0: interrupt source is DSPINT bit INTSRC=1: interrupt is generated at the completion of the master transfer initiated by writing to the START bit-field.
XFRCT	Transfer counter controls the number of elements transferred between the expansion bus and an external slave when the CPU is mastering the bus (range of up to 64k).

8.5.2 Synchronous Host Port Mode

In this mode host port has address and data signals multiplexed and is i960Jx compatible. This allows a minimum glue interface to the PCI bus, since major PCI interface chip manufacturers adopted the i960 bus for local bus on their chips.

The synchronous host port can also easily interface to many other common processors, and essentially act in a slave only mode. This is done by simply not initiating transactions on the expansion bus.

The ‘C6202 expansion bus has the capability to initiate and receive burst transfers.

Table 8–16 lists pin function in the expansion bus synchronous host port mode:

Table 8–16. Expansion Bus Pin Description (Synchronous Host Port Mode)

Signal Symbol	Signal Type	Signal Count	Signal Name	Signal Function
XCLKIN	I	1	Clock Input	Expansion bus clock (maximum clock speed is 1/4 of the CPU clock speed.
$\overline{\text{XCS}}$	I	1	Chip Select	Selects the ‘C6202 as a target of an external master.
XHOLD	I/O/Z	1	Hold Request	Case 1 (Internal bus arbiter enabled) XHOLD is asserted by external device to request use of the expansion bus. The ‘C6202 asserts XHOLDA when control is granted. Case 2 (Internal bus arbiter disabled) The ‘C6202 wakes up from reset as slave on the bus. XHOLD is asserted by ‘C6202 to request use of the expansion bus. The expansion bus arbiter asserts XHOLDA when control is granted.
XHOLDA	I/O/Z	1	Hold acknowledge	Case 1 (Internal bus arbiter disabled) The ‘C6202 wakes up from reset as slave on the bus. The expansion bus arbiter asserts XHOLDA when control is granted in response to XHOLD. The bus should not be granted to ‘C6202 unless requested by XHOLD. Case 2 (Internal bus arbiter enabled) The ‘C6202 wakes up from reset as master of the bus. XHOLDA is asserted by the ‘C6202 when control is granted in response to XHOLD.

Table 8–16. Expansion Bus Pin Description (Synchronous Host Port Mode) (Continued)

Signal Symbol	Signal Type	Signal Count	Signal Name	Signal Function
XD[31:0]	I/O/Z	32	Address/ data bus	Data
XBLAST	I/O/Z	1	Burst last	Signal driven by the current expansion bus master to indicate the last transfer in a bus access. Input polarity selected at boot. Output polarity is always active low.
$\overline{\text{XAS}}$	I/O/Z	1	Address Strobe	Indicates a valid address and the start of a new bus access. Asserted for the first clock of a bus access.
XCNTL	I	1	Control signal	This signal selects between XBD and XBISA register. XCNTL=0: access is made to the XBD register XCNTL=1: access is made to the XBISA register
$\overline{\text{XBE[3:0]}}$/ XA[5:2]	I/O/Z	4	Byte enables	During host-port accesses these signals operate as $\overline{\text{XBE[3:0]}}$. $\overline{\text{BE3}}$ byte enable 3: XD[31:24] $\overline{\text{BE2}}$ byte enable 2: XD[23:16] $\overline{\text{BE1}}$ byte enable 1: XD[15:8] $\overline{\text{BE0}}$ byte enable 0: XD[7:0]
XW/R	I/O/Z	1	Read/write	Write/read enable Polarity of this signal is configured during boot.
XRDY	I/O/Z	1	Ready out Ready in	Active(low) during host-port access. XRDY is an input when the ’C6202 owns the bus. When the ’C6202 does not own the bus, XRDY is not driven until a request is made to the ’C6202.
XBOFF	I	1	Bus Back-Off	When asserted, suspends the current access and the ’C6202 releases ownership of the expansion bus.
$\overline{\text{XWAIT}}$	O	1	Wait	Ready output for master accesses

8.5.2.1 TMS320C6202 Master on the Expansion Bus

When the ‘C6202 is the master of the expansion bus, it can initiate a burst read or write to a peripheral on the bus.

When the DSP controls the bus, data flow is controlled in a manner similar to a DMA transfer; however, the expansion bus host channel controls the actual data transfer. The event flow is as follows:

1) The DSP must initialize the XBEA, which dictates where in the external slave memory map that data is accessed.
2) The XBIMA must be set to specify the source or destination address in the DSP memory map where the transaction starts.
3) The XFRCNT field of the expansion bus host port control (XBHC) register field is set to control the number of elements being transferred.
4) The start field is written,controlling whether the external access is a read or write burst.

An interrupt is generated at the completion of the transfer if specified by the INTSRC bit in the XBHC register.

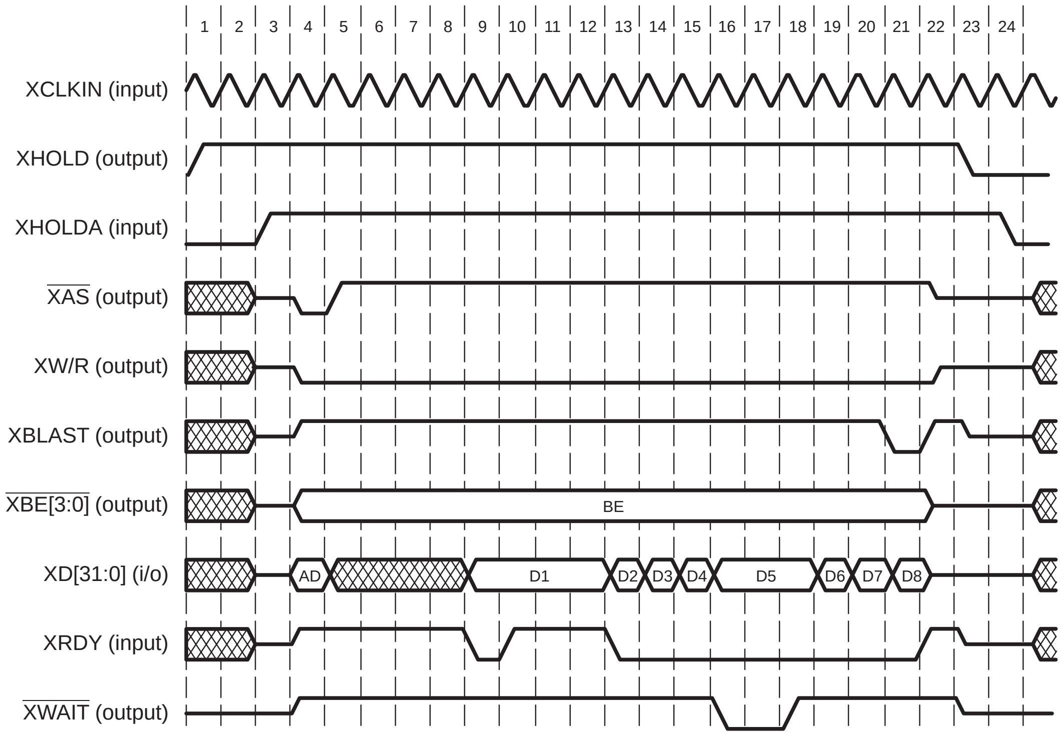
Figure 8–19 and Figure 8–20 show examples of timing diagrams for a burst read and write when the ‘C6202 is mastering the bus. In this case internal bus arbiter is disabled (XHOLD is output and XHOLDA is input) and ‘C6202 wakes up from reset as slave on the expansion bus.

The $\overline{\text{XWAIT}}$ signal prevents data overflow/underflow when the DSP is a master on the expansion bus. The $\overline{\text{XWAIT}}$ signal, which is multiplexed with the $\overline{\text{XWE}}$ signal, can be thought of as a ready output when the ’C6202 initiates transfers on the expansion bus. By asserting the $\overline{\text{XWAIT}}$ signal low, the ’C6202 (the ’C6202 initiated a transaction) indicates that it is not ready to deliver/receive new data.

Burst Read Transfer

The timing presented in Table 8–16 can be referenced for a visual description of the steps required to complete a burst read initiated by the ’C6202 and throttled by the $\overline{\text{XWAIT}}$ and XRDY signals.

Figure 8–19. Read Transfer Initiated by the TMS320C6202 and Throttled by $\overline{\text{XWAIT}}$ and XRDY (Internal Bus Arbiter Disabled)



The step by step description of the events marked above the waveforms in Figure 8–19 follows:

1) The ’C6202 requests the expansion bus by asserting XHOLD output.

2) The DSP waits for the expansion bus.

3) The external bus arbiter asserts the XHOLDA signal, and the ’C6202 starts driving the bus. The $\overline{\text{XAS}}$, XW/R, XBLAST, $\overline{\text{XBE[3:0]}}$ signals become outputs, and the XRDY signal becomes an input.

4) Address phase: During this phase, $\overline{\text{XAS}}$ is asserted and the address is presented on the expansion bus.

5) Data phase: The external device is not ready to deliver data, as indicated by XRDY high.

6) Same as 5.

7) Same as 5.

8) Same as 5.

9) The external device presents requested data (D1), and asserts XRDY.

10) The external device is not ready to deliver next data. The XRDY is negated.

11) Same as 10

12) Same as 10

13) The external device presents next data (D2), and asserts XRDY.

14) The external device presents next data (D3), and XRDY stays asserted.

15) The external device presents next data (D4), and XRDY stays asserted.

16) The external device presents next data (D5), and XRDY stays asserted. The DSP can not accept the new data (D5), and asserts $\overline{\text{XWAIT}}$.

17) The external device recognizes $\overline{\text{XWAIT}}$, and keeps the D5 on the expansion bus. The XRDY is asserted and indicates that the external device is ready waiting for the DSP to accept the data.

18) The DSP deasserts $\overline{\text{XWAIT}}$,, and accepts D5.

19) The external device presents next data (D6), and XRDY stays asserted.

20) The external device presents next data (D7), and XRDY stays asserted.

21) The external device presents the last data (D8), and the ’C6202 asserts the XBLAST.

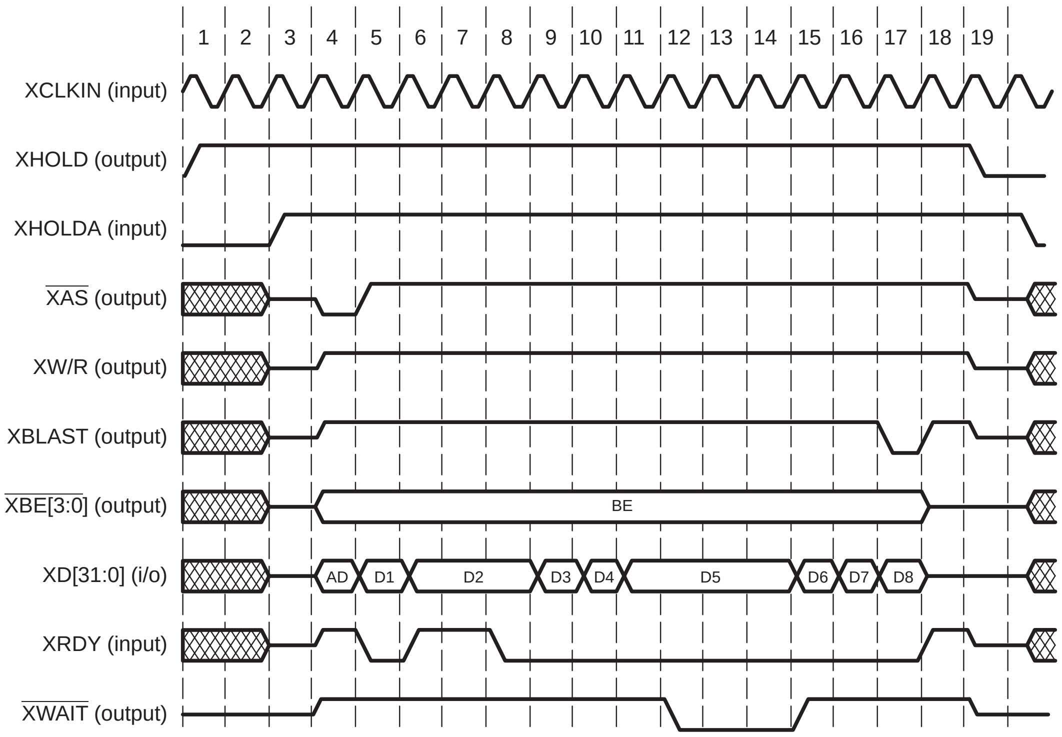
22) The recovery cycle.

23) The DSP negates the expansion bus request (XHOLD), and turns off the outputs.

Burst Write Transfer

The timing presented in Figure 8–20 can be referenced for a visual description of the steps required to complete a burst write initiated by the C6202 and throttled by the $\overline{\text{XWAIT}}$ and XRDY signals.

Figure 8–20. Write Transfer Initiated by the TMS320C6202 and Throttled by $\overline{\text{XWAIT}}$ and XRDY (Internal Bus Arbiter Disabled)



The step by step description of the events marked above the waveforms in Figure 8–20 follows:

1) The DSP requests the expansion bus (XHOLD asserted).

2) The DSP waits for the XHOLDA signal to be asserted by the external arbiter.

3) The external bus arbiter asserts the XHOLDA signal, the $\overline{\text{XAS}}$, XW/R, XBLAST, and $\overline{\text{XBE[3:0]}}$ signals become outputs, and the XRDY signal becomes an input.

4) Address phase: During this phase, the $\overline{\text{XAS}}$ is asserted and the address is presented on the expansion bus.

5) Data phase: During this phase, data (D1) is presented by the DSP and the external device is ready to accept the data, which is indicated by XRDY being active.

6) The DSP presents next data (D2). The external device indicates not ready condition, which is indicated by XRDY being inactive.

7) The ’C6202 is holding data D2 on the expansion bus since the external device is still not ready.

8) External device finally accepts the D2.

9) The DSP presents next data (D3). The external device is ready to take D3.

10) The DSP presents next data (D4). The external device is ready to take D4.

11) The DSP presents next data (D5). The external device is ready to take D5.

12) The DSP is not ready to present D6 and asserts $\overline{\text{XWAIT}}$. The external device is waiting for the DSP to present new data.

13) Same as 12.

14) Same as 12.

15) The DSP presents next data (D6), and negates $\overline{\text{XWAIT}}$. The external device is ready to take D6.

16) The DSP presents next data (D7). The external device is ready to take D7.

17) The DSP presents the last data (D8), and asserts XBLAST. The external device is ready to take D8.

18) Recovery cycle

19) The DSP removes the bus request (XHOLD), and is turns off the outputs.

To prevent contention on the expansion bus, one recovery state between the last data transfer and next address cycle is inserted.

Preventing Deadlocks with Backoff

To prevent deadlocks while the ’C6202 is performing a master transfer, the expansion bus has the XBOFF signal. When asserted, XBOFF suspends the current access and causes the ‘C6202 to release ownership of the expansion bus. Figure 8–21 is timing diagram for the XBOFF signal.

The backoff is only recognized during active master transfers when XRDY indicates a not ready status and:

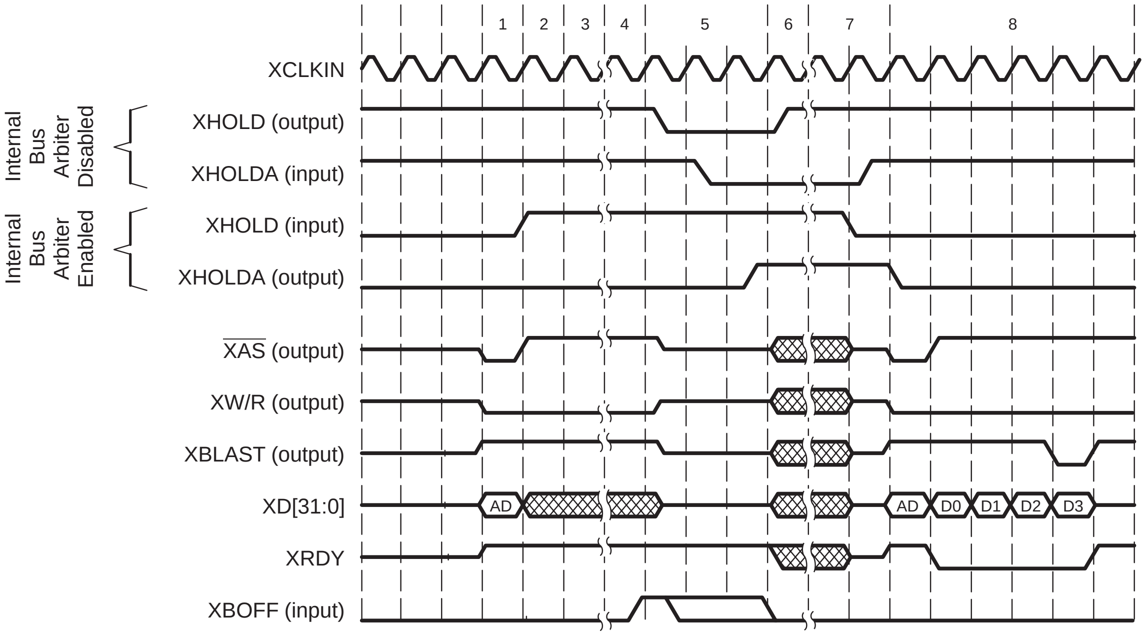
1) The external device is requesting the expansion bus (XHOLD = 1), when the internal bus arbiter is enabled (XARB = 1)

or

2) The DSP is the expansion bus master (XHOLD = 1 and XHOLDA = 1), and the internal bus arbiter is disabled (XARB = 0).

The backoff request is not serviced until all current master transfers are completed internally. This allows read data to be flushed out of the pipeline. The XBOFF signal is not recognized during I/O port transfers.

Figure 8–21. External Device Requests the Bus From the TMS320C6202 Using XBOFF



The timing diagram shown in Figure 8–21 can be referenced for a visual description of the steps involved in release of the expansion bus ownership as initiated by the XBOFF signal. The diagram illustrates the backoff condition for both internal bus arbiter enabled and internal bus arbiter disabled . The step by step description of the events in Figure 8–21 follows:

1) The ‘C6202 is expansion bus master and initiates address phase of a read transaction. The $\overline{\text{XAS}}$ signal is active and valid address is presented.
2) The XRDY signal is high indicating that the external device is not ready to perform the transaction. Also, the external device drives XHOLD active, indicating a bus request.
3) The ‘C6202 is still holding the expansion bus waiting for XRDY to become low.
4) The external device asserts XBOFF, indicating a potential deadlock condition.
5) The DSP responds by releasing the expansion bus. When the internal bus arbiter is enabled, the DSP asserts XHOLDA. When the internal bus arbiter is disabled the DSP deasserts XHOLD. It can take a several clock cycles before ‘C6202 responds to XBOFF. Figure 8–21 shows the fastest response time, one cycle.
6) The expansion bus ownership changes. The new master drives the expansion bus. XBOFF is deasserted.
7) The external device releases the bus after performing the desired transactions.
8) The XHOLDA is removed, and the DSP resumes the expansion bus ownership.
9) The DSP performs a burst read of four words.

The DSP automatically tries to restart the transfer interrupted by a backoff from the point where the interruption took place. The transfer restart is completely transparent to the user.

8.5.2.2 TMS320C6202 Slave on the Expansion Bus

The external host can access the different expansion bus host port registers by driving the XCNTL signal as follows:

❑ XCNTL = 0
Reads or writes the expansion bus data (XBD) register.

❑ XCNTL = 1
Reads or writes the expansion bus internal slave address (XBISA) register.

Every transaction initiated by the host on the expansion bus is a two step process. First, the host has to set the XBISA register, and then transfer the data to/from the address pointed by XBISA register. The data transfer can take place with or without auto-incrementing the internal ‘C6202 memory address register (XBISA). Whether the XBISA gets autoincremented is determined by the AINC bit-field of the XBISA register.

To read/write from the ‘C6202 memory space, the host must follow the following sequence:

1) The host writes the transfer source/destination address to the XBISA register, and sets AINC accordingly (in bit one of the XBISA register).
2) The host reads/writes to/from the address specified by XBISA. Read or write is dictated by the XW/R signal. The XBISA register is auto-incremented or not depending on what is written to the AINC bit during step 1.
3) If the transfer is a burst, dictated by the BLAST signal, data is continuously read or written.

Cycle Description

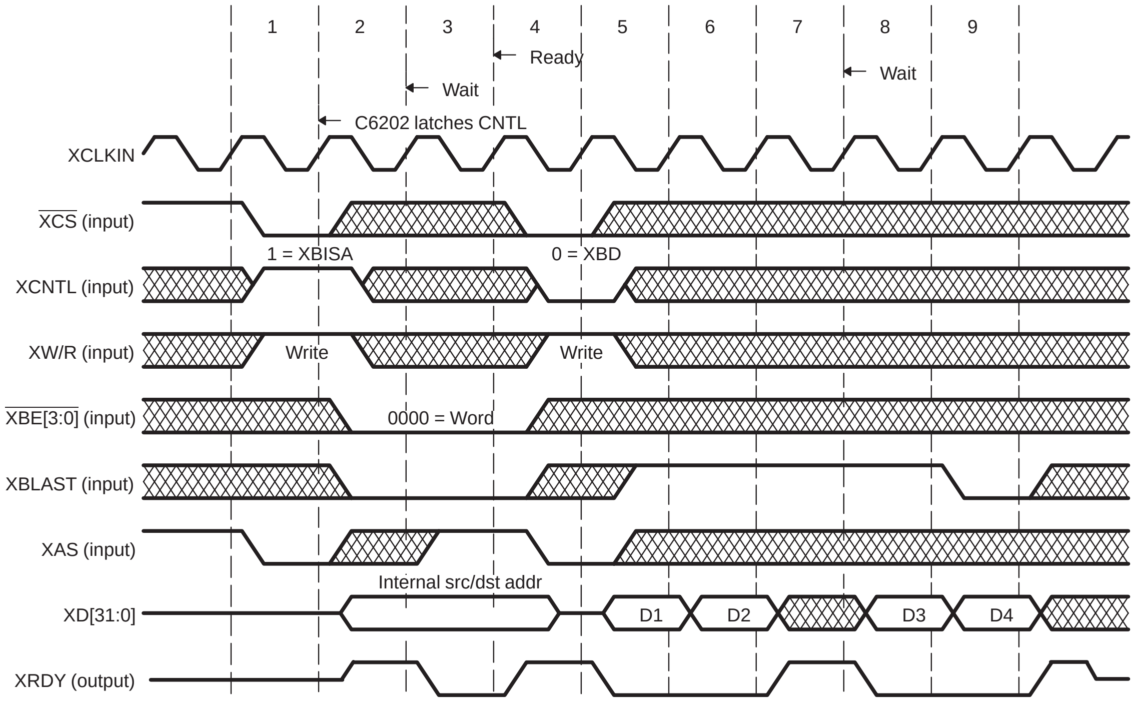
Each access initiated by the external host can be broken up into distinct categories:

- **Address phase (Ta):** During the address phase, the ’C6x is selected with the $\overline{\text{XCS}}$ input and the address phase is started with a low pulse on the $\overline{\text{XAS}}$ signal. During this phase, the ’C6x determines if the external master is doing a read or write cycle (XW/R input) and which expansion bus register is being accessed (via the XCNTL input).
- **Wait/data phase (Tw/Td):** Immediately after the address phase, the transaction enters either the wait phase or data phase. For a read cycle, there is at least one wait phase before the ’C6x presents the data to the external host. This is controlled via the XRDY output of the ’C6x. If the XRDY signal is high, this indicates to the external host that the ’C6x is not ready to receive data for a write, or is not ready to present data for a read, and is in the wait phase. The data phase is entered when the ’C6x asserts XRDY signal, indicating that read data should be latched by the external host or that write data has been latched by the ’C6x.
- **Recovery phase (Tr):** The recovery phase is entered after final data phase of a burst access or after the data phase of a single access. When the ’C6x is a slave, if the external master has a multiplexed address/data bus, it is recommended that the external master insert at least one recovery phase between a read data phase and a subsequent address phase in order to avoid potential bus contention.

Burst Write Transfer

The timing diagram shown in Figure 8–22 can be referenced for a visual description of the steps required to complete a burst write initiated by an external host and throttled by the XRDY signal.

Figure 8–22. The Expansion Bus Master Writes a Burst of Data to the TMS320C6202



The boot configuration for XBLAST and XW/R: BLPOL = 0 and RWPOL = 0. See Table 8–16 for more details.

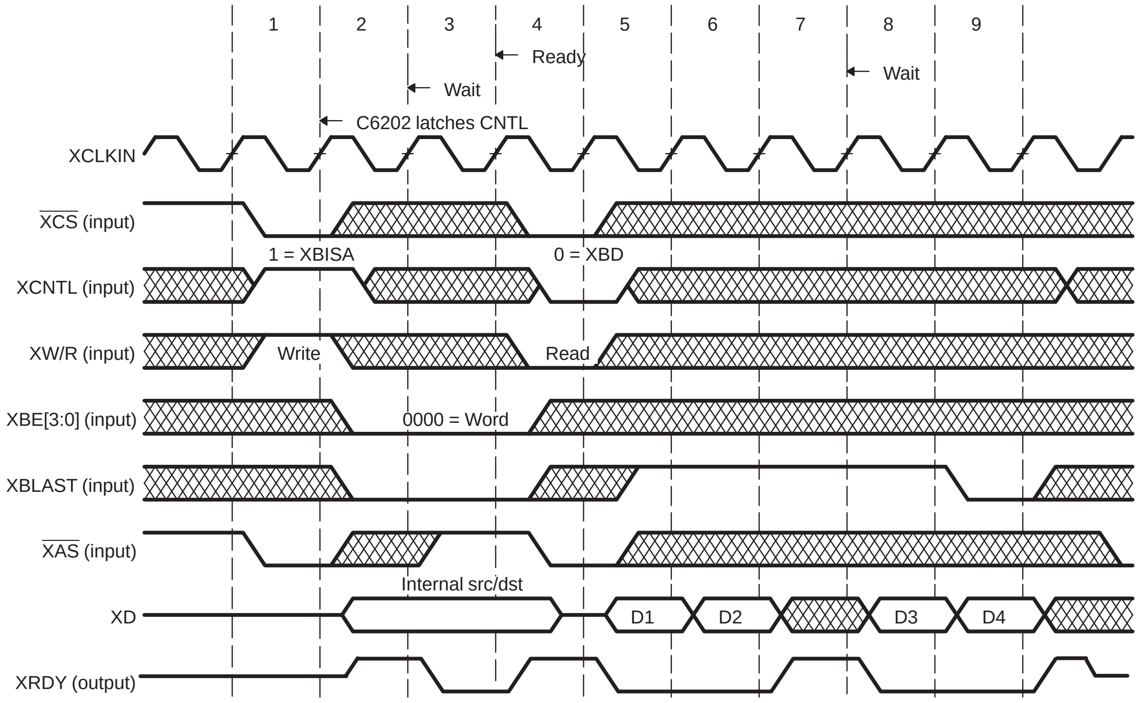
The step by step description of the events marked above the waveforms in Figure 8–22 follows:

1) The $\overline{\text{XCS}}$, $\overline{\text{XAS}}$ and XCNTL signals are low, low, and high respectively, indicating XBISA register as the destination for the following transaction. The XW/R is high specifying that a write access is taking place.
2) The ’C6202 begins driving the XRDY output in response to a transfer initiated by the external host. A high XRDY indicates that the ’C6202 is not ready.
3) The data is written to the XBISA register when the ’C202 asserts the XRDY output low.
4) The $\overline{\text{XAS}}$ and XCNTL signals are both low (and $\overline{\text{XCS}}$ is low), indicating XBD register as the destination for the following transaction. The XW/R is high specifying that a write access is taking place.
5) The expansion bus master presents the valid data. The data is written to the XBD register on the rising edge of the XCLKIN when XRDY is active-low.
6) Same as 5.
7) The ‘C6202 is not ready to accept next data, which is indicated by XRDY high.
8) Same as 5.
9) The expansion bus master indicates that the last write transaction is taking place by asserting the XBLAST signal. The data is written to the XBD register on the rising edge of the XCLKIN.

Burst Read Transfer

The timing diagram shown in Figure 8–23 can be referenced for a visual description of the steps required to complete a burst read initiated by an external host and throttled by the XRDY signal.

Figure 8–23. The Bus Master Reads a Burst of Data From the TMS320C6202



The boot configuration for XBLAST and XRW: BLPOL = 0 and RWPOL = 0. See Table 8–16 for more details.

The step by step description of the events marked above the waveforms in Figure 8–23 follows:

1) The $\overline{\text{XCS}}$, $\overline{\text{XAS}}$ and XCNTL signals are low, low and high respectively, indicating XBISA register as the destination for the following transaction. The XW/R is high specifying that a write access is taking place.
2) The ’C6202 begins driving the XRDY output in response to a transfer initiated by the external host. A high XRDY indicates that the ’C6202 is not ready.
3) The data is written to the XBISA register when the ’C6202 asserts the XRDY output low.
4) The $\overline{\text{XAS}}$ and XCNTL signals are both low (and $\overline{\text{XCS}}$ is low), indicating XBD register as the destination for the following transaction. The XW/R is low specifying that a read access is taking place.
5) The ‘C6202 presents the valid data, and drives XRDY low.
6) Same as 5.
7) The ‘C6202 is not ready to present the next data, which is indicated by XRDY high.
8) Same as 5.
9) The expansion bus master indicates that the last read transaction is taking place by asserting the XBLAST signal.

8.5.3 Asynchronous Host Port Mode

This mode is slave only, it uses a 32-bit data path, and it is similar to the HPI on the ‘C6201. The asynchronous host port mode is used to interface to asynchronous microprocessor buses.

A list of the signals when the expansion bus operates in the asynchronous host port mode is given in Table 8–17.

Table 8–17. Expansion Bus Pin Description (Asynchronous Host Port Mode)

Signal Symbol	Signal Type	Signal Count	Signal Name	Signal Function
$\overline{\text{XCS}}$	I	1	Chip Select	Selects the ‘C6202 as a target of an external master.
XD[31:0]	I/O/Z	32	Data Bus	
$\overline{\text{XBE[3:0]}}$	I	4	Byte Enables	Functionality of these signals is the same as on the ‘C6201 HPI (during a read XBE do not matter). During a write: $\overline{\text{BE3}}$ byte enable 3– XD[31:24] $\overline{\text{BE2}}$ byte enable 2– XD[23:16] $\overline{\text{BE1}}$ byte enable 1– XD[15:8] $\overline{\text{BE0}}$ byte enable 0– XD[7:0]
XCNTL	I	1	Control Signal	This signal selects between XBD and XBISA register. XCNTL=0, access is made to the XBD register XCNTL=1, access is made to the XBISA register
XW/R	I	1	Read/Write	Polarity of this signal is configured during boot.
XRDY	O/Z	1	Ready Out	Ready signal indicates normally not ready condition. This signal is always driven in asynch host mode when the ’C6202 does not own the bus.

The XCNTL signal selects which internal register the host is accessing. The state of this pin selects if access is made to the expansion bus internal slave address (XBISA) register or, expansion bus data (XBD) register.

If the expansion bus host port operates in the asynchronous mode, every transaction initiated by the host on the expansion bus is a two step process. The host first has to set the XBISA register, and then transfer the data to/from the address pointed to by the XBISA register. The data transfer can take place with or without auto-incrementing the XBISA register. Whether or not the XBISA gets auto-incremented is determined by AINC bit-field in bit one of the XBISA register.

In order to read/write from the ‘C6202 memory spaces, the host must follow the following sequence:

1) Host writes address to the XBISA register, and sets AINC accordingly in bit one of XBISA.

2) Host reads/writes to/from the address specified by the XBISA register. Read or write is dictated by the XW/R signal. The XBISA register is auto-incremented or not depending on what is written to the AINC bit during step 1.

If the expansion bus host port is configured to operate in asynchronous mode the $\overline{\text{XCS}}$ signal is used for four purposes:

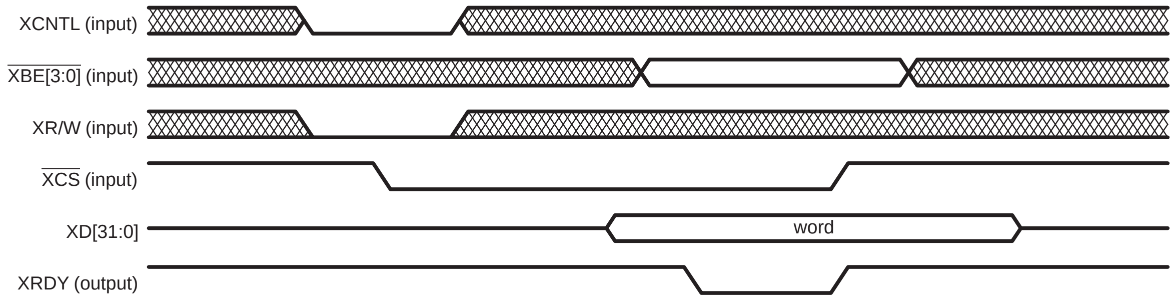
3) To select the expansion bus host port as a target of an external master.
4) On a read, the falling edge of $\overline{\text{XCS}}$ initiates read accesses.
5) On a write, the rising edge of $\overline{\text{XCS}}$ initiates write accesses.
6) The $\overline{\text{XCS}}$ falling edge latches expansion bus host port control inputs including: XW/R and XCNTL.

The XRDY signal of the ’C6202 functions differently than the ‘C6201 HPI READY signal. The XRDY signal indicates normally not ready condition (active low READY signal is internally OR-ed with $\overline{\text{XCS}}$ signal in order to obtain XRDY).

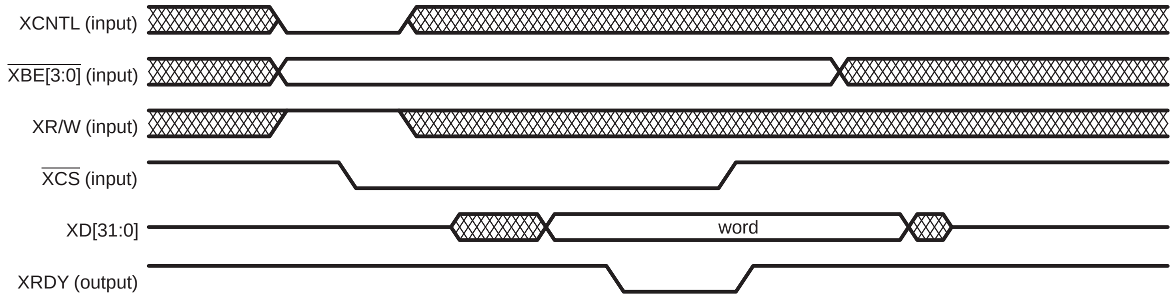
Read and write timing diagrams for asynchronous the expansion bus host port operation in the asynchronous mode are shown in Figure 8–24.

Figure 8–24. Timing Diagrams for Asynchronous Host Port Mode of the Expansion Bus

Asynchronous Host Port Write Timing


XCNTL (input)
$\overline{\text{XBE[3:0]}}$ (input)
XR/W (input)
$\overline{\text{XCS}}$ (input)
XD[31:0]
word
XRDY (output)

Asynchronous Host Port Read Timing


XCNTL (input)
$\overline{\text{XBE[3:0]}}$ (input)
XR/W (input)
$\overline{\text{XCS}}$ (input)
XD[31:0]
word
XRDY (output)

8.6 Expansion Bus Arbitration

Two signals, XHOLD and XHOLDA, are provided for bus arbitration. The internal bus arbiter is disabled or enabled depending on the value on the expansion data bus during reset.

The XARB bit in the expansion bus global control register indicates if the internal bus arbiter is enabled or disabled. This is shown in Table 8–18.

Table 8–18. XARB Bit Value and XHOLD/XHOLDA Signal Functionality

XARB Bit (Read Only)	XHOLD	XHOLDA
0 (Indicates disabled internal bus arbiter)	Output	Input
1 (Indicates enabled internal bus arbiter)	Input	Output

If the internal bus arbiter is enabled, the ‘C6202 wakes up from reset as the bus master. If internal bus arbiter is disabled, the ‘C6202 wakes up from reset as the bus slave. The DMA controller releases the expansion bus between frames if a DMA block transfer is in progress. When the ’C6202 releases the expansion bus, the host port signals become tristated, except for the I/O port signals ($\overline{\text{XWE}}$/XWAIT, $\overline{\text{XOE}}$, $\overline{\text{XRE}}$, $\overline{\text{XCE[3:0]}}$, and XFCLK) which are not affected.

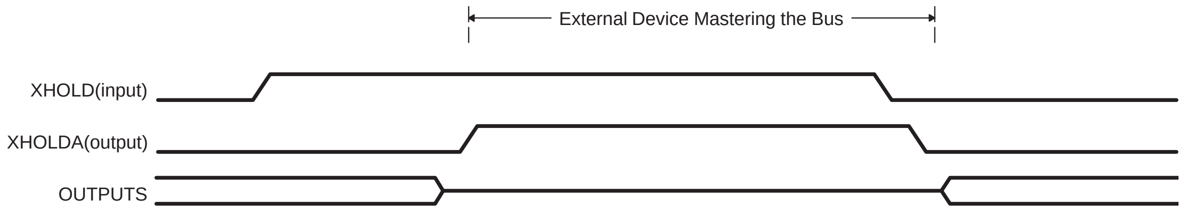
8.6.1 Internal Bus Arbiter Enabled

In this mode the ‘C6202 owns the expansion bus by default. The ‘C6202 wakes up from reset as the master of the expansion bus, and all other devices must request the bus from ‘C6202. This mode is preferred when connecting one ‘C6202 to a PCI interface chip.

When the TMS320C6202 owns the expansion bus, both XHOLD (input) and XHOLDA (output) are low. XHOLD is asserted by an external device to request use of the expansion bus. The ‘C6202 asserts XHOLDA when bus request is granted. The expansion bus is not granted unless requested by XHOLD.

Figure 8–25 illustrates XHOLD and XHOLDA functionality when the internal bus arbiter is enabled. In this mode the DSP grants the expansion bus to the requester only if no internal transfer requests to the expansion bus are pending.

Figure 8–25. Timing Diagrams for Bus Arbitration–XHOLD/XHOLDA (Internal Bus Arbiter Enabled).



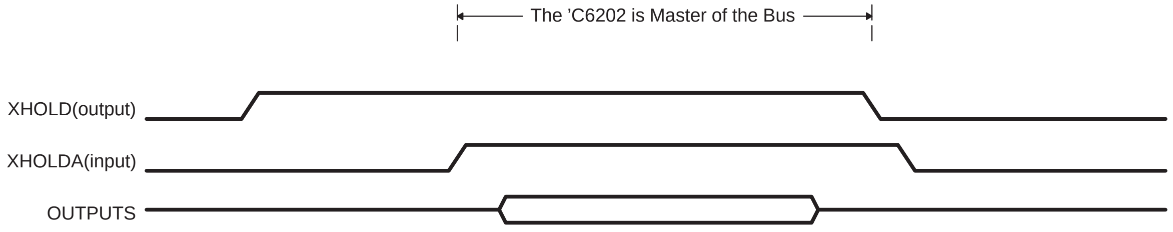
8.6.2 Internal Bus Arbiter Disabled

In this mode, the ‘C6202 acts as slave on the expansion bus by default. This mode is preferred if the ‘C6202 is interfacing to an external host, or if multiple ‘C6202 are connected to a PCI interface chip.

When the ’C6202 owns the expansion bus, both XHOLD (output) and XHOLDA (input) are high. To request the expansion bus (for example to access a FIFO) the ‘C6202 asserts XHOLD. The external expansion bus arbiter asserts XHOLDA when control is granted. The expansion bus should not be granted to the ’C6202 unless requested by XHOLD.

Figure 8–26 illustrates XHOLD and XHOLDA functionality in this mode.

Figure 8–26. Timing Diagrams for Bus Arbitration XHOLD/XHOLDA (Internal Bus Arbiter Disabled)



When the internal bus arbiter is disabled (XARB = ’0’) and the expansion bus master transfer is initiated by writing to the start bit field of the XBHC register, the DSP asserts its XHOLD request. If the host initiates a transfer to the DSP instead of granting the DSP access to the expansion bus, the DSP drops its XHOLD request, as shown in Figure 8–27.

The DSP drops the bus request only if the pending request is for a transfer to the expansion bus host port. The DSP will reassert the bus request for pending master transfers after the host completes its transfer (see Figure 8–27). For more detail see Table 8–19.

Figure 8–27. XHOLD Timing When the External Host Starts a Transfer to DSP Instead of Granting the DSP Access to the Expansion Bus(Internal Bus Arbiter Disabled)

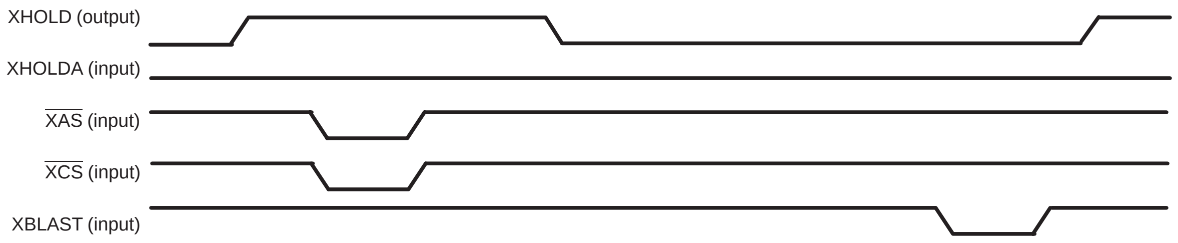

XHOLD (output)
XHOLDA (input)
$\overline{\text{XAS}}$ (input)
$\overline{\text{XCS}}$ (input)
XBLAST (input)

Table 8–19 shows possible scenarios that can happen when the internal bus arbiter is disabled (XARB =0).

Table 8–19. Possible Expansion Bus Arbitration Scenarios (Internal Bus Arbiter Disabled)

XARB = '0'			
XBOFF asserted	**Current External Host Activity**	**Current DSP state**	**Actions**
N/A	Host transfer to the expansion bus in progress	DMA request to expansion bus IO port pending	❑ If the DMA request comes before or at the same time when the host started the transfer, the DSP asserts the XHOLD and keep it asserted during the host transfer. ❑ If the DMA request came after the host started the transfer, the DSP waits for the host transfer to complete and then asserts XHOLD.
		DMA request to expansion bus IO port, and aux. DMA requests are pending	After the DSP gets the expansion bus the pending auxiliary DMA request is executed first (since for the expansion bus, the aux. DMA channel always has priority over the other DMA channels). After the auxiliary DMA transfer is completed, the DSP starts the DMA transfer and does not drop the XHOLD between these two transfers.
		Aux. DMA request pending	❑ If the auxiliary DMA request comes prior to the host starting the transfer, the DSP asserts the XHOLD and keeps it asserted until the host starts the transfer. Once the host starts the transfer, the DSP drops the request (see Figure 8–24). The DSP re-asserts the XHOLD after the host completes the transfer. ❑ If the auxiliary DMA request comes after the host is started the transfer, the DSP waits for the host transfer to complete and asserts the XHOLD.

Table 8–19. Possible Expansion Bus Arbitration Scenarios (Internal Bus Arbiter Disabled)

XARB = ’0’			
XBOFF asserted	**Current External Host Activity**	**Current DSP state**	**Actions**
NO	NONE	DMA request to expansion Bus IO port pending	The DSP asserts the XHOLD, and once it gets the expansion bus the transfer starts.
		DMA request to expansion bus IO port, and auxiliary DMA requests are pending	After the DSP gets the expansion bus the pending auxiliary DMA request is executed first (since for the expansion bus, the auxiliary DMA channel always has priority over the rest of the DMA channels). After the auxiliary DMA transfer is completed, the DSP will start the DMA transfer and does not drop the XHOLD be tween these two transfers.
		Aux. DMA request pending	The DSP asserts the XHOLD, and once it gets the expansion bus the transfer starts.
YES	N/A	DMA transfer to expansion bus IO port in progress	XBOFF is ignored if a DMA transfer to the expansion bus IO port is in progress.
		Aux. DMA transfer in progress	The DSP releases ownership of the expansion bus as soon as possible. After that, the DSP requests the expansion bus to complete the transfer interrupted by the XBOFF.
		Aux. DMA transfer in progress, and DMA request to expansion bus IO port pending	The DSP stops the current auxiliary DMA transfer in progress, and starts executing the pending DMA transfer to the expansion bus IO port. After the pending DMA transfer is completed, the DSP releases the expansion bus to the external device. Some time afterwards, the DSP requests the expansion bus to complete the transfer interrupted by the XBOFF.

8.6.3 Expansion Bus Requestor Priority

For the expansion bus of the ‘C6202, the auxiliary DMA channel is always given the highest priority, followed by the standard DMA priority (DMA0 highest).

Priority	Description
Highest	Auxiliary channel
	DMA0
	DMA1
	DMA2
Lowest	DMA3

In many situations the priority between the auxiliary channel and the standard DMA channels is first come first serve, because the auxiliary channel cannot preempt the standard DMA channels during a frame transfer and the standard DMA channels cannot preempt the auxiliary channel. The standard DMA channels can preempt each other.

The auxiliary channel can only acquire the bus between DMA frames or if no other DMA activity is occurring. For example, if an unsynchronized DMA transfer is set up to perform 4 frames of 32 elements each, and an auxiliary transfer becomes pending, either by an external host asserting the XHOLD request signal if the internal arbiter is enabled or by the ’C6202 attempting to begin a master transfer by writing to the start bits of the XBHC register(internal arbiter enabled or disabled), the auxiliary request will be ignored during the frame transfer to the expansion memory. After the first frame, however, the auxiliary request is recognized and the DMA transfer to the expansion memory stops to allow the host transfer to begin.

To allow host transfers sufficient access to the expansion bus, DMA transactions should be set up so that the frame length is as short as possible. The size of frame transfers to the expansion bus I/O port define the longest amount of time that host transactions can be blocked from accessing the expansion buses.

8.7 Boot Configuration Control via Expansion Bus

The polarity of read/write XW/R and XBLAST control signals on the expansion bus is determined during boot using pull up/pull down resistors on the XD[31:0] pins of the expansion bus. Pull up/pull down resistors on the expansion bus are used for boot mode selection and to enable/disable internal bus arbiter, to define expansion bus host port mode to define memory type used in each expansion bus memory space and to define FIFO mode. All expansion data pins, XD[31:0], should be configured by pull-up/pull-down resistors. Reserved fields should be pulled-down. Detailed description of boot configuration is shown in Figure 8–28 and Table 8–20.

Figure 8–28. Expansion Bus Boot Configuration via Pull Up/Pull Down Resistors on XD[31:0]

31	30 29 28	27	26 25 24	23	22 20	19	18 16
rsvd	MTYPE XCE3	rsvd	MTYPE XCE2	rsvd	MTYPE XCE1	rsvd	MTYPE XCE0

15 14	13	12	11	10	9	8	7 5	4 0
Reserved	BLPOL	RWPOL	HMOD	XARB	FMOD	LEND	Reserved	BOOTMODE

Table 8–20. Description of Expansion Bus Boot Configuration via Pull Up/Pull Down Resistors on XD[31:0]

Field	Description
MTYPE0/1/2/3	Memory type
	MTYPE=010b: 32-bit wide asynchronous interface
	MTYPE=101b: 32-bit wide FIFO interface
	MTYPE=other: reserved
BLPOL	Determines polarity of the XBLAST signal when the DSP is a slave on the expansion bus.
	BLPOL=0: XBLAST is active low.
	BLPOL=1: XBLAST is active high.
	When the DSP initiates a transfer on the expansion bus XBLAST is always active low.
RWPOL	Determines polarity of expansion bus read/write signal.
	RWPOL=0: X$\overline{R}$/W. Write is active-high.
	RWPOL=1, XR/$\overline{W}$. Read is active-high.
HMOD	Host mode (status in XB HPIC)
	HMOD = 0: external host interface operates in asynchronous slave mode.
	HMOD = 1: external host interface is in synchronous master/slave mode.
XARB	Expansion bus arbiter (status in XBGC)
	XARB = 0: Internal expansion bus arbiter is disabled
	XARB = 1: Internal expansion bus arbiter is enabled.
FMOD	FIFO mode (status in XBGC)
	FMOD = 0: Glue is used for FIFO read interface in all XCE spaces operating in FIFO mode. XOE can be used in all XCE spaces
	FMOD = 1: XOE is reserved for use only in $\overline{XCE3}$ for FIFO read mode. XOE is disabled in all other XCE spaces.
LEND	Little endian mode
	LEND = 0: system operates in big endian mode
	LEND = 1: system operates in little endian mode
BOOTMODE[4:0]	Dictates the boot-mode of the device, including host port boot, ROM boot, memory map selection. For a complete list of boot-modes, see Section 10, TMS320C6000 Boot Modes.

Chapter 9

External Memory Interface

This chapter describes the external memory interface used by the CPU to access off-chip memory. This chapter also describes the EMIF control registers and their fields, and it explains how to reset the EMIF. Various memory interfaces are described, along with diagrams showing the connections between the EMIF and each supported memory type.

Topic **Page**

9.1 Overview

The external memory interfaces (EMIFs) of the TMS320C6000 devices support a glueless interface to a variety of external devices, including:

- Synchronous-burst SRAM (SBSRAM)
- Synchronous DRAM (SDRAM)
- Asynchronous devices, including SRAM, ROM, and FIFOs
- An external shared-memory device

The TMS320C6201/C6202/C6701 EMIF services requests of the external bus from four requesters:

- The on-chip program memory controller that services CPU program fetches
- The on-chip data memory controller that services CPU data fetches
- The on-chip DMA controller
- An external shared-memory device controller

If multiple requests arrive simultaneously, the EMIF prioritizes them and performs the necessary number of operations. A block diagram of the TMS320C6201/C6202/C6701 is shown in Figure 9–1, and the signals shown there are summarized in Table 9–1.

The ’C6211/C6711 services requests of the external bus from two requestors:

- An enhanced direct-memory access (EMDA) controller
- An external shared-memory device controller

A block diagram of the TMS320C6201/C6202/C6701 is shown Figure 9–2.

Figure 9–1. External Memory Interface in the TMS320C6201/C6202/C6701 Block Diagram

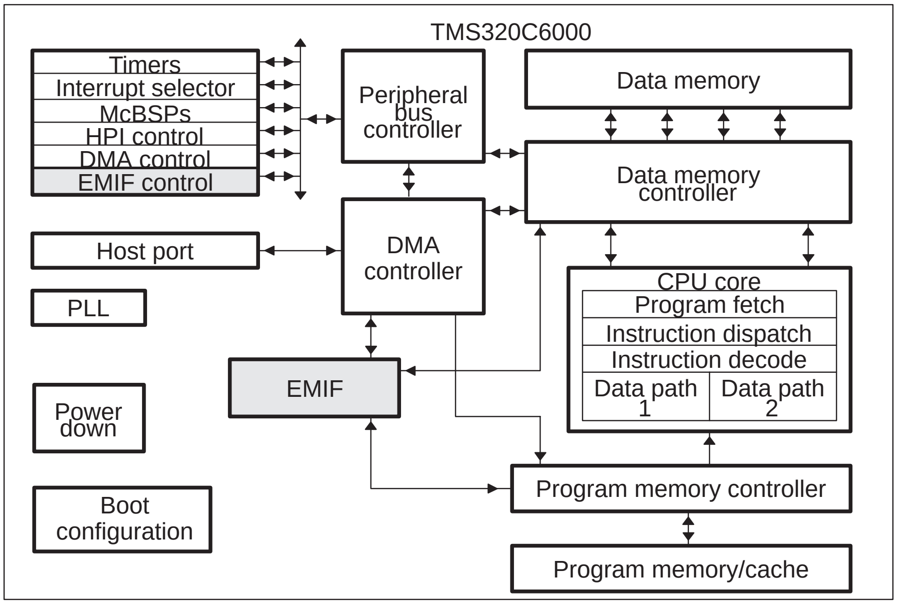


Figure 9–2. External Memory Interface in the TMS320C6211/C6711 Block Diagram

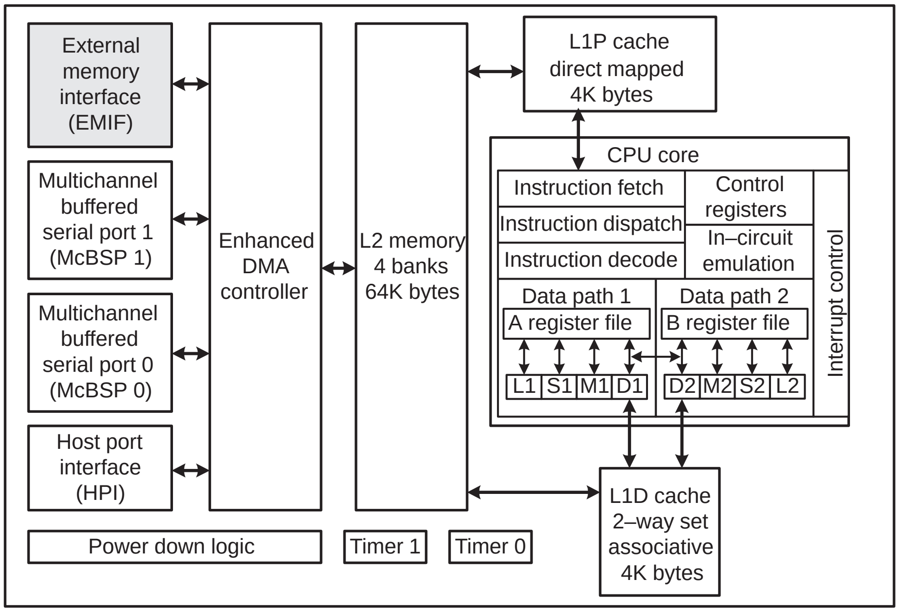
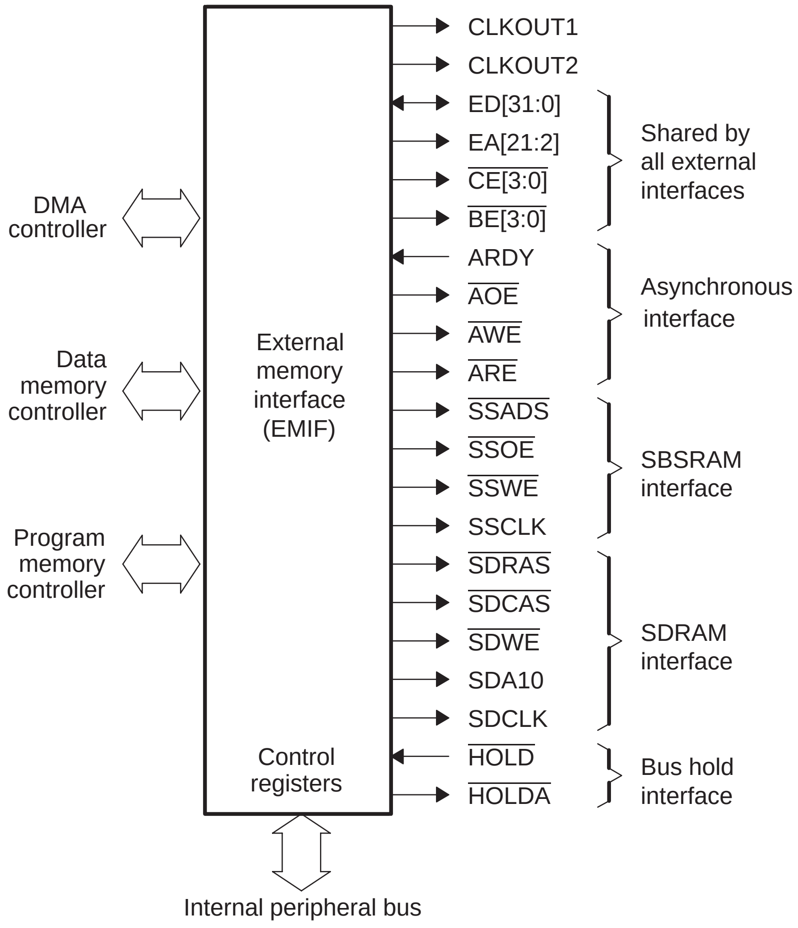
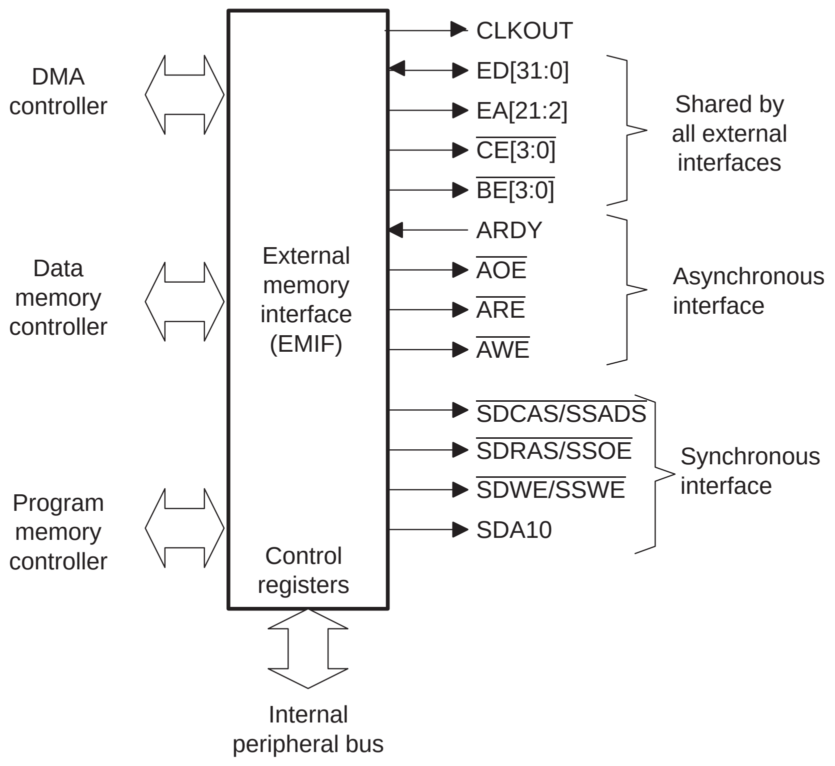

External memory interface (EMIF)
Multichannel buffered serial port 1 (McBSP 1)
Multichannel buffered serial port 0 (McBSP 0)
Host port interface (HPI)
Enhanced DMA controller
L2 memory 4 banks 64K bytes
L1P cache direct mapped 4K bytes
CPU core
Instruction fetch
Instruction dispatch
Instruction decode
Control registers
In–circuit emulation
Interrupt control
Data path 1
A register file
Data path 2
B register file
L1
S1
M1
D1
D2
M2
S2
L2
L1D cache 2–way set associative 4K bytes
Power down logic
Timer 1
Timer 0

Figure 9–3. TMS320C6201/C6701 External Memory Interface



The EMIF signals of the ’C6202 are shown in Figure 9–4. The ’C6202 has combined the SDRAM and SBSRAM signals, such that only one of these two memory types can be used in a system. These memories run off CLKOUT2, which is equal to 1/2x the CPU clock rate.

Figure 9–4. TMS320C6202 External Memory Interface


DMA controller
Data memory controller
Program memory controller
External memory interface (EMIF)
Control registers
Internal peripheral bus
CLKOUT
ED[31:0]
EA[21:2]
$\overline{\text{CE}}$[3:0]
$\overline{\text{BE}}$[3:0]
Shared by all external interfaces
ARDY
$\overline{\text{AOE}}$
$\overline{\text{ARE}}$
$\overline{\text{AWE}}$
Asynchronous interface
$\overline{\text{SDCAS/SSADS}}$
$\overline{\text{SDRAS/SSOE}}$
$\overline{\text{SDWE/SSWE}}$
SDA10
Synchronous interface

The EMIF signals of the ’C6211/C6711 are shown in Figure 9–5. The ’C6211/C6711 has the following features:

- ❑ The ’C6211/C6711 EMIF requires that an external clock source (ECLKIN) be provided by the system. The ECLKOUT signal is produced internally (based on ECLKIN). All of the memories interfacing with the ’C6211/C6711 should operate off of ECLKOUT. If desired, the CLKOUT2 output can be routed back to the ECLKIN input.
- ❑ The synchronized memory interfaces use a four-word burst length which is optimized for the two-level cache architecture.
- ❑ The SDRAM interface is flexible, allowing interfaces to a wide range of SDRAM configurations.
- ❑ The SDA10 pin has been removed. Address pin EA[12] serves the function of the SDA10 pin for the SDRAM memories.

Figure 9–5. TMS320C6211/C6711 External Memory Interface

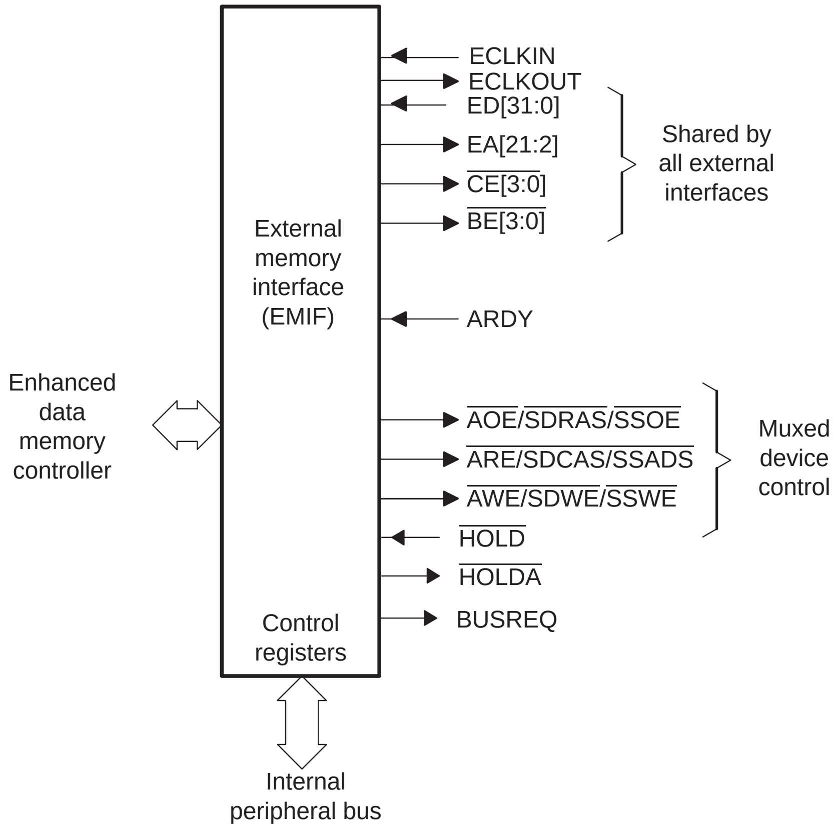

ECLKIN
ECLKOUT
ED[31:0]
EA[21:2]
$\overline{\text{CE}[3:0]}$
$\overline{\text{BE}[3:0]}$
Shared by all external interfaces
External memory interface (EMIF)
ARDY
Enhanced data memory controller
$\overline{\text{AOE}}/\overline{\text{SDRAS}}/\overline{\text{SSOE}}$
$\overline{\text{ARE}}/\overline{\text{SDCAS}}/\overline{\text{SSADS}}$
$\overline{\text{AWE}}/\overline{\text{SDWE}}/\overline{\text{SSWE}}$
Muxed device control
$\overline{\text{HOLD}}$
$\overline{\text{HOLDA}}$
BUSREQ
Control registers
Internal peripheral bus

Table 9–1. EMIF Signal Descriptions

6201	6701	6202	6211	Pin	(I/O/Z)	Description
✓	✓	✓	✓	CLKOUT1	O	Clock output. Runs at the CPU clock rate.
✓	✓	✓	✓	CLKOUT2	O	Clock output. Runs at 1/2 the CPU clock rate. Used for synchronous memory interface on ’C6202
			✓	BUSREQ	O	Active high bus request signal
			✓	ECLKOUT	O	EMIF clock output. All EMIF I/O are clocked relative to ECLKOUT.
			✓	ECLKIN	I	EMIF clock input. Must be provided by system.
✓	✓	✓	✓	ED[31:0]	I/O/Z	Data I/O. 32-bit data input/output from external memories and peripherals
✓	✓	✓	✓	EA[21:2]	O/Z	External address output. Drives bits 21–2 of the byte address.
✓	✓	✓	✓	$\overline{\text{CE0}}$	O/Z	Active low chip select for memory space CE0
✓	✓	✓	✓	$\overline{\text{CE1}}$	O/Z	Active low chip select for memory space CE1
✓	✓	✓	✓	$\overline{\text{CE2}}$	O/Z	Active low chip select for memory space CE2
✓	✓	✓	✓	$\overline{\text{CE3}}$	O/Z	Active low chip select for memory space CE3
✓	✓	✓	✓	$\overline{\text{BE[3:0]}}$	O/Z	Active low byte enables. Individual bytes and halfwords can be selected for both read and write cycles. Decoded from two LSBs of the byte address.
✓	✓	✓	✓	ARDY	I	Ready. Active low asynchronous ready input used to insert wait states for slow memories and peripherals.
✓	✓	✓	M	$\overline{\text{AOE}}$	O/Z	Active low output enable for asynchronous memory interface
✓	✓	✓	M	$\overline{\text{AWE}}$	O/Z	Active low write strobe for asynchronous memory interface
✓	✓	✓	M	$\overline{\text{ARE}}$	O/Z	Active low read strobe for asynchronous memory interface
✓	✓	M	M	$\overline{\text{SSADS}}$	O/Z	Active low address strobe/enable for SBSRAM interface
✓	✓	M	M	$\overline{\text{SSOE}}$	O/Z	Output buffer enable for SBSRAM interface
✓	✓	M	M	$\overline{\text{SSWE}}$	O/Z	Active low write enable for SBSRAM interface
✓	✓			SSCLK	O/Z	SBSRAM interface clock. Programmable to either the CPU clock rate or half of the CPU clock rate.
✓	✓	M	M	$\overline{\text{SDRAS}}$	O/Z	Active low row strobe for SDRAM memory interface
✓	✓	M	M	$\overline{\text{SDCAS}}$	O/Z	Active low column strobe for SDRAM memory interface
✓	✓	M	M	$\overline{\text{SDWE}}$	O/Z	Active low write enable for SDRAM memory interface
✓	✓	✓		SDA10	O/Z	SDRAM A10 address line. Address line/autoprecharge disable for SDRAM memory.
✓	✓			SDCLK	O/Z	SDRAM interface clock. Runs at 1/2 the CPU clock rate. Equivalent to CLKOUT2.
✓	✓	✓	✓	$\overline{\text{HOLD}}$	I	Active low external bus hold (3-state) request
✓	✓	✓	✓	$\overline{\text{HOLDA}}$	O	Active low external bus hold acknowledge

† ’M’ indicates a multiplexed output signal

9.2 Resetting the EMIF

A hardware reset using the $\overline{\text{RESET}}$ pin on the device forces all register values to their reset state. During reset, all outputs are driven to their inactive levels, with the exception of the clock outputs (SDCLK, SSCLK, CLKOUT1, and CLKOUT2). CLKOUT2, SSCLK, and SDCLK are driven high or low during active $\overline{\text{RESET}}$. CLKOUT1 continues clocking unless the values on the PLL configuration pins are changed.On the ’C6211, ECLKIN should be provided during reset in order to drive EMIF signals to the correct reset values. ECLKOUT will continue to clock as long as ECLKIN is provided.

9.3 EMIF Registers

Control of the EMIF and the memory interfaces it supports is maintained through memory-mapped registers within the EMIF. The memory-mapped registers are listed in Table 9–2.

Table 9–2. EMIF Memory-Mapped Registers

Byte Address	Name
0180 0000h	EMIF global control
0180 0004h	EMIF CE1 space control
0180 0008h	EMIF CE0 space control
0180 000Ch	Reserved
0180 0010h	EMIF CE2 space control
0180 0014h	EMIF CE3 space control
0180 0018h	EMIF SDRAM control
0180 001Ch	EMIF SDRAM timing register

9.3.1 Global Control Register

The EMIF global control register (shown in Figure 9–6 and summarized in Table 9–3) configures parameters common to all the CE spaces.

Figure 9–6. EMIF Global Control Register Diagram

31–16
Reserved
R, +0000 0000 0000 0000 00

15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
Rsv	Rsv	Rsv	BUS REQ†	ARDY	HOLD	HOLDA	NOHOLD	SDCEN‡	SSCEN‡	CLK1EN	CLK2EN§	SSCRT§‡	RBTR8‡	MAP‡
R,+0	RW,+0	R,+11	R, +0	R, +x	R, +x	R, +0	RW, +0	RW, +1	RW, +1	RW, +1	RW, +1	RW, +0	RW, +0	R, +x

† Field exists *only* in ’C6211/C6711
‡ Fields do not exist in ’C6211/C6711
§ Fields do not exist in ’C6202.

Table 9–3. EMIF Global Control Register Field Descriptions

Field	Description

BUSREQ[†]	BUSREQ = 0; BUSREQ ouput is low.
	BUSREQ = 1; BUSREQ output is high.

ARDY	ARDY = 0: ARDY input is low.
	ARDY = 1: ARDY input is high.

HOLD	HOLD = 0: $\overline{\text{HOLD}}$ input is low.
	HOLD = 1: $\overline{\text{HOLD}}$ input is high.

HOLDA	HOLDA = 0: $\overline{\text{HOLDA}}$ output is low.
	HOLDA = 1: $\overline{\text{HOLDA}}$ output is high.

NOHOLD	External HOLD disable
	NOHOLD = 0: hold enabled
	NOHOLD = 1: hold disabled

SDCEN[‡]	SDCLK enable
	SDCEN = 0: SDCLK held high
	SDCEN = 1: SDCLK enabled to clock
	
	There is no SDCLK on the ’C6211/C6711. All external memories run off the EMIF external clock, ECLKIN/ECLKOUT. SDCEN enables CLKOUT2 on the ’C6202 if SDRAM is used in the system (specified by the MTYPE field in in the CE space control register).

SSCEN[‡]	SSCLK enable
	SSCEN = 0: SSCLK held high
	SSCEN = 1: SSCLK enabled to clock
	
	There is no SSCLK on the ’C6211/C6711. All external memories run off of the EMIF external clock, ECLKIN/ECLKOUT. SSCEN enables CLKOUT2 on the ’C6202 if SBSRAM is used in the system (specified by the MTYPE field in in the CE space control register).

CLK1EN	CLKOUT1 enable
	CLK1EN = 0: CLKOUT1 held high
	CLK1EN = 1: CLKOUT1 enabled to clock

CLK2EN[§]	CLKOUT2 enable
	CLK2EN = 0: CLKOUT2 held high
	CLK2EN = 1: CLKOUT2 enabled to clock
	
	CLKOUT2 is enabled/disabled using SSCEN/SDCEN on the ’C6202

[†] Field exists *only* in ’C6211/C6711
[‡] Fields do not exist in ’C6211/C6711
[§] Fields do not exist in ’C6202.

Table 9–3. EMIF Global Control Register Field Descriptions (Continued)

Field	Description
SSCRT‡	SBSRAM clock rate select SSCRT = 0: SSCLK runs at 1/2x CPU clock rate SSCRT = 1: SSCLK runs at 1x CPU clock rate There is no SSCLK on the ’C6202. CLKOUT2 is fixed at half the CPU clock. SBSRAM runs at the EMIF clock rate (ECLKIN) on the ’C6211/C6711.
RBTR8‡	Requester arbitration mode RBTR8 = 0: The requester controls the EMIF until a high-priority request occurs. RBTR8 = 1: The requester controls the EMIF for a minimum of eight accesses. ‡All arbitration is performed outside of the EMIF on the ’C6211/C6711.
MAP‡	Map mode, contains the value of the memory map mode of the device MAP = 0: internal memory is used at address 0 MAP = 1: external memory used at address 0 ‡There is only one memory map available on the ’C6211/C6711.

† Field exists *only* in ’C6211/C6711
‡ Fields do not exist in ’C6211/C6711
§ Fields do not exist in ’C6202.

The ’C6202 EMIF registers are similar to those of the ’C6201. Due to the combination of the SDRAM and SBSRAM signals, the user cannot include both SDRAM and SBSRAM in the same system. The EMIF global control register bitfields are modified slightly to reflect this change.

In order to support as many common programming practices as possible between the ’C6201 and ’C6202, SSCEN and SDCEN are still used to enable the memory interface clock, CLKOUT2. If SBSRAM is used in the system, (specified with the MTYPE field in the CEn Control register) then SSCEN enables and disables CLKOUT2. If SDRAM is used, then SDCEN enables and disables CLKOUT2. This is possible since only one synchronous memory type can exist in a given system.

The ’C6211/C6711 has similar EMIF registers to the ’C6201, with the exception that some of the bitfields of the global control register have been removed. The ’C6211/C6711 EMIF global control register contains an additional field BUSREQ.

9.3.2 EMIF CE Space Control Registers

The four CE space control registers, CE0, CE1, CE2, and CE3, are shown in Figure 9–7 and summarized in Table 9–4. These registers correspond to the four CE memory spaces supported by the EMIF. The MTYPE field identifies the memory type for the corresponding CE space. If MTYPE selects SDRAM or SBSRAM, the remaining fields in the register do not have any effect. If an asynchronous type is selected (ROM or asynchronous), the remaining fields specify the shaping of the address and control signals for access to that space. These features are discussed in Section 9.6.

The MTYPE field in the CE space control register should only be set once during system initialization except when CE1 is used for ROM boot mode. In this mode, the CE space can be configured to another asynchronous memory type.

For the ’C6202, only one synchronous memory type is supported at a time. If a CE space is set as a synchronous memory type (SBSRAM or SDRAM), all synchronous memory spaces are changed to the new memory type. For example, if CE2 is configured as SDRAM (MTYPE = 011b), setting CE3 as SBSRAM (MTYPE = 100b) changes CE2 and CE3 to SBSRAM. Changing a CE space memory type to asynchronous memory does not affect the memory type of other CE spaces, and setting a memory space to a synchronous type does not change the type of asynchronous memory spaces.

Figure 9–7. TMS320C6201/C6202/C6701 EMIF CE Space Control Register Diagram

31 28	27 22	21 20	19 16
Write setup	Write strobe	Write hold	Read setup
RW, +1111	RW, +111111	RW, +11	RW, +1111

15 14	13 8	7	6 4	3 2	1 0
Reserved	Read strobe	Rsvd	MTYPE	Reserved	Read hold
R, +00	RW, +111111	R, +0	RW, +010	R, +0	RW, +11

Figure 9–8. TMS320C6211/C6711 EMIF CE Space Control Register

31 28	27 22	21 20	19 16
Write setup	Write strobe	Write hold	Read setup
RW, +1111	RW, +11111	RW, +11	RW, +1111

15 14	13 8	7 4	3	2 0
TA	Read strobe	MTYPE	Write hold MSB	Read hold
RW, +11	RW, +11111	RW, +0010	RW, +0	RW, +011

Table 9–4. EMIF CE Space Control Registers Field Descriptions

Field	Description
Read setup Write setup	Setup width. Number of clock[§] cycles of setup time for address (EA), chip enable ($\overline{\text{CE}}$), and byte enables ($\overline{\text{BE[0-3]}}$) before read strobe or write strobe falls. For asynchronous read accesses, this is also the setup time of $\overline{\text{AOE}}$ before $\overline{\text{ARE}}$ falls.
Read strobe Write strobe	Strobe width. The width of read strobe ($\overline{\text{ARE}}$) and write strobe ($\overline{\text{AWE}}$) in clock[§] cycles
Read hold Write hold	Hold width. Number of clock[§] cycles that address (EA) and byte strobes ($\overline{\text{BE[0-3]}}$) are held after read strobe or write strobe rises. For asynchronous read accesses, this is also the hold time of $\overline{\text{AOE}}$ after $\overline{\text{ARE}}$ rising.

MTYPE[†]	Memory type of the corresponding CE spaces for ’C6201/C6202/C6701
	
	MTYPE = 000b: 8-bit-wide ROM (CE1 only)
	MTYPE = 001b: 16-bit-wide ROM (CE1 only)
	MTYPE = 010b: 32-bit-wide asynchronous interface
	MTYPE = 011b: 32-bit-wide SDRAM (CE0, CE2, CE3 only)
	MTYPE = 100b: 32-bit-wide SBSRAM

MTYPE[‡]	Memory type of the corresponding CE spaces for ’C6211/C6711
	
	MTYPE = 0000b: 8-bit-wide asynchronous interface (previously ROM)
	MTYPE = 0001b: 16-bit-wide asynchronous interface (previously ROM)
	MTYPE = 0010b: 32-bit-wide asynchronous interface
	MTYPE = 0011b: 32-bit-wide SDRAM
	MTYPE = 0100b: 32-bit-wide SBSRAM
	MTYPE = 1000b: 8-bit-wide SDRAM
	MTYPE = 1001b: 16-bit-wide SDRAM
	MTYPE = 1010b: 8-bit-wide SBSRAM
	MTYPE = 1011b: 16-bit-wide SBSRAM
	MTYPE = other: reserved

TA[‡]	Turn around time controls the number of ECLKOUT cycles between a read, and a write, or between reads, to different CE spaces (asynchronous memory types only).[‡]

[†] Applies to TMS320C6201/C6202/C6701
[‡] Applies to TMS320C6211/C6711
[§] Clock cycles are in terms of CLKOUT1 for ’C6201/C6202/C6701, and ECLKOUT for the ’C6211/C6711

The ’C6211/C6711 has a modified version of the CE space control register, in that the MTYPE, write hold, and read hold bit fields have been extended by one bit each. The CE space control register for the ’C6211/C6711 is shown in Figure 9–8. Programmed values refer to ECLKOUT clock cycles, not CLKOUT1 cycles as in the ’C6201,’C6202, and ’C6701.

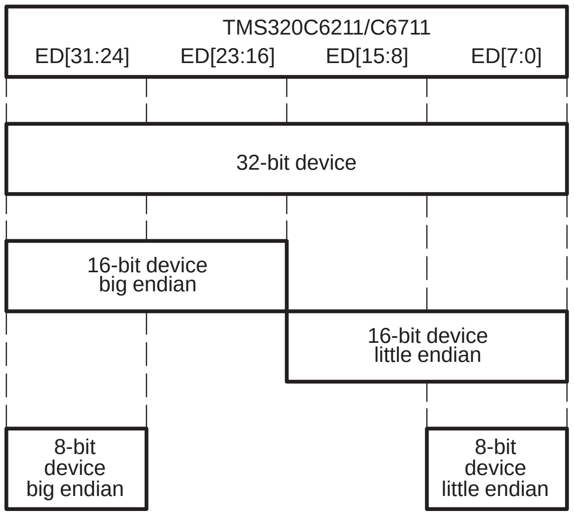
The read hold and write hold fields have been increased by one bit, to allow greater asynchronous configuration possibilities. The MTYPE field has been increased by one bit to allow for 8-, 16-, and 32-bit interface options for all memory types.

The ’C6211/C6711 EMIF supports memory widths of 8-, 16-, and 32-bits, including reads and writes of both big and little endian devices. There is no distinction between ROM and asynchronous interface.

For all memory types, the address is internally shifted to compensate for memory widths of less than 32 bits. The least-significant address bit is always output on external address pin EA[2], regardless of the width of the device. Accesses to 8-bit memories have logical address bit 0 output on EA[2].

Packing and unpacking is automatically performed by the EMIF for word accesses to external memories of less than 32 bits. For a 32-bit write to an 8-bit memory, the data is automatically unpacked into bytes such that the bytes are written to byte address N, N+1, N+2, then N+3. Likewise for 32-bit reads from a 16-bit memory, data is taken from halfword address N then N+1, packed into a 32-bit word, then written to its destination. The byte lane used depends on the endianness of the system as shown in Figure 9–9.

Figure 9–9. TMS320C6211/C6711 Byte Alignment by Endianness



9.3.3 EMIF SDRAM Control Register

The SDRAM control register (shown in Figure 9–10) controls SDRAM parameters for all CE spaces that specify an SDRAM memory type in the MTYPE field of the associated CE space control register. Because the SDRAM control register controls all SDRAM spaces, each space must contain SDRAM with the same refresh, timing, and page characteristics. The fields in this register are shown in Figure 9–10 and Figure 9–11, and described in Table 9–5. These registers should not be modified while accessing SDRAM.

Figure 9–10. TMS320C6201/C6202/C6701 EMIF SDRAM Control Register

31 28	27	26	25	24	23 20	19 16
Reserved	Rsv	SDWID	RFEN	INIT	TRCD	TRP
RW, +000	R,+0	RW, +0	RW, +1	W, +1	RW, +1000	RW, +1000

15 12	11 0
TRC	Reserved
RW, +1111	R, +0000 0000 0000

Figure 9–11. TMS320C6211/C6711 EMIF SDRAM Control Register

31	30	29 28	27 26	25	24	23 20	19 16
Rsv	SDBSZ	SDRSZ	SDCSZ	RFEN	INIT	TRCD	TRP
R,+0	RW, +0	RW, +00	RW, +0	RW, +1	W, +1	RW, +0100	RW, +1000

15 12	11 0
TRC	Reserved
RW, +1111	R, +0000 0000 0000

Table 9–5. EMIF to SDRAM Control Register Field Description

Field	Description
TRC	Specifies the t_{RC} value of the SDRAM TRC = t_{RC} / p[§] – 1
TRP	Specifies the t_{RP} value of the SDRAM in CLKOUT2 cycles TRP = t_{RP} / p[§] – 1
TRCD	Specifies the t_{RCD} value of the SDRAM in CLKOUT2 cycles TRCD = t_{RCD} / p[§] – 1

INIT	Forces initialization of all SDRAM present
	INIT = 0: no effect
	INIT = 1: initialize SDRAM in each CE space configured for SDRAM

RFEN	Refresh enable
	RFEN = 0: SDRAM refresh disabled
	RFEN = 1: SDRAM refresh enabled

SDWID[†]	[†]SDRAM width select SDWID = 0: Each external SDRAM space consists of four 8-bit SDRAMs SDWID = 1: Each external SDRAM space consists of two 16-bit SDRAMs

SDCSZ[‡]	[‡]SDRAM column size
	SDCSZ = 00: 9 column address pins
	SDCSZ = 01: 8 column address pins
	SDCSZ = 10: 10 column address pins
	SDCSZ = 11: reserved

SDRSZ[‡]	[‡]SDRAM column size
	SDCSZ = 00: 11 row address pins
	SDCSZ = 01: 12 row address pins
	SDCSZ = 10: 13 row address pins
	SDCSZ = 11: reserved

SDBSZ[‡]	[‡]SDRAM bank size
	SDBSZ = 0: two banks
	SDBSZ = 1: four banks

[†] Applies to ’C6201/C6202/C6701
[‡] Applies to ’C6211/C6711
[§] p – refers to the EMIF clock period, which is equal to CLKOUT2 period for the ’C6201/C6202/C6701, or ECLKOUT period for the ’C6211/C6711

9.3.4 EMIF SDRAM Timing Register

The SDRAM timing register controls the refresh period in terms of CLKOUT2 cycles for the ’C6201/C6202/C6701 (half of the CPU clock rate), or in terms of ECLKOUT cycles for the ’C6211/C6711. Optionally, the period field can send an interrupt to the CPU. Thus, this counter can be used as a general-purpose timer if SDRAM is not used by the system. The counter field can be read by the CPU. When the counter reaches 0, it is automatically reloaded with the period and an interrupt (SDINT) is sent to the interrupt selector. See section 9.4.3 for more information on SDRAM refresh.

Figure 9–12 and Table 9–6 describe the fields of the SDRAM timing register.

The ’C6211/C6711 can control the number of refreshes performed when the refresh counter expires via the XRFR field. Up to four refreshes can be performed when the refresh counter expires.

Figure 9–12. EMIF SDRAM Timing Register

31–26	25–24	23–12	11–0
Reserved	XRFR‡	COUNTER	PERIOD
R, +0000 00	R, +0† RW,+00‡	R, +0000 1000 0000† R, +0101 1101 1100‡	RW, +0000 1000 0000† RW, +0101 1101 1100‡

† Applies to TMS320C6201/C6202/C6701
‡ Applies to TMS320C6211/C6711 only

Table 9–6. EMIF SDRAM Timing Register Field Descriptions

Field	Description
PERIOD	†Refresh period in CLKOUT2 cycles ‡Refresh period in ECLKOUT cycles
COUNTER	Current value of the refresh counter
XRFR‡	‡Extra refreshes: controls the number of refreshes performed to SDRAM when the refresh counter expires.

† Applies to TMS320C6201/C6202/C6701
‡ Applies to TMS320C6211/C6711 only

9.3.5 TMS320C6211/C6711 SDRAM Extension Register

The SDRAM extension register of the ’C6211/C6711 allows programming of many parameters of SDRAM. The programmability offers two distinct advantages. First, the ’C6211/C6711 can interface to a wide variety of SDRAMs and is not limited to a few configurations or speed characteristics. Second, the ’C6211/C6711 can maintain seamless data transfer from external SDRAM due to features like hidden precharge and multiple open banks. Figure 9–13 shows the SDRAM extension register and Table 9–7 discusses these parameters.

Figure 9–13. TMS320C6211/C6711 SDRAM Extension Register

31 21	20	19 18	17	16 15	14 12	11 10	9	8 7	6 5	4	3 1	0
Rsvd	WR2RD	WR2DEAC	WR2WR	R2WDQM	RD2WR	RD2DEAC	RD2RD	THZP	TWR	TRRD	TRAS	TCL
R, +0	RW,+1	RW,+10	RW,+1	RW,+11	RW,+101	RW,+11	RW,+1	RW,+10	RW,+01	RW,+1	RW,+111	RW,+1

Table 9–7. TMS320C6211/C6711 SDRAM Extension Register Field Descriptions

Field	Description

TCL	Specified Cas latency of the SDRAM in ECLKOUT cycles
	TCL = 0: CAS latency = 2 ECLKOUT cycles
	TCL = 1: CAS latency = 3 ECLKOUT cycles

TRAS	Specifies t_{RAS} value of the SDRAM in ECLKOUT cycles TRAS = t_{RAS} – 1

TRRD	Specifies t_{RRD} value of the SDRAM in ECLKOUT cycles
	TRRD = 0, then T_{RRD} = 2 ECLKOUT cycles
	TRRD = 1, then T_{RRD} = 3 ECLKOUT cycles

TWR	Specifies t_{WR} value of the SDRAM in ECLKOUT cycles TWR = t_{WR} – 1
THZP	Specifies t_{HZP} value of the SDRAM in ECLKOUT cycles THZP = t_{HZP} – 1

RD2RD	Specifies number of cycles between READ to READ command (same CE space) of the SDRAM in ECLKOUT cycles
	RD2RD = 0: READ to READ = 1 ECLKOUT cycle
	RD2RD = 1: READ to READ = 2 ECLKOUT cycle

RD2DEAC	Specifies number of cycles between READ to DEAC/DCAB of the SDRAM in ECLKOUT cycles RD2DEAC = (# of cycles READ to DEAC/DCAB) – 1
RD2WR	Specifies number of cycles between READ to WRITE command of the SDRAM in ECLKOUT cycles RD2WR = (# of cycles READ to WRITE) – 1
R2WDQM	Specifies number of of cycles that BEx signals must be high preceding a WRITE interrupting a READ R2WDQM = (# of cycles BEx high) – 1
WR2WR	Specifies minimum number of cycles between WRITE to WRITE command of the SDRAM in ECLKOUT cycles WR2WR = (# of cycles WRITE to WRITE) – 1
WR2DEAC	Specifies minimum number of cycles between WRITE to DEAC/DCAB command of the SDRAM in ECLKOUT cycles WR2DEAC = (# of cycles WRITE to DEAC/DCAB) – 1
WR2RD	Specifies minimum number of cycles between WRITE to READ command of the SDRAM in ECLKOUT cycles WR2RD = (# of cycles WRITE to READ) – 1

9.4 SDRAM Interface

The TMS320C6201/C6202/C6701 EMIF supports 2-bank, 16M-bit SDRAM and 4 bank, 64M-bit SDRAM, providing an interface to high-speed and high-density memory. The EMIF supports the SDRAM commands shown in Table 9–8. The 16M-bit and 64M-bit SDRAM interfaces are shown in Figure 9–14 and Figure 9–16, respectively. Table 9–9 lists all of the possible SDRAM configurations available via the TMS320C6201/C6202/C6701 EMIF.

The TMS320C6211/C6711 EMIF allows programming of the addressing characteristics of the SDRAM, including the number of column address bits (page size), the row address bits (pages per bank), and banks (maximum number of pages which can be opened). Using this information, the ’C6211/C6711 is able to open up to four pages of SDRAM simultaneously. The pages can all be in a single CE space, or distributed across multiple CE spaces. Table 9–10 summarizes the pin connection and related signals specific to SDRAM operation.

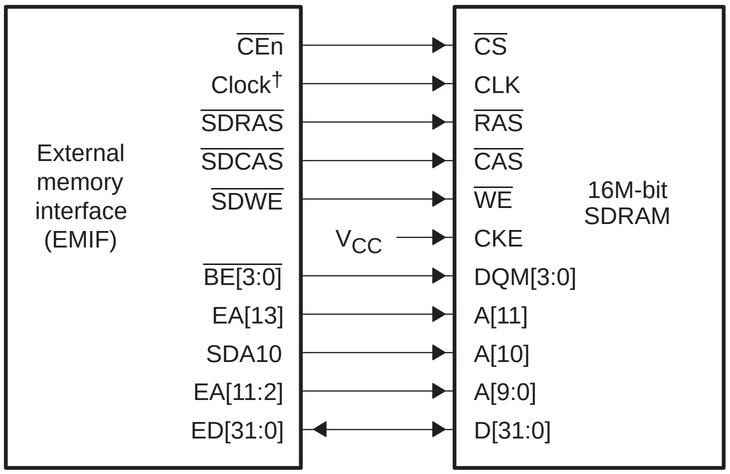
Table 9–8 does not apply to the ’C6211/C6711 because page characteristics are programmable. The ’C6211/C6711 can interface to any SDRAM that has 8 to 10 column address pins, 11 to 13 row address pins, and two or four banks.

Table 9–8. TMS320C6201/C6202/C6701 EMIF SDRAM Commands

Command	Function
DCAB	Deactivate all banks
DEAC†	Deactivate a single bank†
ACTV	Activates the selected bank and select the row
READ	Inputs the starting column address and begins the read operation
WRT	Inputs the starting column address and begins the write operation
MRS	Mode register set, configures SDRAM mode register
REFR	Autorefresh cycle with internal address

† TMS320C6211/C6711 only

Figure 9–14. TMS320C6201/C6202/C6701 EMIF to 16M-Bit SDRAM Interface



† Clock=SDCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.

Figure 9–15. TMS320C6211/C6711 EMIF to 16M-Bit SDRAM Interface

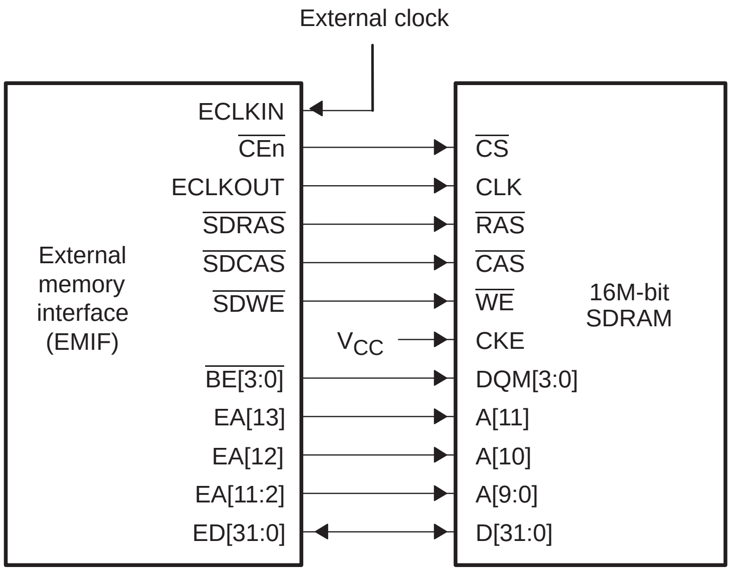
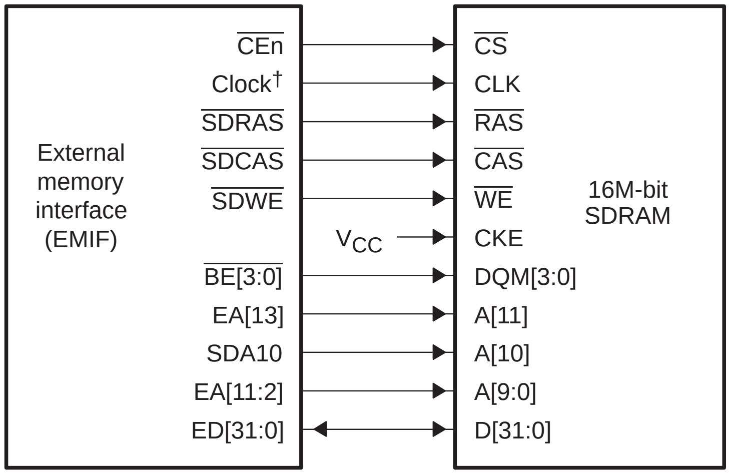


Figure 9–16. TMS320C6201/C6202/C6701 EMIF to 64M-Bit SDRAM Interface



† Clock=SDCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.

Table 9–9. TMS320C6201/C6202/C6701 SDRAM Memory Population†

SDRAM Size	SDRAM Banks	SDRAM Width	Devices per CE Space	Memory Size per CE Space
16M bit	2	16 bits	2	4M bytes
16M bit	2	8 bits	4	8M bytes
64M bit	4	16 bits	2	16M bytes

† The ’C6211/C6711 is not limited to these configurations because of larger possible CE spaces and programmable address shift.

Table 9–10. SDRAM Control Pins

EMIF Signal	SDRAM Signal	SDRAM Function
SDA10	A10	Address line A10/autoprecharge disable. Serves as a row address bit during ACTV commands and also disables the autoprecharging function of SDRAM. (’C6201/C6202/C6701 only)
$\overline{\text{SDRAS}}$	$\overline{\text{RAS}}$	Row address strobe and command input. Latched by the rising edge of CLK to determine current operation. Valid only if $\overline{\text{CS}}$ is active (low) during that clock edge.
$\overline{\text{SDCAS}}$	$\overline{\text{CAS}}$	Column address strobe and command Input. Latched by the rising edge of CLK to determine current operation. Valid only if $\overline{\text{CS}}$ is active (low) during that clock edge.
$\overline{\text{SDWE}}$	$\overline{\text{WE}}$	Write strobe and command input. Latched by the rising edge of CLK to determine current operation. Valid only if $\overline{\text{CS}}$ is active during that clock edge.
$\overline{\text{BE[3:0]}}$	DQM[3:0]	Data/output mask. DQM is an input/output buffer control signal. When high, disables writes and places outputs in the high impedance state during reads. DQM has a 2-CLK-cycle latency on reads and a 0-CLK-cycle latency on writes. DQM pins serve essentially as byte strobes and are connected to $\overline{\text{BE[3:0]}}$ outputs.
$\overline{\text{CE3}}$, $\overline{\text{CE2}}$ or $\overline{\text{CE0}}$	$\overline{\text{CS}}$	Chip select and command enable. $\overline{\text{CS}}$ must be active (low) for a command to be clocked into the SDRAM. $\overline{\text{CS}}$ does not affect data input or output once a write or read has begun. $\overline{\text{CE1}}$ does not support SDRAM.
—	CKE	CKE clock enable. Tied high when interfaced to EMIF to enable clocking always.
CLKOUT2	CLK	SDRAM clock input. Runs at 1/2 the CPU clock rate.Used for synchronous memory interface on the ’C6202.
SDCLK	CLK	SDRAM clock input. Runs at 1/2 the CPU clock rate.Used for SDRAM interface on ’C6201/C6701

The SDRAM interface on the ’C6202 is identical to that of the ’C6201, with the exception that it has been combined with the SBSRAM interface. Only one of these two synchronous memory types can be used on a ’C6202 system. Since the ’C6202 performs background refreshes for SDRAM, SBSRAM accesses could be corrupted during SDRAM refresh if both memory types were present.

The SDRAM interface signals on the ’C6211/C6711 are identical to those of the ’C6201, with the exception that EA12 performs the function of the SDA10 pin. The SDRAM signals have been combined with the SBSRAM and asynchronous memory interface. An external clock source must be provided to the ’C6211/C6711, which generates the ECLKOUT signal used in the SDRAM interface. The ’C6211/C6711 also allows for 8-, 16-, and 32-bit SDRAM interfaces. Since the ’C6211/C6711 does not perform background refreshes, all three memory types may be included in the same system.

9.4.1 SDRAM Initialization

The EMIF performs the necessary tasks to initialize SDRAM if any of the CE spaces are configured for SDRAM. An SDRAM initialization is requested by a write of 1 to the INIT bit in the EMIF SDRAM control register.

The steps of an initialization are as follows:

1) Send a DCAB command to all CE spaces configured as SDRAM.
2) Send three refresh commands.
3) Send an MRS command to all CE spaces configured as SDRAM.

The DCAB cycle is performed immediately after reset, provided the $\overline{\text{HOLD}}$ input is not active (a host request). If $\overline{\text{HOLD}}$ is active, the DCAB command is not performed until the hold condition is removed. In this case the external requester should not attempt to access any SDRAM banks, unless it performs SDRAM initialization and control itself.

9.4.2 Monitoring Page Boundaries

Because SDRAM is a paged memory type, the EMIF SDRAM controller monitors the active row of SDRAM so that row boundaries are not crossed during the course of an access. To accomplish this monitoring, the EMIF stores the address of the open page and performs compares against that address for subsequent accesses to the SDRAM bank. For the ’C6201/C6202/C6701, this storage and comparison is performed independently for each CE space, so that a single page can be open in each CE space.

The number of address bits compared is a function of the page size programmed in the SDWID field in the EMIF SDRAM control register for the ’C6201/C6202/C6701. If SDWID = 0, the EMIF expects CE spaces configured as SDRAM to have four 8-bit-wide SDRAMs that have page sizes of 512. Thus, the logical byte address bits compared are 23–11. If SDWID = 1, the EMIF expects CE spaces with SDRAM to have two 16-bit-wide SDRAMs that have page sizes of 256. Thus, the logical byte address bits compared are 23–10. The logical address bits 25 to 24 determine their CE space. If a page boundary is crossed during an access to the same CE space, the EMIF performs a DCAB command and starts a new row access.

For the ’C6211/C6711, up to four pages of SDRAM can be opened simultaneously. These pages can be within a single CE space, or spread over all CE spaces. For example, two pages can be open in CE0 and CE2, or four pages can be open in CE0. The combination of SDCSZ, SDRSZ, and SDBSZ control which logical address bits are compared to determine if a page is open. For example, a typical 2-bank $\times$ 512K $\times$ 16-bit SDRAM has settings of two banks,eleven row address bits, and eight column address bits. A 32-bit-wide SDRAM uses logical address bits A[9:2] (two-bit offset for word addressing) to specify the column being accessed. Bits A[20:10] specify the row offset, and bit A[21] specifies the bank. Logical address bites A[31:28] determines the CE space used. If a page boundary is crossed during an access to the same CE space, the ’C6211/C6711 performs a DEAC command and starts a new row access.

Simply ending the current access is not a condition that forces the active SDRAM row to be closed. The EMIF leaves the active row open until it becomes necessary to close it. This feature decreases the deactivate-reactivate overhead and allows the interface to capitalize fully on address locality of memory accesses.

9.4.3 SDRAM Refresh

The RFEN bit in the SDRAM control register selects the SDRAM refresh mode of the EMIF. A value of 0 in RFEN disables all EMIF refreshes, and you must ensure that refreshes are implemented in an external device. A value of 1 in RFEN enables the EMIF to perform refreshes of SDRAM.

Refresh commands (REFR) enable all $\overline{\text{CE}}$ signals for all CE spaces selected to use SDRAM (with the MTYPE field of the CE space control register). REFR is automatically preceded by a DCAB command. This ensures that all CE spaces selected with SDRAM are deactivated. Following the DCAB command, the EMIF begins performing trickle refreshes at a rate defined by the period value in the EMIF SDRAM control register, provided no other SDRAM access is pending.

For the ’C6201/C6202/C6701, the SDRAM interface monitors the number of refresh requests posted to it and performs the refreshes. Within the EMIF SDRAM control block, a 2-bit counter monitors the backlog of refresh requests. The counter increments once for each refresh request and decrements once for each refresh cycle performed. The counter saturates at the values of 11 and 00. At reset, the counter is automatically set to 11 to ensure that several refreshes occur before accesses begin.

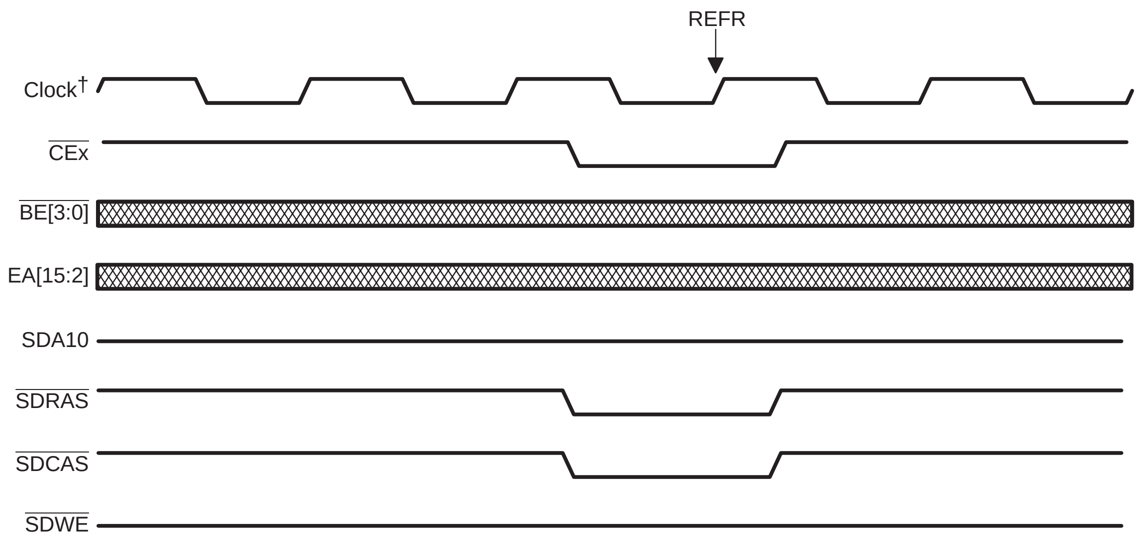
The ’C6201/C6202/C6701 EMIF SDRAM controller prioritizes SDRAM refresh requests with other data access requests posted to it from the EMIF requesters. The following rules apply:

- ❑ A counter value of 11 invalidates the page information register, forcing the controller to close the current SDRAM page. The value 11 indicates an urgent refresh condition. Thus, following the DCAB command, the EMIF SDRAM controller performs three REFR commands, thereby decrementing the counter to 00 before proceeding with the remainder of the current access. If SDRAM is present in multiple CE spaces, the DCAB-refresh sequence occurs in all spaces containing SDRAM.
- ❑ During idle times on the SDRAM interface(s), if no request is pending from the EMIF, the SDRAM interface performs REFR commands as long as the counter value is nonzero. This feature reduces the likelihood of having to perform urgent refreshes during actual SDRAM accesses. If SDRAM is present in multiple CE spaces, this refresh occurs only if all interfaces are idle with invalid page information.

Unlike the ’C6201/C6202/C6701 EMIF, the ’C6211/C6711 REFR requests are considered high priority, and no distinction exists between urgent and trickle refresh. Transfers in progress are allowed to complete. The ’C6211/C6711 SDRAM refresh period has an extra bitfield, XRFR, which controls the number of refreshes performed when the counter reaches zero. This feature allows the XRFR field to be set to perform up to four refreshes when the refresh counter expires.

For all ’C6000 devices, the EMIF SDRAM interface performs $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh cycles for SDRAM. Some SDRAM manufacturers call this autorefresh. Prior to an REFR command, a DCAB command is performed to all CE spaces specifying SDRAM to ensure that all active banks are closed. Page information is always invalid before and after a REFR command; thus, a refresh cycle always forces a page miss. A deactivate cycle is required prior to the refresh command. Figure 9–17 shows the timing diagram for an SDRAM refresh.

Figure 9–17. SDRAM Refresh



† Clock=SDCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.
Clock=ECLKOUT for ’C6211/C6711.

9.4.4 Mode Register Set

The ’C6201/C6202/C6701 EMIF automatically performs a DCAB command followed by an MRS command whenever the INIT field in the EMIF SDRAM control register is set. INIT can be set by device reset or by a user write. Like DCAB and REFR commands, MRS commands are performed to all CE spaces configured as SDRAM through the MTYPE field. Following a hold, the external requester should return the SDRAM MRS register’s original value before returning control of the bus to the EMIF. Alternatively, you could poll the HOLD and HOLDA bits in the EMIF global control register and, upon detecting completion of an external hold, reinitialize the EMIF by writing a 1 to the INIT bit in the EMIF SDRAM control register.

The EMIF always uses a mode register value of 0030h during an MRS command. Figure 9–18 shows the mapping between mode register bits, EMIF pins, and the mode register value. Table 9–11 shows the JDEC standard SDRAM configuration values selected by this mode register value. Figure 9–21 shows the timing diagram during execution of the MRS command.

Figure 9–18. TMS320C6201/C6202/C6701 Mode Register Value

13	12	11	10	9	8	7
EA15	EA14	EA13	SDA10	EA11	EA10	EA9
Rsvd				Write burst length	Rsvd	
0000				0	00	

6	5	4	3	2	1	0
EA8	EA7	EA6	EA5	EA4	EA3	EA2
Read latency			S/I	Burst length		
0	1	1	0	000		

Table 9–11. TMS320C6201/C6202/C6701 Implied SDRAM Configuration by MRS Value

Field	**Selection**
Write burst length	1 word
Read latency	3 cycles
Serial/interleave burst type	Serial
Burst length	1 word

The ’C6211/C6711 uses a mode register value of either 0032h or 0022h. The register value and description are shown in Figure 9–19 and Figure 9–20. Both values program a default burst length of four words for both reads and writes. The value actually used depends on the CASL parameter defined in the SDRAM extension register. If the CAS latency is three, 0032h is written. If the CAS latency is two, 0022h is written during the MRS cycle. Table 9–12 summarizes.

Figure 9–19. TMS320C6211/C6711 Mode Register Value (0032h)

13	12	11	10	9	8	7
EA15	EA14	EA13	SDA10	EA11	EA10	EA9
Rsvd				Write burst length	Rsvd	
0000				0	00	

6	5	4	3	2	1	0
EA8	EA7	EA6	EA5	EA4	EA3	EA2
Read latency			S/I	Burst length		
0	1	1	0	010		

Figure 9–20. TMS320C6211/C6711 Mode Register Value (0022h)

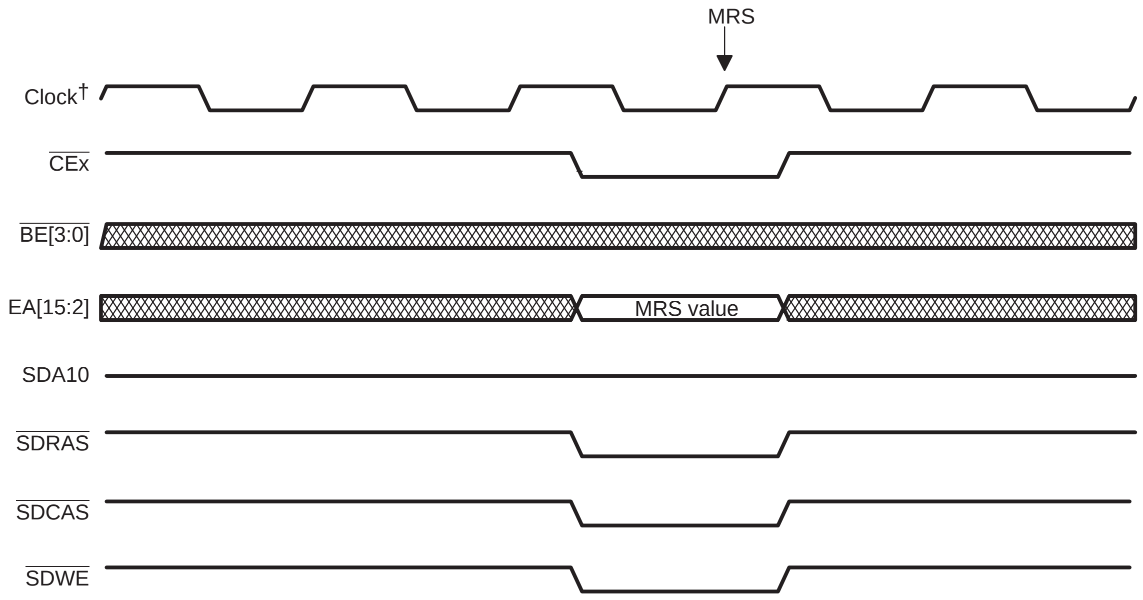
13	12	11	10	9	8	7
EA15	EA14	EA13	SDA10	EA11	EA10	EA9
Rsvd				Write burst length	Rsvd	
0000				0	00	

6	5	4	3	2	1	0
EA8	EA7	EA6	EA5	EA4	EA3	EA2
Read latency			S/I	Burst length		
0	1	0	0	010		

Table 9–12. TMS320C6211/C6711 Implied SDRAM Configuration by MRS

Field	**CASL = 0**	**CASL = 1**
Write burst length	4 words	4 words
Read latency	2 cycles	3 cycles
Serial/interleave burst type	Serial	Serial
Burst length	4 words	4 words

Figure 9–21. SDRAM Mode Register Set: MRS Command



† Clock=SDCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.
Clock=ECLKOUT for ’C6211/C6711.

9.4.5 Address Shift

Because the same EMIF pins determine the row and column address, the EMIF interface appropriately shifts the address in row and column address selection. Table 9–13 and shows the translation between bits of the byte address and how they appear on the EA pins for row and column addresses. SDRAMs use the address inputs for control as well as address.

The following factors apply to the address shifting process for the ’C6201/C6202/C6701:

- ❑ The address line that corresponds to the SDRAM’s bank select field (A11 on 16M-bit SDRAM; A13 and A12 on 64M-bit SDRAM) is latched internally by the SDRAM controller. This ensures that the bank select remains correct during READ and WRT commands. Thus, the EMIF maintains these values as shown in both row and column addresses.
- ❑ The EMIF forces SDA10 to be low when $\overline{\text{RAS}}$ is not active and high during DCAB commands at the end of a page of accesses. This prevents the autoprecharge from occurring following a READ or WRT command.

The following factors apply to the address shifting process for the ’C6211/C6711:

- ❑ The address shift is controlled completely by the column size field (SDCSZ), and is unaffected by the bank and row size fields. The bank and row size are used internally to determine whether a page is opened
- ❑ EA12 is connected directly to A10 signal, instead of using a dedicated precharge pin SDA10.

Table 9–13. TMS320C6201/C6202/C6701 Byte Address to EA Mapping for SDRAM $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$

EMIF Pins				EA[21:17]	EA16	EA15	EA14	EA13	SDA10	EA11	EA10	EA9	EA8	EA7	EA6	EA5	EA4	EA3	EA2
SDRAM Pins	**SDRAM Width**	**SDWID**	**DRAM Cmd**			**A13**	**A12**	**A11**	**A10**	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
Address bit	x16	1	$\overline{\text{RAS}}$	▨	▨	23	22	21	20	19	18	17	16	15	14	13	12	11	10
			$\overline{\text{CAS}}$	▨	▨	23	22	21	20	19	18	9	8	7	6	5	4	3	2
Address bit	x8	0	$\overline{\text{RAS}}$	▨	▨	▨	23	22	21	20	19	18	17	16	15	14	13	12	11
			$\overline{\text{CAS}}$	▨	▨	▨	23	22	21	20	10	9	8	7	6	5	4	3	2

Legend:	░	Bit is internally latched during an ACTV command.
	▨	Reserved for future use. Undefined.

Note: The $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ values indicate the bit of the byte address present on the corresponding EA pin during a $\overline{\text{RAS}}$ or $\overline{\text{CAS}}$ cycle.

Table 9–14 describes the addressing for a 32-bit wide 'C6211/C6711 SDRAM interface. The address presented on the pins are shifted for 8-bit and 16-bit interfaces.

Table 9–14. TMS320C6211/C6711 Byte Address to EA Mapping for 32-bit Interface

		EA[21:17]	**EA16**	**EA15**	**EA14**	**EA13**	**SDA10**	**EA11**	**EA10**	**EA9**	**EA8**	**EA7**	**EA6**	**EA5**	**EA4**	**EA3**	**EA2**
# of column address bits	**DRAM Cmd**		**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
8	$\overline{RAS}$	▓	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
	$\overline{CAS}$	▓	24	23	22	21	20	19	18	9	8	7	6	5	4	3	2
9	$\overline{RAS}$	▓	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
	$\overline{CAS}$	▓	25	24	23	22	21	20	10	9	8	7	6	5	4	3	2
10	$\overline{RAS}$	▓	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12
	$\overline{CAS}$	▓	26	25	24	23	22	11	10	9	8	7	6	5	4	3	2

Legend:

	░	Bit is internally latched during an ACTV command.
	▓	Reserved for future use. Undefined.

9.4.6 Timing Requirements

Table 9–15 shows five SDRAM timing parameters that decouple the EMIF from SDRAM speed limitations. Three of these parameters are programmable via the EMIF SDRAM control register; the remaining two are assumed to be static values. The three programmable values ensure that EMIF control of SDRAM obeys these minimum timing requirements. Consult the SDRAM data sheet for information on the parameters that are appropriate for your particular SDRAM.

Table 9–15. TMS320C6201/C6202/C6701 SDRAM Timing Parameters

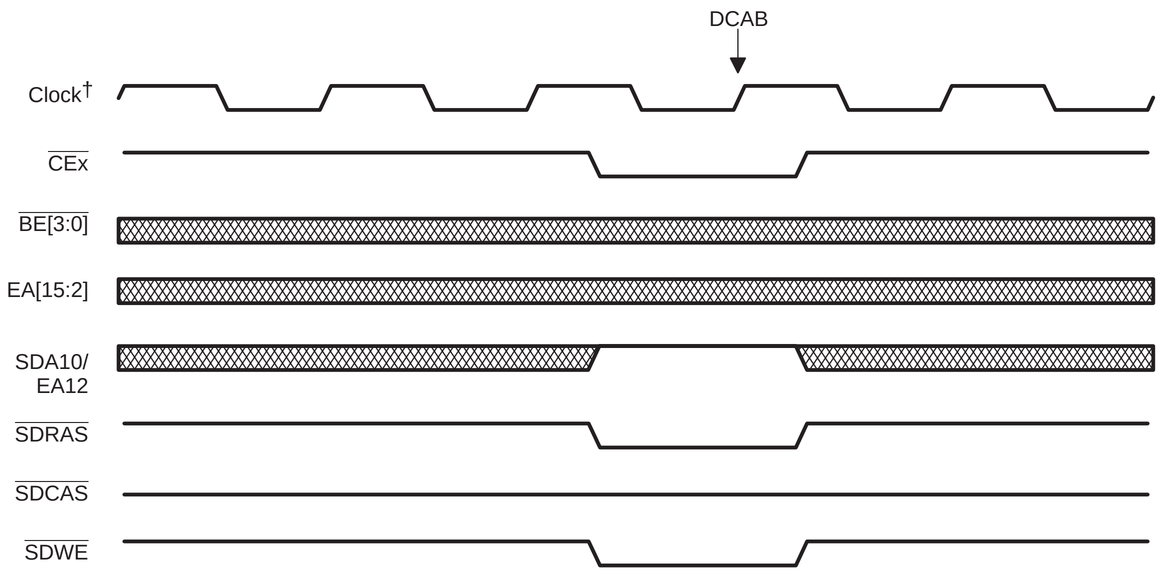
Parameter	Description	Value in CLKOUT2/ ECLKIN2 Cycles
t_{RC}	REFR command to ACTV, MRS, or subsequent REFR command	TRC + 1
t_{RCD}	ACTV command to READ or WRT command	TRCD + 1
t_{RP}	DCAB command to ACTV, MRS, or REFR command	TRP +1
t_{RAS}	ACTV command to DEAC to DCAB command	7
t_{nEP}	Overlap between read data and a DCAB command	2

† CLKOUT2 cycles apply to the ’C6201/C6202/C6701, and ECLKOUT cycles apply to the ’C6211/C6711.

9.4.7 SDRAM Deactivation

The SDRAM deactivation (DCAB) is performed after a hardware reset or when INIT = 1 in the EMIF SDRAM control register. This cycle is also required by the SDRAMs prior to REFR and MRS. On the ’C6201/C6202/C6701, a DCAB is issued when a page boundary is crossed. During the DCAB command, SDA10 is driven high to ensure that all SDRAM banks are deactivated. Figure 9–22 shows the timing diagram for SDRAM deactivation.

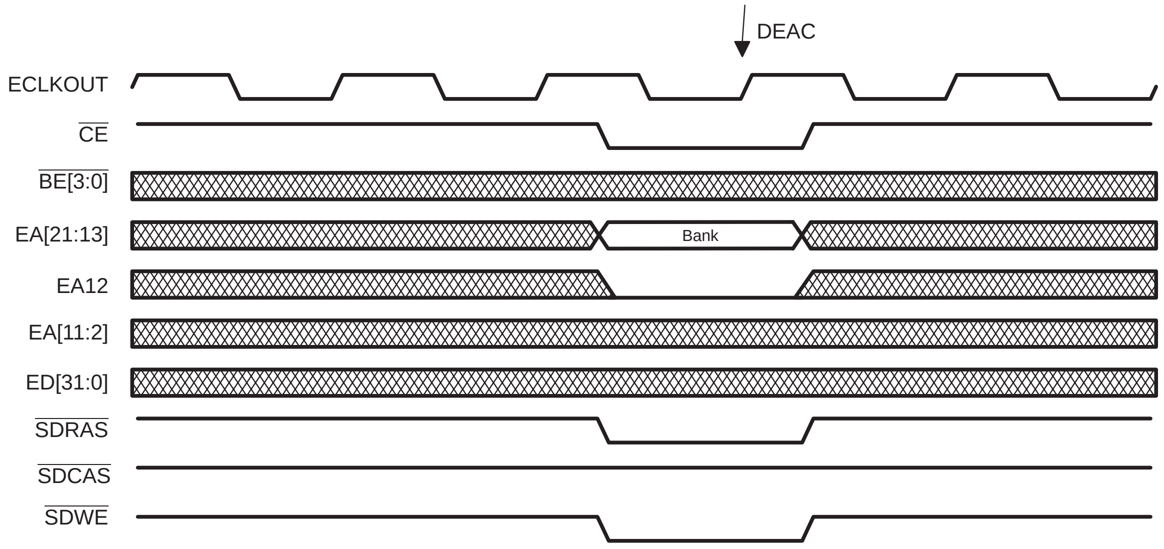
Figure 9–22. SDRAM DCAB — Deactivate all Banks


DCAB
Clock†
$\overline{\text{CEx}}$
$\overline{\text{BE[3:0]}}$
EA[15:2]
SDA10/ EA12
$\overline{\text{SDRAS}}$
$\overline{\text{SDCAS}}$
$\overline{\text{SDWE}}$

† Clock=SDCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.
Clock=ECLKOUT for ’C6211/C6711.

The ’C6211/C6711 also supports the DEAC command, whose operation is depicted in Figure 9–23, which closes a single page of SDRAM specified by the bank select signals. When a page boundary is crossed, the DEAC command is used to close the open page. The ’C6211/C6711 still supports the DCAB command to close all pages prior to REFR and MRS commands.

Figure 9–23. TMS320C6211/C6711 SDRAM DEAC — Deactivate Single Bank

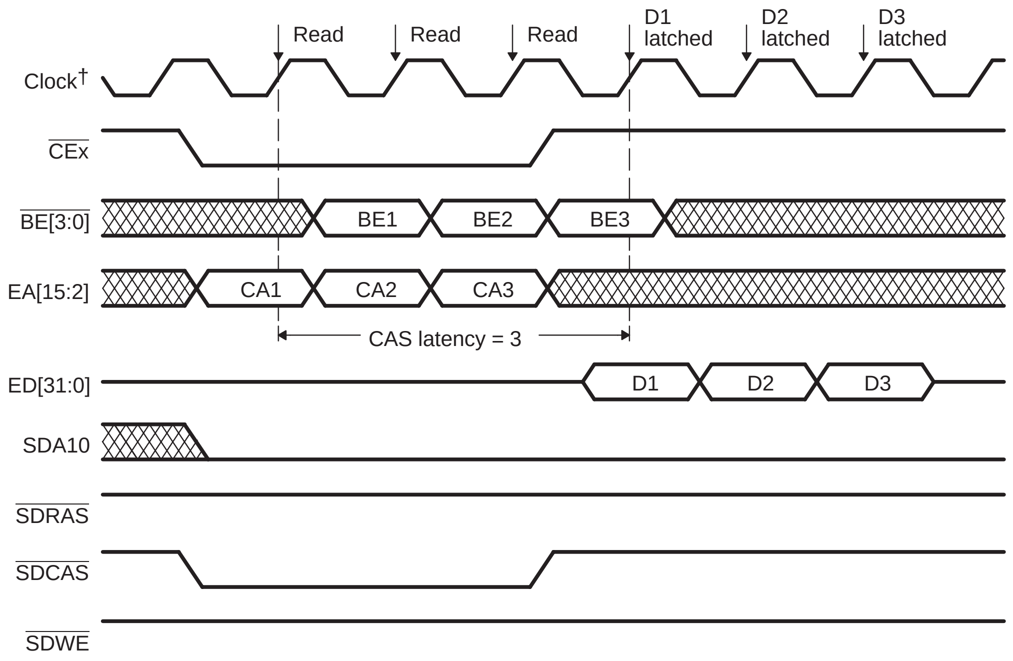


9.4.8 SDRAM Read

9.4.8.1 TMS320C6201/C6202/C6701 SDRAM Read

During an SDRAM read, the selected bank is activated with the row address during the ACTV command. Figure 9–24 shows the timing for the ’C6201/C6202/C6701 issuing three read commands performed at three different column addresses. The EMIF uses a $\overline{\text{CAS}}$ latency of three and a burst length of one. The three-cycle latency causes data to appear three cycles after the corresponding column address. Following the final read command of the ’C6201/C6202/C6701, an idle cycle is inserted to meet timing requirements. If required, the bank is then deactivated with a DCAB command and the EMIF can begin a new page access. If no new access is pending or an access is pending to the same page, the DCAB command is not performed until the page information becomes valid. The values on EA[15:13] during column accesses and execution of the DCAB command are the values latched during the ACTV command.

Figure 9–24. TMS3206201/C6202/C6701 SDRAM Read

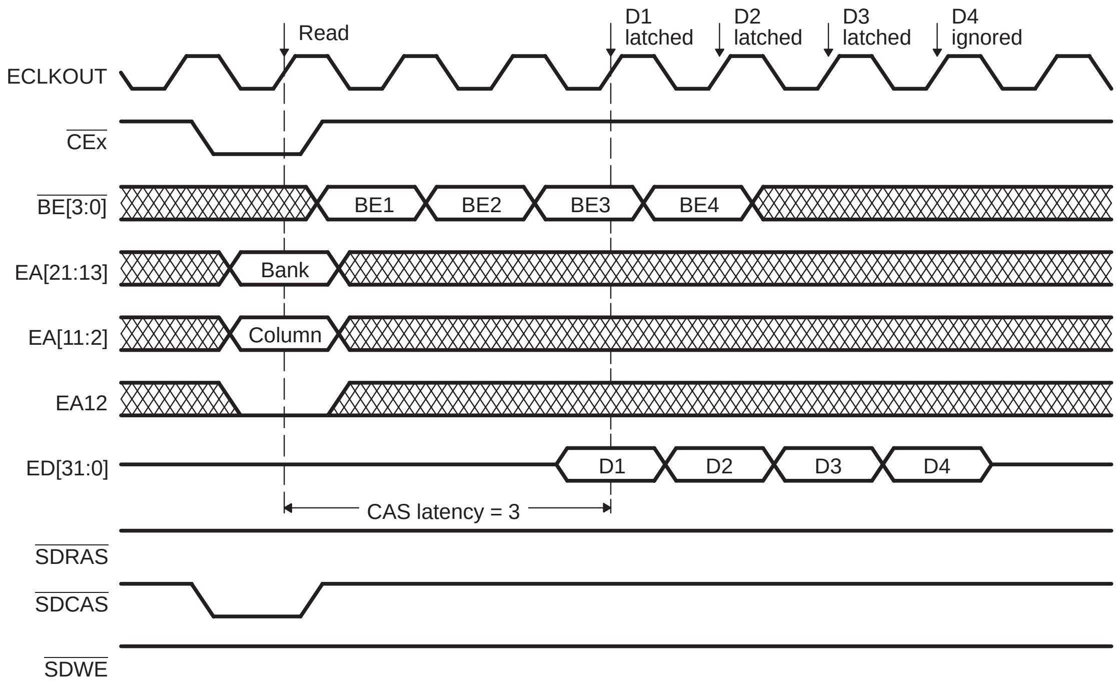


† Clock=SDCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.

9.4.8.2 TMS320C6211/C6711 SDRAM Read

Figure 9–25 shows the 'C6211/C6711 performing a three word read burst from SDRAM. The 'C6211/C6711 uses a burst length of four, and has a programmable $\overline{\text{CAS}}$ latency of either two or three cycles. The CAS latency is three cycles in this example (CASL = 1). Since the default burst length is four words, the SDRAM returns four pieces of data for every read command. If no additional access are pending to the EMIF, as in Figure 9–25, the read burst completes and the unneeded data is disregarded. If accesses are pending, the read burst can be interrupted with a new command (READ,WRT,DEAC,DCAB), controlled by the SDRAM extension register. If a new access is not pending, the DCAB/DEAC command is not performed until the page information becomes invalid.

Figure 9–25. TMS320C6211 SDRAM Read

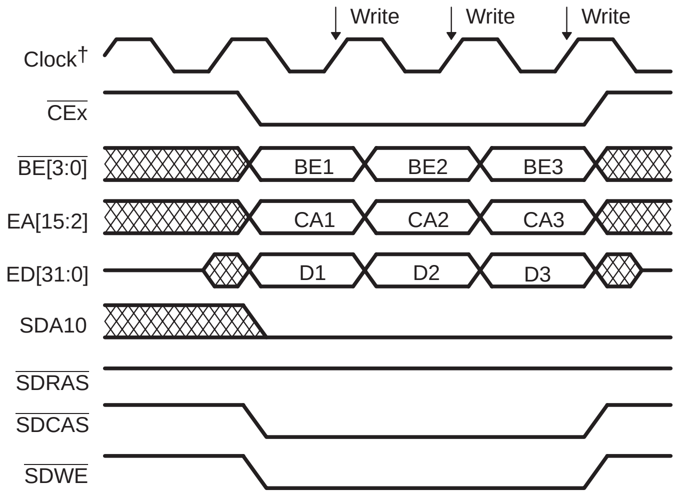


9.4.9 SDRAM Write

9.4.9.1 TMS320C6201/C6202/C6701 SDRAM Write

All SDRAM writes have a burst length of one on the ’C6201/C6202/C6701 . The bank is activated with the row address during the ACTV command. There is no latency on writes, so data is output on the same cycle that the column address. Writes to particular bytes are disabled via the appropriate DQM inputs; this feature allows for byte and halfword writes. Figure 9–26 shows the timing for a three-word write on the ’C6201/C6202/C6701. Since the default write burst length is one-word, a new write command is issued each cycle to perform the three word burst. Following the final write command, the ’C6201/C6202/C6701 inserts an idle cycle to meet SDRAM timing requirements. The bank is then deactivated with a DCAB command, and the memory interface can begin a new page access. If no new access is pending, the DCAB command is not performed until the page information becomes invalid (see section 9.4.2). The values on EA[15:13] during column accesses and the DCAB command are the values latched during the ACTV command.

Figure 9–26. TMS320C6201/C6202/C6701 SDRAM Three Word Write

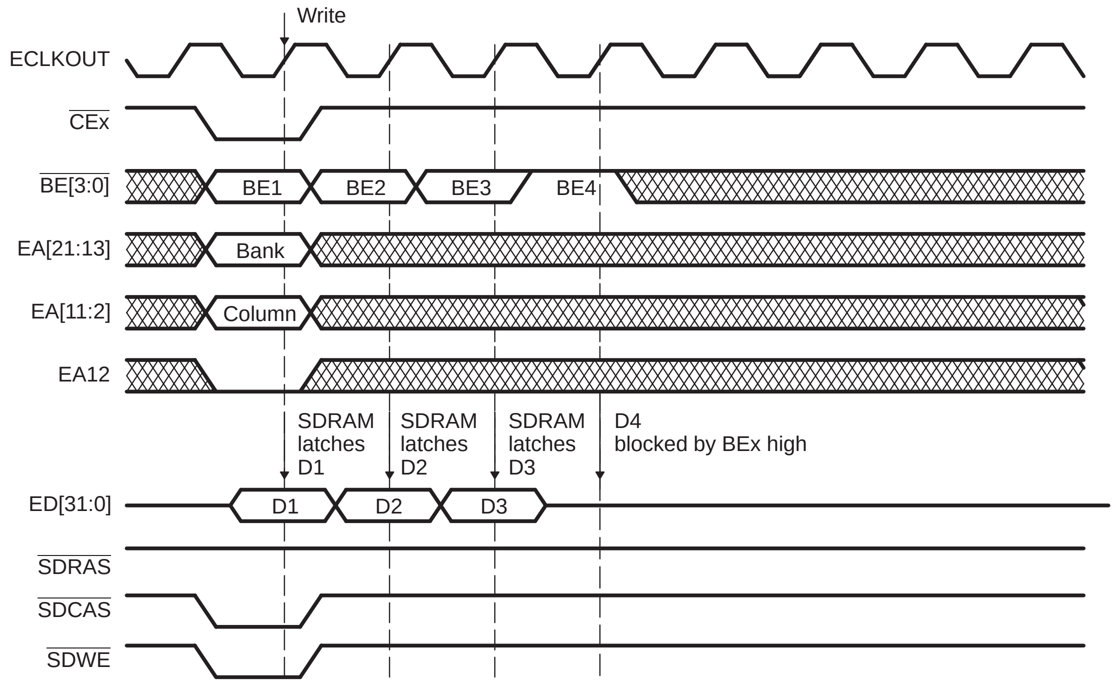


† Clock=SDCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.

9.4.9.2 TMS320C6211/C6711 SDRAM Write

All SDRAM writes have a burst length of four on the 'C6211/C6711. The bank is activated with the row address during the ACTV command. There is no latency on writes, so data is output on the same cycle as the column address. Writes to particular bytes are disabled via the appropriate DQM inputs; this feature allows for byte and halfword writes. Figure 9–27 shows the timing for a three-word write on the 'C6211/C6711. Since the default 'C6211/C6711 write-burst length is four words, the last write is masked out via the byte enable signals. On the 'C6211/C6711, idle cycles are inserted as controlled by the parameters of the SDRAM extension register fields (WR2RD, WR2DEAC, WR2WR, TWR). The bank is then deactivated with a DEAC command for 'C6211/C6711, and the memory interface can begin a new page access. If no new access is pending, the DEAC command is not performed until the page information becomes invalid (see section 9.4.2). The values on EA[15:13] during column accesses and the DEAC command are the values latched during the ACTV command.

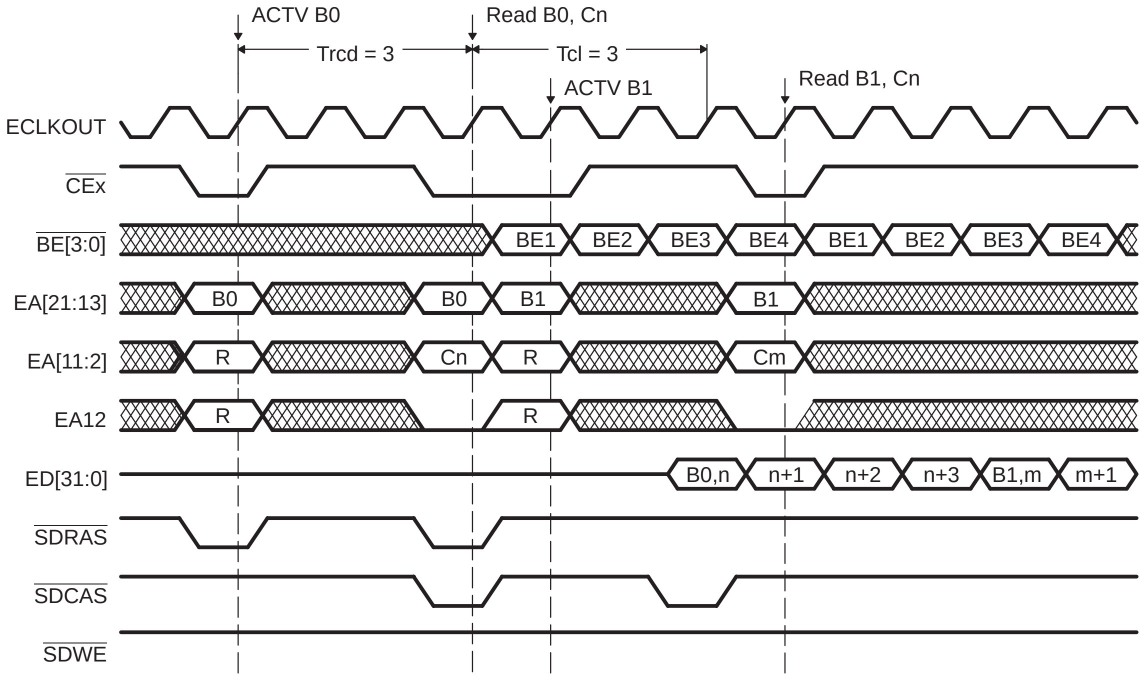
Figure 9–27. TMS320C6211/C6711 SDRAM Three Word Write



9.4.10 TMS320C6211/C6711 Seamless Data Access

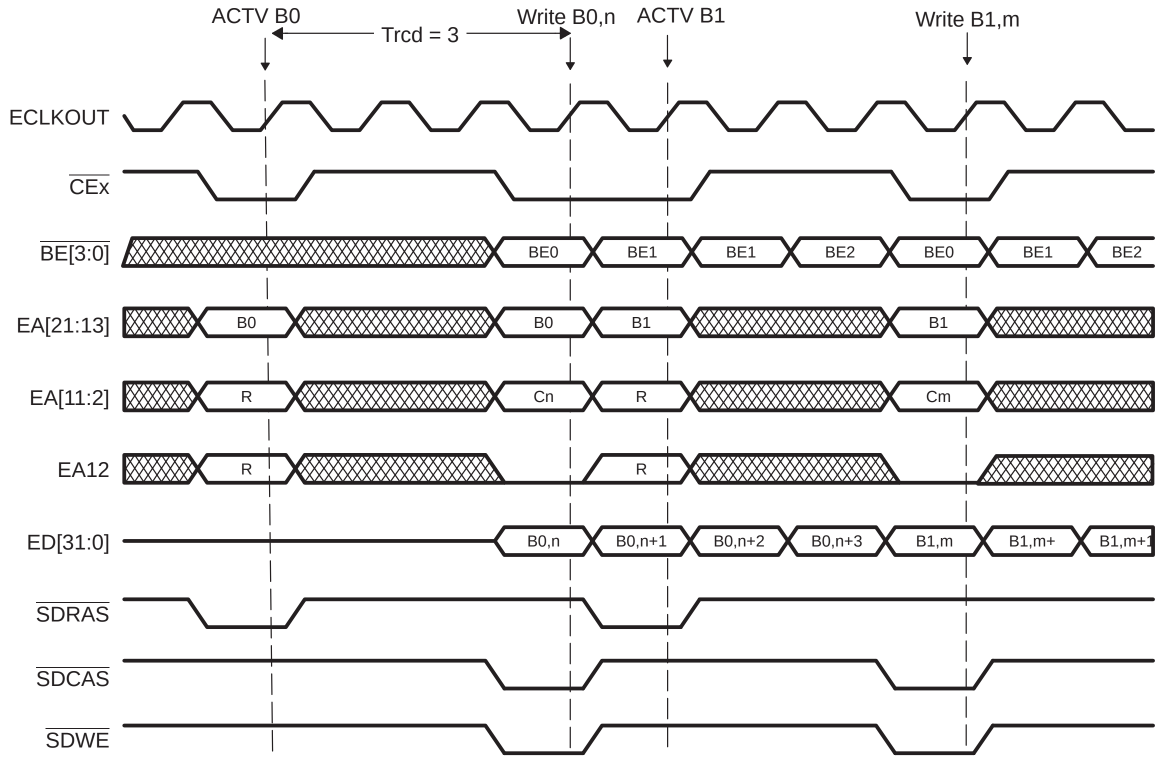
Since the ’C6211/C6711 performs data transfers to SDRAM in bursts of 4 words and can maintain up to 4 open pages in a single CE space, this device is capable of sustaining seamless data transfer to and from multiple pages of SDRAM. Figure 9–28 shows an example of the ’C6211/C6711 performing two consecutive burst reads to different pages in a single CE space. The first page is opened with an ACTV command and after a delay controlled by T_{rcd}, the first read burst begins to bank 0. Since a 4 word burst is done by default, the ’C6211/C6711 takes advantage of the extra cycles by issuing an ACTV command to open bank 1 while the first read burst takes place. When the first read burst is scheduled to end, the read burst to bank 1 is issued such that the 3 cycle CAS latency forces data to continue uninterrupted.

Figure 9–28. Burst Reads to 2 Pages of SDRAM



Seamless write transfers are accomplished in the same way. First, bank 0 is opened and after Trcd cycles, the write burst can begin. During the first write burst, a page in bank 1 can be opened. This allows the write to bank 1 to begin immediately after the write burst to bank 0 ends, as shown in Figure 9–29.

Figure 9–29. Seamless SDRAM Write



9.5 SBSRAM Interface

As shown in Figure 9–30 (’C6201/C6202/C6701) and Figure 9–31 (’C6211/C6711), the EMIF interfaces directly to industry-standard synchronous burst SRAMs (SBSRAMS). This memory interface allows a high-speed memory interface without some of the limitations of SDRAM. Most notably, since SBSRAMs are SRAM devices, random accesses in the same direction can occur in a single cycle. The SBSRAM interface can run at either the CPU clock speed or at 1/2 of this rate for the ’C6201 and ’C6701. The selection is made based on the setting of the SSCRT bit in the EMIF global control register. For the ’C6202 the interface operates at the 1/2 rate only, and for the ’C6211/C6711, the SBSRAM runs off an externally provided clock.

The four SBSRAM control pins are latched by the SBSRAM on the rising SSCLK edge to determine the current operation. These pins are listed in Table 9–17. These signals are valid only if the chip select line for the SBSRAM is low.

For the ’C6201/6202/6701, the $\overline{\text{ADV}}$ signal of the SBSRAM is pulled high. This disables the internal burst advance counter of the SBSRAM. This interface allows bursting by strobing a new address into the SBSRAM on every cycle.

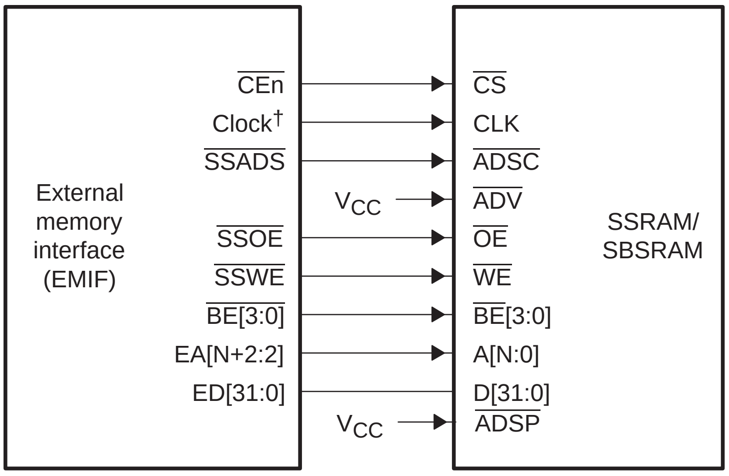
The ’C6211/C6711 interface takes advantage of the internal advance counter of the SBSRAM. For this interface, the $\overline{\text{ADV}}$ signal is pulled low, so that every access to the SBSRAM from the ’C6211/C6711 is assumed to be a four word burst. If random addressing is required for a given access, the ’C6211/C6711 can perform this by overriding the burst feature of the SBSRAM and strobing a new command into the SBSRAM on every cycle, as done by the other devices. Table 9–16 shows the 4 word burst sequencing of standard SBSRAMs in linear burst mode. In order to avoid the SBSRAM wrapping around to an unintended address (indicated in gray), the ’C6211/C6711 strobes a new address into the SBSRAM. This is also done if random reads are done or if the burst order should be non-incrementing or reverse order burst.

Table 9–16. SBSRAM in Linear Burst Mode

	Case 1	**Case 2**	**Case 3**	**Case 4**
SBSRAM Address	A[1:0]	A[1:0]	A[1:0]	A[1:0]
EMIF Address	EA[3:2]	EA[3:2]	EA[3:2]	EA[3:2]
First address	00	01	10	11
	01	10	11	00
	10	11	00	01
Fourth Address	11	00	01	10

The SBSRAM interface on the ’C6202 is identical to that of the ’C6201, with the exception that it has been combined with the SDRAM interface. Only one of these two synchronous memory types can be used on a ’C6202 system.

Figure 9–30. TMS320C6201/C6202/C6701 SBSRAM Interface



† Clock=SSCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.

Figure 9–31. TMS320C6211/C6711 SBSRAM interface

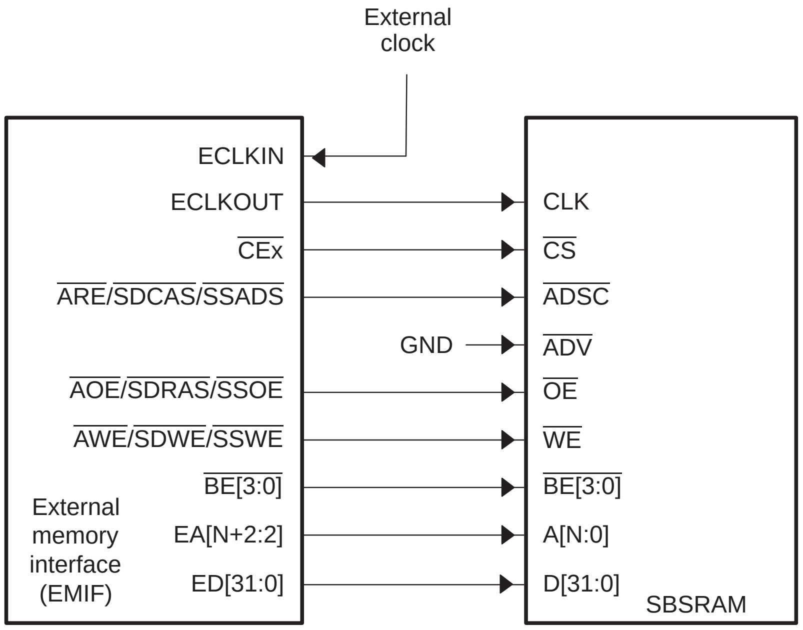


Table 9–17. EMIF SBSRAM Pins

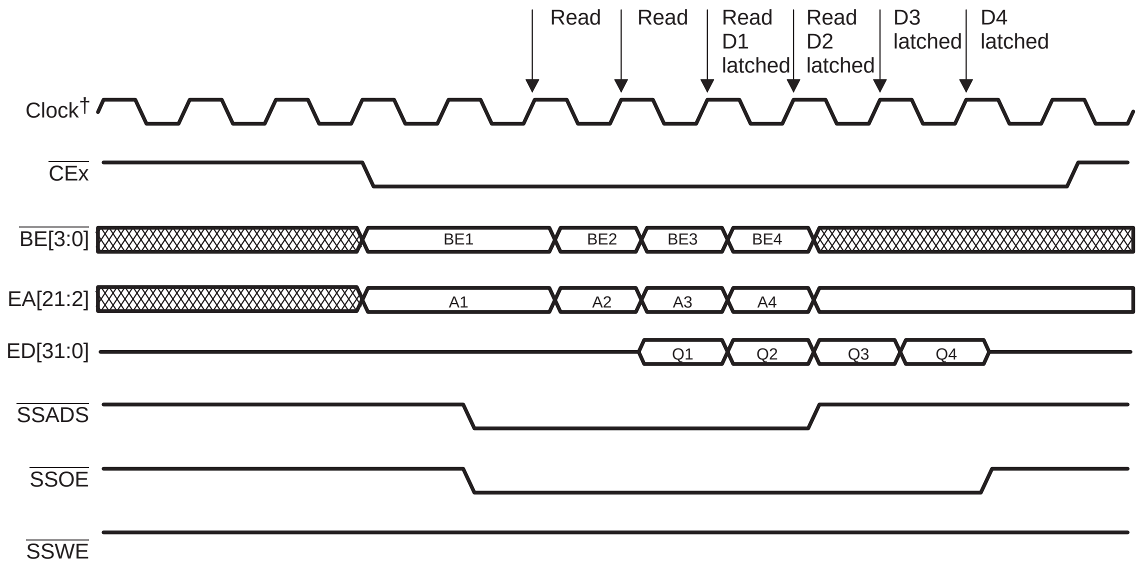
EMIF Signal	SBSRAM Signal	SBSRAM Function
$\overline{\text{SSADS}}$	$\overline{\text{ADSC}}$	Address strobe
$\overline{\text{SSOE}}$	$\overline{\text{OE}}$	Output enable
$\overline{\text{SSWE}}$	$\overline{\text{WE}}$	Write enable
SSCLK/CLKOUT2/ECLKOUT	CLK	SBSRAM clock

SBSRAMs are latent by their architecture, meaning that read data follows address and control information. Consequently, the EMIF inserts cycles between read and write commands to ensure that no conflict exists on the ED[31:0] bus. The EMIF keeps this turnaround penalty to a minimum. The initial 2-cycle penalty occurs when the direction changes on the bus. In general, the first access of a burst sequence incurs a 2-cycle start-up penalty.

9.5.1 SBSRAM Reads

Figure 9–32 shows a four-word read of an SBSRAM for the ’C6201/C6202/C6701. Every access strobes a new address into the SBSRAM, indicated by the $\overline{\text{SSADS}}$ strobe low. The first access requires an initial start-up penalty of two cycles; thereafter, all accesses occur in a single SSCLK cycle.

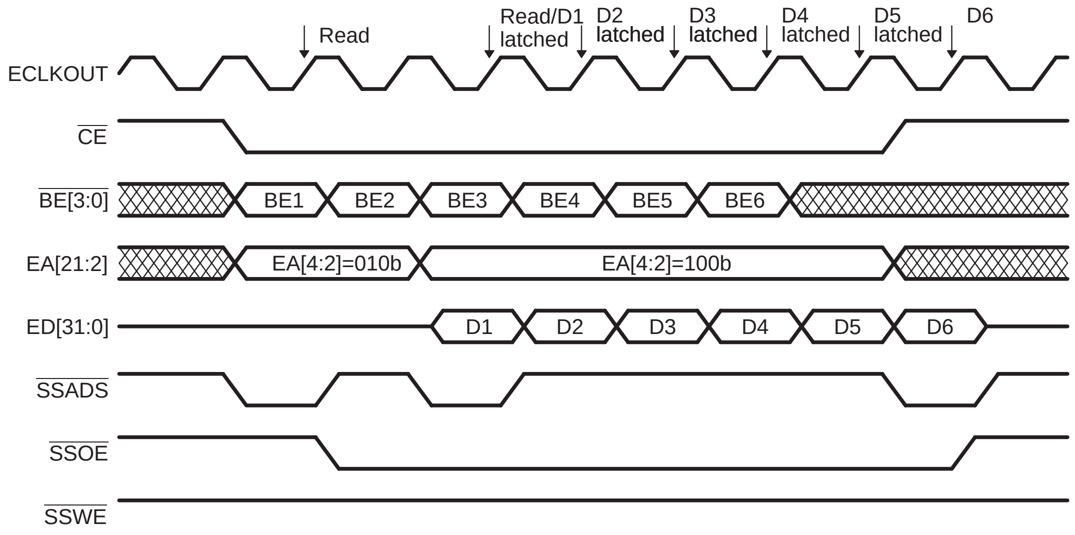
Figure 9–32. SBSRAM Four-Word Read



† Clock=SSCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.

Figure 9–33 shows the timing for ’C6211/C6711 six word read. The address starts with EA[3:2] equal to 10b. A new address is strobed into the SBSRAM on the third cycle to prevent the internal burst counter from rolling over to 000b. The burst is terminated by deasserting the $\overline{\text{CEn}}$ signal while $\overline{\text{SSADS}}$ is strobed low.

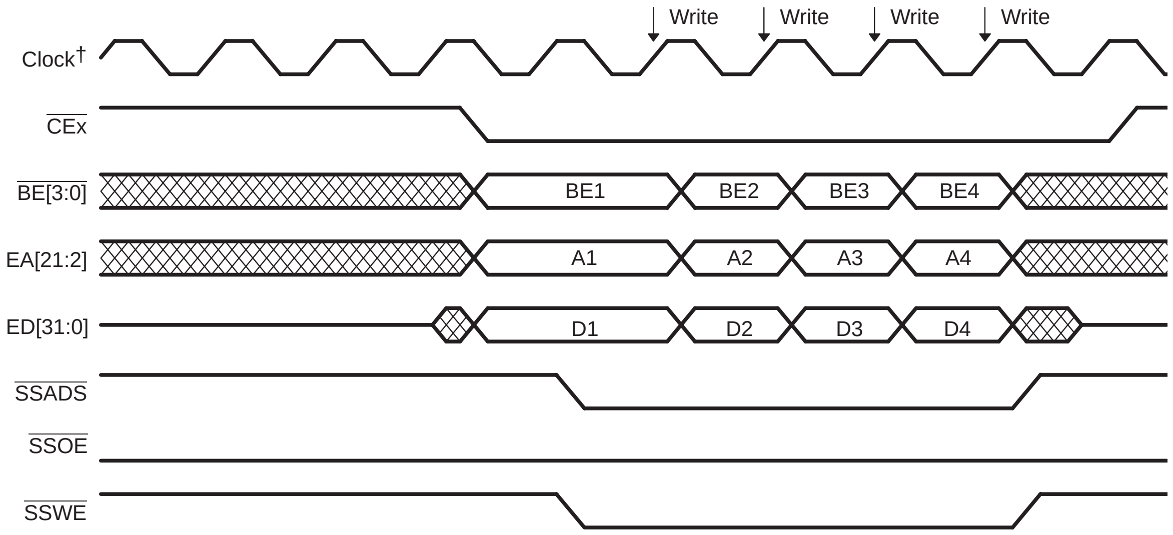
Figure 9–33. TMS320C6211/C6711 SBSRAM Six-Word Read



9.5.2 SBSRAM Writes

Figure 9–34 shows a four-word write to an SBSRAM. Every access strobes a new address into the SBSRAM. The first access requires an initial start-up penalty of two cycles; thereafter, all accesses can occur in a single SSCLK cycle.

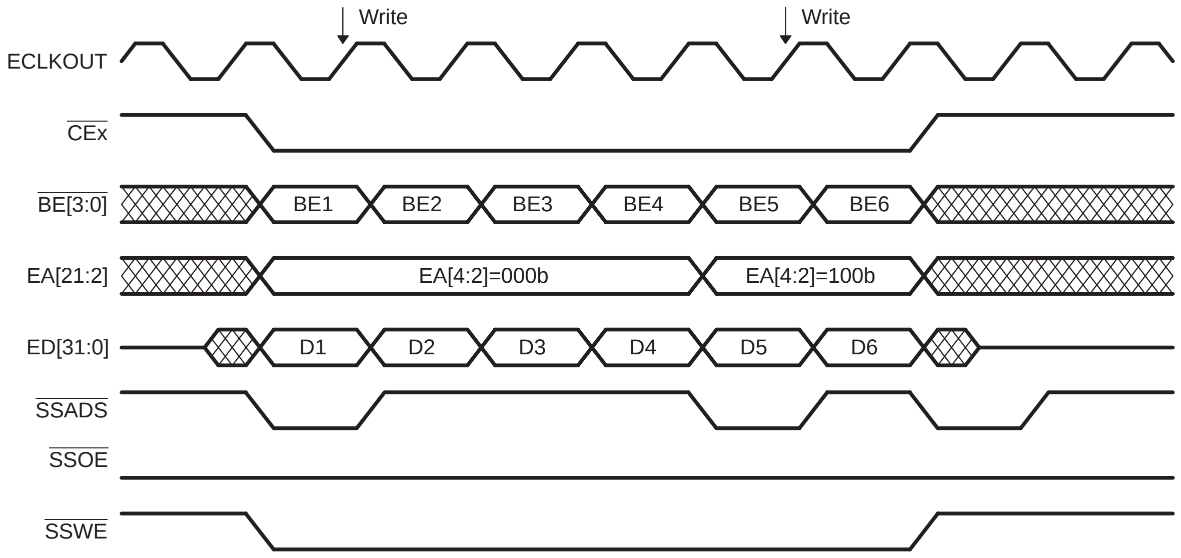
Figure 9–34. TMS320C6201/C6202/C6701 SBSRAM Four Word Write



† Clock=SSCLK for ’C6201/C6701.
Clock=CLKOUT2 for ’C6202.

Figure 9–35 shows a ’C6211/C6711 six-word write to SBSRAM. The new address is strobed into SBSRAM on the fifth cycle to prevent the internal burst counter from rolling over to 000b.

Figure 9–35. TMS320C6211/C6711 SBSRAM Write



9.6 Asynchronous Interface

The asynchronous interface offers configurable memory cycle types to interface to a variety of memory and peripheral types, including SRAM, EPROM, and flash memory, as well as FPGA and ASIC designs.

Table 9–18 lists the asynchronous interface pins.

Figure 9–36 shows an interface to standard SRAM, and Figure 9–38, Figure 9–39, and Figure 9–40 show interfaces to 8-, 16-, and 32-bit ROM for the ’C6201/C6202/C6701 and for the ’C6211/C6711 in little-endian mode. Although ROM can be interfaced at any of the CE spaces, it is often used at CE1 because that space can be configured for widths of less than 32 bits on the ’C6201/C6202/C6701. The ’C6211/C6711 allows 8/16 bit asynchronous mode in any CE space. Figure 9–37 shows the ’C6211/C6711 interface to 16-bit asynchronous SRAM in big endian mode. The only difference is that ED[31:16] pins are used instead of ED[15:0]. The asynchronous interface signals on the ’C6211/C6711 are similar to the ’C6201, except that the signals have been combined with the SDRAM and SBSRAM memory interface. It has also been enhanced to allow for longer read hold and write hold times, and the 8- and 16-bit interface modes have been extended to include writable asynchronous memories, instead of ROM devices. A programmable turnaround time (TA) also allows the user to control the number of cycles between a read and a write to avoid bus contention.

Table 9–18. EMIF Asynchronous Interface Pins

EMIF Signal	Function
$\overline{\text{AOE}}$	Output enable. Active (low) during the entire period of a read access.
$\overline{\text{AWE}}$	Write enable. Active (low) during a write transfer strobe period.
$\overline{\text{ARE}}$	Read enable. Active (low) during a read transfer strobe period.
ARDY	Ready. Input used to insert wait states into the memory cycle.

Figure 9–36. TMS6201/C6202/C6701 EMIF to 32-bit SRAM Interface

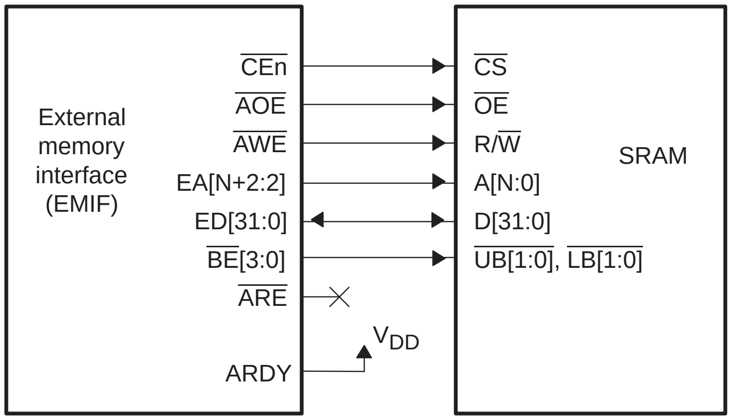


Figure 9–37. TMS320C6211/C6711 EMIF to 16-bit SRAM (Big Endian)

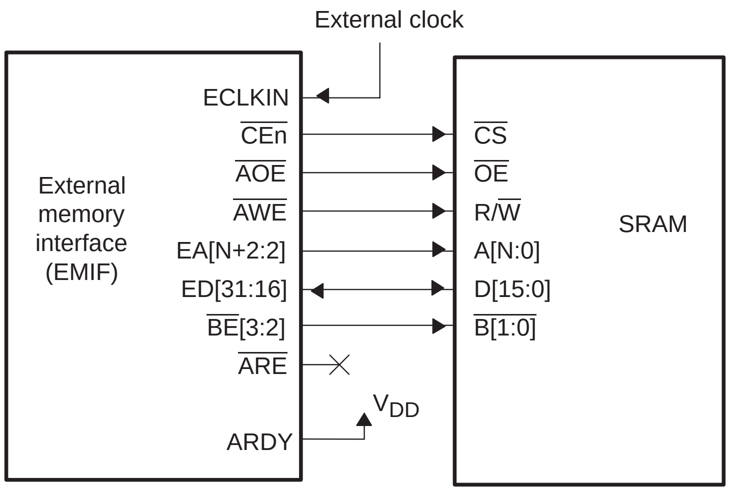


Figure 9–38. EMIF to 8-Bit ROM Interface

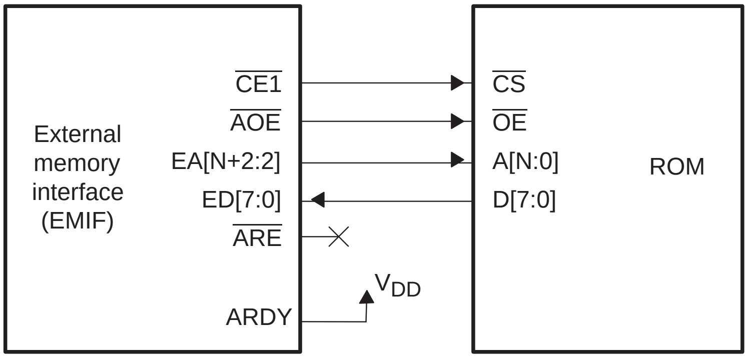


Figure 9–39. EMIF to 16-Bit ROM Interface

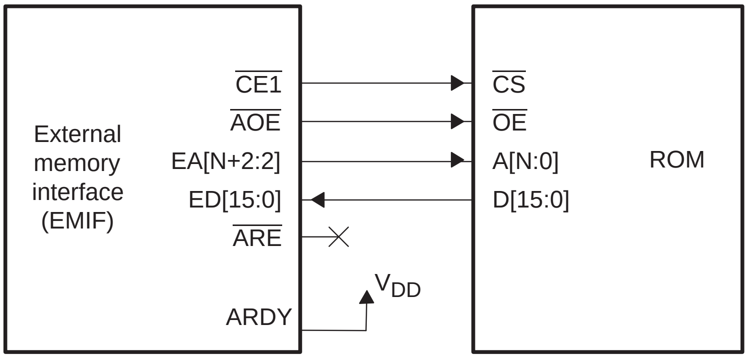
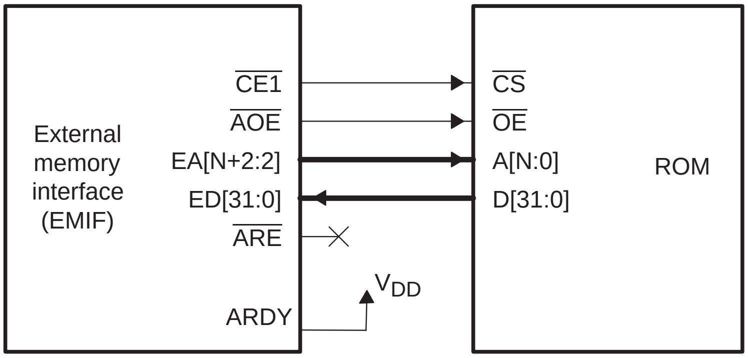


Figure 9–40. EMIF to 32-Bit ROM Interface



9.6.1 TMS320C6201/C6202/C6701 ROM Modes

The EMIF supports 8- and 16-bit-wide ROM access modes which are selected by the MTYPE field in the EMIF CE space control registers. In reading data from these narrow memory spaces, the EMIF packs multiple reads into one 32-bit-wide value. This mode is primarily intended for word accesses to 8-bit and 16-bit ROM devices. The following restrictions apply:

- ❑ Read operations always read 32 bits, regardless of the access size or the memory width.
- ❑ The address is shifted up appropriately to provide the correct address to the narrow memory. The shift amount is 1 for 16-bit ROM and 2 for 8-bit ROM. Thus, the high address bits are shifted out, and accesses wrap around if the CE space spans the entire EA bus. Table 9–19 shows the address bits on the EA bus during an access to CE1 space for all possible asynchronous memory widths.
- ❑ The EMIF always reads the lower addresses first and packs these into the LSbytes. It packs subsequent accesses into the higher order bytes. Thus, the expected packing format in ROM is always little-endian, regardless of the value of the LENDIAN bit.

Table 9–19. Byte Address to EA Mapping for Asynchronous Memory Widths

	EA Line																				
	22	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Width	**Logical Byte Address**																				
×32	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
×16	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
×8	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

9.6.1.1 8-Bit ROM Mode

In 8-bit ROM mode, the address is left-shifted by 2 to create a byte address on EA to access byte-wide ROM. The EMIF always packs four consecutive bytes aligned on a 4-byte boundary (byte address = 4N) into a word access. The bytes are fetched in the following address order: 4N, 4N + 1, 4N + 2, 4N + 3. Bytes are packed into the 32-bit word from MSByte to LSByte in the following little endian order: 4N + 3, 4N + 2, 4N + 1, 4N.

9.6.1.2 16-Bit ROM Mode

In 16-bit ROM mode, the address is left-shifted by 1 to create a half-word address on EA to access 16-bit-wide ROM. The EMIF always packs two consecutive half-words aligned on a 4-byte boundary (byte address = 4N) into a word access. The halfwords are fetched in the following address order: 4N, 4N + 2. Halfwords are packed into the 32-bit word from the most significant halfword to the least significant halfword in the following little-endian order: 4N + 2, 4N.

9.6.2 Programmable ASRAM Parameters

The EMIF allows a high degree of programmability for shaping asynchronous accesses. The programmable parameters that allow this are:

- **Setup:** The time between the beginning of a memory cycle ($\overline{CE}$ low, address valid) and the activation of the read or write strobe
- **Strobe:** The time between the activation and deactivation of the read ($\overline{ARE}$) or write strobe ($\overline{AWE}$)
- **Hold:** The time between the deactivation of the read or write strobe and the end of the cycle (which can be either an address change or the deactivation of the $\overline{CE}$ signal)

For the 'C6201/C6202/C6701 these parameters are programmable in terms of CPU clock cycles via fields in the EMIF CE space control registers. For the 'C6211/C6711, these parameters are programmed in terms of ECLKOUT cycles. Separate set-up, strobe, and hold timing parameters are available for read and write accesses. Minimum values for ASRAM are as follows:

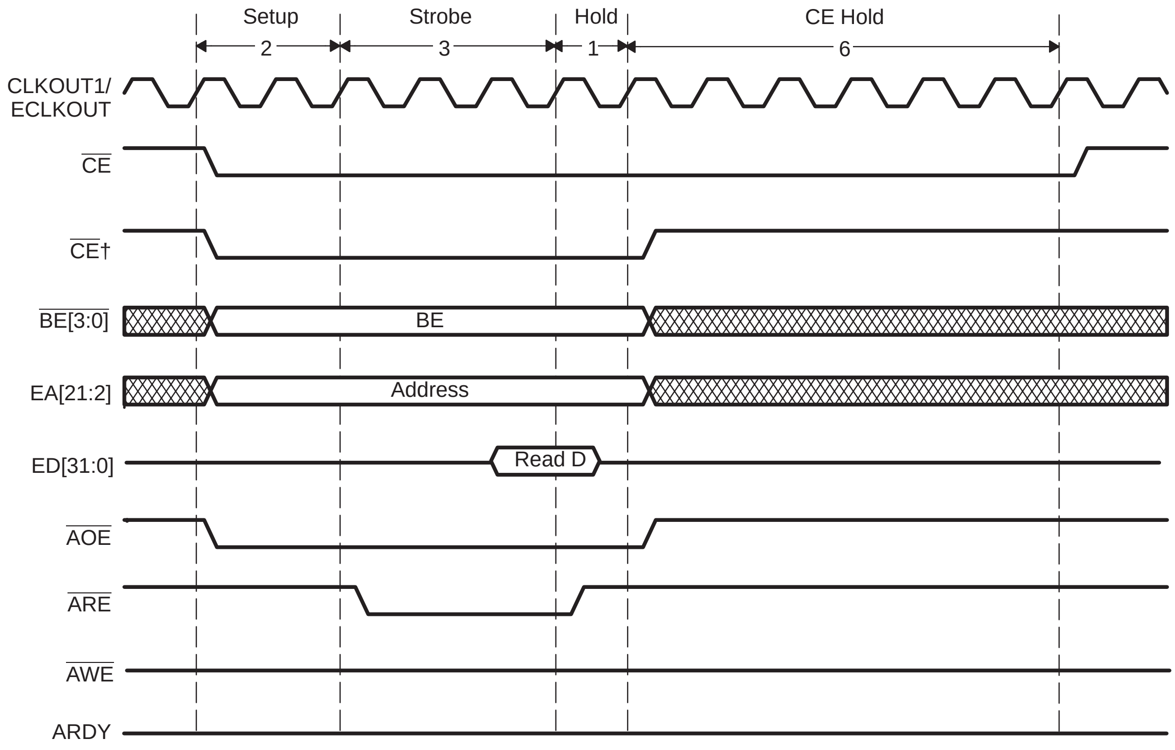
- SETUP ≥ 1 (0 treated as 1)
- STROBE ≥ 1 (0 treated as 1)
- HOLD ≥ 0
- On the 'C6201/C6202/C6701 first access in a set of consecutive accesses or a single access, the setup period has a minimum count of 2.

9.6.3 Asynchronous Reads

Figure 9–41 show an asynchronous read with the setup, strobe, and hold parameter programmed with the values 2,3, and 1, respectively. An asynchronous read proceeds as follows:

- At the beginning of the setup period:
  - $\overline{\text{CE}}$ becomes active.
  - $\overline{\text{AOE}}$ becomes active.
  - $\overline{\text{BE[3:0]}}$ becomes valid.
  - EA becomes valid.
- At the beginning of a strobe period, $\overline{\text{ARE}}$ becomes active
- At the beginning of a hold period:
  - $\overline{\text{ARE}}$ becomes inactive (high).
  - Data is sampled on the CLKOUT1 on the ECLKOUT rising edge concurrent with the beginning of the hold period (the end of the strobe period) and just prior to the $\overline{\text{ARE}}$ low-to-high transition.
- At the end of the hold period: $\overline{\text{AOE}}$ becomes inactive as long as another read access to the same CE space is not scheduled for the next cycle.
- For the ’C6201/C6202/C6701, CE stays active for seven minus the value of Read Hold cycles after the last access (DMA transfer or CPU access). For example, if read HOLD = 1, then CE stays active for six more cycles. This does not affect performance and merely reflects the EMIF’s overhead.
- For the ’C6211/C6711, the $\overline{\text{CEn}}$ signal goes high just after the programmed hold period.

Figure 9–41. Asynchronous Read Timing Example



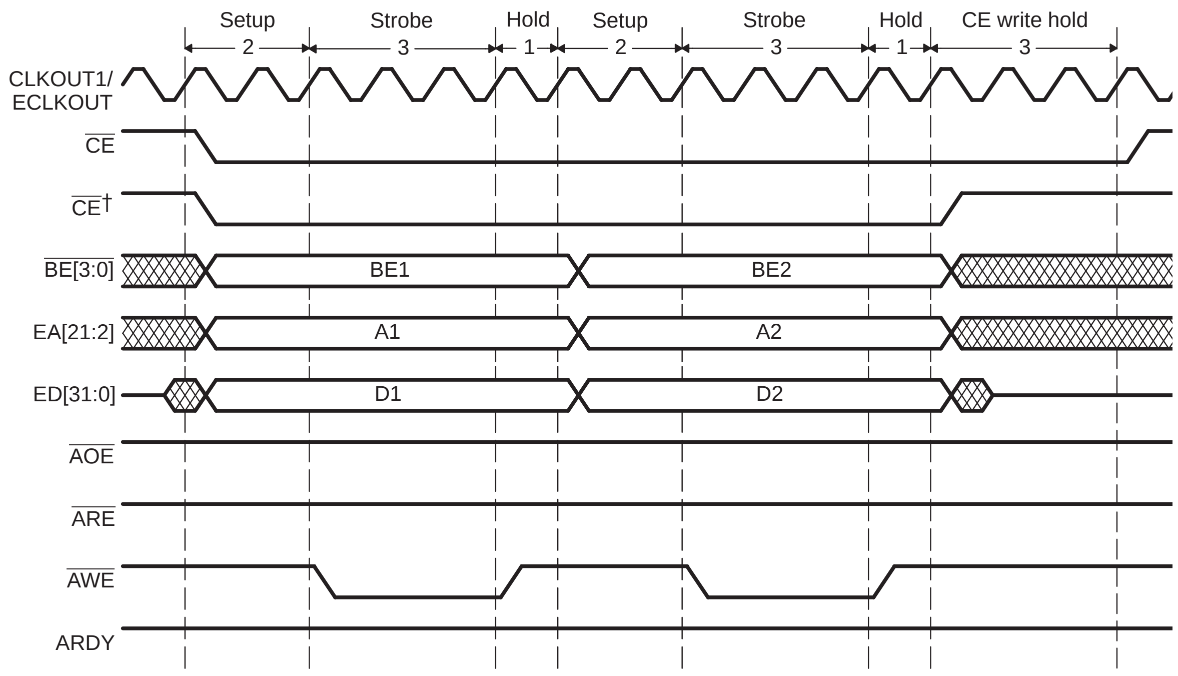
† On the ’C6211/C6711, $\overline{CE}$ goes high immediately after the programmed hold period.
‡ CLKOUT1 referenced for ’C6201/C6202/C6701, ECLKOUT reference for ’C6211/C6711

9.6.4 Asynchronous Writes

Figure 9–42 shows two back-to-back asynchronous write cycles with the ARDY signal pulled high (always ready). The $\overline{\text{SETUP}}$, STROBE and HOLD are programmed to 2,3,and 1.

- At the beginning of the setup period:
  - $\overline{\text{CE}}$ becomes active.
  - $\overline{\text{BE[3:0]}}$ becomes valid.
  - EA becomes valid.
  - ED becomes valid.
  - For the first access, setup has a minimum value of 2. After the first access, setup has a minimum value of 1.
- At the beginning of a strobe period, $\overline{\text{AWE}}$ becomes active.
- At the beginning of a hold period:
  - $\overline{\text{AWE}}$ becomes inactive.
- At the end of the hold period:
  - ED goes into the high-impedance state only if another write access to the same CE space is not scheduled for the next cycle.
  - $\overline{\text{CE}}$ becomes inactive only if another write access to the same CE space is not scheduled for the next cycle.
- For the ’C6201/C6202/C6701, if no write accesses are scheduled for the next cycle and write hold is set to 1 or greater, then CE stays active for 3 cycles after the value of the programmed hold period. If write hold is set to 0, then CE stays active ffom four more cycles. This does not affect performance and merely reflects the EMIF’s overhead.
- For the ’C6211/C6711, the $\overline{\text{CEn}}$ signal goes high immediately after the programmed hold period.

Figure 9–42. Asynchronous Write Timing Example



† On the ’C6211/C6711, $\overline{\text{CE}}$ goes high immediately after the programmed hold period.
‡ CLKOUT1 referenced for ’C6201/C6202/C6701, ECLKOUT reference for ’C6211/C6711

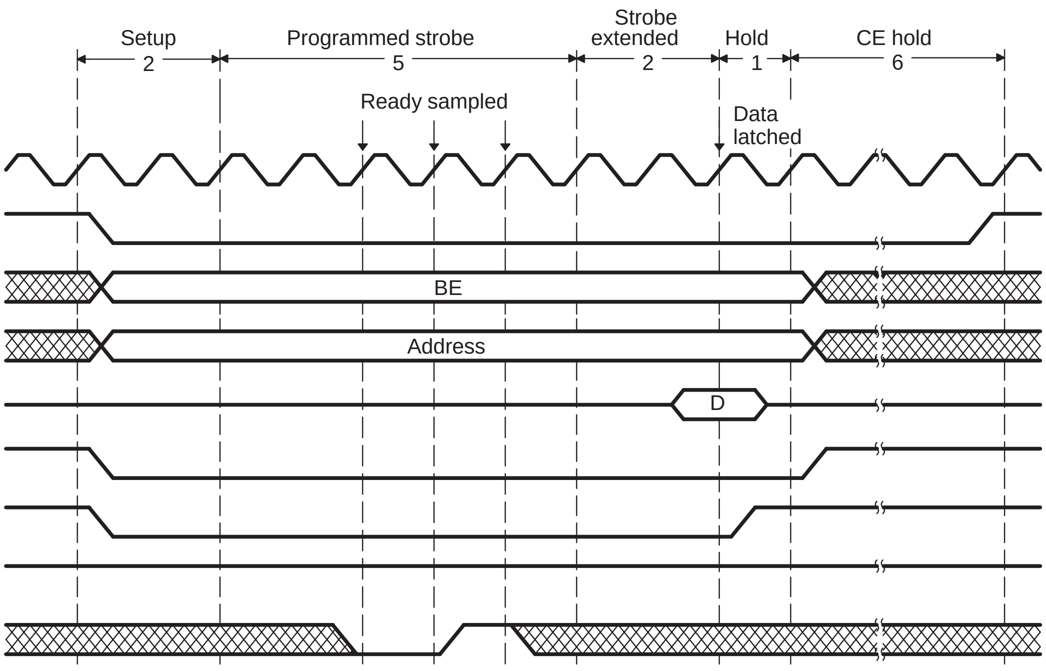
9.6.5 Ready Input

In addition to programmable access shaping, you can insert extra cycles into the strobe period by deactivating the ARDY input. The ready input is internally synchronized to the CPU clock. This synchronization allows an asynchronous ARDY input while avoiding metastablility.

- **TMS320C6201/C6202/C6701 Operation:** If ARDY is low on the third rising edge of CLKOUT1 before the end of the programmed strobe period, then the strobe period is extended by one CLKOUT1 cycle. For each subsequent CLKOUT1 rising edge that ARDY is sampled low, the strobe period is extended by one CLKOUT1 cycle. Thus to effectively use CE to generate ARDY inactive with external logic the minimum of SETUP and STROBE should be four.

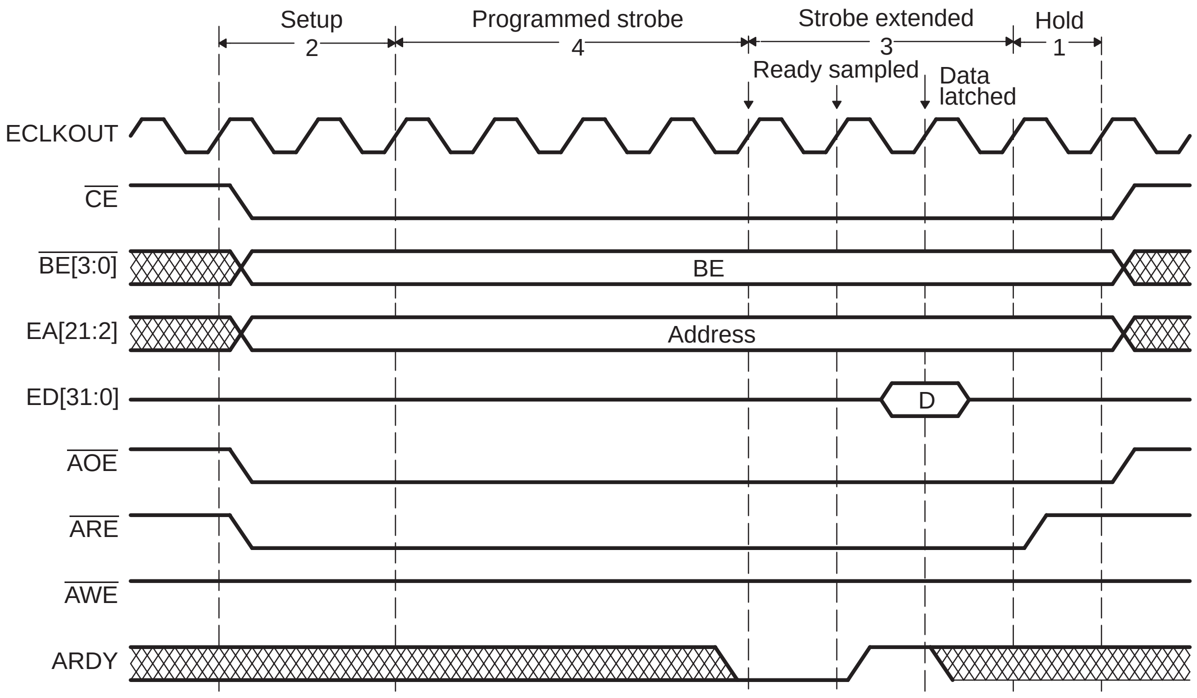
The read cycle in Figure 9–43 illustrates ready operation for the ’C6201/C6202/C6701.

Figure 9–43. TMS320C6201/C6202/C6701 Ready Operation



- T**MS320C6211/C6711 Operation:** ARDY is sampled for the first time on the ECLKOUT cycle at the end of the programmed strobe period. If sampled low, the strobe period is extended and ARDY is sampled again on the next ECLKOUT cycle. Read data is latched by the ’C6211 on the cycle that ARDY is sampled high. The $\overline{\text{ARE}}$ signal goes high on the the following cycle. Therefore, the strobe period is visibly extended by three cycles in Figure 9–44, although data is latched by the ’C6211 after the second cycle.

Figure 9–44. TMS320C6211/C6711 Ready Operation



9.7 Hold Interface

The EMIF responds to hold requests for the external bus. The hold handshake allows an external device and the EMIF to share the external bus. The handshake mechanism uses two signals:

- $\overline{\text{HOLD}}$: hold request input. $\overline{\text{HOLD}}$ is synchronized internally to the CPU clock. This synchronization allows an asynchronous input while avoiding metastability. The external device drives this pin low to request bus access. $\overline{\text{HOLD}}$ is the highest priority request that the EMIF can receive during active operation. When the hold is requested, the EMIF stops driving the bus at the earliest possible moment, which may entail completion of the current accesses, device deactivation, and SDRAM bank deactivation. The external device must continue to drive $\overline{\text{HOLD}}$ low for as long as it wants to drive the bus. If any memory spaces are configured for SDRAM, these memory spaces are deactivated and refreshed after $\overline{\text{HOLD}}$ is released by the external master.
- $\overline{\text{HOLDA}}$: Hold acknowledge output. The EMIF asserts this signal active after it has placed its signal outputs in the high-impedance state. The external device can then drive the bus as required. The EMIF places all outputs in the high-impedance state with the exception of the clock outputs: CLKOUT1, CLKOUT2, SDCLK, and/or SSCLK, depending on the device. If any memory spaces are configured for SDRAM, these memory spaces are deactivated and refreshed before $\overline{\text{HOLDA}}$ is asserted to the external master.
- BUSREQ. Bus request output ('C6211/C6711 only). The EMIF asserts this signal active when any request is either pending to the EMIF or is in progress. The BUSREQ signal is driven without regard to the state of the $\overline{\text{HOLD}}$/$\overline{\text{HOLDA}}$ signals or the type of access pending. This signal can be used by an external master to release control of the bus if desired and may be ignored in some systems.

Note:

There is no mechanism to ensure that the external device does not attempt to drive the bus indefinitely. You should be aware of system-level issues, such as refresh, that you may need to perform.

During host requests, the refresh counters within the EMIF continue to log refresh requests; however, no refresh cycles can be performed until bus control is again granted to the EMIF when the $\overline{\text{HOLD}}$ input returns to the inactive level. You can prevent an external hold by setting the NOHOLD bit in the EMIF global control register.

9.8 Memory Request Priority

9.8.1 TMS320C6201/C6202/C6701 Memory Request Priority

The ’C6201/C6202/C6701 EMIF has multiple requestors competing for the interface. Table 9–20 summarizes the priority scheme that the EMIF uses in the case of multiple pending requests. The priority scheme may change if the DMA channel that is issuing a request through the DMA controller is of high priority. This mode is set in the DMA controller by setting the PRI bit in the DMA channel primary control register.

Once a requester (in this instance, the refresh controller is considered a requester) is prioritized and chosen, no new requests are recognized until either the chosen requester stops making requests or a subsequent higher priority request occurs. In this case, all issued requests of the previous requester are allowed to finish while the new requester starts making its requests.

If the arbitration bit of the EMIF global control register is set (RBTR8 = 1) and if a higher priority requester needs the EMIF, the higher priority requester does not gain control until the current controller relinquishes control or until eight word requests have finished. If the arbitration bit is not set (RBTR8 = 0), a requester maintains control of the EMIF as long as it needs the EMIF or until a higher priority requester requests the EMIF. When the RBTR8 is not set, the current controller is interrupted by a higher priority requester regardless of the number of requests that have occurred.

Table 9–20. TMS320C6201/C6202/C6701 EMIF Prioritization of Requests

Priority	Requestor PRI = 1	Requestor PRI = 0
Highest	External hold	External hold
	Mode register set	Mode register set
	Urgent refresh	Urgent refresh
	DMA controller	DMC
	DMC	PMC
	PMC	DMA controller
Lowest	Trickle refresh	Trickle refresh

9.8.2 TMS320C6211/C6711 Memory Request Priority

The ’C6211/C6711 has fewer interface requestors because the data memory controller (DMC), program memory controller (PMC), and EDMA transactions are processed by the EDMA. Other requestors include the hold interface and internal EMIF operations, including mode register set (MRS) and refresh (REFR).

Table 9–21. TMS320C6211/C6711 EMIF Prioritization of Requests

Priority	**Requestor**
Highest	External hold
	Mode register set
	refresh
	EDMA – DMC
	EDMA – PMC
Lowest	EDMA – DMA

9.9 Boundary Conditions When Writing to EMIF Registers

The EMIF has internal registers that change memory type, asynchronous memory timing, SDRAM refresh, SDRAM initialization (MRS COMMAND), clock speed, arbitration type, HOLD/NOHOLD condition, etc.

The following actions can cause improper data reads or writes:

- Writing to the CE0, CE1, CE2, or CE3 space control registers while an external access to that CE space is active
- Changing the memory type (MTYPE) in the CE space control register while any external operation is in progress (SDRAM type while SDRAM initialization is active)
- Changing the state of NOHOLD in the configuration while HOLD is active at the pin
- Changing the RBTR8 in the EMIF global control register while multiple EMIF requests are pending
- Initiating an SDRAM INIT (MRS) while the $\overline{\text{HOLD}}$ input or the $\overline{\text{HOLDA}}$ output is active
  - The EMIF global control register can be read before the SDRAM INIT bit is set to determine if the HOLD function is active, and it must be read immediately after the SDRAM INIT bit is written to make sure that the two events did not occur simultaneously.
  - The EMIF global control register has status on the HOLD/HOLDA, DMC/PMC/DMA active access and false access detection.

9.10 Clock Output Enabling

To reduce electromagnetic interference (EMI) radiation, the EMIF allows the disabling (holding high) of CLKOUT2, CLKOUT1, SSCLK, and SDCLK. This disabling is performed by setting the CLK2EN, CLK1EN, SSCEN, and SDCEN bits to 0 in the EMIF global control register, which is shown in Figure 9–6 on page 9-9 and summarized in Table 9–3 on page 9-10.ECLKOUT cannot be disabled using software.

9.11 Emulation Halt Operation

The EMIF continues operating during emulation halts. Emulator accesses through the EMIF can work differently than the way the actual device works during EMIF accesses. This discrepancy can cause start-up penalties after a halt operation.

9.12 Power Down

In power-down 2 mode, refresh is enabled. SSCLK, CLKOUT1, and CLKOUT2 are held low during power-down 2 and power-down 3 modes. In power-down 3 mode, the EMIF acts as if it were in reset. See Chapter 14, *Power-Down Logic,* for further details on power-down modes.

For the ’C6211/C6711, refreshes are issued to SDRAM if ECLKIN is provided.

Chapter 10

Boot Modes and Configuration

This chapter describes the boot modes and device configuration used by the TMS320C6000 platform. It also describes the available boot processes and explains how the device is reset.

Topic **Page**

10.1 Overview

The TMS320C6000 platform uses a variety of boot configurations to determine what actions the devices are to perform after reset for proper device initialization. Each ‘C6000 device has some or all of the following boot configuration options:

- Selection of the memory map, which determines whether internal or external memory is mapped at address 0
- Selection of the type of external memory mapped at address 0 if external memory is mapped there
- Selection of the boot process used to initialize the memory at address 0 before the CPU is released from reset.

10.2 Device Reset

The external device reset uses an active (low) signal, $\overline{\text{RESET}}$. While $\overline{\text{RESET}}$ is low, the device is held in reset and is initialized to the prescribed reset state. All 3-state outputs are placed in the high-impedance state, and all other outputs are returned to their default states. The rising edge of $\overline{\text{RESET}}$ starts the processor running with the prescribed boot configuration. The $\overline{\text{RESET}}$ pulse may have to be increased if the phase-locked loop (PLL) requires synchronization following power up or when PLL configuration pins change during reset.

10.3 Boot Configuration

External pins BOOTMODE[4:0] select the boot configuration. The values of BOOTMODE[4:0] are latched during the low period of $\overline{\text{RESET}}$. Table 10–1 lists all the values for BOOTMODE[4:0] as well as the associated memory maps and boot processes. For example, the value 00000b on BOOTMODE[4:0] selects memory map 0 and indicates that the memory type at address 0 is synchronous DRAM organized as four 8-bit-wide banks and that no boot process is selected. SDWID is a bit in the EMIF SDRAM control register.

Table 10–1. Boot Configuration Summary

BOOTMODE [4:0]	Memory Map	Memory at Address 0	Boot Process
00000	MAP 0	SDRAM: four 8-bit devices (SDWID = 0)	None
00001	MAP 0	SDRAM: two 16-bit devices (SDWID = 1)	None
00010	MAP 0	32-bit asynchronous with default timing	None
00011	MAP 0	1/2x rate SBSRAM	None
00100	MAP 0	1x rate SBSRAM	None
00101	MAP 1	Internal	None
00110	MAP 0	External: default values	HPI
00111	MAP 1	Internal	HPI
01000	MAP 0	SDRAM: four 8-bit devices (SDWID = 0)	8-bit ROM with default timings
01001	MAP 0	SDRAM: two16-bit devices (SDWID = 1)	8-bit ROM with default timings
01010	MAP 0	32-bit asynchronous with default timing	8-bit ROM with default timings
01011	MAP 0	1/2x rate SBSRAM	8-bit ROM with default timings
01100	MAP 0	1x rate SBSRAM	8-bit ROM with default timings
01101	MAP 1	Internal	8-bit ROM with default timings
01110		Reserved	
01111		Reserved	
10000	MAP 0	SDRAM: four 8-bit devices(SDWID=0)	16-bit ROM with default timings
10001	MAP 0	SDRAM: two 16-bit devices (SDWID = 1)	16-bit ROM with default timings

Table 10–1. Boot Configuration Summary (Continued)

BOOTMODE [4:0]	Memory Map	Memory at Address 0	Boot Process
10010	MAP 0	32-bit asynchronous with default timing	16-bit ROM with default timings
10011	MAP 0	1/2x rate SBSRAM	16-bit ROM with default timings
10100	MAP 0	1x rate SBSRAM	16-bit ROM with default timings
10101	MAP 1	Internal	16-bit ROM with default timings
10110		Reserved	
10111		Reserved	
11000	MAP 0	SDRAM: four 8-bit devices (SDWID = 0)	32-bit ROM with default timings
11001	MAP 0	SDRAM: two 16-bit devices (SDWID = 1)	32-bit ROM with default timings
11010	MAP 0	32-bit asynchronous with default timing	32-bit ROM with default timings
11011	MAP 0	1/2x rate SBSRAM	32-bit ROM with default timings
11100	MAP 0	1x rate SBSRAM	32-bit ROM with default timings
11101	MAP 1	Internal	32-bit ROM with default timings
11110		Reserved	
11111		Reserved	

The TMS320C6201 and ’C6701 devices latch their boot configuration setting at reset from dedicated BOOTMODE pins.

The TMS3206202 latches its boot configuration from five data lines of the expansion bus, XD[4:0]. The XD[4:0] lines directly map to BOOTMODE[4:0], and should be configured using external pull-up and pull-down resistors.

The TMS320C6211/C6711 latches its boot configuration from the host-port data lines. Only two of the five BOOTMODE bits are required because the ’C6211/C6711 only has one memory map, which places internal memory at address 0. The HD[4:3] pins map to the BOOTMODE[4:3] pins. The complete boot configuration shown in Table 10–1 can be significantly reduced for the ’C6211/C6711 as shown in Table 10–2. External pull-down resistors should be used on HD[4:3] to configure the boot mode.

Table 10–2. TMS320C6211/C6711 Boot Configuration Summary

BOOTMODE[4:0]	Boot Process
00xxx	Host-port interface
01xxx	8-bit ROM with default timings
10xxx	16-bit ROM with default timings
11xxx	32-bit ROM with default timings

10.3.1 Memory Map

The two memory maps of the 'C6201 and 'C6701, MAP 0 and MAP 1, are summarized in Table 10–3. They differ in that MAP 0 has external memory mapped at address 0, and MAP 1 has internal memory mapped at address 0. Refer to Chapter 2 and Chapter 3 for program and data memory descriptions.

Table 10–3. TMS320C6201/C6701 Memory Map Summary

Address Range (Hex)	Size (Bytes)	Description of Memory Block in ...	
		MAP 0	MAP 1
0000 0000 – 0000 FFFF	64K	External memory interface CE 0	Internal program RAM
0001 0000 – 003F FFFF	4M–64K	External memory interface CE 0	Reserved
0040 0000 – 00FF FFFF	12M	External memory interface CE 0	External memory interface CE 0
0100 0000 – 013F FFFF	4M	External memory interface CE 1	External memory interface CE 0
0140 0000 – 0140 FFFF	64K	Internal program RAM	External memory interface CE 1
0141 0000 – 017F FFFF	4M–64K	Reserved	External memory interface CE 1
0180 0000 – 0183 FFFF	256K	Internal peripheral bus EMIF registers	
0184 0000 – 0187 FFFF	256K	Internal peripheral bus DMA controller registers	
0188 0000 – 018B FFFF	256K	Internal peripheral bus HPI register	
018C 0000 – 018F FFFF	256K	Internal peripheral bus McBSP 0 registers	
0190 0000 – 0193 FFFF	256K	Internal peripheral bus McBSP 1 registers	
0194 0000 – 0197 FFFF	256K	Internal peripheral bus Timer 0 registers	
0198 0000 – 019B FFFF	256K	Internal peripheral bus Timer 1 registers	
019C 0000 – 019F FFFF	256K	Internal peripheral bus interrupt selector registers	
01A0 0000 – 01FF FFFF	6M	Internal peripheral bus (reserved)	
0200 0000 – 02FF FFFF	16M	External memory interface CE 2	
0300 0000 – 03FF FFFF	16M	External memory interface CE 3	
0400 0000 – 7FFF FFFF	2G–64M	Reserved	
8000 0000 – 803F FFFF	64K	Internal data RAM	
8040 0000 – FFFF FFFF	2G–64K	Reserved	

The ’C6202 has two memory maps that are supersets of the ’C6201/’C6701 memory maps. All valid ’C6201/’C6701 address ranges are valid on the ’C6202. There are three primary differences: the ’C6202 has larger internal memory spaces, four external memory locations for the expansion bus (XCE[3:0]), and a third serial port. The memory maps for the ‘C6202 are shown in Table 10–4.

Table 10–4. TMS320C6202 Memory Map Summary

	Size (Bytes)	**Description of Memory Block In…**	
Address Range (Hex)		**MAP 0**	**MAP 1**
0000 0000–003F FFFF	256K	External memory interface CE0	Internal program RAM
0004 0000–003F FFFF	4M–256K	External memory interface CE0	Reserved
0040 0000–00FF FFFF	12M	External memory interface CE0	External memory interface CE0
0100 0000–013F FFFF	4M	External memory interface CE1	External memory interface CE0
0140 0000–0143 FFFF	256K	Internal program RAM	External memory interface CE1
0144 0000–017F FFFF	4M–256K	Reserved	External memory interface CE1
0180 0000–0183 FFFF	256K	Internal peripheral bus EMIF registers	
0184 0000–0187 FFFF	256K	Internal peripheral bus DMA controller registers	
0188 0000–018B FFFF	256K	Internal peripheral bus expansion bus registers	
018C 0000–018F FFFF	256K	Internal peripheral bus McBSP 0 registers	
0190 0000–0193 FFFF	256K	Internal peripheral bus McBSP 1 registers	
0194 0000–0197 FFFF	256K	Internal peripheral bus timer 0 registers	
0198 0000–019B FFFF	256K	Internal peripheral bus timer 1 registers	
019C 0000–019C 01FF	512	Internal peripheral bus interrupt selector registers	
019C 0200–019C FFFF	256K–512	Internal peripheral bus power-down registers	
01A0 0000–01A3 FFFF	256K	Reserved	
01A4 0000–01A7 FFFF	256K	Internal peripheral bus McBSP2 registers	
01A8 0000–01FF FFFF	5.5M	Reserved	
0200 0000–02FF FFFF	16M	External memory interface CE2	
0300 0000–03FF FFFF	16M	External memory interface CE3	
0400 0000–3FFF FFFF	1G–64M	Reserved	
4000 0000–4FFF FFFF	256M	Expansion bus XCE0	
5000 0000–5FFF FFFF	256M	Expansion bus XCE1	
6000 0000–6FFF FFFF	256M	Expansion bus XCE2	
7000 0000–7FFF FFFF	256M	Expansion bus XCE3	
8000 0000–8001 FFFF	128K	Internal data RAM	
8000 2000–FFFF FFFF	1G–128K	Reserved	

The ‘C6211 and ’C6711 have only one memory map, which is shown in Table 10–5. Internal memory is always located at address 0, but can be used as both program and data memory. The configuration register for those peripherals common to the ’C6201, ’C6211, and ’C6711 are located at the same addresses in both processors. The external memory address ranges begin at 8000 0000h in the ’C6211/C6711, which is the location of internal data memory in the ’C6201.

Table 10–5. TMS320C6211/C6711 Memory Map Summary

Address Range (Hex)	Size (Bytes)	Description of Memory Block
0000 0000–0000 FFFF	64K	Internal RAM (L2)
0001 0000–017F FFFF	24M–64K	Reserved
0180 0000–0183 FFFF	256K	Internal configuration bus EMIF registers
0184 0000–0187 FFFF	256K	Internal configuration bus L2 control registers
0188 0000–018B FFFF	256K	Internal configuration bus HPI register
018C 0000–018F FFFF	256K	Internal configuration bus McBSP 0 registers
0190 0000–0193 FFFF	256K	Internal configuration bus McBSP 1 registers
0194 0000–0197 FFFF	256K	Internal configuration bus timer 0 registers
0198 0000–019B FFFF	256K	Internal configuration bus timer 1 registers
019C 0000–019F FFFF	256K	Internal configuration bus interrupt selector registers
01A0 0000–01A3 FFFF	256K	Internal configuration bus EDMA RAM and registers
01A4 0000–1FFF FFFF	1G–288M	Reserved
3000 0000–2FFF FFFF	256M	McBSP 0/1 data
4000 0000–3FFF FFFF	1G	Reserved
8000 0000–8FFF FFFF	256M	External memory interface CE0
9000 0000–9FFF FFFF	256M	External memory interface CE1
A000 0000–AFFF FFFF	256M	External memory interface CE2
B000 0000–BFFF FFFF	256M	External memory interface CE3
C000 0000–FFFF FFFF	1G	Reserved

10.3.2 Memory at Reset Address

For ’C6000 processors with multiple memory maps, the boot configuration determines the type of memory located at the reset address for processor operation, address 0 as shown in Table 10–1. When the BOOTMODE [4:0] pins select MAP 1, this memory is internal. When the device mode is in MAP 0, the memory is external. When external memory is selected, BOOTMODE [4:0] also determine the type of memory at the reset address. These options effectively provide alternative reset values to the appropriate EMIF control registers.

The ‘C6211/C6711 always has internal RAM at address 0, regardless of the BOOTMODE[4:0] configuration.

10.3.3 Boot Processes

The boot process is determined by the BOOTMODE[4:0] pins, as shown in Table 10–1. Up to three types of boot processes are available:

- ❑ No boot process: The CPU begins direct execution from the memory located at address 0. If SDRAM is used in the system, the CPU is held until SDRAM initialization is complete. This feature is not available on the ’C6211/C6711.

- ❑ ROM boot process: The program located in external ROM is copied to address 0 by the DMA/EDMA controller. Although the boot process begins when the device is released from external reset, this transfer occurs while the CPU is internally held in reset. This boot process also lets you choose the width of the ROM. In this case, the EMIF automatically assembles consecutive 8-bit bytes or 16-bit halfwords to form the 32-bit instruction words to be moved. These values are expected to be stored in little endian format in the external memory, typically a ROM device.

  The transfer is automatically done by the DMA/EDMA as a single-frame block transfer from the to ROM address 0.

  After completion of the block transfer, the CPU is removed from reset and allowed to run from address 0.

  The ROM boot process differs slightly between specific ’C6000 devices.

  - ■ **’C6201,’C6202,’C6701:** The DMA copies 32K bytes from CE1 to address 0, using default ROM timings. After the transfer the CPU begins executing from address 0.

  - ■ **’C6211, ’C6711:** THE EDMA copies 1K bytes from the beginning of CE1 to address 0, using default ROM timings. After the transfer the CPU begins executing from address 0.

- ❑ Host boot process: The CPU is held in reset while the remainder of the device is released. During this period, an external host can initialize the CPU’s memory space as necessary through the host interface, including external memory configuration registers. Once the host is finished with all necessary initialization, it must set the DSPINT to complete the boot process. This transition causes the boot configuration logic to remove the CPU from its reset state. The CPU then begins execution from address 0. The DSPINT condition is not latched by the CPU, because it occurs while the CPU is still in reset. Also, DSPINT wakes the CPU from internal reset only if the HPI boot process is selected. All memory may be written to and read by the host. This allows for the host to verify what it sends to the processor, if required.

Note:

The host interface used during host boot varies between different devices, as follows:

- ❑ **’C6201, ’C6701, ’C6211, ’C6711:** The HPI is used for the host boot. The HPI is always a slave interface, and needs no special configuration.
- ❑ **’C6202:** The expansion bus is used for the host boot. The type of host interface is determined by a set of latched signals during rest.

10.4 Device Configuration

Several device settings are configured at reset to determine how the device operates.

10.4.1 Input Clock Mode

The on-chip PLL frequency multiplier is configured through static CLKMODE input pins. Different devices in the ’C6000 platform have different numbers of CLKMODE pins. Only ×1(PLL bypass) and ×4(CLKIN × 4) are supported. The DLL mode selection is shown in Table 10–6.

Table 10–6. DLL Multiplier Select

	’C6201/C6701	**’C6202**	**’C6211/C6711**
PLL Mode	**CLKMODE[1:0]**	**CLKMODE[2:0]**	**CLKMODE0**
×1	00b	000b	0b
×4	11b	001b	1b
Reserved	other	other	

10.4.2 Endian Mode

Each ’C6000 device can be configured to operate in either big or little endian mode. Set the LENDIAN flag to 1 to select little endian, and 0 to select big endian. The selection method varies slightly among different devices. The ’C6201 and ’C6701 have a dedicated LENDIAN input pin. The ’C6211 and ’C6711 sample the ninth data line of the host-port interface, HD[8]. The ’C6202 samples the ninth data line of the expansion bus, XD[8]. The device pin should be configured via a pull-up or pull-down resister.

For more details on endian mode refer to section 2.6.7, *Data Endianness*.

10.4.3 TMS320C6202 Expansion Bus

The expansion bus of the ’C6202 is configured through pull-up an pull-down resistors on the remaining data lines of the expansion bus, XD[31:9] and XD[7:5]. For more details on how to configure the expansion bus, see section 8.7, *Boot Configuration Control via Expansion Bus*.

Chapter 11

Multichannel Buffered Serial Ports

This chapter describes the operation and hardware of the two multichannel buffered serial ports (McBSPs). It also includes register definitions and timing diagrams for the McBSPs.

Topic **Page**

11.1 Features

The multichannel buffered serial port (McBSP) is based on the standard serial port interface on the TMS320C2x, ’C3x, ’C5x, and ’C54x devices. The McBSP provides:

- Full-duplex communication
- Double-buffered data registers, which allow a continuous data stream
- Independent framing and clocking for receive and transmit
- Direct interface to industry-standard codecs, analog interface chips (AICs), and other serially connected A/D and D/A devices
- External shift clock or an internal, programmable frequency shift clock for data transfer
- Autobuffering capability through the 5-channel DMA controller

In addition, the McBSP has the following capabilities:

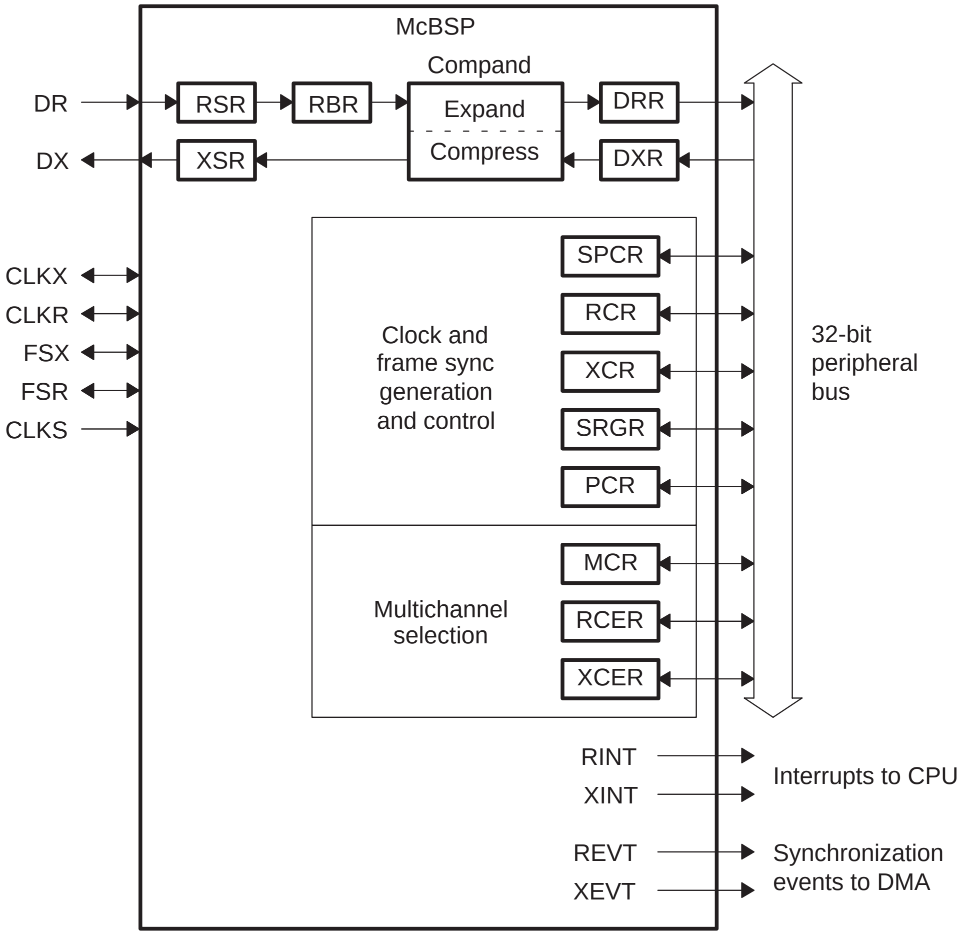
- Direct interface to:
  - T1/E1 framers
  - MVIP switching compatible and ST-BUS compliant devices including:
    - MVIP framers
    - H.100 framers
    - SCSA framers
  - IOM-2 compliant devices
  - AC97 compliant devices. (The necessary multi phase frame synchronization capability is provided.)
  - IIS compliant devices
  - SPI™ devices
- Multichannel transmit and receive of up to 128 channels
- A wide selection of data sizes, including 8, 12, 16, 20, 24, and 32 bits
- µ-Law and A-Law companding
- 8-bit data transfers with the option of LSB or MSB first
- Programmable polarity for both frame synchronization and data clocks
- Highly programmable internal clock and frame generation

11.2 McBSP Interface Signals and Registers

The multichannel buffered serial port (McBSP) consists of a data path and a control path, which connect to external devices. Data is communicated to these external devices via separate pins for transmission and reception. Control information (clocking and frame synchronization) is communicated via four other pins. The device communicates to the McBSP via 32-bit-wide control registers accessible via the internal peripheral bus.

The McBSP consists of a data path and control path, as shown in Figure 11–1. Seven pins listed in Table 11–1 connect the control and data paths to external devices.

Figure 11–1. McBSP Block Diagram



Data is communicated to devices interfacing to the McBSP via the data transmit (DX) pin for transmission and the data receive (DR) pin for reception. Control information (clocking and frame synchronization) is communicated via CLKX, CLKR, FSX, and FSR. The ’C6201/C6701 communicates to the McBSP via 32-bit-wide control registers accessible via the internal peripheral bus (see section 2.7, *Peripheral Bus*). Either the CPU or the DMA controller reads the received data from the data receive register (DRR) and writes the data to be transmitted to the data transmit register (DXR). Data written to the DXR is shifted out to DX via the transmit shift register (XSR). Similarly, receive data on the DR pin is shifted into the receive shift register (RSR) and copied into the receive buffer register (RBR). RBR is then copied to DRR, which can be read by the CPU or the DMA controller. This allows simultaneous internal data movement and external data communications.

The data receive and transmit registers (DRR and DXR) are mapped to locations shown in Table 11–2. For the TMS320C6211/C6711 device, the DRR and DXR are also mapped to memory locations 30000000h–33FFFFFFh (McBSP 0) and 34000000h–3FFFFFFFh (McBSP 1), as shown in Table 11–3. Both the CPU and the EDMA in the TMS320C6211/C6711 device can access the DRR and DXR in all the memory-mapped locations shown in Table 11–3. A write to *any* location in 30000000h – 33FFFFFFh is equivalent to a write to the DXR of McBSP 0 at 018C0004h. A read from *any* location in 30000000h–3FFFFFFh is equivalent to a read from the DRR of McBSP 0 at 018C0000h. Similarly, a read from any location in 34000000h–3FFFFFFFh is equivalent to a read from the DRR of McBSP 1 at 01900000h, while a write to any location in 34000000h–3FFFFFFFh is equivalent to a write to the DXR of McBSP 1 at 01900004h. You have a choice of reading from/writing to the DRR and DXR in either the 3xxxxxxxh or the 018Cxxxxh/0190xxxxh location. Accesses to the 018Cxxxxh and 0190xxxxh locations go through the peripheral bus. Therefore, it is recommended that you set up the EDMA to use the 3xxxxxxxh addresses for serial port servicing in order to free up the peripheral bus for other functions. The McBSP control registers are only mapped to the 018Cxxxxh/0190xxxxh locations.

The remaining registers accessible to the CPU configure the control mechanism of the McBSP. The McBSP registers are listed in Table 11–2. The control block consists of internal clock generation, frame-synchronization signal generation and control of these signals, and multichannel selection. This control block sends notification of important interrupts to the CPU and events to the DMA controller via the four signals shown in Table 11–4.

Table 11–1. McBSP Interface Signals

Pin	I/O/Z	Description
CLKR	I/O/Z	Receive clock
CLKX	I/O/Z	Transmit clock
CLKS	I	External clock
DR	I	Received serial data
DX	O/Z	Transmitted serial data
FSR	I/O/Z	Receive frame synchronization
FSX	I/O/Z	Transmit frame synchronization

Note: I = Input, O = Output, Z = High Impedance

Table 11–2. McBSP Registers

Hex Byte Address					
McBSP 0	McBSP 1	McBSP 2[§]	Abbreviation	McBSP Register Name[†]	Section
–	–	–	RBR	Receive buffer register	11.2
–	–	–	RSR	Receive shift register	11.2
–	–	–	XSR	Transmit shift register	11.2
018C 0000	0190 0000	01A4 0000	DRR	Data receive register[‡¶]	11.2
018C 0004	0190 0004	01A4 0004	DXR	Data transmit register[#]	11.2
018C 0008	0190 0008	01A4 0008	SPCR	Serial port control register	11.2.1
018C 000C	0190 000C	01A4 000C	RCR	Receive control register	11.2.2
018C 0010	0190 0010	01A4 0010	XCR	Transmit control register	11.2.2
018C 0014	0190 0014	01A4 0014	SRGR	Sample rate generator register	11.5.1.1
018C 0018	0190 0018	01A4 0018	MCR	Multichannel control register	11.6.1
018C 001C	0190 001C	01A4 001C	RCER	Receive channel enable register	11.6.3.1
018C 0020	0190 0020	01A4 0020	XCER	Transmit channel enable register	11.6.3.1
018C 0024	0190 0024	01A4 0024	PCR	Pin control register	11.2.1

† The RBR, RSR, and XSR are not directly accessible via the CPU or the DMA controller.
‡ The CPU and DMA controller can only read this register; they cannot write to it.
§ Applicable only to ’C6202 and ’C6203
¶ For the TMS320C6211/C6711, the DRR is also mapped at 30000000–33FFFFFF for McBSP 0, and at 34000000h–3FFFFFFFh for McBSP 1.
# For the TMS320C6211/C6711, the DXR is also mapped at 30000000–33FFFFFF for McBSP 0, and at 34000000h–3FFFFFFFh for McBSP 1.

Table 11–3. TMS320C6211/C6711 Data Receive and Transmit Registers (DRR/DXR) Mapping

	Accessible Via	
Serial Port	Peripheral Bus	EDMA Bus
McBSP 0	0x018C0000	0x30000000–0x33FFFFFF
McBSP 1	0x01900000	0x34000000–0x3FFFFFFF

Table 11–4. McBSP CPU Interrupts and DMA Synchronization Events

Interrupt Name	Description	Section
RINT	Receive interrupt to CPU	11.3.3
XINT	Transmit interrupt to CPU	11.3.3
REVT	Receive synchronization event to the DMA controller	11.3.2.1
XEVT	Transmit synchronization event to the DMA controller	11.3.2.2

11.2.1 Serial Port Configuration

The serial port is configured via the 32-bit serial port control register (SPCR) and the pin control register (PCR) shown in Figure 11–2 and Figure 11–3, respectively. The SPCR and PCR contain McBSP status control bits. Table 11–5 and Table 11–6 summarize the SPCR and the PCR fields, respectively.

The PCR is also used to configure the serial port pins as general purpose inputs or outputs during receiver and/or transmitter reset (for more information see Section 11.8).

Figure 11–2. Serial Port Control Register (SPCR)

31 24	23	22	21 20	19	18	17	16
reserved†	$\overline{\text{FRST}}$	$\overline{\text{GRST}}$	XINTM	XSYNCERR‡	$\overline{\text{XEMPTY}}$	XRDY	$\overline{\text{XRST}}$
R, +0	RW, +0	RW, +0	RW, +0	RW, +0	R, +0	R, +0	RW, +0

15	14 13	12 11	10 8	7	6	5 4	3	2	1	0
DLB	RJUST	CLKSTP	Rsvd†	DXENA§	Rsvd†	RINTM	RSYNCERR‡	RFULL	RRDY	$\overline{\text{RRST}}$
RW,+0	RW, +0	RW,+0	R, +0	RW, +0	R, +0	RW, +0	RW, +0	R, +0	R, +0	RW, +0

† Reserved-fields have *no* storage associated with them. However, they are always read as 0.
‡ Writing a 1 to this bit will set the error condition. Thus, it is used mainly for testing purposes or if this operation is desired.
§ The DXENA feature is only available in the ’C6211/C711.

Table 11–5. Serial Port Control Register (SPCR) Field Descriptions

Name	Function	Section
$\overline{\text{FRST}}$	Frame sync generator reset $\overline{\text{FRST}}$ = 0: The frame sync generation logic is reset. Frame sync signal is not generated by the sample rate generator. $\overline{\text{FRST}}$ = 1: Frame sync signal is generated after eight CLKG clocks. All frame counters are loaded with their programmed values.	11.5.3
$\overline{\text{GRST}}$	Sample rate generator reset $\overline{\text{GRST}}$ = 0: Sample rate generator is reset. $\overline{\text{GRST}}$ = 1: Sample rate generator is pulled out of reset; CLKG is driven according to the programmed values in the sample rate generator register (SRGR).	11.5.1.2
RINTM	Receive interrupt mode RINTM = 00b: RINT driven by RRDY RINTM = 01b: RINT generated by end-of-subframe in multichannel operation RINTM = 10b: RINT generated by a new frame synchronization RINTM = 11b: RINT generated by RSYNCERR	11.3.3
XINTM	Transmit interrupt mode XINTM = 00b: XINT driven by XRDY XINTM = 01b: XINT generated by end-of-subframe in multichannel operation XINTM = 10b: XINT generated by a new frame synchronization XINTM = 11b: XINT generated by XSYNCERR	11.3.3
RSYNCERR	Receive synchronization error RSYNCERR = 0: No frame synchronization error RSYNCERR = 1: Frame synchronization error detected by McBSP	11.3.7.2 11.3.7.5
XSYNCERR	Transmit synchronization error XSYNCERR = 0: No frame synchronization error XSYNCERR = 1: Frame synchronization error detected by McBSP	11.3.7.2 11.3.7.5
$\overline{\text{XEMPTY}}$	Transmit shift register (XSR) empty $\overline{\text{XEMPTY}}$ = 0: XSR is empty. $\overline{\text{XEMPTY}}$ = 1: XSR is not empty.	11.3.7.4

Table 11–5. Serial Port Control Register (SPCR) Field Descriptions (Continued)

Name	Function	Section
RFULL	Receive shift register (RSR) full error condition RFULL = 0: Receiver is not in overrun condition. RFULL = 1: DRR is not read, RBR is full, and RSR is full with a new element.	11.3.7.1
RRDY	Receiver ready RRDY = 0: Receiver is not ready. RRDY = 1: Receiver is ready with data to be read from DRR.	11.3.2
XRDY	Transmitter ready XRDY = 0: The transmitter is not ready. XRDY = 1: The transmitter is ready for data to be written to DXR.	11.3.2
$\overline{\text{RRST}}$	Receiver reset. This resets or enables the receiver. $\overline{\text{RRST}}$ = 0: The serial port receiver is disabled and is in reset state. $\overline{\text{RRST}}$ = 1: The serial port receiver is enabled.	11.3.1
$\overline{\text{XRST}}$	Transmitter reset. This resets or enables the transmitter. $\overline{\text{XRST}}$ = 0: The serial port transmitter is disabled and is in reset state. $\overline{\text{XRST}}$ = 1: The serial port transmitter is enabled.	11.3.1
DLB	Digital loopback mode DLB = 0: Digital loopback mode disabled DLB = 1: Digital loopback mode enabled	11.5.2.5 11.5.2.6 11.5.3.2

RJUST	Receive data sign-extension and justification mode
	RJUST = 00b: Right-justify and zero-fill MSBs in DRR.
	RJUST = 01b: Right-justify and sign-extend MSBs in DRR.
	RJUST = 10b: Left-justify and zero-fill LSBs in DRR.
	RJUST = 11b: Reserved
		11.3.8

Table 11–5. Serial Port Control Register (SPCR) Field Descriptions (Continued)

Name	Function	Section
CLKSTP	Clock stop mode CLKSTP = 0Xb: Clock stop mode disabled. Normal clocking enabled for non-SPI mode. Clock stop mode enabled for various SPI™ modes when: CLKSTP = 10b and CLKXP = 0: Clock starts with rising edge without delay. CLKSTP = 10b and CLKXP = 1: Clock starts with falling edge without delay. CLKSTP = 11b and CLKXP = 0: Clock starts with rising edge with delay. CLKSTP = 11b and CLKXP = 1: Clock starts with falling edge with delay.	11.7

DXENA	DX Enabler – applicable only for the ’C6211/C6711 device. Enable extra delay for DX turn-on time. This bit controls the Hi-Z enable on the DX pin, not the data itself, so only the first bit of data is delayed.
	DXENA = 0: DX enabler is off.
	DXENA = 1: DX enabler is on.
		11.6.4

Figure 11–3. Pin Control Register (PCR)

31–16
reserved
R, +0

15 14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd	XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM	Rsvd	CLKS_STAT	DX_STAT	DR_STAT	FSXP	FSRP	CLKXP	CLKRP
R,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW, +0	R,+0	RW,+0	RW,+0	R,+0	RW,+0	RW,+0	RW,+0	RW,+0

Table 11–6. Pin Control Register (PCR) Field Descriptions

Name	Function	Section
RIOEN	Receiver in general-purpose I/O mode *only* when $\overline{\text{RRST}}$ = 0 in SPCR RIOEN = 0: DR and CLKS pins are not general-purpose inputs. FSR and CLKR are not general-purpose I/Os and perform serial port operation. RIOEN = 1: DR and CLKS pins are general-purpose inputs. FSR and CLKR are general-purpose I/Os. These serial port pins do not perform serial port operation.	11.8
XIOEN	Transmitter in general-purpose I/O mode *only* when $\overline{\text{XRST}}$ = 0 in SPCR XIOEN = 0: CLKS pin is not a general-purpose input. DX pin is not a general purpose output. FSX and CLKX are not general-purpose I/Os. XIOEN = 1: CLKS pin is a general-purpose input. DX pin is a general-purpose output. FSX and CLKX are general-purpose I/Os. These serial port pins do not perform serial port operation.	11.8
FSXM	Transmit frame synchronization mode FSXM = 0: Frame synchronization signal is provided by an external source. FSX is an input pin. FSXM = 1: Frame synchronization generation is determined by the sample rate generator frame synchronization mode bit FSGM in the SRGR.	11.5.3.3 and 11.8
FSRM	Receive frame synchronization mode FSRM = 0: Frame synchronization signals are generated by an external device. FSR is an input pin. FSRM = 1: Frame synchronization signals are generated internally by the sample rate generator. FSR is an output pin except when GSYNC = 1 (see section 11.5.1.1) in SRGR.	11.5.3.2 and 11.8

Table 11–6. Pin Control Register (PCR) Field Descriptions (Continued)

Name	Function	Section
CLKRM	Receiver clock mode Case 1: Digital loopback mode not set (DLB = 0) in SPCR CLKRM = 0: Receive clock (CLKR) is an input driven by an external clock. CLKRM = 1: CLKR is an output pin and is driven by the sample rate generator. Case 2: Digital loopback mode set (DLB = 1) in SPCR CLKRM = 0: Receive clock (not the CLKR pin) is driven by the transmit clock (CLKX), which is based on the CLKXM bit in PCR. CLKR is in high impedance. CLKRM = 1: CLKR is an output pin and is driven by the transmit clock. The transmit clock is derived from CLKXM bit in the PCR.	11.5.2.6 and 11.8
CLKXM	Transmitter clock mode CLKXM = 0: Transmitter clock is driven by an external clock with CLKX as an input pin. CLKXM = 1: CLKX is an output pin and is driven by the internal sample rate generator.	11.5.2.7 and 11.8
	During SPI mode (CLKSTP in SPCR is a nonzero value): CLKXM = 0: McBSP is a slave and (CLKX) is driven by the SPI master in the system. CLKR is internally driven by CLKX. CLKXM = 1: McBSP is a master and generates the transmitter clock (CLKX) to drive its receiver clock (CLKR) and the shift clock of the SPI-compliant slaves in the system.	11.7
CLKS_STAT	CLKS pin status. Reflects the value on the CLKS pin when selected as a general-purpose input.	11.8
DX_STAT	DX pin status. Reflects the value driven onto the DX pin when selected as a general-purpose output.	11.8
DR_STAT	DR pin status. Reflects the value on the DR pin when selected as a general-purpose input.	11.8

FSRP	Receive frame synchronization polarity
	FSRP = 0: Frame synchronization pulse FSR is active high
	FSRP = 1: Frame synchronization pulse FSR is active low
		11.3.4.1 and 11.8

FSXP	Transmit frame synchronization polarity
	FSXP = 0: Frame synchronization pulse FSX is active high
	FSXP = 1: Frame synchronization pulse FSX is active low
		11.3.4.1 and 11.8

Table 11–6. Pin Control Register (PCR) Field Descriptions (Continued)

Name	Function	Section
CLKXP	Transmit clock polarity CLKXP = 0: Transmit data driven on rising edge of CLKX CLKXP = 1: Transmit data driven on falling edge of CLKX	11.3.4.1 and 11.8
CLKRP	Receive clock polarity CLKRP = 0: Receive data sampled on falling edge of CLKR CLKRP = 1: Receive data sampled on rising edge of CLKR	11.3.4.1 and 11.8

11.2.2 Receive and Transmit Control Registers: RCR and XCR

The receive and transmit control registers (RCR and XCR), shown in Figure 11–4 and Figure 11–5, configure parameters of the receive and transmit operations, respectively. The fields of RCR and XCR are summarized in Figure 11–4.

Figure 11–4. Receive Control Register (RCR)

31	30 24	23 21	20 19	18	17 16
RPHASE	RFRLEN2	RWDLEN2	RCOMPAND	RFIG	RDATDLY
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0

15	14 8	7 5	4	3 0
RPHASE 2[†]	RFRLEN1	RWDLEN1	RWDREVRS[‡]	Reserved
RW, +0	RW, +0	RW, +0	RW, +0	R, +0

† (R/X) PHASE 2 feature's available only on 'C6211/C6711
‡ RWDREVRS and XWDREVRS 32-bit reversal feature is applicable only to the 'C6211/C6711 device.

Figure 11–5. Transmit Control Register (XCR)

31	30 24	23 21	20 19	18	17 16
XPHASE	XFRLEN2	XWDLEN2	XCOMPAND	XFIG	XDATDLY
RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0

15	14 8	7 5	4	3 0
XPHASE 2[†]	XFRLEN1	XWDLEN1	XWDREVRS[‡]	Reserved
RW, +0	RW, +0	RW, +0	RW, +0	R, +0

† (R/X) PHASE 2 feature's available only on 'C6211/C6711
‡ RWDREVRS and XWDREVRS 32-bit reversal feature is applicable only to the 'C6211/C6711 device.

Table 11–7. Receive/Transmit Control Register (RCR/XCR) Field Descriptions

Name	Function	Section

RPHASE	Receive phases
	RPHASE = 0: Single phase frame
	RPHASE = 1: Dual phase frame
		11.3.4.2

XPHASE	Transmit phases
	XPHASE = 0: Single phase frame
	XPHASE = 1: Dual phase frame
		11.3.4.2

RFRLEN(1/2)	Receive frame length in phase 1 and phase 2 RFRLEN(1/2) = 000 0000b: 1 word per phase RFRLEN(1/2) = 000 0001b: 2 words per phase • • • RFRLEN(1/2) = 111 1111b: 128 words per phase	11.3.4.4
XFRLEN(1/2)	Transmit frame length in phase 1 and phase 2 XFRLEN(1/2) = 000 0000b: 1 word per phase XFRLEN(1/2) = 000 0001b: 2 words per phase • • • XFRLEN(1/2) = 111 1111b: 128 words per phase	11.3.4.4
RWDLEN(1/2)	Receive element length in phase 1 and phase 2 RWDLEN(1/2) = 000b: 8 bits RWDLEN(1/2) = 001b: 12 bits RWDLEN(1/2) = 010b: 16 bits RWDLEN(1/2) = 011b: 20 bits RWDLEN(1/2) = 100b: 24 bits RWDLEN(1/2) = 101b: 32 bits RWDLEN(1/2) = 11Xb: Reserved	11.3.4.5

Table 11–7. Receive/Transmit Control Register (RCR/XCR) Field Descriptions (Continued)

Name	Function	Section
XWDLEN(1/2)	Transmit element length in phase 1 and phase 2 XWDLEN(1/2) = 000b: 8 bits XWDLEN(1/2) = 001b: 12 bits XWDLEN(1/2) = 010b: 16 bits XWDLEN(1/2) = 011b: 20 bits XWDLEN(1/2) = 100b: 24 bits XWDLEN(1/2) = 101b: 32 bits XWDLEN(1/2) = 11Xb: Reserved	11.3.4.5
RCOMPAND	Receive companding mode. Modes other than 00b are only applicable when the appropriate RWDLEN is 000b, indicating 8-bit data. RCOMPAND = 00b: No companding. Data transfer starts with MSB first. RCOMPAND = 01b: No companding, 8-bit data. Transfer starts with LSB first. RCOMPAND = 10b: Compand using μ-law for receive data. RCOMPAND = 11b: Compand using A-law for receive data.	11.4
XCOMPAND	Transmit companding mode. Modes other than 00b are only applicable when the appropriate XWDLEN is 000b, indicating 8-bit data. XCOMPAND = 00b: No companding, Data transfer starts with MSB first. XCOMPAND = 01b: No companding, 8-bit data. Transfer starts with LSB first. XCOMPAND = 10b: Compand using μ-law for transmit data. XCOMPAND = 11b: Compand using A-law for transmit data.	11.4
RFIG	Receive frame ignore RFIG = 0: Unexpected receive frame synchronization pulses restart the transfer. RFIG = 1: Unexpected receive frame synchronization pulses are ignored.	11.3.6.1
XFIG	Transmit frame ignore XFIG = 0: Unexpected transmit frame synchronization pulses restart the transfer. XFIG = 1: Unexpected transmit frame synchronization pulses are ignored.	11.3.6.1

Table 11–7. Receive/Transmit Control Register (RCR/XCR) Field Descriptions (Continued)

Name	Function	Section

RDATDLY	Receive data delay
	RDATDLY = 00b: 0-bit data delay
	RDATDLY = 01b: 1-bit data delay
	RDATDLY = 10b: 2-bit data delay
	RDATDLY = 11b: Reserved
		11.3.4.7

XDATDLY	Transmit data delay
	XDATDLY = 00b: 0-bit data delay
	XDATDLY = 01b: 1-bit data delay
	XDATDLY = 10b: 2-bit data delay
	XDATDLY = 11b: Reserved
		11.3.4.7

RPHASE 2	Receive PHASE 2. Applicable only for dual-phase frames. Mainly used for I^{2}S feature. Applicable only for ’C6211/C6711 device. RPHASE 2 = 0: The start of phase 2 is unaffected by receive frame sync. RPHASE 2 = 1: The second phase in a dual-phase frame starts when the receive frame sync transitions to the opposite edge that started the first phase. This is applicable when frame syncs are inputs or outputs.	11.3.4.3
XPHASE 2	Transmit PHASE 2. Applicable only for dual-phase frames. Mainly used for I^{2}S feature. Applicable only for ’C6211/C6711 device. XPHASE 2 = 0: The start of phase 2 is unaffected by transmit frame sync. XPHASE 2 = 1: The second phase in a dual-phase frame starts when the transmit frame sync transitions to the opposite edge that started the first phase. This is applicable when frame syncs are inputs or outputs	11.3.4.3
RWDREVRS	Receive 32-bit bit reversal feature. Applicable only for ’C6211/C6711 device. RWDREVRS = 0: 32-bit reversal disabled RWDREVRS = 1: 32-bit reversal enabled. 32-bit data is received LSB first. RWDLEN should be set for 32-bit operation; else operation is undefined.	11.3.9
XWDREVRS	Transmit 32-bit bit reversal feature. Applicable only for ’C6211/C6711 device. XWDREVRS = 0: 32-bit reversal disabled XWDREVRS = 1: 32-bit reversal enabled. 32-bit data is transmitted LSB first. XWDLEN should be set for 32-bit operation; else operation is undefined.	11.3.9

11.3 Data Transmission and Reception

As shown in Figure 11–1 on page 11-3, the receive operation is triple-buffered and the transmit operation is double-buffered. Receive data arrives on the DR and is shifted into the RSR. Once a full element (8, 12, 16, 20, 24, or 32 bits) is received, the RSR is copied to the receive buffer register (RBR) only if the RBR is not full. The RBR is then copied to the DRR unless the DRR has not been read by the CPU or the DMA controller.

Transmit data is written by the CPU or the DMA controller to the DXR. If there is no data in the XSR, the value in the DXR is copied to the XSR. Otherwise, the DXR is copied to the XSR when the last bit of data is shifted out on the DX. After transmit frame synchronization, the XSR begins shifting out the transmit data on the DX.

11.3.1 Resetting the Serial Port: $\overline{\text{(R/X)RST}}$, $\overline{\text{GRST}}$, and $\overline{\text{RESET}}$

The serial port can be reset in the following two ways:

- ❑ Device reset ($\overline{\text{RESET}}$ pin is low) places the receiver, the transmitter, and the sample rate generator in reset. When the device reset is removed ($\overline{\text{RESET}}$ = 1), $\overline{\text{FRST}}$ = $\overline{\text{GRST}}$ = $\overline{\text{RRST}}$ = $\overline{\text{XRST}}$ = 0, keeping the entire serial port in the reset state.

- ❑ The serial port transmitter and receiver can be independently reset by the $\overline{\text{XRST}}$ and $\overline{\text{RRST}}$ bits in the SPCR. The sample rate generator is reset by the $\overline{\text{GRST}}$ bit in the SPCR.

Table 11–8 shows the state of the McBSP pins when the serial port is reset by these methods.

Table 11–8. Reset State of McBSP Pins

McBSP Pins	**Direction**	**Device Reset ($\overline{\text{RESET}}$ = 0)**	**McBSP Reset**
			Receiver Reset ($\overline{\text{RRST}}$ = 0 and $\overline{\text{GRST}}$ = 1)
DR	I	Input	Input
CLKR	I/O/Z	Input	Known state if input; CLKR if output
FSR	I/O/Z	Input	Known state if input; FSRP(inactive state) if output
CLKS	I	Input	Input
			Transmitter Reset ($\overline{\text{XRST}}$ = 0 and $\overline{\text{GRST}}$ = 1)
DX	O/Z	High impedance	High impedance
CLKX	I/O/Z	Input	Known state if input; CLKX if output
FSX	I/O/Z	Input	Known state if input; FSXP(inactive state) if output
CLKS	I	Input	Input

- **Device reset or McBSP reset:** When the McBSP is reset by device reset or McBSP reset, the state machine is reset to its initial state. All counters and status bits are reset. This includes the receive status bits RFULL, RRDY, and RSYNCERR and the transmit status bits $\overline{\text{XEMPTY}}$, XRDY, and XSYNCERR.

- **Device reset:** When the McBSP is reset due to device reset, the entire serial port (including the transmitter, receiver, and the sample rate generator) is reset. All input-only pins and 3-state pins should be in a known state. The output-only pin, DX, is in the high impedance state. Since the sample rate generator is also reset ($\overline{\text{GRST}}$ = 0), the sample rate generator clock, CLKG, is driven by a divide-by-2 internal clock source, and the frame sync signal, FSG, is not generated. The internal clock source for the ’C6211/C6711 is CPU clock, while the internal clock source for ’C6211/C6711 is CPU/2 clock (half of the CPU clock frequency). See section 11.5.1.2 for more information on sample rate generator reset. When the device is pulled out of reset, the serial port remains in the reset condition ($\overline{\text{(R/X)RST}}$ = $\overline{\text{FRST}}$ = 0). In this reset condition, the serial port pins can be used as general-purpose I/O (see section 11.8).

- **McBSP reset:** When the receiver and transmitter reset bits, $\overline{\text{RRST}}$ and $\overline{\text{XRST}}$, are written with 0, the respective portions of the McBSP are reset and activity in the corresponding section stops. All input-only pins, such as DR and CLKS, and all other pins that are configured as inputs are in a known state. FS(R/X) is driven to its inactive state (same as its polarity bit, FS(R/X)P) if it is an output. If CLK(R/X) are programmed as outputs, they are driven by CLKG, provided that $\overline{\text{GRST}}$ = 1. The DX pin is in the high-impedance state when the transmitter is reset. During normal operation, the sample rate generator can be reset by writing a 0 to $\overline{\text{GRST}}$. $\overline{\text{GRST}}$ should be low 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) is driven inactive (low). When the sample rate generator is not in the reset state ($\overline{\text{GRST}}$ = 1), FSR and FSX are in an inactive state when $\overline{\text{RRST}}$ = 0 and $\overline{\text{XRST}}$ = 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 $\overline{\text{FRST}}$ = 1 and frame sync is driven by FSG.

- **Sample-rate generator reset:** As mentioned previously, the sample rate generator is reset when the device is reset or when its reset bit, $\overline{\text{GRST}}$, is written with 0. In the case of device reset, the CLKG signal is driven by a divide-by-2 internal clock source and FSG is driven inactive (low). The internal clock source for the ’C6211/C6711 is CPU clock, while the internal clock source for ’C6211/C6711 is CPU/2 clock (half of the CPU clock frequency). If you want to reset the sample rate generator when neither the transmitter nor receiver is fed by the CLKG and FSG, you can program $\overline{\text{GRST}}$ in the SRGR to 0. CLKG and FSG are driven inactive (low). When $\overline{\text{GRST}}$ = 1, CLKG runs as programmed in the SRGR. If $\overline{\text{FRST}}$ = 1, FSG is driven active (high) after eight cycles have elapsed.

The serial port initialization procedure is as follows:

1) Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = $\overline{\text{FRST}}$ = 0 in SPCR. If the device has been reset, this step is not required.
2) Program only the McBSP configuration registers, not the data registers, that are listed in Table 11–2, as required when the serial port is in the reset state ($\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = $\overline{\text{FRST}}$ = 0).
3) Wait two bit clocks to ensure proper internal synchronization.
4) Set up data acquisition as desired.
5) Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 1 to enable the serial port. The value written to the SPCR should have only the reset bits changed to 1 and the remaining bit fields should have the same value as in step 2.
6) Set $\overline{\text{FRST}}$ = 1. If it is the frame master, the McBSP is now ready to transmit and/or receive.

Alternatively, on either write (steps 1 and 5 above), the transmitter and receiver can be placed in or taken out of reset individually by modifying only the desired bit. The necessary duration of the active(low) period of $\overline{\text{XRST}}$ or $\overline{\text{RRST}}$ is at least two bit clocks (CLKR/CLKX). This procedure for reset initialization can be applied generally 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. The sample-rate generator reset procedure is explained in section 11.5.1.2.

Notes:

1) The appropriate fields in the serial port configuration registers SPCR, PCR, RCR, XCR, and SRGR should be modified only when the affected portion of the serial port is in reset.

2) The data transmit register, DXR, should be loaded by the CPU or DMA only when the transmitter is not in reset ($\overline{\text{XRST}}$ = 1). The exception to this rule occurs during non-digital loop-back mode, which is described in section 11.4.1.

3) The multichannel selection registers MCR, XCER, and RCER can be modified at any time as long as they are not being used by the current block in the multichannel selection. See section 11.6.3.2 for more information.

11.3.2 Determining Ready Status

RRDY and XRDY indicate the ready state of the McBSP receiver and transmitter, respectively. Writes and reads from the serial port can be synchronized by any of the following methods:

- ❑ Polling RRDY and XRDY
- ❑ Using the events sent to the DMA controller (REVT and XEVT)
- ❑ Using the interrupts to the CPU (RINT and XINT) that the events generate.

Note:

Note that reading the DRR and writing to DXR affects RRDY and XRDY, respectively.

11.3.2.1 Receive Ready Status: REVT, RINT, and RRDY

RRDY = 1 indicates that the RBR contents have been copied to the DRR and that the data can be read by either the CPU or the DMA controller. Once that data has been read by either the CPU or the DMA controller, RRDY is cleared to 0. Also, at device reset or serial port receiver reset ($\overline{\text{RRST}}$ = 0), the RRDY is cleared to 0 to indicate that no data has yet been received and loaded into DRR. RRDY directly drives the McBSP receive event to the DMA controller (via REVT). Also, the McBSP receive interrupt (RINT) to the CPU can be driven by RRDY if RINTM = 00b (default value) in the SPCR.

11.3.2.2 Transmit Ready Status: XEVT, XINT, and XRDY

XRDY = 1 indicates that the DXR contents have been copied to XSR and that DXR is ready to be loaded with a new data word. When the transmitter transitions from reset to non-reset ($\overline{\text{XRST}}$ transitions from 0 to 1), XRDY also transitions from 0 to 1 indicating that the DXR is ready for new data. Once new data is loaded by the CPU or the DMA controller, XRDY is cleared to 0. However, once this data is copied from the DXR to the XSR, XRDY transitions again from 0 to 1. The CPU or the DMA controller can write to DXR although XSR has not yet been shifted out on DX. XRDY directly drives the transmit synchronization event to the DMA controller (via XEVT). Also, the transmit interrupt (XINT) to the CPU can be driven by XRDY if XINTM = 00b (default value) in the SPCR.

11.3.3 CPU Interrupts: (R/X)INT

The receive interrupt (RINT) and transmit interrupt (XINT) signal the CPU of changes to the serial port status. Four options exist for configuring these interrupts. These options are set by the receive/transmit interrupt mode field, (R/X)INTM, in the SPCR. The possible values of the mode and the configurations they represent are:

- (R/X)INTM = 00b. Interrupt on every serial element by tracking the (R/X)RDY bits in the SPCR.

- (R/X)INTM = 01b. Interrupt at the end of a subframe (16 elements or less) within a frame. See subsection 11.6.3.3 for details.

- (R/X)INTM = 10b. Interrupt on detection of frame synchronization pulses. This generates an interrupt even when the transmitter/receiver is in reset. This is done by synchronizing the incoming frame sync pulse to the CPU clock and sending it to the CPU via (R/X)INT. See subsection 11.5.3.4 for more information.

- (R/X)INTM = 11b. Interrupt on frame synchronization error. Note that if any of the other interrupt modes are selected, (R/X)SYNCERR may be read when servicing the interrupts to detect this condition. See subsections 11.3.7.2 and 11.3.7.5 for more details on synchronization error.

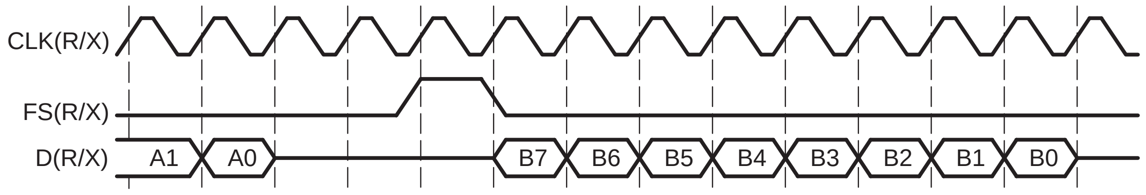
11.3.4 Frame and Clock Configuration

Figure 11–6 shows typical operation of the McBSP clock and frame sync signals. Serial clocks CLKR and CLKX define the boundaries between bits for receive and transmit, respectively. Similarly, frame sync signals FSR and FSX define the beginning of an element transfer. The McBSP allows configuration of the following parameters for data and frame synchronization:

- ❑ Polarities of FSR, FSX, CLKX, and CLKR
- ❑ A choice of single- or dual-phase frames
- ❑ For each phase, the number of elements per frame
- ❑ For each phase, the number of bits per element
- ❑ Whether subsequent frame synchronization restarts the serial data stream or is ignored
- ❑ The data delay from frame synchronization to first data bit which can be 0-, 1-, or 2-bit delays
- ❑ Right or left justification as well as sign extension or zero filling for receive data

The configuration can be independent for receive and transmit.

Figure 11–6. Frame and Clock Operation



11.3.4.1 Frame and Clock Operation

Receive and transmit frame sync pulses can either be generated internally by the sample rate generator (see section 11.5.1) or be driven by an external input. The source of frame sync is selected by programming the mode bit, FS(R/X)M, in the PCR. FSR is also affected by the GSYNC bit in the SRGR (see section 11.5.3.2 for details). Similarly, receive and transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLK(R/X)M, in the PCR.

When FSR and FSX are inputs (FSXM = FSRM = 0), the McBSP detects them on the internal falling edge of clock, CLKR_int and CLKX_int, respectively (see

Figure 11–37 on page 11-53). The receive data arriving at the DR pin is also sampled on the falling edge of CLKR_int. These internal clock signals are either derived from external source via the CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of the internal clock, CLK(R/X)_int. Similarly, data on DX is output on the rising edge of CLKX_int. See section 11.3.4.7 for more information.

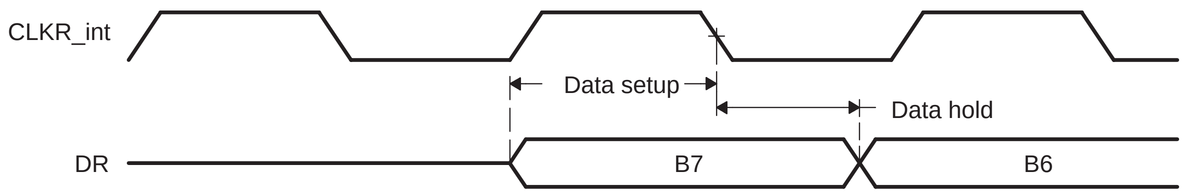
FSRP, FSXP, CLKRP, and CLKXP configure the polarities of FSR, FSX, CLKR, and CLKX, as indicated in Table 11–6. All frame sync signals (FSR_int and FSX_int) internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to the McBSP) and FSRP = FSXP = 1, the external active (low) frame sync signals are inverted before being sent to the receiver signal (FSR_int) and transmitter signal (FSX_int). Similarly, if internal synchronization is selected (FSR/FSX are outputs and GSYNC = 0), the internal active (high) sync signals are inverted if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin. Figure 11–37 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. Data is always transmitted on the rising edge of CLKX_int. If CLKXP = 1 and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge-triggered input clock on CLKX is inverted to a rising-edge-triggered clock before being sent to the transmitter. If CLKXP = 1 and internal clocking is selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge-triggered) clock, CLKX_int, is inverted before being sent out on the CLKX pin.

Similarly, the receiver can reliably sample data that is clocked (by the transmitter) with a rising-edge clock. The receive clock polarity bit, CLKRP, sets the edge used to sample received data. The receive data is always sampled on the falling edge of CLKR_int. 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 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 before being sent out on the CLKR pin.

In a system where the same clock (internal or external) is used to clock the receiver and transmitter, CLKRP = CLKXP. The receiver uses the opposite edge as the transmitter to ensure valid setup and hold times of data around this edge. Figure 11–7 shows how data clocked by an external serial device using a rising-edge clock can be sampled by the McBSP receiver with the falling edge of the same clock.

Figure 11–7. Receive Data Clocking



11.3.4.2 Frame Synchronization Phases

Frame synchronization indicates the beginning of a transfer on the McBSP. The data stream following frame synchronization can have up to two phases, phase 1 and phase 2. The number of phases can be selected by the phase bit, (R/X)PHASE, in the RCR and XCR. The number of elements per frame and bits per element can be independently selected for each phase via (R/X)FRLEN(1/2) and (R/X)WDLEN(1/2), respectively. Figure 11–8 shows a frame in which the first phase consists of two elements of 12 bits each followed by a second phase of three elements of 8 bits each. The entire bit stream in the frame is contiguous; no gaps exist either between elements or phases. Table 11–9 shows the fields in the receive/transmit control registers (RCR/XCR) that control the frame length and element length for each phase for both the receiver and the transmitter. The maximum number of elements per frame is 128 for a single-phase frame and 256 elements in a dual-phase frame. The number of bits per element can be 8, 12, 16, 20, 24, or 32.

Note:

For a dual-phase frame with internally generated frame sync, the maximum number of elements per phase depends on the word length. This is because the frame period, FPER is only 12-bits wide and, therefore, provides 4096 bits per frame. Hence, the maximum number of 128 elements per single-phase frame for a total of 256 elements per dual-phase frame applies only when the WDLEN is 16-bits.

Figure 11–8. Dual-Phase Frame Example

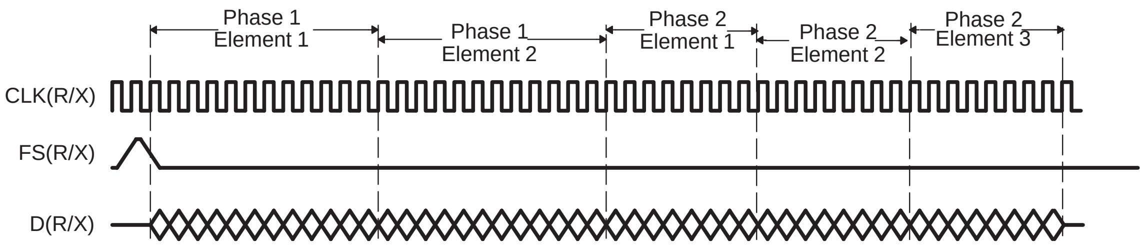


Table 11–9. RCR/XCR Fields Controlling Elements per Frame and Bits per Element

Serial Port McBSP0/1	Frame Phase	RCR/XCR field Control: Elements per Frame	RCR/XCR field Control: Bits per Element
Receive	1	RFRLEN1	RWDLEN1
Receive	2	RFRLEN2	RWDLEN2
Transmit	1	XFRLEN1	XWDLEN1
Transmit	2	XFRLEN2	XWDLEN2

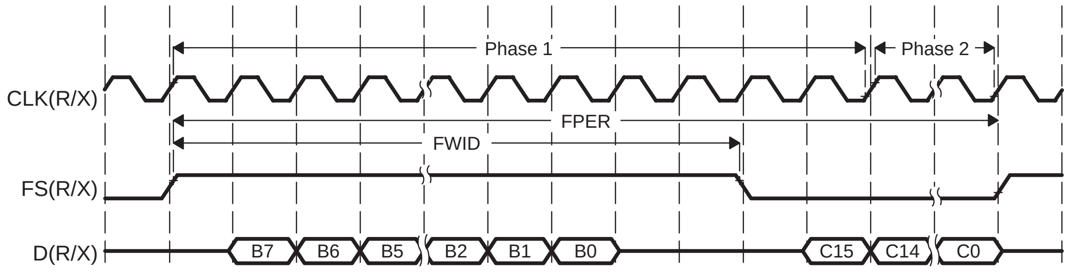
11.3.4.3 Phase 2 Control: (R/X) PHASE2

This feature is available only in the ’C6211/C6711 device. The (R/X)PHASE2 bits in the (R/X)CR register determine when the second phase starts in a dual phase frame. This feature is to support more varieties of IIS formats. Note that the McBSP in the other ’C6000 family of devices does support some IIS formats.

The start of second phase can be controlled by setting the (R/X)PHASE2 bit. When (R/X)PHASE 2 is zero, the start of phase 2 is unaffected by the receive/ transmit frame sync. As shown in Figure 11–8, phase 2 starts as soon as phase 1 is finished. When (R/X)PHASE2 = 1, the first phase starts as soon as the frame sync goes active (low if FS(R/X)P = 1, high if FS(R/X)P = 0). The second phase starts when the frame sync transitions to the opposite edge that started the first phase as shown in Figure 11–9. If FS(R/X) is an output driven by the McBSP, FWID determines the duration of Phase 1, and FPER determines the total frame period for the two phases. If FS(R/X) is an input, the frame sync transition after the first phase is detected and the second phase transmission/reception is initiated.

For all dual phase frames, phase 1 and phase 2 can have elements with different word lengths. Hence, WDLEN1 can be different than WDLEN2. Setting the (R/X)PHASE2 bit also allows dead time between Phase 1 and Phase 2 as shown in Figure 11–9. All data delays are still valid with this set up.

Figure 11–9. Inter-IC Sound (IIS) Timing



11.3.4.4 Frame Length: (R/X)FRLEN(1/2)

Frame length can be defined as the number of serial elements transferred per frame. The length corresponds to the number of elements or logical time slots or channels per frame synchronization signal. The 7-bit (R/X)FRLEN(1/2) field in the (R/X)CR supports up to 128 elements per frame, as shown in Table 11–10. (R/X)PHASE = 0 selects a single-phase data frame, and (R/X)PHASE = 1 selects a dual-phase frame for the data stream. For a single-phase frame, the value of FRLEN2 does not matter. Program the frame length fields with (*w* minus 1), where *w* represents the number of elements per frame. For Figure 11–8, (R/X)FRLEN1 = 1 or 0000001b and (R/X)FRLEN2 = 2 or 0000010b.

Table 11–10. McBSP Receive/Transmit Frame Length 1/2 Configuration

(R/X)PHASE	(R/X)FRLEN1	(R/X)FRLEN2	Frame Length
0	0 ≤ n ≤ 127	x	Single-phase frame; (n+1) words per frame
1	0 ≤ n ≤ 127	0 ≤ m ≤ 127	Dual-phase frame; (n+1) plus (m+1) words per frame

11.3.4.5 Element Length: (R/X)WDLEN(1/2)

The (R/X)WDLEN(1/2) fields in the receive/transmit control register determine the element length in bits per element for the receiver and the transmitter for each phase of the frame, as indicated in Table 11–9. Table 11–11 shows how the value of these fields selects particular element lengths in bits. For the example in Figure 11–8, (R/X)WDLEN1 = 001b and (R/X)WDLEN2 = 000b. If (R/X)PHASE = 0, indicating a single-phase frame, (R/X)WDLEN2 is not used by the McBSP and its value does not matter.

Table 11–11. McBSP Receive/Transmit Element Length Configuration

(R/X)WDLEN (1/2)	McBSP Element Length (Bits)
000	8
001	12
010	16
011	20
100	24
101	32
110	Reserved
111	Reserved

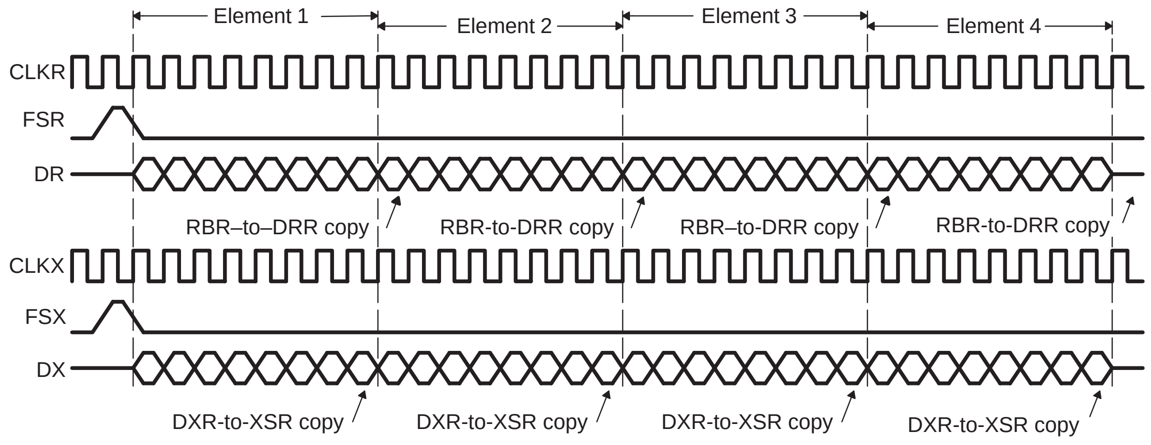
11.3.4.6 Data Packing using Frame Length and Element Length

The frame length and element length can be manipulated to effectively pack data. For example, consider a situation in which four 8-bit elements are transferred in a single-phase frame, as shown in Figure 11–10. In this case:

- ❑ (R/X)PHASE = 0, indicating a single-phase frame
- ❑ (R/X)FRLEN1 = 0000011b, indicating a 4-element frame
- ❑ (R/X)WDLEN1 = 000b, indicating 8-bit elements

In this situation, four 8-bit data elements are transferred to and from the McBSP by the CPU or the DMA controller. Four reads of DRR and four writes of DXR are necessary for each frame.

Figure 11–10. Single-Phase Frame of Four 8-Bit Elements

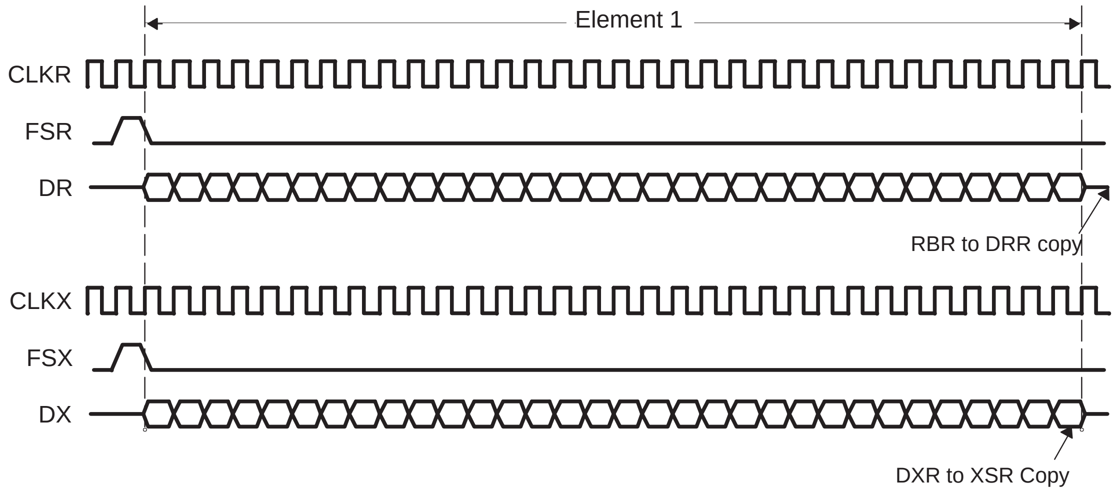


The example in Figure 11–10 can also be viewed as a data stream of a single-phase frame of one 32-bit data element, as shown in Figure 11–11. In this case:

- ❑ (R/X)PHASE = 0, indicating a single phase frame
- ❑ (R/X)FRLEN1 = 0b, indicating a 1-element frame
- ❑ (R/X)WDLEN1 = 101b, indicating 32-bit elements

In this situation, one 32-bit data element is transferred to and from the McBSP by the CPU or the DMA controller. Thus, one read of DRR and one write of DXR is necessary for each frame. As a result, the number of transfers is one fourth that of the previous case. This manipulation reduces the percentage of bus time required for serial port data movement.

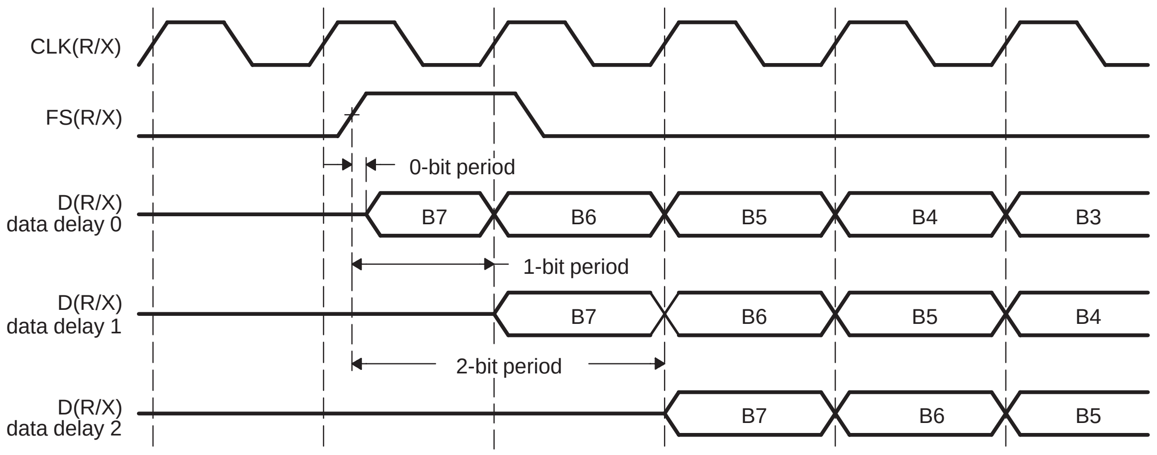
Figure 11–11. Single-Phase Frame of One 32-Bit Element



11.3.4.7 Data Delay: (R/X)DATDLY

The start of a frame is defined by the first clock cycle in which frame synchronization is 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 and XDATDLY specify the data delay for reception and transmission, respectively. The range of programmable data delay is zero to two bit clocks ((R/X)DATDLY = 00b to10b), as indicated in Table 11–7 and shown in Figure 11–12. Typically, a 1-bit delay is selected because data often follows a 1-cycle active frame sync pulse.

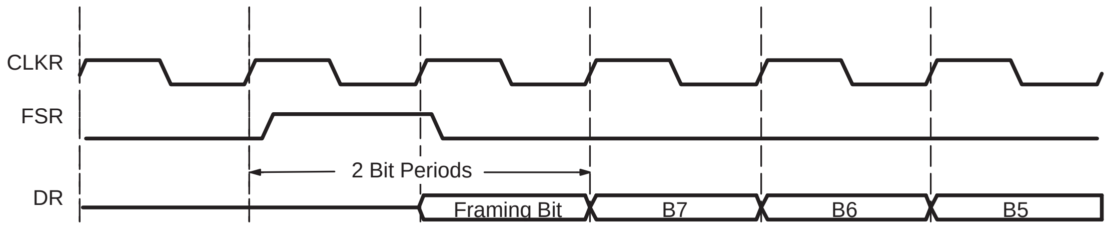
Figure 11–12. Data Delay



Normally a frame sync pulse is detected or sampled with respect to an edge of serial clock CLK(R/X). Thus, on a subsequent cycle (depending on data delay value), data can be received or transmitted. However, in the case of a 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 by receive data being sampled on the first falling edge of CLKR when an active (high) FSR is detected. However, data transmission must begin on the rising edge of CLKX that generated the frame synchronization. Therefore, the first data bit is assumed to be in the XSR and DX. The transmitter then asynchronously detects the frame synchronization, FSX goes active, and it immediately starts driving the first bit to be transmitted on the DX pin.

Another common operation uses a data delay of 2. This configuration allows the serial port to interface to different types of T1 framing devices in which the data stream is preceded by a framing bit. During the reception of such a stream with a data delay of two bits, the 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 11–13. In transmission, by delaying the first transfer bit, the serial port essentially inserts a blank period (a high-impedance period) in place of the framing bit. Here, it is expected that the framing device inserts its own framing bit or that the framing bit is generated by another device. Alternatively, you may pull up or pull down DX to achieve the desired value.

Figure 11–13. 2-Bit Data Delay Used to Discard Framing Bit

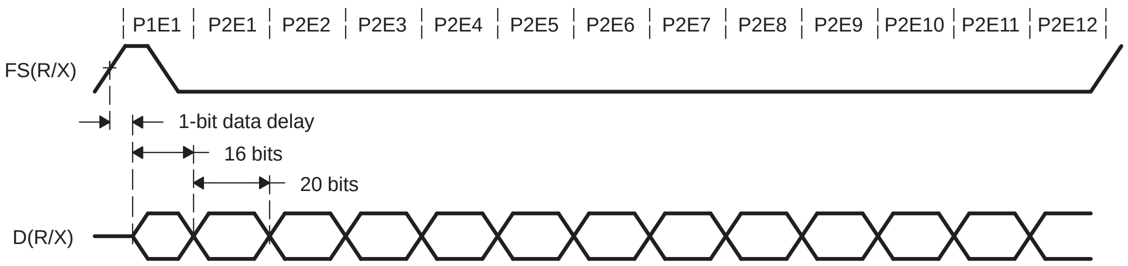


11.3.4.8 Multiphase Frame Example: AC97

Figure 11–14 shows an example of the Audio Codec ’97 (AC97) standard, which uses the dual-phase frame feature. The first phase consists of a single 16-bit element. The second phase consists of 12 20-bit elements. The phases are configured as follows:

- ❑ (R/X)PHASE = 1b: specifying a dual-phase frame
- ❑ (R/X)FRLEN1 = 0b: specifying one element per frame in phase 1
- ❑ (R/X)WDLEN1 = 010b: specifying 16 bits per element in phase 1
- ❑ (R/X)FRLEN2 = 0001011b: specifying 12 elements per frame in phase 2
- ❑ (R/X)WDLEN2 = 011b: specifying 20 bits per element in phase 2
- ❑ CLK(R/X)P = 0: specifying that the receive data sampled on the falling edge of CLKR and the transmit data are clocked on the rising edge of CLKX
- ❑ FS(R/X)P = 0: indicating that active frame sync signals are used
- ❑ (R/X)DATDLY = 01b: indicating a data delay of one bit clock

Figure 11–14. AC97 Dual-Phase Frame Format†

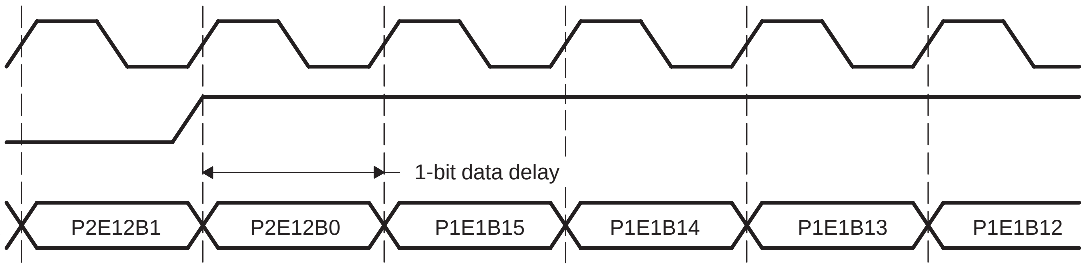


† PxEy denotes phase x and element y.

Figure 11–14 shows the AC97 timing near frame synchronization. First the frame sync pulse itself overlaps the first element. In McBSP operation, the inactive-to-active transition of the frame synchronization signal actually indicates frame synchronization. For this reason, frame synchronization can be high for an arbitrary number of bit clocks. Only after the frame synchronization is recognized as inactive and then active again is the next frame synchronization recognized.

In Figure 11–15, there is 1-bit data delay. Regardless of the data delay, transmission can occur without gaps. The last bit of the previous (last) element in phase 2 is immediately followed by the first data bit of the first element in phase 1 of the next data frame.

Figure 11–15. AC97 Bit Timing Near Frame Synchronization†



† † PxEyBz denotes phase x, element y, and bit z.

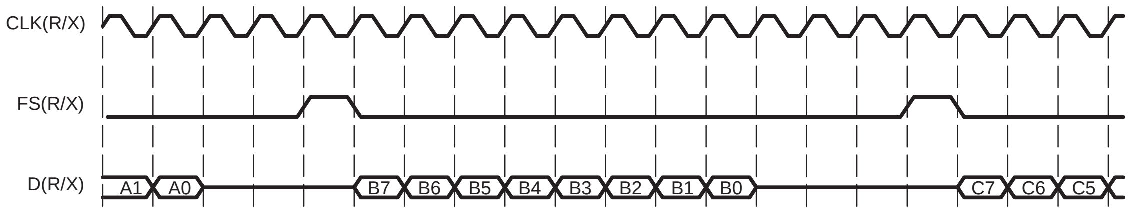
11.3.5 McBSP Standard Operation

During a serial transfer, there are typically periods of serial port inactivity between packets or transfers. The receive and transmit frame synchronization pulse occurs for every serial transfer. When the McBSP is not in the reset state and has been configured for the desired operation, a serial transfer can be initiated by programming (R/X)PHASE = 0 for a single-phase frame with the required number of elements programmed in (R/X)FRLEN1. The number of elements can range from 1 to 128 ((R/X)FRLEN1 = 00h to 7Fh). The required serial element length is set in the (R/X)WDLEN1 field in the (R/X)CR. If a dual-phase frame is required for the transfer, RPHASE = 1 and each (R/X)FRLEN(1/2) can be set to any value between 00h and 7Fh.

Figure 11–16 shows a single-phase data frame of one 8-bit element. Since the transfer is configured for a 1-bit data delay, the data on the DX and DR pins are available one bit clock after FS(R/X) goes active. This figure as well as all others in this section use the following assumptions:

- ❑ (R/X)PHASE = 0, specifying a single-phase frame
- ❑ (R/X)FRLEN1 = 0b, specifying one element per frame
- ❑ (R/X)WDLEN1 = 000b, specifying eight bits per element
- ❑ (R/X)FRLEN2 = (R/X)WDLEN2 = Value is ignored.
- ❑ CLK(R/X)P = 0, specifying that the receive data is clocked on the falling edge and that transmit data is clocked on the rising edge
- ❑ FS(R/X)P = 0, specifying that active (high) frame sync signals are used
- ❑ (R/X)DATDLY = 01b, specifying a 1-bit data delay

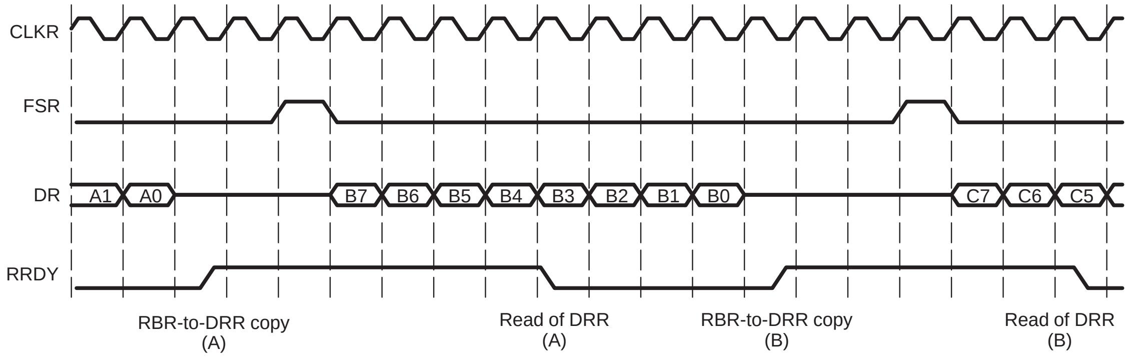
Figure 11–16. McBSP Standard Operation



11.3.5.1 Receive Operation

Figure 11–17 shows serial reception. Once the receive frame synchronization signal (FSR) transitions to its active state, it is detected on the first falling edge of the receiver's CLKR. The data on the DR pin is then shifted into the receive shift register (RSR) after the appropriate data delay as set by RDATDLY. The contents of RSR is copied to RBR at the end of every element on the rising edge of the clock, provided RBR is not full with the previous data. Then, an RBR-to-DRR copy activates the RRDY status bit to 1 on the following falling edge of CLKR. This indicates that the receive data register (DRR) is ready with the data to be read by the CPU or the DMA controller. RRDY is deactivated when the DRR is read by the CPU or the DMA controller.

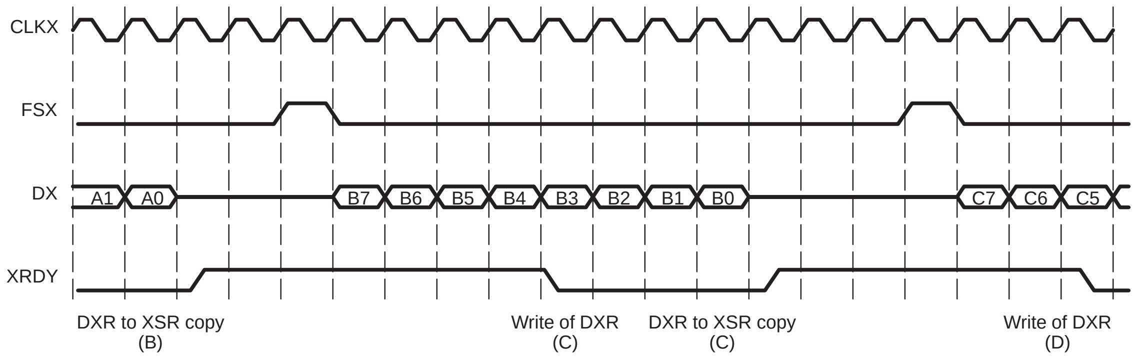
Figure 11–17. Receive Operation



11.3.5.2 Transmit Operation

Once transmit frame synchronization occurs, the value in the transmit shift register, XSR, is shifted out and driven on the DX pin after the appropriate data delay as set by XDATDLY. XRDY is activated after every DXR-to-XSR copy on the following falling edge of CLKX, indicating that the data transmit register (DXR) can be written with the next data to be transmitted. XRDY is deactivated when the DXR is written by the CPU or the DMA controller. Figure 11–18 illustrates serial transmission. See section 11.3.7.4 for information on transmit operation when the transmitter is pulled out of reset ($\overline{\text{XRST}}$ = 1).

Figure 11–18. Transmit Operation



11.3.5.3 Maximum Frame Frequency

The frame frequency is determined by the following equation, which calculates the period between frame synchronization signals:

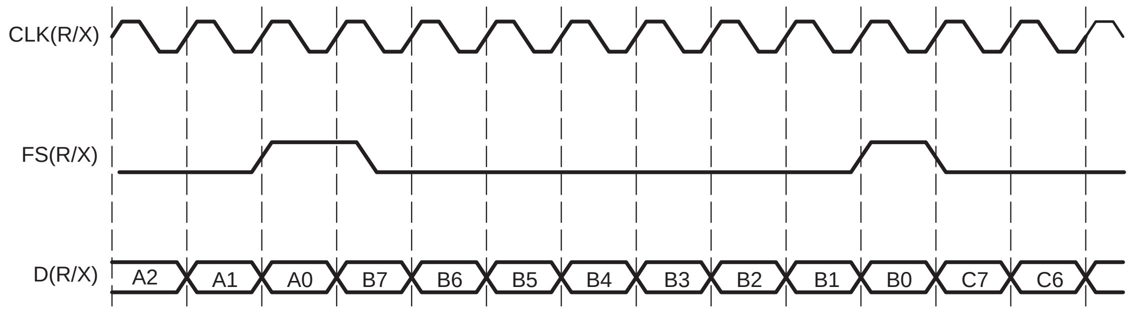
$$\textit{Frame frequency} = \frac{\textit{Bit clock frequency}}{\textit{Number of bit clocks between frame sync signals}}$$

The frame frequency may be increased by decreasing the time between frame synchronization signals in bit clocks (which is limited only by the number of bits per frame). As the frame transmit frequency is increased, the inactivity period between the data frames for adjacent transfers decreases to 0. The minimum time between frame synchronization pulses is the number of bits transferred per frame. This time also defines the maximum frame frequency, which is calculated by the following equation:

$$\textit{Maximum frame frequency} = \frac{\textit{Bit clock frequency}}{\textit{Number of bits per frame}}$$

Figure 11–19 shows the McBSP operating at maximum frame frequency. The data bits in consecutive frames are transmitted continuously with no inactivity between bits. If there is a 1-bit data delay, as shown, the frame synchronization pulse overlaps the last bit transmitted in the previous frame.

Figure 11–19. Maximum Frame Frequency Transmit and Receive



Note:

For (R/X)DATDLY = 0, the first bit of data transmitted is asynchronous to CLKX, as shown in Figure 11–12.

11.3.6 Frame Synchronization Ignore

The McBSP can be configured to ignore transmit and receive frame synchronization pulses. The (R/X)FIG bit in the (R/X)CR can be set to 0 to recognize frame sync pulses, or it can be set to 1 to ignore frame sync pulses. This way, you can use (R/X)FIG either to pack data, if operating at maximum frame frequency, or to ignore unexpected frame sync pulses.

11.3.6.1 Frame Sync Ignore and Unexpected Frame Sync Pulses

RFIG and XFIG are used to ignore unexpected frame sync pulses. Any frame sync pulse is considered unexpected if it occurs one or more bit clocks earlier than the programmed data delay from the end of the previous frame specified by ((R/X)DATDLY). Setting the frame ignore bits to 1 causes the serial port to ignore these unexpected frame sync signals.

In reception, if not ignored (RFIG = 0), an unexpected FSR pulse discards the contents of RSR in favor of the incoming data. Therefore, if RFIG = 0, an unexpected frame synchronization pulse aborts the current data transfer, sets RSYNCERR in the SPCR to 1, and begins the reception of a new data element. When RFIG = 1, the unexpected frame sync pulses are ignored.

In transmission, if not ignored (XFIG = 0), an unexpected FSX pulse aborts the ongoing transmission, sets the XSYNCERR bit in the SPCR to 1, and reinitiates transmission of the current element that was aborted. When XFIG = 1, unexpected frame sync signals are ignored.

Figure 11–20 shows that element B is interrupted by an unexpected frame sync pulse when (R/X)FIG = 0. The reception of B is aborted (B is lost), and a new data element (C) is received after the appropriate data delay. This condition causes a receive synchronization error and thus sets the RSYNCERR bit. However, for transmission, the transmission of B is aborted and the same data (B) is retransmitted after the appropriate data delay. This condition is a transmit synchronization error and thus sets the XSYNCERR bit. Synchronization errors are discussed in sections 11.3.7.2 and 11.3.7.5.

Figure 11–20. Unexpected Frame Synchronization With (R/X)FIG = 0

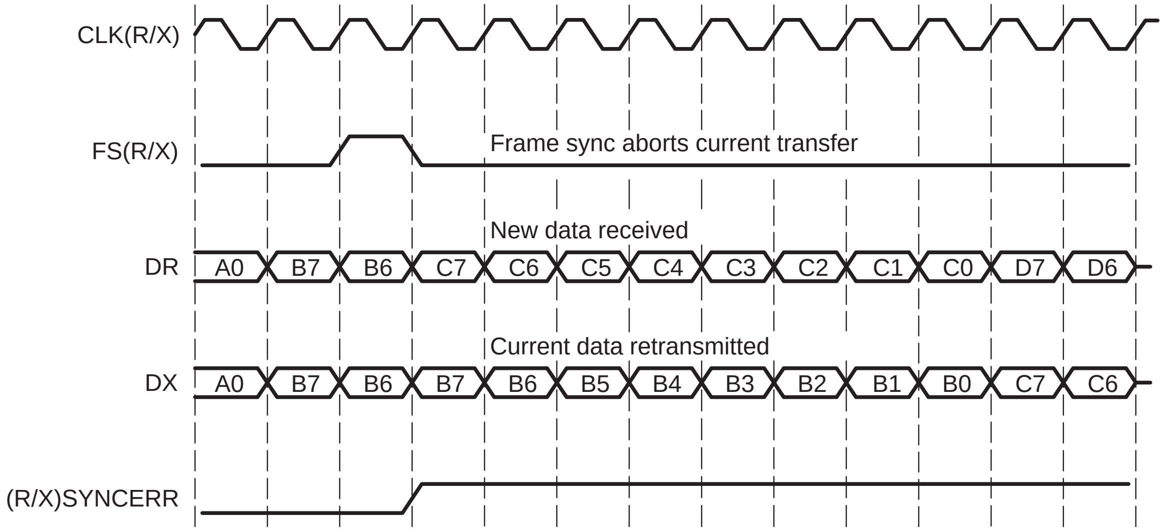
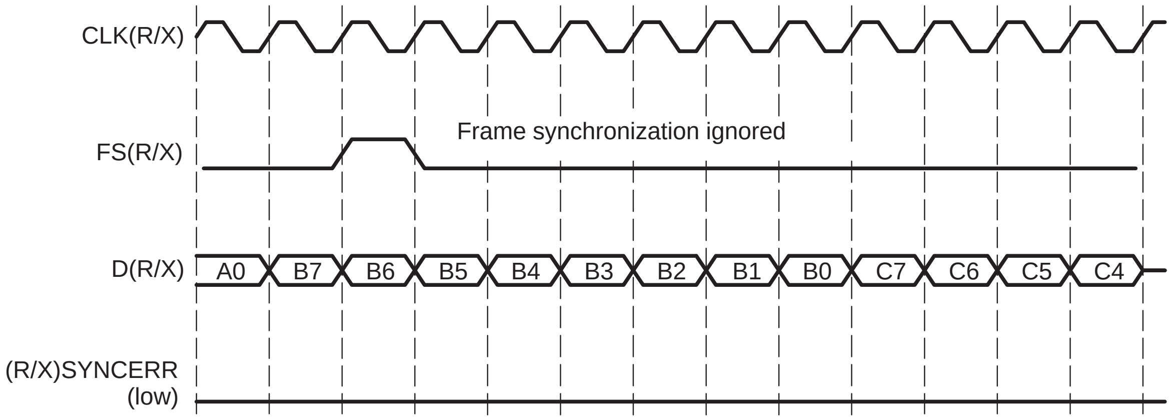

CLK(R/X)
FS(R/X)
Frame sync aborts current transfer
New data received
DR
A0 B7 B6 C7 C6 C5 C4 C3 C2 C1 C0 D7 D6
Current data retransmitted
DX
A0 B7 B6 B7 B6 B5 B4 B3 B2 B1 B0 C7 C6
(R/X)SYNCERR

Figure 11–21 shows McBSP operation when unexpected frame synchronization signals are ignored by setting (R/X)FIG = 1. Here, the transfer of element B is not affected by an unexpected frame synchronization.

Figure 11–21. Unexpected Frame Synchronization With (R/X)FIG = 1



11.3.6.2 Data Packing using Frame Sync Ignore Bits

Section 11.3.4.6 describes one method of changing the element length and frame length to simulate 32-bit serial element transfers, thus requiring much less bus bandwidth than four 8-bit transfers require. This example works when there are multiple elements per frame. Now consider the case of the McBSP operating at maximum packet frequency, as shown in Figure 11–22. Here, each frame has only a single 8-bit element. This stream takes one read transfer and one write transfer for each 8-bit element. Figure 11–23 shows the McBSP configured to treat this stream as a continuous stream of 32-bit elements. In this example, (R/X)FIG is set to 1 to ignore unexpected subsequent frames. Only one read transfer and one write transfer is needed every 32-bits. This configuration effectively reduces the required bus bandwidth to one-fourth of the bandwidth needed to transfer four 8-bit blocks.

Figure 11–22. Maximum Frame Frequency Operation With 8-Bit Data

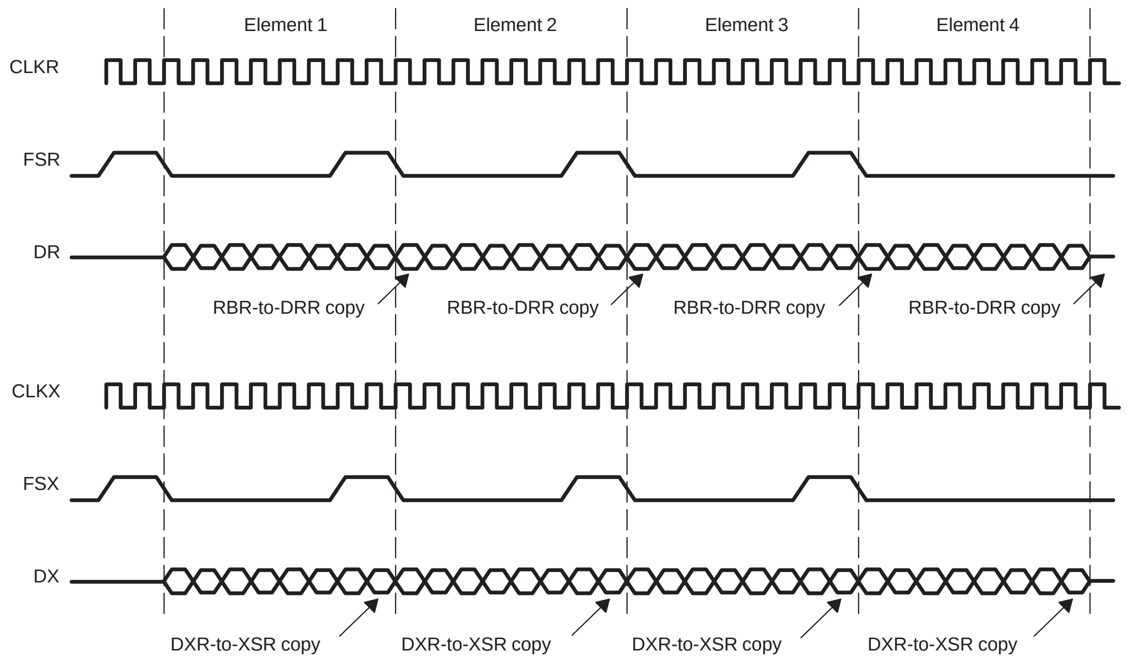
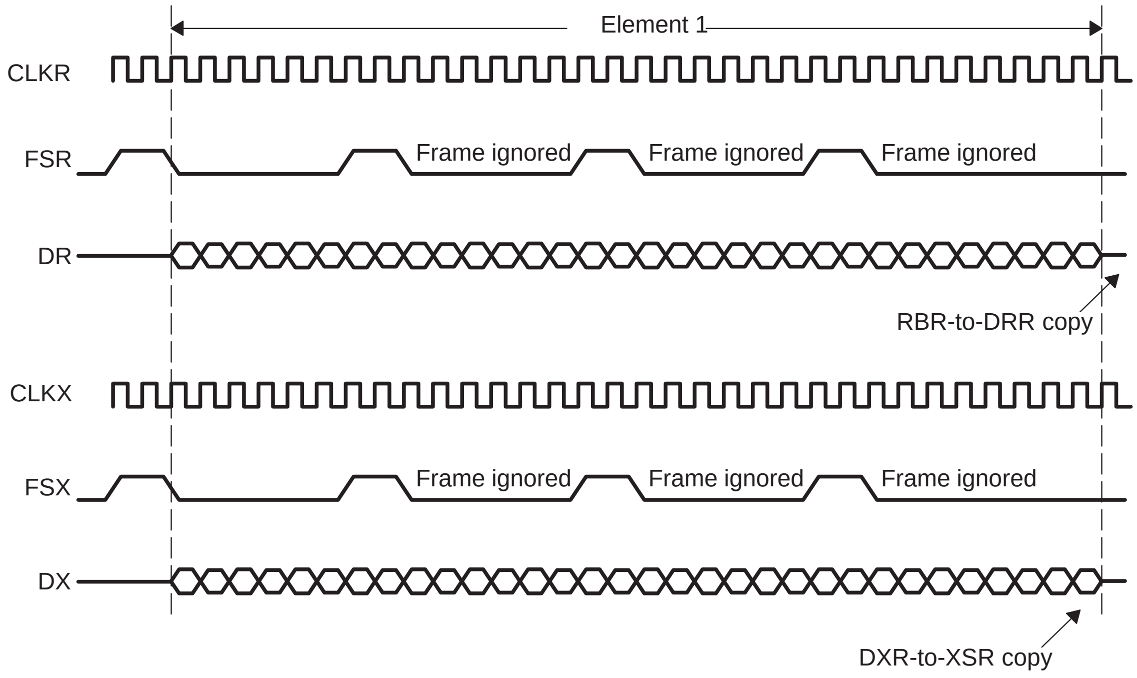


Figure 11–23. Data Packing at Maximum Frame Frequency With (R/X)FIG = 1



11.3.7 Serial Port Exception Conditions

There are five serial port events that can constitute a system error:

- Receive overrun (RFULL = 1)
- Unexpected receive frame synchronization (RSYNCERR = 1)
- Transmit data overwrite
- Transmit empty ($\overline{\text{XEMPTY}}$ = 0)
- Unexpected transmit frame synchronization (XSYNCERR = 1)

11.3.7.1 Reception With Overrun: RFULL

RFULL = 1 in the SPCR indicates that the receiver has experienced overrun and is in an error condition. RFULL is set when the following conditions are met:

- DRR has not been read since the last RBR-to-DRR transfer.
- RBR is full and an RBR-to-DRR copy has not occurred.
- RSR is full and an RSR-to-RBR transfer has not occurred.

The data arriving on DR is continuously shifted into RSR. Once a complete element is shifted into RSR, an RSR-to-RBR transfer can occur only if an RBR-to-DRR copy is complete. Therefore, if DRR has not been read by the CPU or the DMA controller since the last RBR-to-DRR transfer (RRDY = 1), an RBR-to-DRR copy does not take place until RRDY = 0. This prevents an RSR-to-RBR copy. New data arriving on the DR pin is shifted into RSR, and the previous contents of RSR is lost. After the receiver starts running from reset, a minimum of three elements must be received before RFULL can be set, because there was no last RBR-to-DRR transfer before the first element.

This data loss can be avoided if DRR is read no later than two and a half CLKR cycles before the end of the third element (data C) in RSR, as shown in Figure 11–25.

Either of the following events clears the RFULL bit to 0 and allows subsequent transfers to be read properly:

- Reading DRR
- Resetting the receiver ($\overline{\text{RRST}}$ = 0) or the device

Another frame synchronization is required to restart the receiver.

Figure 11–24 shows the receive overrun condition. Because element A is not read before the reception of element B is complete, B is not transferred to DRR yet. Another element, C, arrives and fills RSR. DRR is finally read, but not earlier than two and one half cycles before the end of element C. New data D overwrites the previous element C in RSR. If RFULL is still set after the DRR is read, the next element can overwrite D if DRR is not read in time.

Figure 11–24. Serial Port Receive Overrun

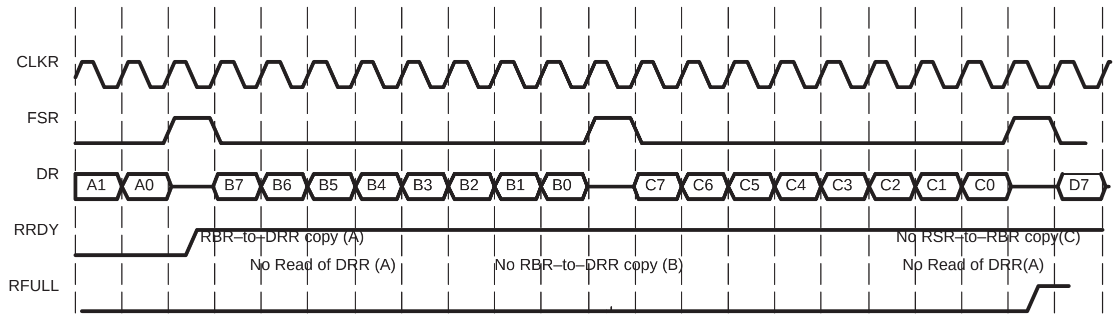
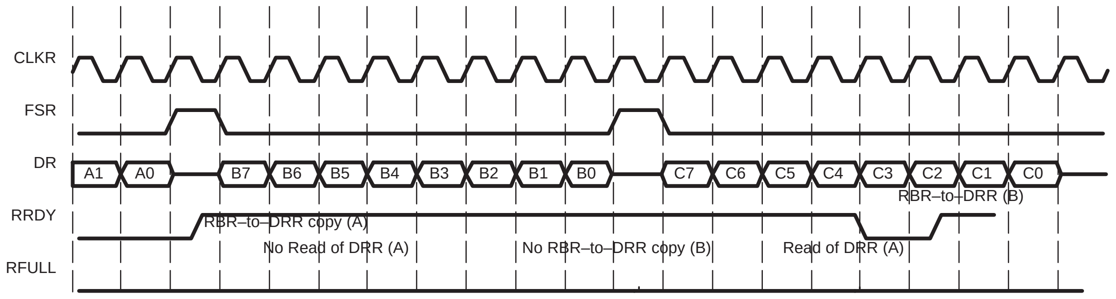


Figure 11–25 shows the case in which RFULL is set but the overrun condition is averted by reading the contents of DRR at least two and a half cycles before the next element, C, is completely shifted into RSR. This ensures that a RBR-to-DRR copy of data B occurs before the next element is transferred from RSR to RBR.

Figure 11–25. Serial Port Receive Overrun Avoided



11.3.7.2 Unexpected Receive Frame Synchronization: RSYNCERR

Figure 11–26 shows the decision tree that the receiver uses to handle all incoming frame synchronization pulses. The diagram assumes that the receiver has been activated ($\overline{\text{RRST}}$ = 1). Unexpected frame sync pulses can originate from an external source or from the internal sample rate generator. An unexpected frame sync pulse is defined as a sync pulse which occurs RDATDLY bit clocks earlier than the last transmitted bit of the previous frame. Any one of three cases can occur:

❑ Case 1: Unexpected FSR pulses with RFIG = 1. This case is discussed in section 8.3.6.1 and shown in Figure 11–21. Here, receive frame sync pulses are ignored and the reception continues.

❑ Case 2: Normal serial port reception. There are three reasons for a receive *not* to be in progress:

  - This FSR is the first after $\overline{\text{RRST}}$ = 1.
  - This FSR is the first after DRR has been read clearing an RFULL condition.
  - The serial port is in the inter-packet intervals. The programmed data delay (RDATDLY) for reception may start during these inter-packet intervals for the first bit of the next element to be received. Thus, at maximum frame frequency, frame synchronization can still be received RDATDLY bit clocks before the first bit of the associated element.

For this case, reception continues normally, because these are not unexpected frame sync pulses.

❑ Case 3: Unexpected receive frame synchronization with RFIG = 0 (unexpected frame not ignored). This case was shown in Figure 11–20 for maximum packet frequency. Figure 11–27 shows this case during normal operation of the serial port with time intervals between packets. Unexpected frame sync pulses are detected when they occur the value in RDATDLY bit clocks before the last bit of the previous element is received on DR. In both cases, RSYNCERR in the SPCR is set. RSYNCERR can be cleared only by receiver reset or by writing a 0 to this bit in the SPCR. If RINTM = 11b in the SPCR, RSYNCERR drives the receive interrupt (RINT) to the CPU.

Note:

Note that the RSYNCERR bit in the SPCR is a read/write bit, so writing a 1 to it sets the error condition. Typically, writing a 0 is expected.

Figure 11–26. Decision Tree Response to Receive Frame Synchronization Pulse

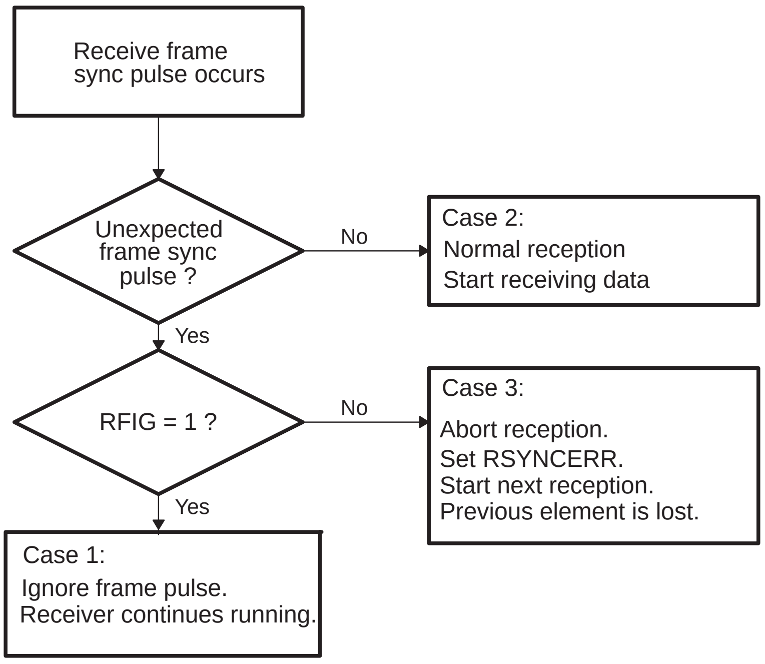
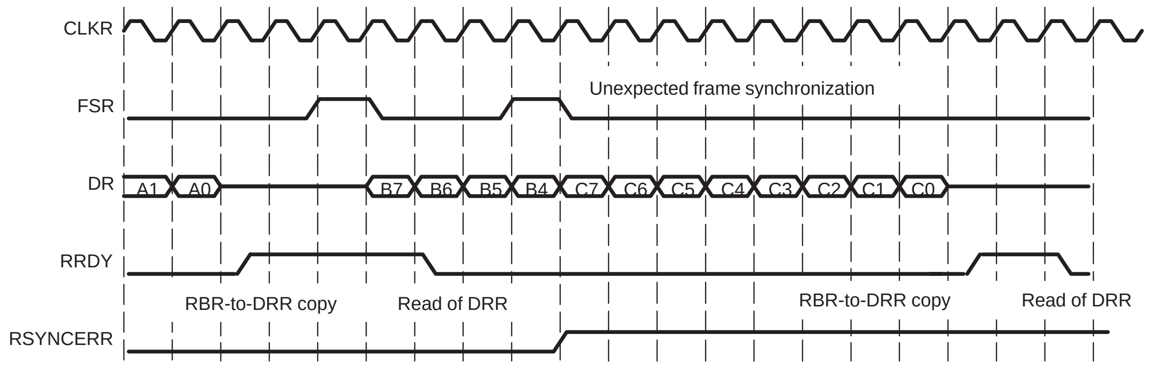


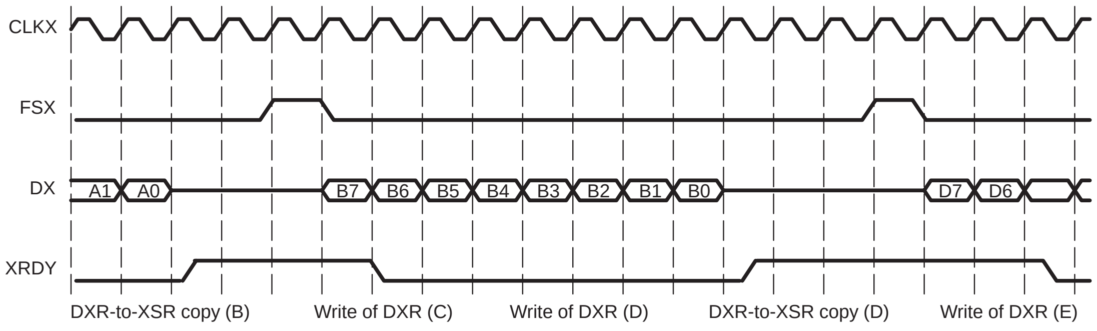
Figure 11–27. Unexpected Receive Synchronization Pulse



11.3.7.3 Transmit With Data Overwrite

Figure 11–28 shows what happens if the data in DXR is overwritten before it is transmitted. Suppose you load the DXR with data C. A subsequent write to the DXR overwrites C with D before C is copied to the XSR. Thus, C is never transmitted on DX. The CPU can avoid overwriting data by polling XRDY before writing to DXR or by waiting for a programmed XINT to be triggered by XRDY (XINTM = 00b). The DMA controller can avoid overwriting by synchronizing data writes with XEVT.

Figure 11–28. Transmit With Data Overwrite



11.3.7.4 Transmit Empty: $\overline{\text{XEMPTY}}$

$\overline{\text{XEMPTY}}$ indicates whether the transmitter has experienced under flow. Either of the following conditions causes $\overline{\text{XEMPTY}}$ to become active ($\overline{\text{XEMPTY}}$ = 0):

❑ During transmission, DXR has not been loaded since the last DXR-to-XSR copy, and all bits of the data element in the XSR have been shifted out on DX.

❑ The transmitter is reset ($\overline{\text{XRST}}$ = 0 or the device is reset) and then restarted.

During underflow condition, the transmitter continues to transmit the old data in DXR for every new frame sync signal that arrives on FSX until a new element is loaded into DXR by the CPU or the DMA controller. $\overline{\text{XEMPTY}}$ is deactivated ($\overline{\text{XEMPTY}}$ = 1) when this new element in DXR is transferred to XSR. In the case of internal frame sync generation, the transmitter regenerates a single FSX initiated by a DXR-to-XSR copy (FSXM = 1 in the PCR and FSGM = 0 in SRGR). Otherwise, the transmitter waits for the next frame synchronization.

When the transmitter is taken out of reset ($\overline{\text{XRST}}$ = 1), it is in a transmit ready (XRDY = 1) and transmit empty ($\overline{\text{XEMPTY}}$ = 0) condition. If DXR is loaded by the CPU or the DMA controller before FSX goes active, a valid DXR-to-XSR transfer occurs. This allows for the first element of the first frame to be valid even before the transmit frame sync pulse is generated or detected. Alternatively, if a transmit frame sync is detected before DXR is loaded, 0s are output on DX.

Figure 11–29 depicts a transmit underflow condition. After B is transmitted, B is retransmitted on DX if you fail to reload the DXR before the subsequent frame synchronization. Figure 11–30 shows the case of writing to DXR just before a transmit underflow condition that would otherwise occur. After B is transmitted, C is written to DXR before the next transmit frame sync pulse occurs so that C is successfully transmitted on DX, averting a transmit empty condition.

Figure 11–29. Transmit Empty

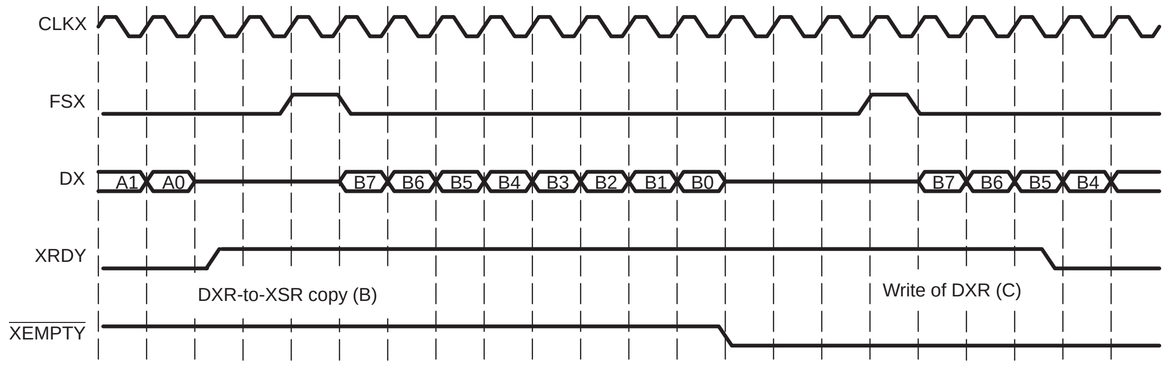
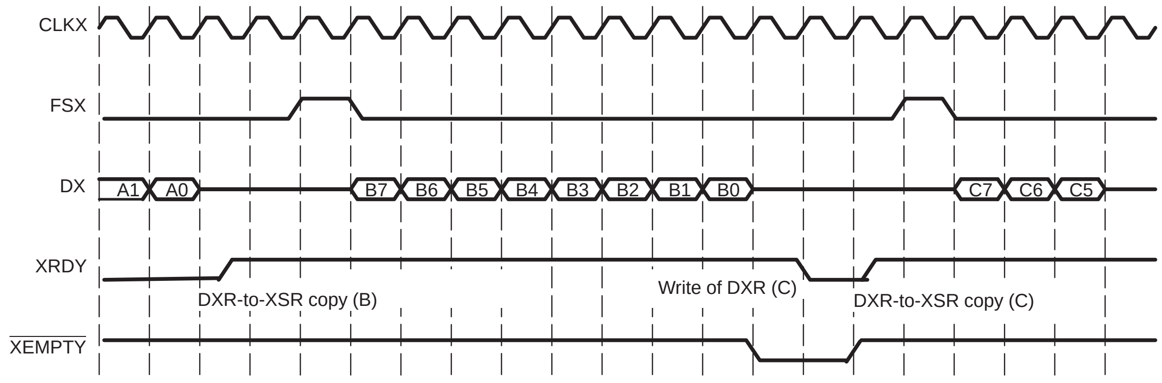


Figure 11–30. Transmit Empty Avoided

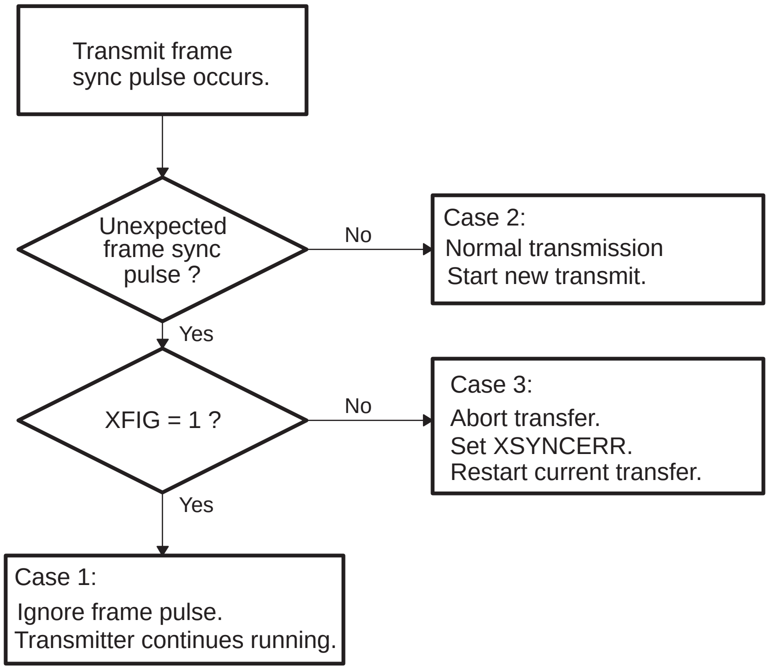


11.3.7.5 Unexpected Transmit Frame Synchronization: XSYNCERR

Figure 11–26 shows the decision tree that the transmitter uses to handle all incoming frame synchronization signals. The diagram assumes that the transmitter has been started ($\overline{\text{XRST}}$ = 1). An unexpected transmit frame sync pulse is defined as a sync pulse that occurs XDATDLY bit clocks earlier than the last transmitted bit of the previous frame. Any one of three cases can occur:

- ❑ Case 1: Unexpected FSX pulses with XFIG = 1. This case is discussed in section 11.3.6.1 and shown in Figure 11–21. In this case, unexpected FSX pulses are ignored, and the transmission continues.
- ❑ Case 2: FSX pulses with normal serial port transmission. This situation is discussed in section 11.3.5.3. There are two possible reasons for a transmit *not* to be in progress:
  - ■ This FSX pulse is the first one to occur after $\overline{\text{XRST}}$ = 1.
  - ■ The serial port is in the interpacket intervals. The programmed data delay (XDATDLY) may start during these interpacket intervals before the first bit of the next element is transmitted. Thus, if operating at maximum packet frequency, frame synchronization can still be received XDATDLY bit clocks before the first bit of the associated element.

Figure 11–31. Response to Transmit Frame Synchronization

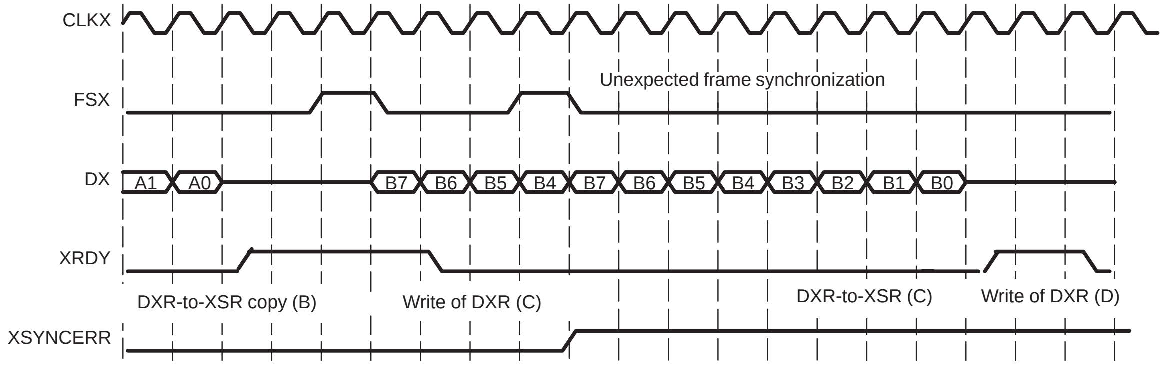

Transmit frame sync pulse occurs.
Unexpected frame sync pulse ?
No
Case 2: Normal transmission Start new transmit.
Yes
XFIG = 1 ?
No
Case 3: Abort transfer. Set XSYNCERR. Restart current transfer.
Yes
Case 1: Ignore frame pulse. Transmitter continues running.

- ❑ Case 3: Unexpected transmit frame synchronization with XFIG = 0. The case for frame synchronization with XFIG = 0 at maximum packet frequency is shown in Figure 11–20. Figure 11–32 shows the case for normal operation of the serial port with interpacket intervals. In both cases, XSYNCERR in the SPCR is set. XSYNCERR can be cleared only by transmitter reset or by writing a 0 to this bit in the SPCR. If XINTM = 11b in the SPCR, XSYNCERR drives the receive interrupt (XINT) to the CPU.

Note:

The XSYNCERR bit in the SPCR is a read/write bit, so writing a 1 to it sets the error condition. Typically, writing a 0 is expected.

Figure 11–32. Unexpected Transmit Frame Synchronization Pulse



11.3.8 Receive Data Justification and Sign Extension: RJUST

RJUST in the SPCR selects whether data in the RBR is right- or left-justified (with respect to the MSB) in the DRR. If right justification is selected, RJUST further selects whether the data is sign-extended or zero-filled. Table 11–12 summarizes the effect that various values of RJUST have on an example 12-bit receive data value ABCh.

Table 11–12. Effect of RJUST Values With 12-Bit Example Data ABCh

RJUST value	Justification	Extension	Value in DRR
00	Right	Zero-fill MSBs	0000 0ABCh
01	Right	Sign-extend MSBs	FFFF FABCh
10	Left	Zero-fill LSBs	ABC0 0000h
11	Reserved	Reserved	Reserved

11.3.9 32-Bit Bit Reversal: (R/X)WDREVRS

The 32-bit bit reversal feature is only available for the ’C6211/C6711 device. Normally all transfers are sent and received with the MSB first. However, if you set the RWDREVRS field to 1 in the RCR, or the XWDREVRS field to 1 in the XCR, the bit ordering of the 32-bit elements is reversed (LSB first) before being received by or sent from the serial port. The (R/X)WDLEN(1/2) fields in the (R/X)CR should be set to 101b to indicate 32-bit elements. Otherwise the operation is undefined.

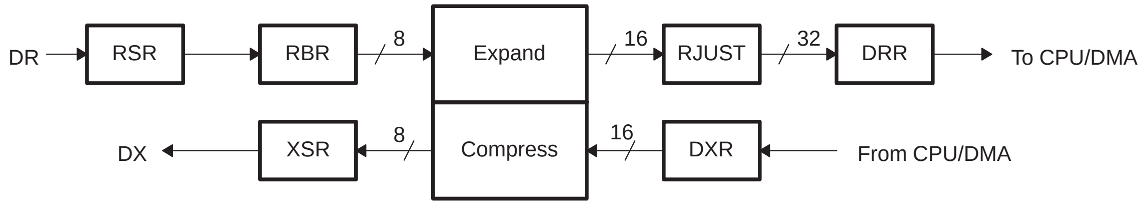
11.4 μ-LAW/A-LAW Companding Hardware Operation

Companding (*com*pressing and ex*panding*) hardware allows compression and expansion of data in either μ-law or A-law format. The specification for μ-law and A-law log PCM is part of the CCITT G.711 recommendation. The companding standard employed in the United States and Japan is μ-law and allows 14 bits of dynamic range. The European companding standard is A-law and allows 13 bits of dynamic range. Any values outside these ranges are set to the most positive or most negative value. Thus, for companding to work best here, the data transferred to and from the McBSP via the CPU or the DMA controller must be at least 16 bits wide.

The μ-law and A-law formats encode data into 8-bit code elements. Companded data is always 8-bits-wide, so the appropriate (R/X)WDLEN(1/2) must be set to 0, indicating an 8-bit serial data stream. If companding is enabled and either phase of the frame does not have an 8-bit element length, companding continues as if the element length is eight bits.

When companding is used, transmit data is encoded according to the specified companding law, and receive data is decoded to 2s-complement format. Companding is enabled and the desired format is selected by appropriately setting (R/X)COMPAND in the (R/X)CR, as indicated in Table 11–7. Compression occurs during the process of copying data from DXR to XSR and expansion occurs from RBR to DRR, as shown in Figure 11–33.

Figure 11–33. Companding Flow



For transmit data to be compressed, it should be 16-bit, left-justified data, such as LAW16 as shown in Figure 11–34. The value can be either 13 or 14 bits wide, depending on the companding law. This 16-bit data is aligned in DXR, as shown in Figure 11–35.

Figure 11–34. Companding Data Formats

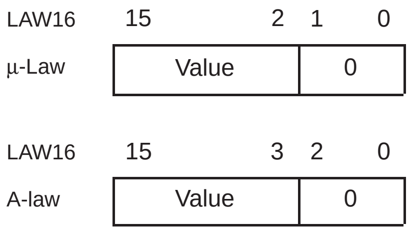
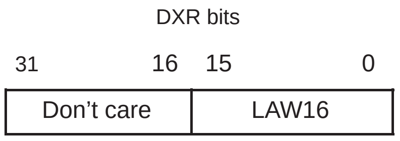


Figure 11–35. Transmit Data Companding Format in DXR



For reception, the 8-bit compressed data in RBR is expanded to a left-justified 16-bit data, LAW16. This can be further justified to a 32-bit data by programming the RJUST field in the SPCR as shown in Table 11–13.

Table 11–13. Justification of Expanded Data in DRR

	DRR Bits	
RJUST	31 16	15 0
00	0	LAW16
01	sign	LAW16
10	LAW16	0
11	Reserved	

11.4.1 Companding Internal Data

If the McBSP is otherwise unused, the companding hardware can compand internal data. This hardware can be used to:

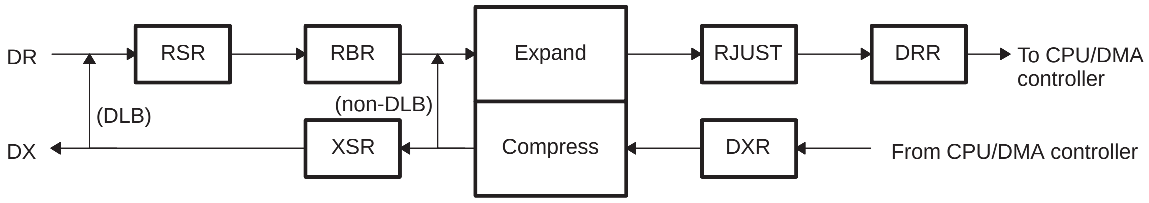
- ❑ Convert linear data to the appropriate μ-law or A-law format
- ❑ Convert μ-law or A-law data to the linear format
- ❑ Observe the quantization effects in companding by transmitting linear data and compressing and re-expanding this data. This is useful only if both XCOMPAND and RCOMPAND enable the same companding format.

Figure 11–36 shows two methods by which the McBSP can compand internal data. Data paths for these two methods are indicated by (DLB) and (non-DLB) arrows.

- **Non-DLB:** When both the transmit and receive sections of the serial port are reset, the DRR and DXR are internally connected through the companding logic. Values from the DXR are compressed as determined by XCOMPAND and then expanded as determined by RCOMPAND. RRDY and XRDY bits are not set. However, data is available in DRR four CPU clocks after being written to DXR. The advantage of this method is its speed. The disadvantage is that there is no synchronization available to the CPU and the DMA controller to control the flow of data.

- **DLB:** The McBSP is enabled in digital loopback (DLB) 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 the DMA controller to these conversions, respectively. Here, the time for this companding depends on the serial bit rate selected.

Figure 11–36. Companding of Internal Data



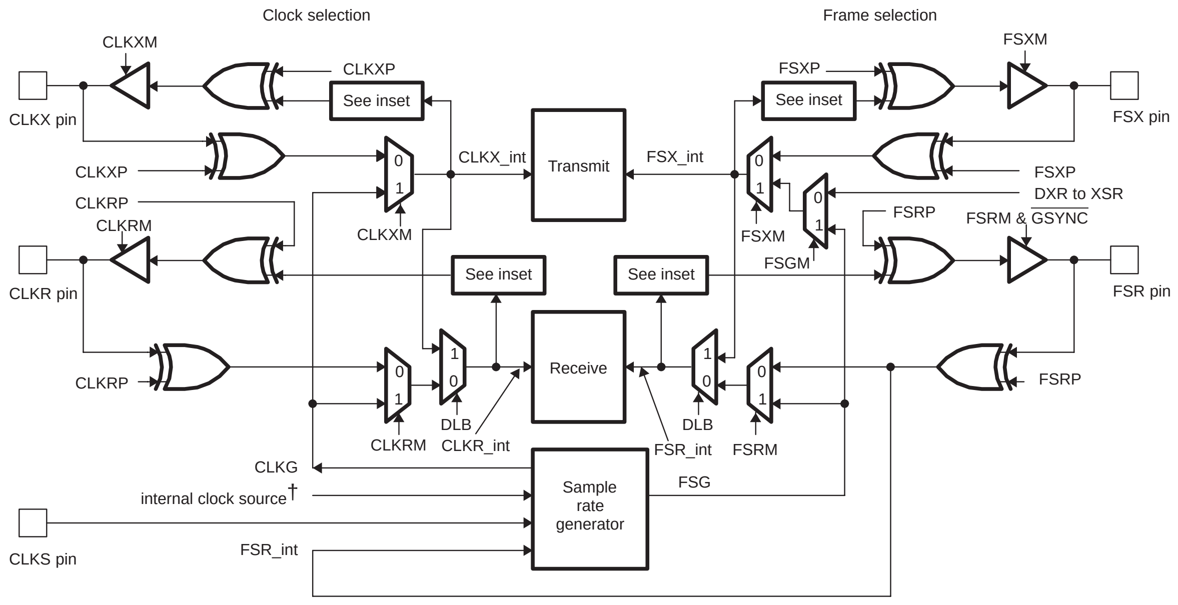
11.4.1.1 Bit Ordering

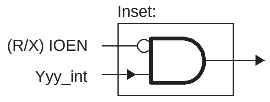
Normally, all transfers on the McBSP are sent and received with the MSB first. However, certain 8-bit data protocols (that do not use companded data) require the LSB to be transferred first. By setting the (R/X)COMPAND = 01b in the (R/X)CR, the bit ordering of 8-bit elements is reversed (LSB first) before being sent to the serial port. Like the companding feature, this feature is enabled only if the appropriate (R/X)WDLEN(1/2) bit is set to 0, indicating that 8-bit elements are to be transferred serially.

11.5 Programmable Clock and Framing

The McBSP has several means of selecting clocking and framing for both the receiver and transmitter. Clocking and framing can be sent to both portions by the sample rate generator. Each portion can select external clocking and/or framing independently. Figure 11–37 is a block diagram of the clock and frame selection circuitry.

Figure 11–37. Clock and Frame Generation





† ’C6201/C6202/C6701 uses CPU clock as the internal clock source to the sample rate generator. ’C6211/C6711 uses CPU/2 clock as the internal clock source to the sample rate generator.

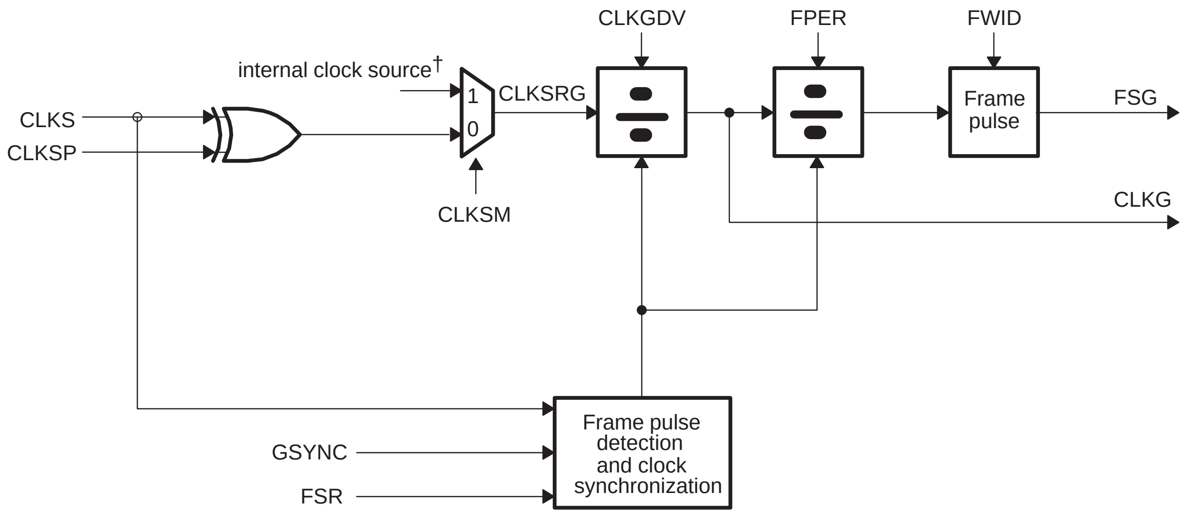
11.5.1 Sample Rate Generator Clocking and Framing

The sample rate generator is composed of a 3-stage clock divider that provides a programmable data clock (CLKG) and framing signal (FSG), as shown in Figure 11–38. CLKG and FSG are McBSP internal signals that can be programmed to drive receive and/or transmit clocking, CLK(R/X), and framing, FS(R/X). The sample rate generator can be programmed to be driven by an internal clock source or an internal clock derived from an external clock source. The three stages of the sample rate generator circuit compute:

- ❑ Clock divide-down (CLKGDV): The number of input clocks per data bit clock
- ❑ Frame period (FPER): The frame period in data bit clocks
- ❑ Frame width (FWID): The width of an active frame pulse in data bit clocks

In addition, a frame pulse detection and clock synchronization module allows synchronization of the clock divide-down with an incoming frame pulse. The operation of the sample rate generator during device reset is described in section 11.3.1.

Figure 11–38. Sample Rate Generator



† ’C6201/C6202/C6701 uses CPU clock as the internal clock source to the sample rate generator. ’C6211/C6711 uses CPU/2 clock as the internal clock source to the sample rate generator.

11.5.1.1 Sample Rate Generator Register (SRGR)

The sample rate generator register (SRGR) shown in Figure 11–39 and summarized in Table 11–14, controls the operation of various features of the sample rate generator. This section describes the fields in the SRGR.

Figure 11–39. Sample Rate Generator Register (SRGR)

31	30	29	28	27–16
GSYNC	CLKSP	CLKSM	FSGM	FPER
RW, +0	RW, +0	RW, +1	RW, +0	RW, +0

15–8	7–0
FWID	CLKGDV
RW, +0	RW, +1

Table 11–14. Sample Rate Generator Register (SRGR) Field Summary

Name	Function	Section
GSYNC	Sample rate generator clock synchronization. Used only when the external clock (CLKS) drives the sample rate generator clock (CLKSM = 0). GSYNC = 0: The sample rate generator clock (CLKG) is free running. GSYNC = 1: (CLKG) is running but is resynchronized, and the frame sync signal (FSG) is generated only after the receive frame synchronization signal (FSR) is detected. Also, the frame period (FPER) is a don't care because the period is dictated by the external frame sync pulse.	11.5.2.4
CLKSP	CLKS polarity clock edge select. Used only when the external clock CLKS drives the sample rate generator clock (CLKSM = 0). CLKSP = 0: The rising edge of CLKS generates CLKG and FSG. CLKSP = 1: The falling edge of CLKS generates CLKG and FSG.	11.5.2.3
CLKSM	McBSP sample rate generator clock mode CLKSM = 0: The sample rate generator clock is derived from CLKS. CLKSM = 1: (Default value) The sample rate generator clock is derived from the internal clock source.	11.5.2.1
FSGM	Sample rate generator transmit frame synchronization mode. Used when FSXM = 1 in PCR. FSGM = 0: The transmit frame sync signal (FSX) is generated on every DXR-to-XSR copy. FSGM = 1: The transmit frame sync signal is driven by the sample rate generator frame sync signal, FSG.	11.5.3.3

Table 11–14. Sample Rate Generator Register (SRGR) Field Summary (Continued)

Name	Function	Section
FPER	Frame period. This field's value plus 1 determines when the next frame sync signal should become active. Valid values: 0 to 4095	11.5.3.1
FWID	Frame width. This field's value plus 1 is the width of the frame sync pulse, FSG, during its active period. Valid values: 0 to 255	11.5.3.1
CLKGDV	Sample rate generator clock divider. This value is used as the divide-down number to generate the required sample rate generator clock frequency. The default value is 1. Valid values: 0 to 255	11.5.2.2

11.5.1.2 Sample Rate Generator Reset Procedure

The sample rate generator reset and initialization procedure is as follows:

1) During device reset, $\overline{\text{GRST}}$ = 0. Otherwise, during normal operation, reset the sample rate generator with $\overline{\text{GRST}}$ = 0 in SPCR, provided CLKG and/or FSG ($\overline{\text{FRST}}$ = 1) are not used by any portion of the McBSP. If $\overline{\text{GRST}}$ is low due to device reset, CLKG is driven by a divide-by-2 internal clock and FSG is driven inactive. The internal clock source for the 'C6211/C6711 is CPU clock, while the internal clock source for 'C6211/C6711 is CPU/2 clock (half of the CPU clock frequency). CLKG and FSG are inactive when $\overline{\text{GRST}}$ = 0. If necessary, set $\overline{\text{(R/X)RST}}$ = 0.
2) Program SRGR as required. If necessary, other control registers can be written with desired values if the respective portion (R/X) is in reset.
3) Wait two CLKSRG clocks. This is to ensure proper internal synchronization.
4) Set $\overline{\text{GRST}}$ = 1 to enable the sample rate generator.
5) Wait two CLKG bit clocks.
6) Pull the receiver and/or transmitter out of reset ($\overline{\text{(R/X)RST}}$ = 1) if required.
7) On the next rising edge of CLKSRG, CLKG transitions to 1 and starts clocking with a frequency equal to either (internal clock / (1+CLKGDV)) if CLKSM =1 or CLKS clock/(1 + CLKGDV) if CLKSM = 0. The internal clock source for the 'C6211/C6711 is CPU clock, while the internal clock source for 'C6211/C6711 is CPU/2 clock (half of the CPU clock frequency).
8) After the required data acquisition setup, such as writing to DXR, $\overline{\text{FRST}}$ can be written with 1 in the SPCR if an internally generated frame pulse is required. FSG is generated on an active edge after eight CLKG clocks have elapsed.

11.5.2 Data Clock Generation

When the receive/transmit clock mode is set to 1 (CLK(R/X)M = 1), the data clocks (CLK(R/X)) are driven by the internal sample rate generator output clock, CLKG. You can select for the receiver and transmitter from a variety of data bit clocks including:

- The input clock to the sample rate generator, which can be either the internal clock source or a dedicated external clock source (CLKS). The internal clock source for the ’C6211/C6711 is CPU clock, while the internal clock source for ’C6211/C6711 is CPU/2 clock (half of the CPU clock frequency).
- The input clock source (internal clock source or external clock CLKS) to the sample rate generator can be divided down by a programmable value (CLKGDV) to drive CLKG.

Regardless of the source to the sample rate generator, the rising edge of CLKSRG (see Figure 11–38) generates CLKG and FSG (see section 11.5.2.3).

11.5.2.1 Input Clock Source Mode: CLKSM

The CLKSM bit in the SRGR selects either the CPU clock (CLKSM = 1) or the external clock input (CLKSM = 0), CLKS, as the source for the sample rate generator input clock. Any divide periods are divide-downs calculated by the sample rate generator and are timed by this input clock selection. The McBSP cannot run faster than half of the CPU clock frequency. Therefore, when CLKSM = 1, the minimum value of CLKGDV should be 1 for the ’C6201/C6202/C6701. For the ’C6211/C6711, even if CLKSM = 1 you can set CLKGDV to the minimum of 0 because a CPU/2 clock drives the sample rate generator.

11.5.2.2 Sample Rate Generator Data Bit Clock Rate: CLKGDV

The first divider stage generates the serial data bit clock from the input clock. This divider stage uses a counter that is preloaded by CLKGDV and that contains the divide ratio value. The output of this stage is the data bit clock that is output on the sample rate generator output, CLKG, and that serves as the input for the second and third divider stages.

CLKG has a frequency equal to 1/(CLKGDV+1) of the sample rate generator input clock. Thus, the sample-rate 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) the high state duration is p + 1 cycles and the low state duration is p cycles.

11.5.2.3 Bit Clock Polarity: CLKSP

The external clock (CLKS) is selected to drive the sample rate generator clock divider by selecting CLKSM = 0. In this case, the CLKSP bit in the SRGR selects the edge of CLKS on which sample rate generator data bit clock (CLKG) and frame sync signal (FSG) are generated. Since the rising edge of CLKSRG generates CLKG and FSG, the rising edge of CLKS when CLKSP = 0 or the falling edge of CLKS when CLKSP = 1 causes the transition on CLKG and FSG.

11.5.2.4 Bit Clock and Frame Synchronization

When CLKS is selected to drive the sample rate generator (CLKSM = 0), GSYNC can be used to configure the timing of CLKG relative to CLKS. GSYNC = 1 ensures that the McBSP and the external device to which it is communicating are dividing down the CLKS with the same phase relationship. If GSYNC = 0, this feature is disabled and CLKG runs freely and is not resynchronized. If GSYNC = 1, an inactive-to-active transition on FSR triggers a resynchronization of CLKG and the generation of FSG. CLKG always begins at a high state after synchronization. Also, FSR is always detected at the same edge of CLKS that generates CLKG, regardless of the length the FSR pulse. Although an external FSR is provided, FSG can still drive internal receive frame synchronization when GSYNC = 1. When GSYNC = 1, FPER is a don’t care, because the frame period is determined by the arrival of the external frame sync pulse.

Figure 11–40 and Figure 11–41 show this operation with various polarities of CLKS and FSR. These figures assume that FWID is 0, for a FSG = 1 CLKG wide.

Figure 11–40. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 1

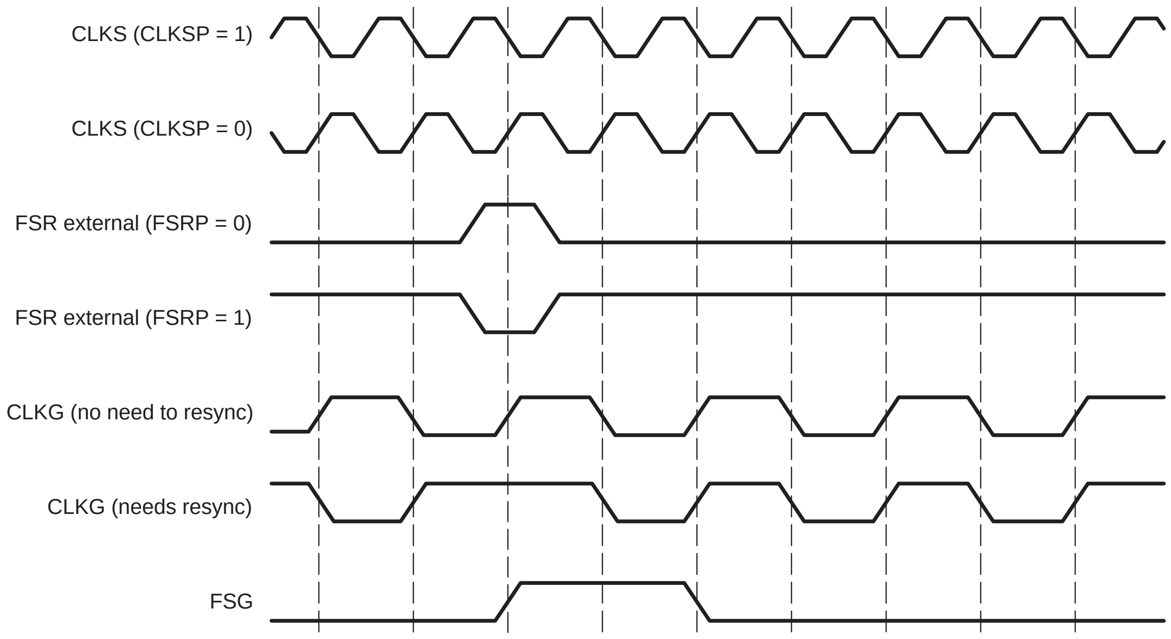
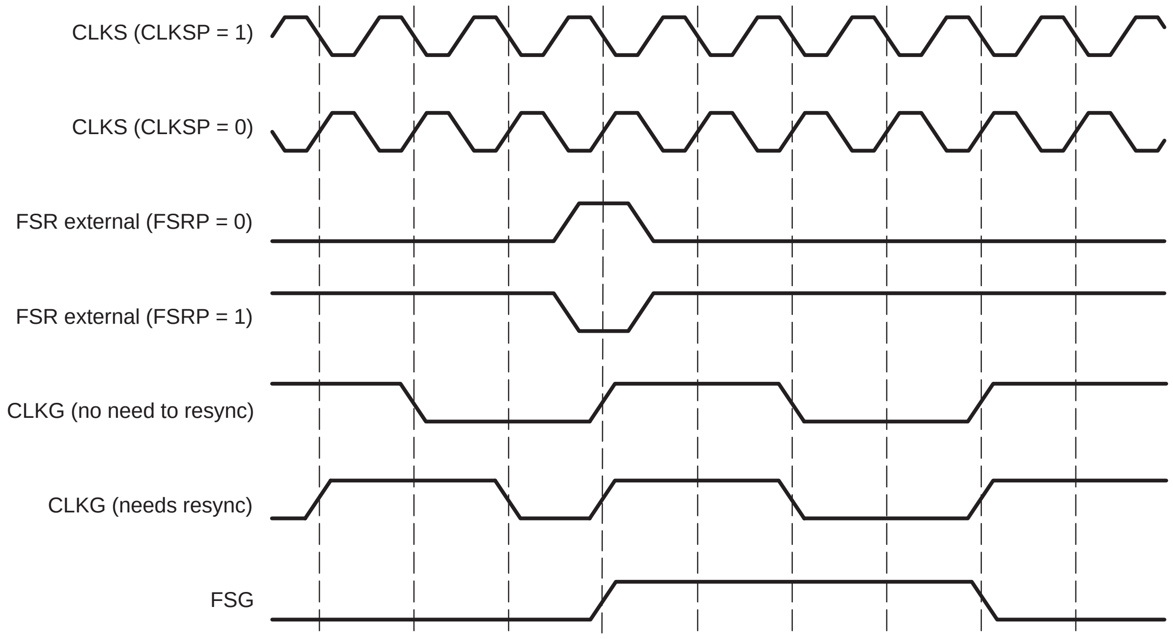


Figure 11–41. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 3



These figures show what happens to CLKG when it is initially in sync and GSYNC = 1, as well as when it is not in sync with the frame synchronization and GSYNC = 1.

When GSYNC = 1, the transmitter can operate synchronously with the receiver, provided that the following conditions are met:

- FSX is programmed to be driven by the sample rate generator frame sync, FSG (FSGM = 1 in the SRGR and FSXM = 1 in the PCR). If the input FSR has timing that enables it to be sampled by the falling edge of CLKG, it can be used instead by setting FSXM = 0 in the PCR and connecting FSR to FSX externally.
- The sample-rate generator clock should drive the transmit and receive bit clock (CLK(R/X)M = 1 in the SPCR). Therefore, the CLK(R/X) pin should not be driven by any other source.

11.5.2.5 Digital Loopback Mode: DLB

Setting DLB = 1 in the SPCR enables digital loopback mode. In DLB mode, DR, FSR, and CLKR are internally connected through multiplexers to DX, FSX, and CLKX, respectively, as shown in Figure 11–37. DLB mode allows testing of serial port code with a single DSP device.

11.5.2.6 Receive Clock Selection: DLB, CLKRM

Table 11–15 shows how the digital loopback bit (DLB) and the CLKRM bit in the PCR select the receiver clock. In digital loopback mode (DLB = 1), the transmitter clock drives the receiver. CLKRM determines whether the CLKR pin is an input or an output.

Table 11–15. Receive Clock Selection

DLB in SPCR	CLKRM in PCR	Source of Receive Clock	CLKR Function
0	0	CLKR acts as an input driven by the external clock and inverted as determined by CLKRP before being used.	Input
0	1	The sample rate generator clock (CLKG) drives CLKR.	Output. CLKG inverted as determined by CLKRP before being driven out on CLKR.
1	0	CLKX_int drives the receive clock CLKR_int as selected and is inverted. See Table 11–16.	High impedance
1	1	CLKX_int drives CLKR_int as selected and is inverted. See Table 11–16.	Output. CLKR (same as CLKX) inverted as determined by CLKRP before being driven out.

11.5.2.7 Transmit Clock Selection: CLKXM

Table 11–16 shows how the CLKXM bit in the PCR selects the transmit clock and whether the CLKX pin is an input or output.

Table 11–16. Transmit Clock Selection

CLKXM in PCR	Source of Transmit Clock	CLKX Function
0	The external clock drives the CLKX input pin. CLKX is inverted as determined by CLKXP before being used.	Input
1	The sample rate generator clock, CLKG, drives the transmit clock	Output. CLKG is inverted as determined by CLKXP before being driven out on CLKX.

11.5.3 Frame Sync Signal Generation

Data frame synchronization is independently programmable for the receiver and the transmitter for all data delay values. When set to 1 the $\overline{\text{FRST}}$ bit in the SPCR activates the frame generation logic to generate frame sync signals, provided that FSGM = 1 in SRGR. The frame sync programming options are:

❑ A frame pulse with a programmable period between sync pulses and a programmable active width specified in the sample rate generator register (SRGR).

❑ The transmitter can trigger its own frame sync signal that is generated by a DXR-to-XSR copy. This causes a frame sync to occur on every DXR-to-XSR copy. The data delays can be programmed as required. However, maximum packet frequency cannot be achieved in this method for data delays of 1 and 2.

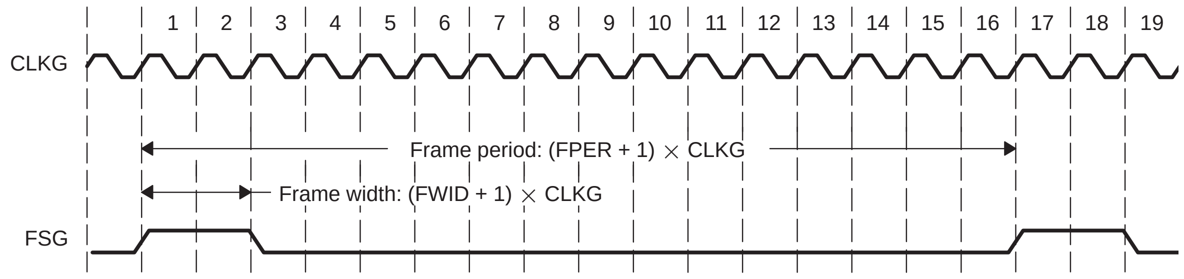
❑ Both the receiver and transmitter can independently select an external frame synchronization on the FSR and FSX pins, respectively.

11.5.3.1 Frame Period and Frame Width: FPER and FWID

The FPER block is a 12-bit down counter that can count down the generated data clocks from 4095 to 0. FPER controls the period of active frame sync pulses. The FWID block in the sample rate generator is an 8-bit down counter. The FWID field controls the active width of the frame sync pulse.

When the sample rate generator comes out of reset, FSG is in an inactive (low) state. After this, when $\overline{\text{FRST}}$ = 1 and FSGM = 1, frame sync signals are generated. The frame width value (FWID + 1) is counted down on every CLKG cycle until it reaches 0 when FSG goes low. Thus, the value of FWID+1 determines an active frame pulse width ranging from 1 to 256 data bit clocks. At the same time, the frame period value (FPER + 1) is also counting down, and when this value reaches 0, FSG goes high again, indicating a new frame is beginning. Thus, the value of FPER + 1 determines a frame length from 1 to 4096 data bits. When GSYNC = 1, the value of FPER does not matter. Figure 11–42 shows a frame of 16 CLKG periods (FPER = 15 or 00001111b).

Figure 11–42. Programmable Frame Period and Width


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
CLKG
Frame period: (FPER + 1) × CLKG
Frame width: (FWID + 1) × CLKG
FSG

11.5.3.2 Receive Frame Sync Selection: DLB, FSRM, GSYNC

Table 11–17 explains how you can select various sources to provide the receive frame synchronization signal. Note that in digital loopback mode (DLB = 1) the transmit frame sync signal is used as the receive frame sync signal and that DR is internally connected to DX.

Table 11–17. Receive Frame Synchronization Selection

DLB in SPCR	FSR in PCR	GSYNC in SRGR	Source of Receive Frame Synchronization	FSR Pin Function
0	0	X	External frame sync signal drives the FSR input pin, whose signal is then inverted as determined by FSRP before being used as FSR_int.	Input
0	1	0	Sample rate generator frame sync signal (FSG) drives FSR_int, $\overline{\text{FRST}}$ = 1.	Output. FSG is inverted as determined by FSRP before being driven out on the FSR pin.
0	1	1	Sample rate generator frame sync signal (FSG) drives FSR_int, $\overline{\text{FRST}}$ = 1.	Input. The external frame sync input on FSR is used to synchronize CLKG and generate FSG.
1	0	0	FSX_int drives FSR_int. FSX is selected as shown in Table 11–18.	High impedance
1	X	1	FSX_int drives FSR_int and is selected as shown in Table 11–18.	Input. External FSR is not used for frame synchronization but is used to synchronize CLKG and generate FSG since GSYNC = 1.
1	1	0	FSX_int drives FSR_int and is selected as shown in Table 11–18.	Output. Receive (same as transmit) frame synchronization is inverted as determined by FSRP before being driven out.

11.5.3.3 Transmit Frame Sync Signal Selection: FSXM, FSGM

Table 11–18 shows how you can select the source of transmit frame synchronization pulses. The three choices are:

- ❑ External frame sync input
- ❑ The sample rate generator frame sync signal, FSG
- ❑ A signal that indicates a DXR-to-XSR copy has been made

Table 11–18. Transmit Frame Synchronization Selection

FSXM in PCR	FSGM in SRGR	Source of Transmit Frame Synchronization	FSX Pin Function
0	X	External frame sync input on the FSX pin. This is inverted by FSXP before being used as FSX_int.	Input
1	1	Sample rate generator frame sync signal (FSG) drives FSX_int. $\overline{\text{FRST}}$ = 1	Output. FSG is inverted by FSXP before being driven out on FSX.
1	0	A DXR-to-XSR copy activates transmit frame sync signal.	Output. 1-bit-clock-wide signal inverted as determined by FSXP before being driven out on FSX.

11.5.3.4 Frame Detection for Initialization

To facilitate detection of frame synchronization, the receive and transmit CPU interrupts (RINT and XINT) can be programmed to detect frame synchronization by setting RINTM = XINTM = 10b in the SPCR. Unlike other types of serial port interrupts, this one can operate while the associated portion of the serial port is in reset (for example, RINT can be activated while the receiver is in reset). In that case, the FS(R/X)M and FS(R/X)P still select the appropriate source and polarity of frame synchronization. Thus, even when the serial port is in reset, 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 receive and transmit portions of the serial port. A new frame synchronization pulse can be detected, after which the CPU can safely take the serial port out of reset.

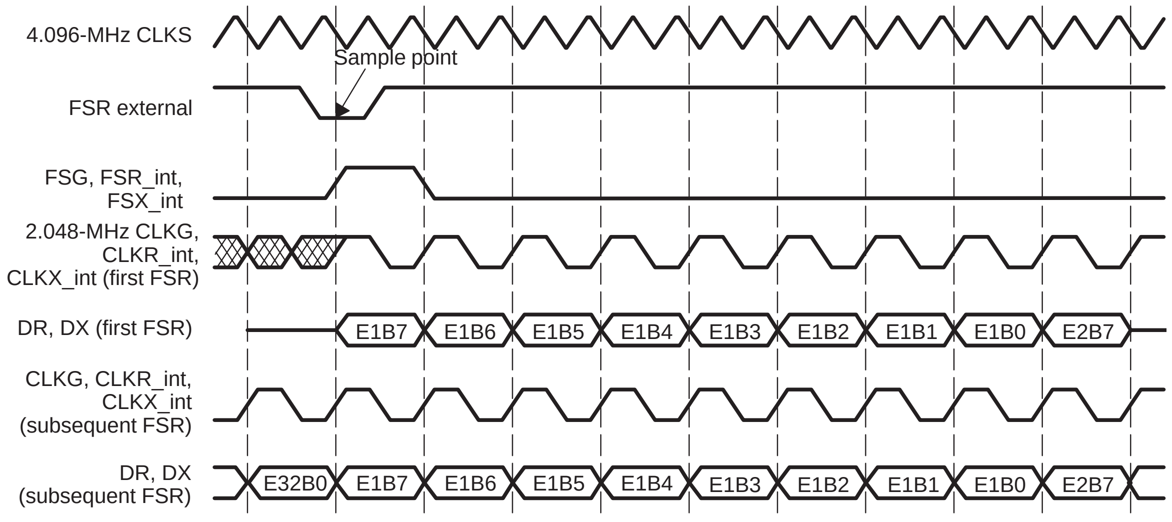
11.5.4 Clocking Examples

11.5.4.1 Double-Rate ST-BUS Clock

Figure 11–43 shows the McBSP timing to be compatible with the Mitel ST-Bus™. The operation is running at maximum frame frequency.

- ❑ CLK(R/X)M = 1: CLK(R/X)_int generated internally by sample rate generator
- ❑ GSYNC = 1: CLKG is synchronized with the external frame sync signal input on FSR. CLKG is not synchronized (it runs freely) until the frame sync signal is active. Also, FSR is regenerated internally to form a minimum pulse width.
- ❑ CLKSM = 0: external clock (CLKS) drives the sample rate generator
- ❑ CLKSP = 1: falling edge of CLKS generates CLKG and thus CLK(R/X)_int
- ❑ CLKGDV = 1: receive clock (shown as CLKR) is half of CLKS frequency
- ❑ FS(R/X)P = 1: active (low) frame sync pulse
- ❑ (R/X)FRLEN1 = 11111b: 32 elements per frame
- ❑ (R/X)WDLEN1 = 0: 8-bit element
- ❑ (R/X)PHASE = 0: single-phase frame and thus (R/X)FRLEN2 = (R/X)WDLEN2 = X
- ❑ (R/X)DATDLY = 0: no data delay

Figure 11–43. ST-BUS and MVIP Example



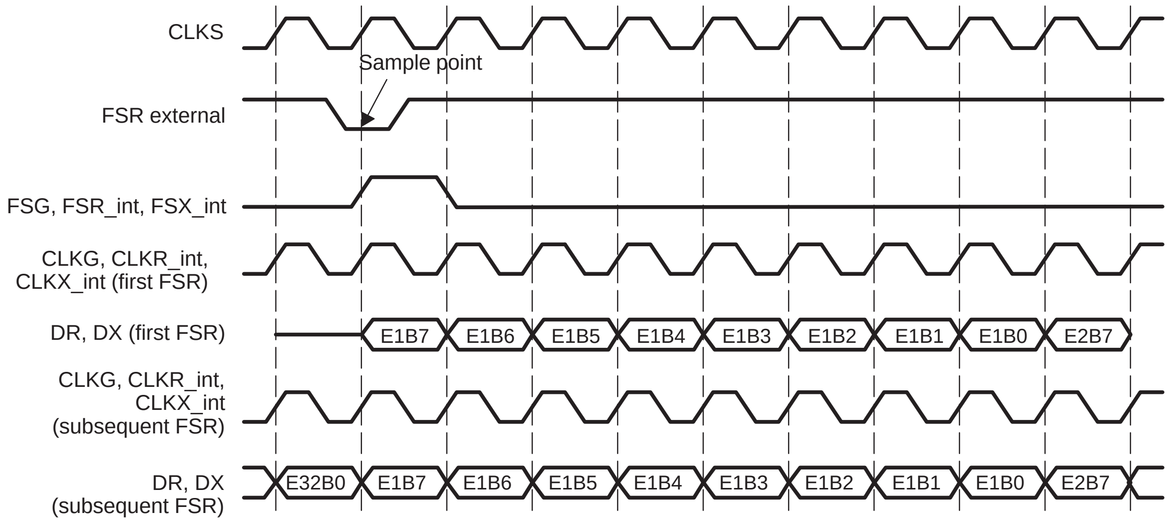
11.5.4.2 Single-Rate ST-BUS Clock

This example is the same as the ST-BUS example except for the following items:

❑ CLKGDV = 0: CLKS drives CLK(R/X)_int without any divide down (single-rate clock).

❑ CLKSP = 0: The rising edge of CLKS generates internal clocks CLKG and CLK(R/X)_int.

Figure 11–44. Single-Rate Clock Example



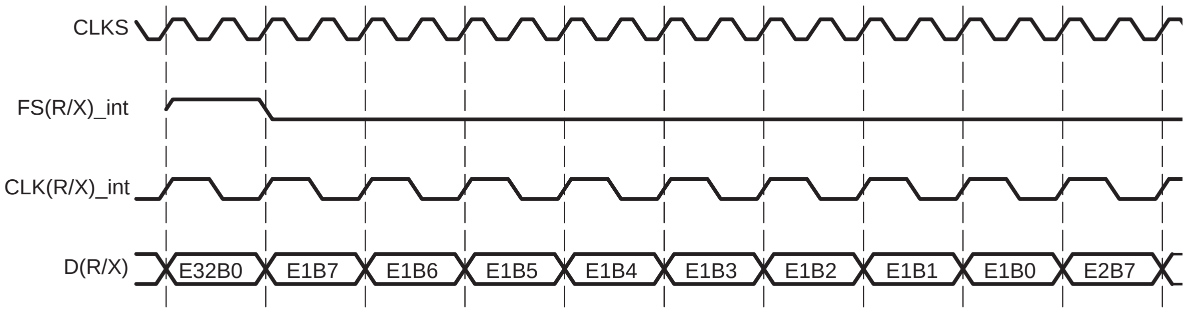
The rising edge of CLKS detects the external FSR. This external frame sync pulse resynchronizes the internal McBSP clocks and generates the frame sync for internal use. The internal frame sync is generated so that it is wide enough to be detected on the falling edge of the internal clocks.

11.5.4.3 Double-Rate Clock

This example is the same as the ST-BUS example except for the following:

- ❑ CLKSP = 0: The rising edge of CLKS generates CLKG and CLK(R/X).
- ❑ CLKGDV = 1: CLKG, CLKR_int, and CLKX_int frequencies are half of the CLKS frequency.
- ❑ GSYNC = 0: CLKS drives CLKG. CLKG runs freely and is not resynchronized by FSR.
- ❑ FS(R/X)M = 0: Frame synchronization is externally generated. The framing pulse is wide enough to be detected.
- ❑ FS(R/X)P = 0: Active (high) input frame sync signal
- ❑ (R/X)DATDLY = 1: Specifies a data delay of one bit

Figure 11–45. Double-Rate Clock Example



11.6 Multichannel Selection Operation

Multiple channels can be independently selected for the transmitter and receiver by configuring the McBSP with a single-phase frame. Each frame represents a time-division multiplexed data stream. The number of elements per frame represented by (R/X)FRLEN1 denotes the number of channels available for selection. Thus, to save memory and bus bandwidth, multichannel selection allows independent enabling of particular elements for transmission and reception. Up to 32 elements in a bit stream of up to 128 elements can be enabled at any given time.

If a receive element is not enabled:

- RRDY is not set to 1 upon reception of the last bit of the element.
- RBR is not copied to DRR upon reception of the last bit of the element. Thus, RRDY is not set active. This feature also implies that no interrupts or synchronization events are generated for this element.

If a transmit element is not enabled:

- DX is in the high impedance state.
- A DXR-to-XSR transfer is not automatically triggered at the end of serial transmission of the related element.
- $\overline{\text{XEMPTY}}$ and XRDY are not affected by the end of transmission of the related serial element.

An enabled transmit element can have its data masked or transmitted. When data is masked, the DX pin is forced to the high-impedance state even though the transmit channel is enabled.

11.6.1 Multichannel Operation Control Registers

The following control registers are used in multichannel operation:

- The multichannel control register (MCR)
- The transmit channel enable register (XCER)
- The receive channel enable register (RCER)

Figure 11–46. Multichannel Control Register

31–25	24–23	22–21	20–18	17–16
rsvd	XPBBLK	XPABLK	XCBLK	XMCM
R, +0000 000	RW, +00	RW, +00	R, +000	RW, +00

15–9	8–7	6–5	4–2	1	0
rsvd	RPBBLK	RPABLK	RCBLK	rsvd	RMCM
R, +0000 000	RW, +00	RW, +00	R, +000	R, +0	RW, +0

Table 11–19. Multichannel Control Register Field Descriptions

Name	**Function**	**Section**
RMCM	Receive multichannel selection enable RMCM = 0: All channels are enabled. RMCM = 1: All elements are disabled. Required channels are selected by enabling RP(A/B)BLK and RCER appropriately.	11.6.2
XMCM	Transmit multichannel selection enable XMCM = 00b: All elements are enabled without masking (DX is always driven during transmission of data). DX is masked or driven to hi-Z during inter-packet intervals, when a channel is masked (regardless of whether it is enabled), or when an element is disabled. XMCM = 01b: All elements are disabled and therefore masked by default. Required elements are selected by enabling XP(A/B)BLK and XCER appropriately and these selected elements are not masked, DX is always driven. XMCM = 10b: All elements are enabled but masked. Selected elements that are enabled via XP(A/B)BLK and XCER are unmasked. XMCM = 11b: All elements are disabled and therefore masked by default. Required elements are selected by enabling RP(A/B)BLK and RCER appropriately. Selected elements can be unmasked by RP(A/B)BLK and XCER. This mode is used for symmetric transmit and receive operation. (See section 11.6.3 for more details on symmetric operation).	11.6.3

Table 11–19. Multichannel Control Register Field Descriptions (Continued)

Name	Function	Section
RCBLK	Receive current subframe	11.6.3.2
	RCBLK = 000b: Subframe 0. Element 0 to element 15	
	RCBLK = 001b: Subframe 1. Element 16 to element 31	
	RCBLK = 010b: Subframe 2. Element 32 to element 47	
	RCBLK = 011b: Subframe 3. Element 48 to element 63	
	RCBLK = 100b: Subframe 4. Element 64 to element 79	
	RCBLK = 101b: Subframe 5. Element 80 to element 95	
	RCBLK = 110b: Subframe 6. Element 96 to element 111	
	RCBLK = 111b: Subframe 7. Element 112 to element 127	
XCBLK	Transmit current subframe	11.6.3.2
	XCBLK = 000b: Subframe 0. Element 0 to element 15	
	XCBLK = 001b: Subframe 1. Element 16 to element 31	
	XCBLK = 010b: Subframe 2. Element 32 to element 47	
	XCBLK = 011b: Subframe 3. Element 48 to element 63	
	XCBLK = 100b: Subframe 4. Element 64 to element 79	
	XCBLK = 101b: Subframe 5. Element 80 to element 95	
	XCBLK = 110b: Subframe 6. Element 96 to element 111	
	XCBLK = 111b: Subframe 7. Element 112 to element 127	
RPBBLK	Receive partition B subframe	11.6.3
	RPBBLK = 00b: Subframe 1. Element 16 to element 31	
	RPBBLK = 01b: Subframe 3. Element 48 to element 63	
	RPBBLK = 10b: Subframe 5. Element 80 to element 95	
	RPBBLK = 11b: Subframe 7. Element 112 to element 127	
XPBBLK	Transmit partition B subframe	11.6.3
	XPBBLK = 00b: Subframe 1. Element 16 to element 31	
	XPBBLK = 01b: Subframe 3. Element 48 to element 63	
	XPBBLK = 10b: Subframe 5. Element 80 to element 95	
	XPBBLK = 11b: Subframe 7. Element 112 to element 127	

Table 11–19. Multichannel Control Register Field Descriptions (Continued)

Name	Function	Section

RPABLK	Receive partition A subframe
	RPABLK = 00b: Subframe 0. Element 0 to element 15
	RPABLK = 01b: Subframe 2. Element 32 to element 47
	RPABLK = 10b: Subframe 4. Element 64 to element 79
	RPABLK = 11b: Subframe 6. Element 96 to element 111
		11.6.3

XPABLK	Transmit partition A subframe
	XPABLK = 00b: Subframe 0. Element 0 to element 15
	XPABLK = 01b: Subframe 2. Element 32 to element 47
	XPABLK = 10b: Subframe 4. Element 64 to element 79
	XPABLK = 11b: Subframe 6. Element 96 to element 111
		11.6.3

11.6.2 Enabling Multichannel Selection

Multichannel mode can be enabled independently for reception and transmission by setting RMCM to 1 and XMCM to a nonzero value in the MCR, respectively.

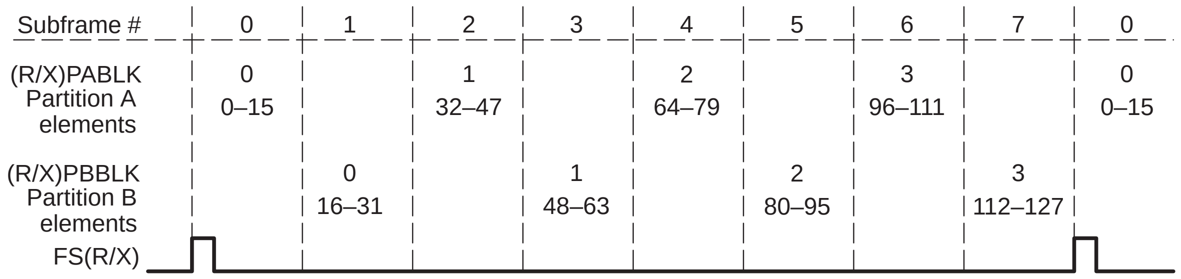
11.6.3 Enabling and Masking of Channels

A total of 32 of the available 128 elements can be enabled at any given time. The 128 elements comprise eight subframes (0 through 7), and each subframe has 16 contiguous elements. Further, even-numbered subframes 0, 2, 4, and 6 belong to partition A, and odd-numbered subframes 1, 3, 5, and 7 belong to partition B.

The number of elements enabled can be updated during the course of a frame to allow any arbitrary group of elements to be enabled. This update is accomplished using an alternating ping-pong scheme for controlling two subframes (one odd-numbered and the other even-numbered) of 16 contiguous elements within a frame at any time. One subframe belongs to partition A and the other to partition B.

Any one 16-element block from partition A and partition B can be selected, yielding a total of 32 elements that can be enabled at one time. The subframes are allocated on 16-element boundaries within the frame, as shown in Figure 11–47. The (R/X)PABLK and (R/X)PBBLK fields in the MCR select the subframes in partition A and B respectively. This enabling is performed independently for transmit and receive.

Figure 11–47. Element Enabling by Subframes in Partitions A and B



Transmit data masking allows an element enabled for transmit to have its DX pin set to the high-impedance state during its transmit period. In systems where symmetric transmit and receive provides software benefits, this feature allows transmit elements to be disabled on a shared serial bus. A similar feature is not needed for receive, because multiple receptions cannot cause serial bus contention.

Note:

DX is masked or driven to the high-impedance state.

- ❑ During inter-packet intervals
- ❑ When an element is masked regardless of whether it is enabled
- ❑ When an element is disabled.

Following are descriptions of how each XMCM value affects operation:

- ❑ XMCM = 00b: The serial port transmits data over the DX pin for the number of elements programmed in XFRLEN1. Thus, DX is driven during transmit.
- ❑ XMCM = 01b: Only those elements that need to be transmitted are selected via XP(A/B)BLK and XCER. Only these selected elements are written to DXR and ultimately transmitted. In other words, if XINTM = 00b, which implies that an XINT is generated for every DXR-to-XSR copy, the number of XINT generated is equal to the number of elements selected via XCER (and *not* equal to XFRLEN1).
- ❑ XMCM = 10b: All elements are enabled, which means all the elements in a data frame (XFRLEN1) are written to DXR and DXR-to-XSR copies occur at their respective times. However, DX is driven only for those elements that are selected via XP(A/B)BLK and XCER and is placed in the high-impedance state otherwise. In this case, if XINTM = 00b, the number of interrupts generated due to every DXR-to-XSR copy would equal the number of elements in that frame (XFRLEN1).

- XMCM = 11b: In this mode, symmetric transmit and receive operation is forced. Symmetric operation occurs when a device transmits and receives on the same set of subframes. These subframes are determined by setting RP(A/B)BLK. The elements in each of these subframes can then be enabled/selected using the RCER register for receive. The transmit side uses the same blocks as the receive side (thus the value of X(P/A)BLK does not matter). In this mode, all elements are disabled, so DR and DX are in the high-impedance state. For receiving, a RBR-to-DRR copy occurs only for those elements that are selected via RP(A/B)BLK and RCER. If RINT were to be generated for every RBR-to-DRR copy, it would occur as many times as the number of elements selected in RCER (and *not* the number of elements programmed in RFRLEN1). For transmitting, the same subframe that is used for reception is used to maintain symmetry, so the value XP(A/B)BLK does not matter. DXR is loaded, and DXR-to-XSR copy occurs for all the elements that are enabled via RP(A/B)BLK . However, DX is driven only for those elements that are selected via XCER. The elements enabled in XCER can be either a subset of or the same as those selected in RCER. Therefore, if XINTM = 00b, transmit interrupts to the CPU would be generated the same number of times as the number of elements selected in RCER (not XCER).

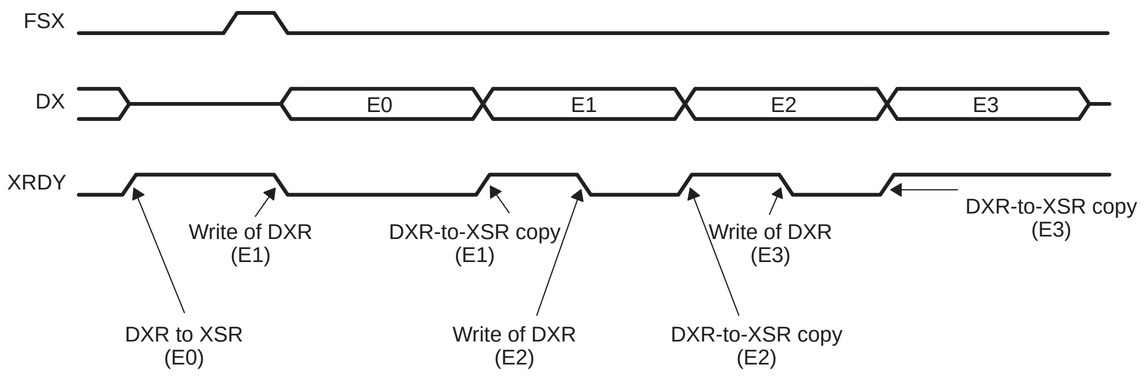
Figure 11–48 shows the activity on the McBSP pins for all of the preceding XMCM values with the following conditions:

- (R/X)PHASE = 0: Single-phase frame for multichannel selection enabled
- FRLEN1 = 011b: 4-element frame
- WDLEN1 = Any valid serial element length

In the following illustrations, the arrows indicating the occurrence of events are only sample indications.

Figure 11–48. XMCM Operation

(a) XMCM = 00b



(b) XMCM = 01b, XPABLK = 00b, XCER = 1010b

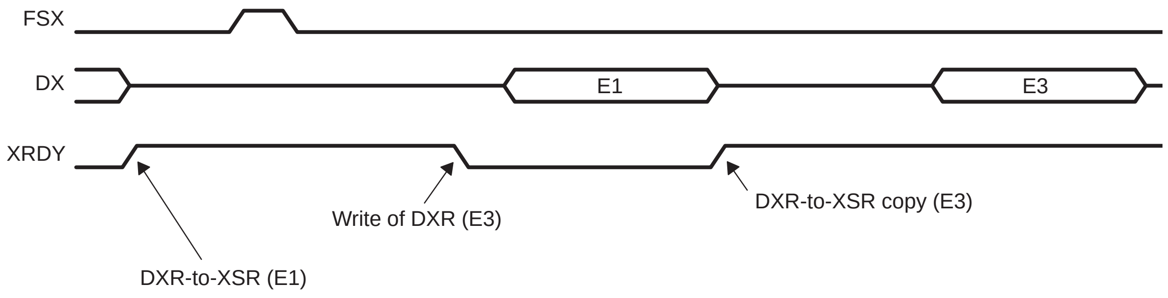
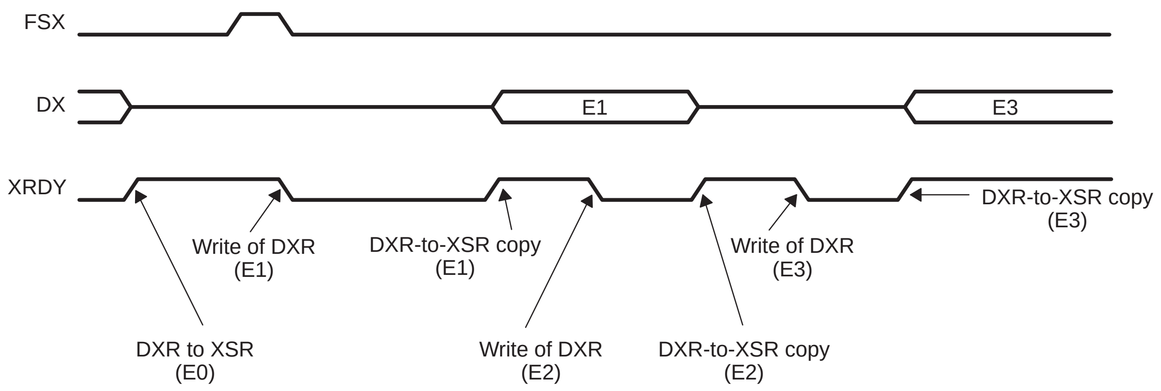
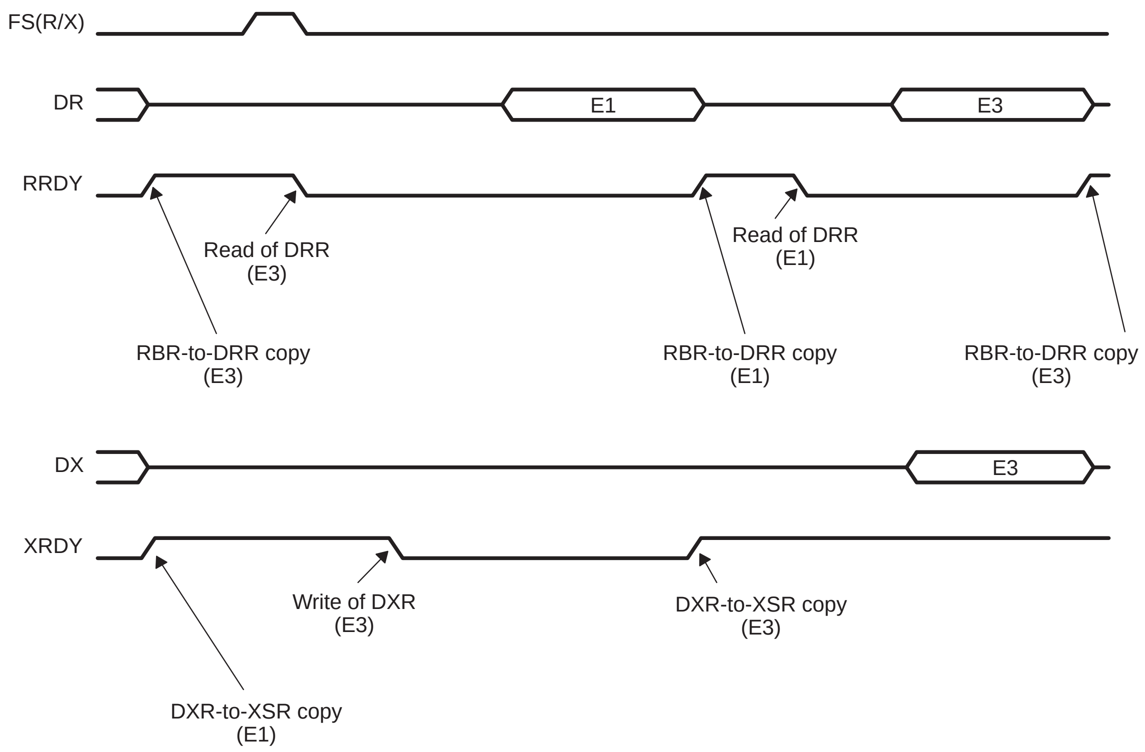


Figure 11–48. XMCM Operation (Continued)

(c) XMCM = 10b, XPABLK = 00b, XCER = 1010b



(d) XMCM = 11b, RPABLK = 00b, XPABLK = X, RCER = 1010b, XCER = 1000b



11.6.3.1 Channel Enable Registers: (R/X)CER

The receive channel enable register (RCER) and transmit channel enable register (XCER) are used to enable any of the 32 elements for receive and transmit, respectively. Of the 32 elements, 16 belong to a subframe in partition A and the other 16 belong to a subframe in partition B. They are shown in Figure 11–49 and Figure 11–50. The (R/X)CEA and (R/X)CEB register fields shown in Table 11–20 enable elements within the 16-channel elements in partitions A and B, respectively. The (R/X)PABLK and (R/X)PBBLK fields in the MCR determine which 16-element subframes are selected.

Figure 11–49. Receive Channel Enable Register (RCER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RCEB 15	RCEB 14	RCEB 13	RCEB 12	RCEB 11	RCEB 10	RCEB 9	RCEB 8	RCEB 7	RCEB 6	RCEB 5	RCEB 4	RCEB 3	RCEB 2	RCEB 1	RCEB 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RCEA 15	RCEA 14	RCEA 13	RCEA 12	RCEA 11	RCEA 10	RCEA 9	RCEA 8	RCEA 7	RCEA 6	RCEA 5	RCEA 4	RCEA 3	RCEA 2	RCEA 1	RCEA 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Figure 11–50. Transmit Channel Enable Register (XCER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XCEB 15	XCEB 14	XCEB 13	XCEB 12	XCEB 11	XCEB 10	XCEB 9	XCEB 8	XCEB 7	XCEB 6	XCEB 5	XCEB 4	XCEB 3	XCEB 2	XCEB 1	XCEB 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XCEA 15	XCEA 14	XCEA 13	XCEA 12	XCEA 11	XCEA 10	XCEA 9	XCEA 8	XCEA 7	XCEA 6	XCEA 5	XCEA 4	XCEA 3	XCEA 2	XCEA 1	XCEA 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Table 11–20. Receive/Transmit Channel Enable Register Field Description

Name	Function
RCEA*n* 0 ≤ n ≤ 15	Receive channel enable RCEA*n* = 0: Disables reception of the *n*th element in an even-numbered subframe in partition A RCEA*n* = 1: Enables reception of the *n*th element in an even-numbered subframe in partition A
XCEA*n* 0 ≤ n ≤ 15	Transmit channel enable XCEA*n* = 0: Disables transmission of the *n*th element in an even-numbered subframe in partition A XCEA*n* = 1: Enables transmission of the *n*th element in an even-numbered subframe in partition A
RCEB*n* 0 ≤ n ≤ 15	Receive channel enable (R/X)CEB*n* = 0: Disables reception of the *n*th element in an odd-numbered subframe in partition B (R/X)CEB*n* = 1: Enables reception of the *n*th element in an odd-numbered subframe in partition B
XCEB*n* 0 ≤ n ≤ 15	Transmit channel enable XCEB*n* = 0: Disables transmission of the *n*th element in anodd-numbered subframe in partition B XCEB*n* = 1: Enables transmission of the *n*th element in an odd-numbered subframe in partition B

11.6.3.2 Changing Element Selection

Using the multichannel selection feature, a static group of 32 elements can be enabled and remains enabled with no CPU intervention until this allocation is modified. An arbitrary number of, group of, or all of the elements within a frame can be accessed by updating the block allocation registers during the course of the frame in response to the end-of-subframe interrupts (see section 11.6.3.3 for information about these interrupts).

Note:

You must be careful not to affect the currently selected subframe when changing the selection.

The currently selected subframe is readable through the RCBLK and XCBLK fields in the MCR for receive and transmit, respectively. The associated channel enable register cannot be modified if it is selected by the appropriate (R/X)P(A/B)BLK register to point toward the current subframe. Similarly, the (R/X)PABLK and (R/X)PBBLK fields in the MCR cannot be modified while pointing to or being changed to point to the currently selected subframe. If the total number of elements is 16 or less, the current partition is always pointed to. In this case, only a reset of the serial port can change the element enabling.

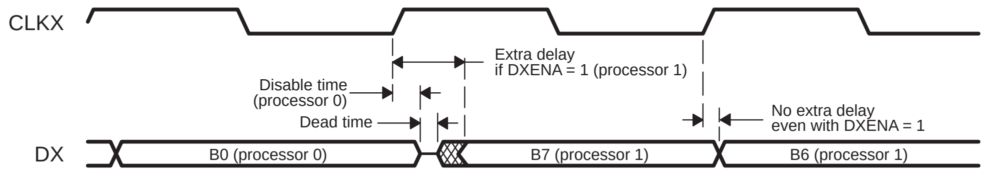
11.6.3.3 End-of-Subframe Interrupt

At the end of every subframe (16 elements or less) boundary during multichannel operation, the receive interrupt (RINT) or transmit interrupt (XINT) to the CPU is generated if RINTM = 01b or XINTM = 01b in the SPCR, respectively. This interrupt indicates that a new partition has been crossed. You can then check the current partition and change the selection of subframes in the A and/or B partitions if they do not point to the current subframe. These interrupts are two CPU-clock high pulses. If RINTM = XINTM = 01b when (R/X)MCM = 0 (nonmultichannel operation), interrupts are not generated.

11.6.4 DX Enabler: DXENA

The DX enabler is only available for the ’C6211/C6711 device. The DXENA field in the serial port control register (SPCR) controls the high impedance enable on the DX pin. When DXENA = 1, the McBSP enables extra delay for the DX pin turn-on time. This feature is useful for McBSP multichannel operations, such as in a time-division multiplexed (TDM) system. The McBSP supports up to 128 channels in a multichannel operation. These channels can be driven by different devices in a TDM data communication line, such as the T1/E1 line. In any multichannel operation where multiple devices transmit over the same DX line, you need to ensure that no two devices transmit data simultaneously, which results in bus contention. Enough dead time should exist between the transmission of the first data bit of the current device and the transmission of the last data bit of the previous device. In other words, the last data bit of the previous device needs to be disabled to a high impedance state before the next device begins transmitting data to the same data line, as shown in Figure 11–51.

Figure 11–51. DX Timing for Multichannel Operation



In the case when two McBSPs are used to transmit data over the same TDM line, bus contention occurs if DXENA = 0. The first McBSP turns off the transmission of the last data bit (changes DX from valid to Hi–Z) after a disable time specified in the datasheet. As shown in Figure 11–51, this disable time is measured from the CLKX active clock edge. The next McBSP turns on its DX pin (changes from Hi–Z to valid) after a delay time. Again, this delay time is measured from the CLKX active clock edge. Bus contention occurs because the dead time between the two devices is not enough. You need to apply alternative software or hardware methods to ensure proper multichannel operation in this case.

If you set DXENA = 1 in the second McBSP, the second McBSP turns on its DX pin after two CPU-clock cycles of extra delay time. This ensures that the previous McBSP on the same DX line is disabled before the second McBSP starts driving out data. The DX enabler controls only the high impedance enable on the DX pin, not the data itself. Data is shifted out to the DX pin at the same time as in the case when DXENA = 0. The only difference is that with DXENA = 1, the DX pin is masked to high impedance for two extra CPU cycles before the data is seen on the TDM data line. Therefore only the first bit of data is delayed.

11.7 SPI Protocol: CLKSTP

A system conforming to this protocol has a master-slave configuration. The SPI™ protocol is a 4-wire interface composed of serial data in (master in slave out or MISO), serial data out (master out slave in or MOSI), shift clock (SCK), and an active (low) slave enable ($\overline{\text{SS}}$) signal. Communication between the master and the slave is determined by the presence or absence of the master clock. Data transfer is initiated by the detection of the master clock and is terminated on absence of the master clock. The slave has to be enabled during this period of transfer. When the McBSP is the master, the slave enable is derived from the master transmit frame sync pulse, FSX. Example block diagrams of the McBSP as a master and as a slave are shown in Figure 11–52 and Figure 11–53, respectively.

Figure 11–52. SPI Configuration: McBSP as the Master

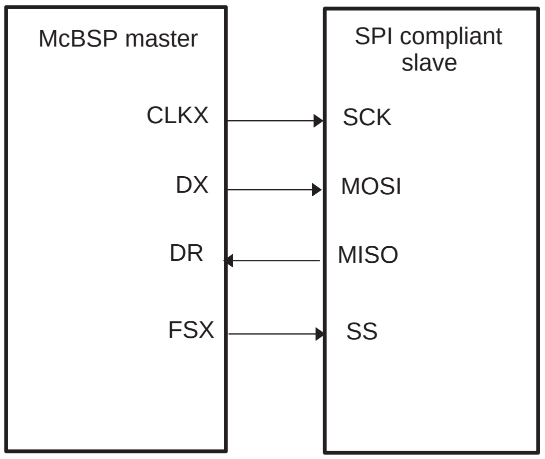
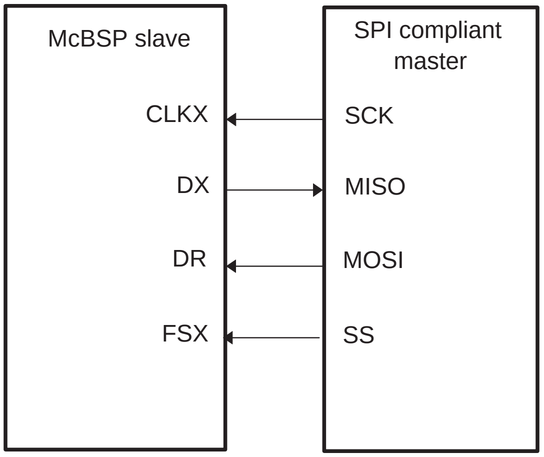


Figure 11–53. SPI Configuration: McBSP as the Slave



The clock stop mode (CLKSTP) of the McBSP provides compatibility with the SPI protocol. The McBSP supports two SPI transfer formats which are specified by the clock stop mode field (CLKSTP) in the SPCR. The clock stop mode field (CLKSTP) in conjunction with the CLKXP bit in the PCR allows serial clocks to be stopped between transfers using one of four possible timing variations, as shown in Table 11–21. Figure 11–54 and Figure 11–55 show the timing diagrams of the two SPI transfer formats and the four timing variations.

Table 11–21. SPI-Mode Clock Stop Scheme

CLKSTP	CLKXP	Clock Scheme
0X	X	Clock stop mode disabled. Clock enabled for non-SPI mode.
10	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.
11	0	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.
10	1	High inactive state without delay. The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of CLKR.
11	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.

Figure 11–54. SPI Transfer with CLKSTP = 10b

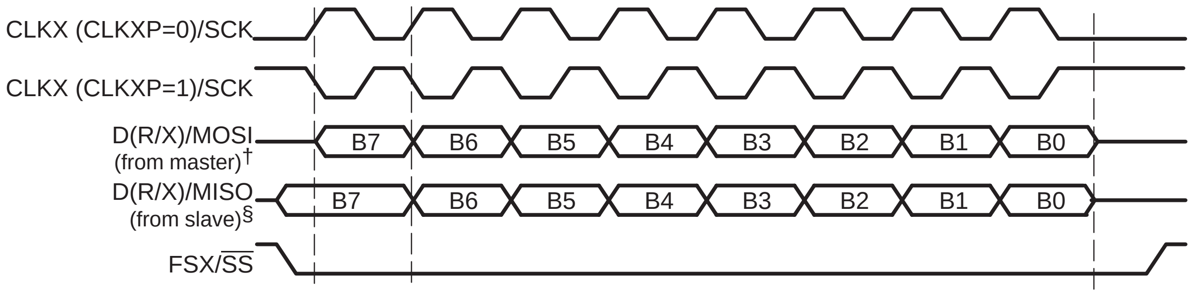
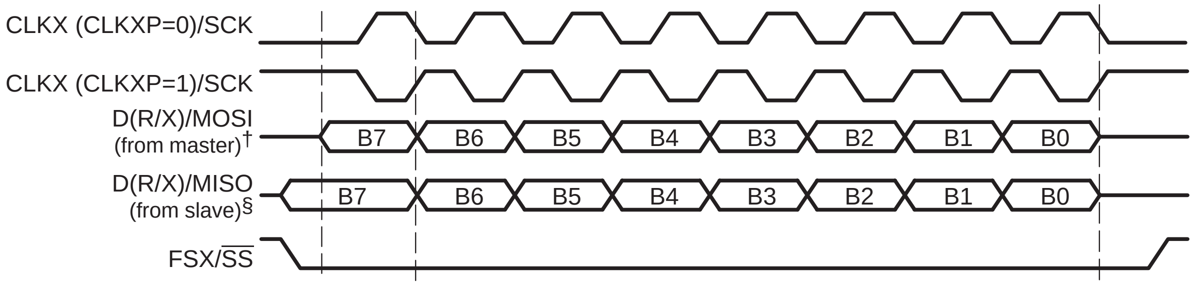


Figure 11–55. SPI Transfer with CLKSTP = 11b



† If the McBSP is the SPI master (CLKXM = 1), MOSI=DX. If the McBSP is the SPI slave (CLKXM = 0), MOSI = DR.
§ If the McBSP is the SPI master (CLKXM = 1), MISO=DR. If the McBSP is the SPI slave (CLKXM = 0), MISO = DX.

The CLKSTP and CLKXP fields of the serial port control register (SPCR) select the appropriate clock scheme for a particular SPI interface, as shown in Table 11–21. The CLKSTP and CLKXP fields in the SPCR determine the following conditions:

- ❑ Whether clock stop mode is enabled or not
- ❑ In clock stop mode, whether the clock is high or low when stopped
- ❑ In clock stop mode, whether the first clock edge occurs at the start of the first data bit or at the middle of the first data bit

The CLKXP bit selects the edge on which data is transmitted (driven) and received (sampled), as shown in Table 11–21.

Figure 11–54 is the timing diagram when CLKSTP = 10b. In this SPI transfer format, the transition of the first clock edge (CLKX) marks the beginning of data transfer, provided the slave enable (FSX/$\overline{\text{SS}}$) is already asserted. Data transfer is synchronized to the first clock edge. Figure 11–55 is the timing diagram when CLKSTP = 11b. Data transfer begins before the transition of the serial clock. Therefore, the transition of the slave enable signal FSX/$\overline{\text{SS}}$ from high to low, instead of the transition of the serial clock, marks the beginning of transfer in this SPI transfer format. The McBSP clock stop mode requires single-phase frames ((R/X)PHASE = 0) and one element per frame ((R/X)FRLEN = 0).

When the McBSP is configured to operate in SPI mode, both the transmitter and the receiver operate together as a master or a slave. The McBSP is a master when it generates clocks. When the McBSP is the SPI master, CLKX drives both its own internal receive clock CLKR and the serial clock SCK of the SPI slave. In conjunction with CLKSTP enabled, CLKXM = 1 (in PCR) indicates that the McBSP is a master, and CLKXM = 0 indicates that the McBSP is an SPI slave. The slave enable signal (FSX/$\overline{\text{SS}}$) enables the serial data input and output driver on the slave device (the device not providing the output clock).

11.7.1 McBSP Operation as the SPI Master

When the McBSP is the SPI master, it generates the master clock CLKX and the slave enable FSX. Therefore, CLKX should be configured as an output (CLKXM = 1) and FSX should be configured as an output that can be connected to the slave enable ($\overline{\text{SS}}$) input on the slave device (FSXM = 1). The DXR-to-XSR transfer of each element generates the slave enable FSX. Therefore, the FSGM field of the sample rate generator register (SRGR) must be set to zero. The SPI protocol specifies that the slave needs to be enabled before the transfer of the data. In other words, FSX needs to be asserted (low) before the McBSP starts shifting out data on the DX pin. Refer to the MOSI and FSX waveforms in Figure 11–54 and Figure 11–55. Therefore, XDATDLY must be programmed to 1. When the McBSP is the SPI master, an XDATDLY value of 0 or 2 causes undefined operation.

As the SPI master, the McBSP generates CLKX and FSX through the internal sample rate generator. As discussed in section 11.5.2.1, the CLKSM bit in the SRGR should be set to specify either the CPU clock or the external clock input (CLKS) as the clock source to the internal sample rate generator. The CLKGDV (clock divide ratio) in SRGR should be programmed to generate CLKX at the required SPI data rate. The McBSP generates a continuous clock (CLKX) internally and gates the clock off (stops the clock) to the external interface when transfers are finished. The McBSP’s receive clock is provided from the internal continuously running clock, so the receiver and transmitter both work internally as if clocks do not stop. Selection of the clock stop modes overrides the frame generator bit fields (FPER and FWID) of the the sample rate generator register (SRGR).

11.7.2 McBSP Operation as the SPI Slave

When the McBSP is an SPI slave device, the master clock CLKX and slave enable FSX are generated by an external SPI master, as shown in Figure 11–53. Thus, the CLKX and FSX pins are configured as inputs by setting the CLKXM and FSXM fields to zero in the PCR. In SPI mode, the FSX and CLKX inputs are also utilized as the internal FSR and CLKR signals for data reception. Data transfer is synchronized to the master clock CLKX and the internal serial port logic performs transfers using only the exact number of input clock pulses CLKX per data bit. The external master needs to assert FSX (low) before the transfer of data begins. FSX is used in its asynchronous form and it controls the McBSP's initial drive of data to the DX pin.

When the McBSP is a slave, (R/X)DATDLY in the receive/transmit control register ((R/X)CR) should be set to zero. XDATDLY = 0 ensures that the first data to be transmitted is available on the DX pin. The MISO waveform in Figure 11–54 and Figure 11–55 shows how the McBSP transmits data as an SPI slave. Setting RDATDLY = 0 ensures that the McBSP is ready to receive data from the SPI master as soon as it detects the serial clock CLKX. Depending on the clock stop mode used, data is received at various clock edges according to Table 11–21.

Although the CLKX signal is generated externally by the master, the internal sample rate generator of the McBSP must be enabled for proper SPI slave mode operation. The internal sample rate clock is then used to synchronize the input clock (CLKX) and frame sync (FSX) from the master to the CPU clock. Accordingly the CLKSM field of the sample rate generator (SRGR) should be left at the default value (CLKSM = 1) to specify the CPU clock as the clock source of the sample rate generator. Furthermore, the CLKGDV in the SRGR must be set to a value such that the rate of the internal clock CLKG is at least eight times that of the SPI data rate. This rate is achieved by programming the sample rate generator to its maximum speed (CLKGDV = 1) for all SPI transfer rates.

11.7.3 McBSP Initialization for SPI Mode

The operation of the serial port during device reset, transmitter reset, and receiver reset is described in section 11.3.1. For McBSP operation as a master or a slave in SPI mode, you must follow these steps for proper initialization:

1) Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 0 in SPCR.

2) Program the necessary McBSP configuration registers (and not the data registers) listed in Table 11–2 as required when the serial port is in the reset state ($\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 0) *except* for CLKSTP, which should be disabled. Program CLKSTP to 0Xb if CLKSTP is not disabled.

3) Set $\overline{\text{GRST}}$ = 1 in SPCR to get the sample rate generator out of reset.

4) Wait two bit clocks for the McBSP to reinitialize.

5) Write the desired value into the CLKSTP field in the SPCR. Table 11–21 shows the various CLKSTP modes.

6) Depending on whether the CPU or DMA services the McBSP, either (a) or (b) should be followed.

   a) This step should be performed if the CPU is used to service the McBSP. Set /XRST = /RRST = 1 to enable the serial port. Note that the value written to the SPCR at this time should have only the reset bits changed to 1 and the remaining bit–fields should have the same values as in Step 2 and 4 above.

   b) If DMA is used to perform data transfers, the DMA should be initialized first with the appropriate read/write syncs and the start bit set to run. The DMA waits for the synchronization events to occur. Now, pull the McBSP out of reset by setting $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 1.

7) Wait two bit clocks for the receiver and transmitter to become active.

11.8 McBSP Pins as General-Purpose I/O

Two conditions allow the serial port pins (CLKX, FSX, DX, CLKR, FSR, DR, and CLKS) to be used as general-purpose I/O rather than serial port pins:

- ❑ The related portion (transmitter or receiver) of the serial port is in reset: $\overline{\text{(R/X)RST}}$ = 0 in the SPCR
- ❑ General-purpose I/O is enabled for the related portion of the serial port: (R/X)IOEN = 1 in the PCR

Figure 11–3 shows the PCR bits that configure each of the McBSP pins as general-purpose inputs or outputs. Table 11–22 shows how this is achieved. In the case of FS(R/X), FS(R/X)M = 0 configures the pin as an input and FS(R/X)M = 1 configures that pin as an output. When configured as an output, the value driven on FS(R/X) is the value stored in FS(R/X)P. If configured as an input, the FS(R/X)P becomes a read-only bit that reflects the status of that signal. CLK(R/X)M and CLK(R/X)P work similarly for CLK(R/X). When the transmitter is selected as general-purpose I/O, the value of the DX_STAT bit in the PCR is driven onto DX. DR is always an input, and its value is held in the DR_STAT bit in the PCR. To configure CLKS as a general-purpose input, both the transmitter and receiver have to be in the reset state and (R/X)IOEN has to be set to 1, because (R/X)IOEN is always an input to the McBSP and it affects both transmit and receive operations.

Table 11–22. Configuration of Pins as General Purpose I/O

Pin	General-Purpose I/O Enabled When...	Selected as Output When...	Output Value Driven From	Selected as Input When ...	Input Value Readable on
CLKX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	CLKXM = 1	CLKXP	CLKXM = 0	CLKXP
FSX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	FSXM = 1	FSXP	FSXM = 0	FSXP
DX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	Always	DX_STAT	Never	N/A
CLKR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	CLKRM = 1	CLKRP	CLKRM = 0	CLKRP
FSR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	FSRM = 1	FSRP	FSRM = 0	FSRP
DR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	Never	N/A	Always	DR_STAT
CLKS	$\overline{\text{RRST}}$ = $\overline{\text{XRST}}$ = 0 RIOEN = XIOEN = 1	Never	N/A	Always	CLKS_STAT

Chapter 12

Timers

This chapter describes the 32-bit timer functionality, registers, and signals.

Topic **Page**

12.1 Overview

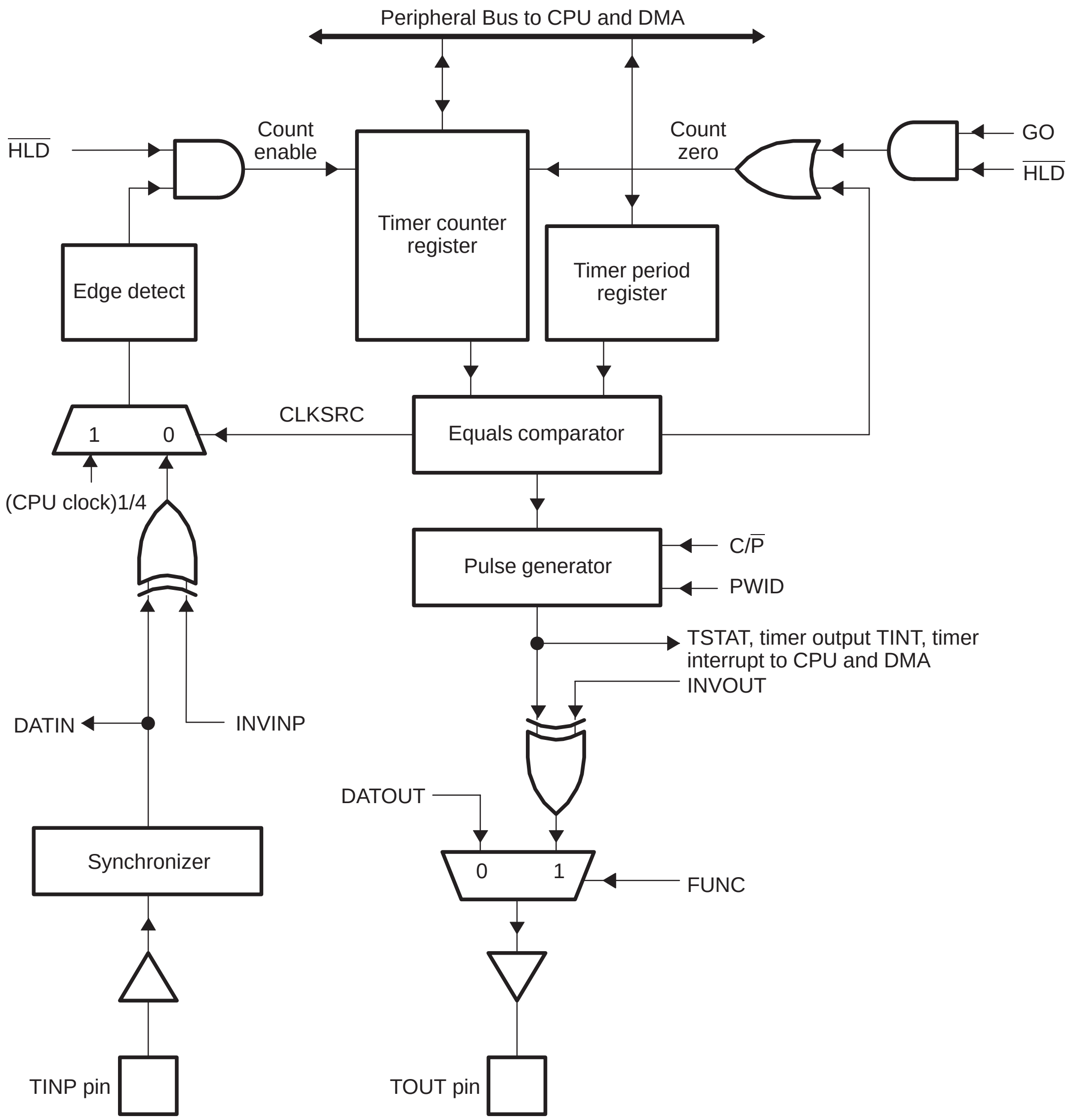
The device has two 32-bit general-purpose timers that you can use to:

- ❑ Time events
- ❑ Count events
- ❑ Generate pulses
- ❑ Interrupt the CPU
- ❑ Send synchronization events to the DMA

The timers have two signaling modes and can be clocked by an internal or an external source. The timers have an input pin and an output pin. The input and output pins, (TINP and TOUT) can function as timer clock input and clock output. They can also be configured for general-purpose input and output, respectively.

With an internal clock, for example, the timer can signal an external A/D converter to start a conversion, or it can trigger the DMA controller to begin a data transfer. With an external clock, the timer can count external events and interrupt the CPU after a specified number of events. Figure 12–1 shows a block diagram of the timers.

Figure 12–1. Timer Block Diagram



12.2 Timer Registers

Table 12–1 describes the three registers that configure timer operation.

Table 12–1. Timer Registers

Hex Byte Address				
Timer 0	**Timer 1**	**Name**	**Description**	**Section**
01940000	01980000	Timer Control	Determines the operating mode of the timer, monitors the timer status, and controls the function of the TOUT pin.	12.2.1
01940004	01980004	Timer Period	Contains the number of timer input clock cycles to count. This number controls the TSTAT signal frequency.	12.2.2
01940008	01980008	Timer Counter	Current value of the incrementing counter	12.2.3

12.2.1 Timer Control Register

Figure 12–2 shows the timer control register. Table 12–2 describes the fields in this register.

Figure 12–2. Timer Control Register

31–12	11	10	9	8
Rsvd	TSTAT	INVINP	CLKSRC	C/$\overline{\text{P}}$
R, +0	R, +0	RW, +0	RW, +0	RW, +0

7	6	5	4	3	2	1	0
$\overline{\text{HLD}}$	GO	Rsvd	PWID	DATIN	DATOUT	INVOUT	FUNC
RW, +0	RW, +0	R, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0

Table 12–2. Timer Control Register Field Descriptions

Bitfield	Description	Section
FUNC	Function of TOUT pin FUNC = 0: TOUT is a general-purpose output pin. FUNC = 1: TOUT is a timer output pin.	12.6
DATOUT	Data output When FUNC = 0: The DATOUT is driven on TOUT. When FUNC = 1: The TSTAT is driven on TOUT after inversion by INVOUT.	12.6
DATIN	Data in: Value on TINP pin	12.5
GO	GO bit. Resets and starts the timer counter. GO = 0: No effect on the timers. GO = 1: If $\overline{\text{HLD}}$ = 1, the counter register is zeroed and begins counting on the next clock.	12.3

Table 12–2. Timer Control Register Field Descriptions (Continued)

Bitfield	Description	Section
$\overline{\text{HLD}}$	Hold. Counter may be read or written regardless of $\overline{\text{HLD}}$ value. $\overline{\text{HLD}}$ = 0: Counter is disabled and held in the current state. $\overline{\text{HLD}}$ = 1: Counter is allowed to count.	12.3
C/$\overline{\text{P}}$	Clock/pulse mode C/$\overline{\text{P}}$ = 0: Pulse mode. TSTAT is active one CPU clock after the timer reaches the timer period. PWID determines when it goes inactive. C/$\overline{\text{P}}$ = 1: Clock mode. TSTAT has a 50% duty cycle with each high and low period one countdown period wide.	12.6
PWID	Pulse width. Only used in pulse mode (C/$\overline{\text{P}}$ = 0). PWID = 0: TSTAT goes inactive one timer input clock cycle after the timer counter value equals the timer period value. PWID = 1: TSTAT goes inactive two timer input clock cycles after the timer counter value equals the timer period value.	12.6
CLKSRC	Timer input clock source CLKSRC = 0: External clock source drives the TINP pin. CLKSRC = 1: CPU clock/4.	12.5

INVINP	TINP inverter control. Only affects operation if CLKSRC = 0.
	INVINP = 0: Uninverted TINP drives timer.
	INVINP = 1: Inverted TINP drives timer.
		12.5

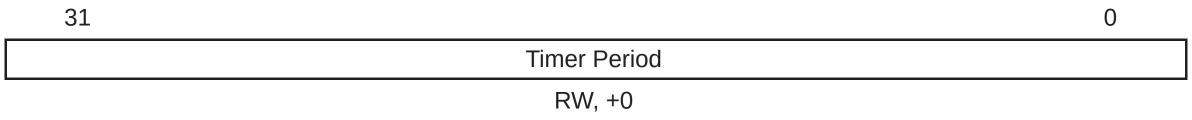
TSTAT	Timer status. Value of timer output.	12.6

INVOUT	TOUT inverter control. Used only if FUNC = 1.
	INVOUT = 0: Uninverted TSTAT drives TOUT.
	INVOUT = 1: Inverted TSTAT drives TOUT.	

12.2.2 Timer Period Register

The timer period register (Figure 12–3) contains the number of timer input clock cycles to count. This number controls the frequency of TSTAT.

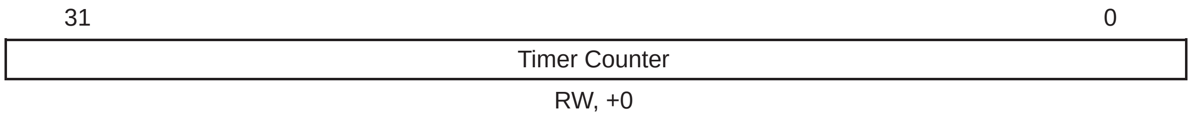
Figure 12–3. Timer Period Register



12.2.3 Timer Counter Register

The timer counter register (Figure 12–4) increments when it is enabled to count. It resets to 0 on the next CPU clock after the value in the timer period register is reached.

Figure 12–4. Timer Counter Register



12.3 Resetting the Timers and Enabling Counting: GO and $\overline{\text{HLD}}$

Table 12–3 shows how the GO and $\overline{\text{HLD}}$ enable basic features of timer operation.

Table 12–3. Timer GO and $\overline{\text{HLD}}$ Field Operation

Operation	**GO**	**$\overline{\text{HLD}}$**	**Description**
Holding the timer	0	0	Counting is disabled.
Restarting the timer after hold	0	1	Timer continues from the value before hold. The timer counter is *not* reset.
Reserved	1	0	Undefined
Starting the timer	1	1	Timer counter resets to 0 and starts counting whenever enabled. Once set, GO self-clears.

Configuring a timer requires three basic steps:

1) If the timer is not currently in the hold state, place the timer in hold ($\overline{\text{HLD}}$ = 0). Note that after device reset, the timer is already in the hold state.

2) Write the desired value to the timer period register.

3) Start the timer by setting the GO and $\overline{\text{HLD}}$ bits of the timer control register to 1 and simultaneously writing the desired values to the timer control register.

12.4 Timer Counting

The timer counter runs at the CPU clock rate. However, counting is enabled on the low-to-high transition of the timer count enable source. This transition is detected by the edge detect circuit shown in Figure 12–1. Each time an active transition is detected, one CPU-clock-wide clock enable pulse is generated. To the user, this makes the counter appear as if it were getting clocked by the count enable source. Thus, this count enable source is referred to as the timer input clock source.

Once the timer reaches a value equal to the value in the timer period register, the timer is reset to 0 on the next CPU clock. Thus, the counter counts from 0 to N. Consider the case where the period is 2 and the CPU clock/4 is selected as the timer clock source (CLKSRC = 1). Once started, the timer counts the following sequence: 0, 0, 0, 0, 1, 1, 1, 1, 2, 0, 0, 0, 1, 1, 1, 1, 2, 0, 0, 0…. Note that although the counter counts from 0 to 2, the period is 8 (2*4) CPU clock cycles rather than 12 (3*4) CPU clock cycles. Thus, the countdown period is the value of TIMER PERIOD, not TIMER PERIOD+1.

12.5 Timer Clock Source Selection: CLKSRC

Low-to-high transitions (or high-to-low transitions if INVINP = 1) of the timer input clock allow the timer counter to increment. Two sources are available to drive the timer input clock:

- The input value on the TINP pin, selected by CLKSRC = 0. This signal is synchronized to prevent any metastability caused by asynchronous external inputs. The value present on the TINP pin is reflected by DATIN.
- The CPU clock/4, selected by CKSRC = 1.

12.6 Timer Pulse Generation

The two basic pulse generation modes are pulse mode and clock mode, as shown in Figure 12–5 and Figure 12–6, respectively. You can select the mode with the C/$\overline{\text{P}}$ bit of the timer control register. Note that in pulse mode, PWID in the timer control register can set the pulse width to either one or two input clock periods. The purpose of this feature is to provide minimum pulse widths in the case in which TSTAT drives the TOUT output. TSTAT drives this pin when TOUT is used as a timer pin (FUNC = 1), and may be inverted by setting INVOUT = 1. The value actually driven out to the TOUT pin is reflected by DATOUT. Table 12–4 gives equations for various TSTAT timing parameters in pulse and clock modes.

Figure 12–5. Timer Operation in Pulse Mode (C/$\overline{P}$ = 0)

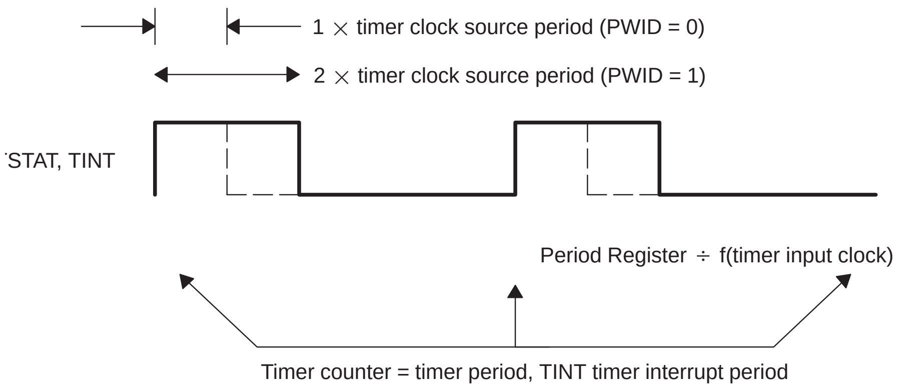

1 × timer clock source period (PWID = 0)
2 × timer clock source period (PWID = 1)
TSTAT, TINT
Period Register ÷ f(timer input clock)
Timer counter = timer period, TINT timer interrupt period

Figure 12–6. Timer Operation in Clock Mode (C/$\overline{P}$ = 1)

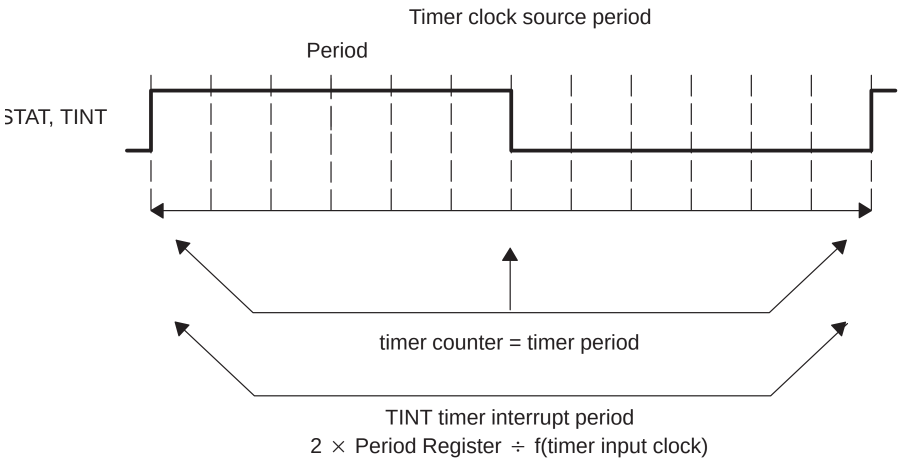

Timer clock source period
Period
TSTAT, TINT
timer counter = timer period
TINT timer interrupt period
2 × Period Register ÷ f(timer input clock)

Table 12–4. TSTAT Parameters in Pulse and Clock Modes

Mode	Frequency	Period	Width High	Width Low
Pulse	$\frac{f\ (\text{clock source})}{\text{timer period register}}$	$\frac{\text{timer period register}}{f\ (\text{clock source})}$	$\frac{(PWID + 1)}{f\ (\text{clock source})}$	$\frac{\text{timer period register} - (PWID + 1)}{f\ (\text{clock source})}$
Clock	$\frac{f\ (\text{clock source})}{2 * \text{timer period register}}$	$\frac{2 * \text{timer period register}}{f\ (\text{clock source})}$	$\frac{\text{timer period register}}{f\ (\text{clock source})}$	$\frac{\text{timer period register}}{f\ (\text{clock source})}$

12.7 Boundary Conditions in the Control Registers

The following boundary conditions affect timer operation:

1) Timer period and counter register value is 0: After device reset and before the timer starts counting, TSTAT is held at 0. After the timer starts running by setting $\overline{\text{HLD}}$ = 1 and GO = 1, while the period and counter registers are zero, the operation of the timer depends on the C/$\overline{\text{P}}$ mode selected. In pulse mode, the TSTAT = 1 regardless of whether or not the timer is held. In clock mode, when the timer is held ($\overline{\text{HLD}}$ = 0), TSTAT keeps it's previous value and when $\overline{\text{HLD}}$ = 1, TSTAT toggles with a frequency of 1/2 of the CPU clock frequency.

2) Counter overflow: When the counter register is set to a value greater than the value of the period register, the counter reaches its maximum value (FFFF FFFFh), rolls over to 0, and continues.

3) Writing to registers of an active timer: Writes from the peripheral bus override register updates to the counter register and new status updates to the control register.

4) Small timer period values in pulse mode: Note that small periods in pulse mode can cause TSTAT to remain high. This condition occurs when TIMER PERIOD ≤ PWID + 1.

12.8 Timer Interrupts

The TSTAT signal directly drives the CPU interrupt as well as a DMA synchronization event. The frequency of the interrupt is the same as the frequency of TSTAT.

12.9 Emulation Operation

During debug using the emulator, the CPU may be halted on an execute packet boundary for single stepping, benchmarking, profiling, or other debug uses. During an emulation halt, the timer halts when the CPU clock/4 is selected as the clock source (CLKSRC = 1). Here, the counter is only enabled to count during those cycles when the CPU is not stalled due to the emulation halt. Thus, counting will be re-enabled during single-step operation. If CLKSRC = 0, the timer continues counting as programmed.

Chapter 13

Interrupt Selector and External Interrupts

This chapter describes the interrupt selector and registers available.

Topic **Page**

13.1 Available Interrupt Sources

The ‘C6000 peripheral set has up to 32 interrupt sources. The CPU however has 12 interrupts available for use. The interrupt selector allows you to choose and prioritize which 12 of the 32 your system needs to use. The interrupt selector also allows you to effectively change the polarity of external interrupt inputs.

Table 13–1 lists the available interrupts. Note that this table is similar to the DMA synchronization events in Chapter 4, *DMA Controller* except for two differences. One difference is that the McBSP generates separate interrupts and DMA synchronization events. The second difference is that DSPINT has been moved from 10000b to 00000b.

Table 13–1. TMS320C6201/C6202/C6701 Available Interrupts

Interrupt Selection Number	Interrupt Acronym	Interrupt Description
00000b	DSPINT	Host processor to DSP interrupt
00001b	TINT0	Timer 0 interrupt
00010b	TINT1	Timer 1 interrupt
00011b	SD_INT	EMIF SDRAM timer interrupt
00100b	EXT_INT4	External interrupt pin 4
00101b	EXT_INT5	External interrupt pin 5
00110b	EXT_INT6	External interrupt pin 6
00111b	EXT_INT7	External interrupt pin 7
01000b	DMA_INT0	DMA channel 0 interrupt
01001b	DMA_INT1	DMA channel 1 interrupt
01010b	DMA_INT2	DMA channel 2 interrupt
01011b	DMA_INT3	DMA channel 3 interrupt
01100b	XINT0	McBSP 0 transmit interrupt
01101b	RINT0	McBSP 0 receive interrupt
01110b	XINT1	McBSP 1 transmit interrupt
01111b	RINT1	McBSP 1 receive interrupt
10000b		Reserved
10001b	XINT2	McBSP 2 transmit interrupt[†]
10010b	RINT2	McBSP 2 receive interrupt[†]
other		Reserved

[†] Only available on the ’C6202

For more information on interrupts, including the interrupt vector table, see the *TMS320C6000 CPU and Instruction Set Reference Guide*.

The EDMA controller in the ‘C6211/C6711 device has 16 channels; each triggered by a specific event. As in the other ’C6000 platform of devices, the ’C6211/C6711 CPU has 12 interrupts available for use. Although there is provision for 32 interrupt sources, the ‘C6211/C6711 provides for 13 interrupt sources. As shown in Table 13–2, the four DMA interrupts in existing ’C6201/’C6701/’C6202 devices are replaced with a single EDMA interrupt (EDMA_INT) which is described in *EDMA Controller* chapter 6, section 6.13 *EDMA Interrupt Generation*.

Table 13–2. TMS320C6211/C6711 Available Interrupts

Interrupt Selection Number	Interrupt Acronym	Interrupt Description
00000b	DSPINT	Host port host to DSP interrupt
00001b	TINT0	Timer 0 interrupt
00010b	TINT1	Timer 1 interrupt
00011b	SD_INT	EMIF SDRAM timer interrupt
00100b	EXT_INT4	External interrupt 4
00101b	EXT_INT5	External interrupt 5
00110b	EXT_INT6	External interrupt 6
00111b	EXT_INT7	External interrupt 7
01000b	EDMA_INT	EDMA channel (0 through 15) interrupt
01001b	Reserved	Not used
01010b	Reserved	Not used
01011b	Reserved	Not used
01100b	XINT0	McBSP 0 transmit interrupt
01101b	RINT0	McBSP 0 receive interrupt
01110b	XINT1	McBSP 1 transmit interrupt
01111b	RINT1	McBSP 1 receive interrupt
other		Reserved

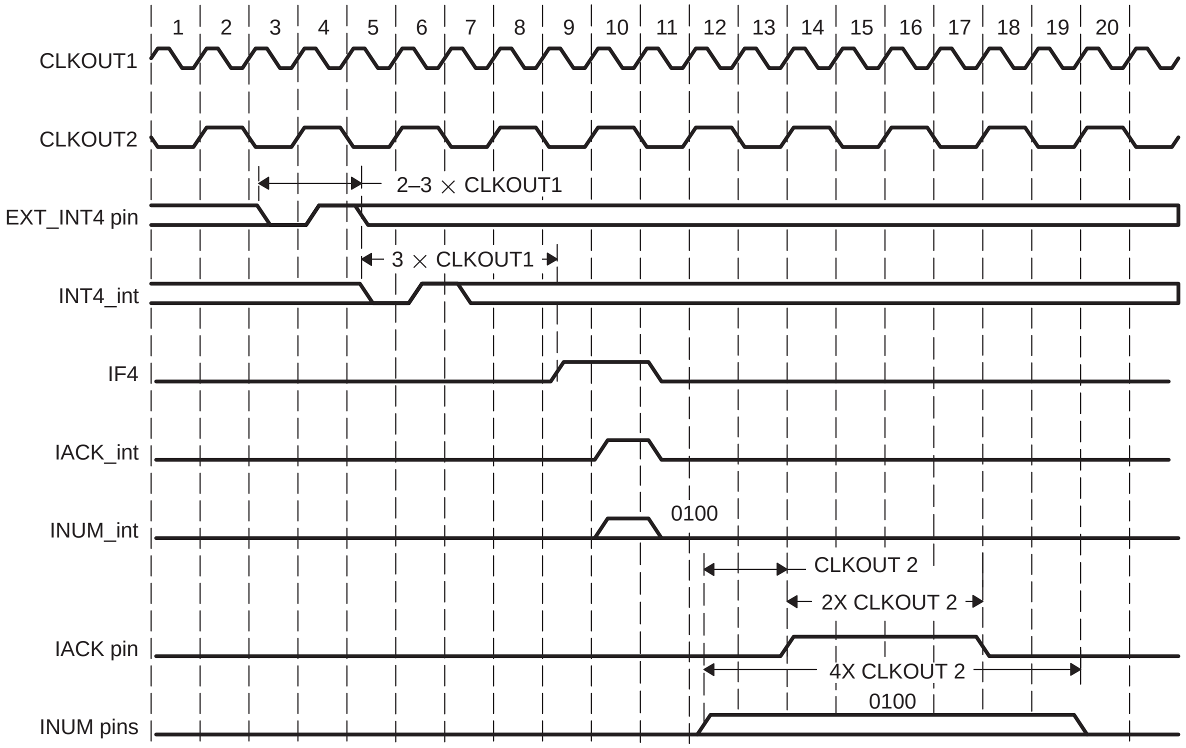
13.2 External Interrupt Signal Timing

EXT_INT4–7, and NMI are dedicated external interrupt sources. In addition, the FSR and FSX can be programmed to directly drive the RINT and XINT signals. Because these signals are asynchronous, they are passed through two registers before being sent to either the DMA or CPU. Figure 13–1 shows the timing of external interrupt signals using INT4 as an example. This diagram is similar to the one in the CPU Reference Guide. However, this diagram also shows the delays for the external interrupt through the two synchronization flip-flops. Note, that this delay is two CPU clock (CLKOUT1) cycles. However, if the EXT_INT4 input transitions during the setup and hold time with respect to the CLKOUT1 rising edge, this delay could be as long as 3 CLKOUT1 cycles. Once synchronized, an additional 3 CLKOUT1 cycle delay occurs before the related interrupt flag (IF4) is set.

The earliest cycle that the interrupt can be scheduled is one CLKOUT1 cycle after IF4 is set. This is indicated by the active internal interrupt acknowledge (IACK) signal as shown in Figure 13–1. The interrupt can be postponed or inhibited if not properly enabled as described in other chapters of the CPU Reference Guide. In that case, IACK will be also be postponed. Along with IACK, the CPU sets the INUM signal to indicate which interrupt was taken. Externally, the IACK pin pulse is extended to two CLKOUT2 cycles wide and synchronized to CLKOUT2. Also, the INUM pin signal frames this external IACK with one CLKOUT2 cycle of setup and hold, for a width of 4 CLKOUT2 cycles. Note that even though INUM and IACK in the diagram are not valid on a CLKOUT2 rising edge, the internal circuitry still catches the transition and produces the desired waveforms on the IACK and INUM pins.

The NMI can interrupt a maskable interrupt’s fetch packet (ISFP) just before the interrupt reaches E1. In this case an IACK and INUM for the NMI is not seen because the IACK and INUM corresponding to the maskable interrupt is on the pins.

Figure 13–1. Timing of External Interrupt Related Signals



13.3 Interrupt Selector Registers

Table 13–3 shows the interrupt selector registers. The interrupt multiplexer registers determine the mapping between the interrupt sources in Table 13–1 and the CPU interrupts 4 through 15 (INT4–INT15). The external interrupt polarity register sets the polarity of external interrupts.

Table 13–3. Interrupt Selector Registers

Byte Address	Name	Description	Section
019C0000h	Interrupt multiplexer high	Selects which interrupts drive CPU interrupts 10–15 (INT10–15)	13.3.2
019C0004h	Interrupt multiplexer low	Selects which interrupts drive CPU interrupts 4–9 (INT4–INT9)	13.3.2
019C0008h	External interrupt polarity	Sets the polarity of the external interrupts (EXT_INT4–EXT_INT7)	13.3.1

13.3.1 External Interrupt Polarity Register

The external interrupt polarity register allows you to change the polarity of the four external interrupts (EXT_INT4 to EXT_INT7). When XIP is its default value of 0, a low-to-high transition on an interrupt source is recognized as an interrupt. By setting the related XIP bit in this register to 1, you can invert the external interrupt source and effectively have the CPU detect high-to-low transitions of the external interrupt. Changing an XIP bit's value creates transitions on the related CPU interrupt (INT4–INT15) that the external interrupt, EXT_INT, is selected to drive. For example, if XIP4 is changed from 0 to 1 and EXT_INT4 is low, or if XIP4 is changed from 1 to 0 and EXT_INT4 is high, the CPU interrupt that is mapped to EXT_INT4 becomes set. The external interrupt polarity register only affects interrupts to the CPU, and has no effect on DMA events.

Figure 13–2. External Interrupt Polarity Register

31–4	3	2	1	0
Rsvd	XIP7	XIP6	XIP5	XIP4
R, +0	R, +0	RW, +0	RW, +0	RW, +0

13.3.2 Interrupt Multiplexer Register

The INTSEL fields in the interrupt multiplexer registers, shown in Figure 13–3 and Figure 13–4 allow mapping the interrupt sources in to particular interrupts. The INTSEL4–INTSEL15 correspond to CPU interrupts INT4–INT15. By setting the INTSEL fields to the value of the desired interrupt selection number in Table 13–1 or Table 13–2, you may map any interrupt source to any CPU interrupt. Table 13–4 shows the default mapping of interrupt sources to CPU interrupts.

Figure 13–3. Interrupt Multiplexer Low Register Diagram

31	30 26	25 21	20 16
Reserved	INTSEL9	INTSEL8	INTSEL7
R, +0	RW, +01001	RW, +01000	RW, +00111

15	14 10	9 5	4 0
Reserved	INTSEL6	INTSEL5	INTSEL4
R, +0	RW, +00110	RW, +00101	RW, +00100

Figure 13–4. Interrupt Multiplexer High Register Diagram

31	30 26	25 21	20 16
Reserved	INTSEL15	INTSEL14	INTSEL13
R, +0	RW, +00010	RW, +00001	RW, +00000

15	14 10	9 5	4 0
Reserved	INTSEL12	INTSEL11	INTSEL10
R, +0	RW, +01011	RW, +01010	RW, +00011

Table 13–4. Default Interrupt Mapping

CPU Interrupt	Related INTSEL field	INTSEL Reset Value	Interrupt Acronym	Interrupt Description
INT4	INTSEL4	00100b	EXT_INT4	External interrupt pin 4
INT5	INTSEL5	00101b	EXT_INT5	External interrupt pin 5
INT6	INTSEL6	00110b	EXT_INT6	External interrupt pin 6
INT7	INTSEL7	00111b	EXT_INT7	External interrupt pin 7
INT8	INTSEL8	01000b	DMA_INT0/ EDMA_INT	DMA Channel 0 Interrupt/ EDMA interrupt
INT9	INTSEL9	01001b	DMA_INT1	DMA Channel 1 interrupt†
INT10	INTSEL10	00011b	SD_INT	EMIF SDRAM timer interrupt
INT11	INTSEL11	01010b	DMA_INT2	DMA Channel 2 interrupt†
INT12	INTSEL12	01011b	DMA_INT3	DMA Channel 3 interrupt†
INT13	INTSEL13	00000b	DSPINT	Host port to DSP interrupt
INT14	INTSEL14	00001b	TINT0	Timer 0 interrupt
INT15	INTSEL15	00010b	TINT1	Timer 1 interrupt

† Reserved on ’C6211/C6711

13.4 Configuring the Interrupt Selector

The interrupt selector registers are meant to be configured once after reset during initialization and before enabling interrupts.

Note:

Once the registers have been set, the interrupt flag register should be cleared by the user after some delay to remove any spurious transitions caused by the configuration.

You may reconfigure the interrupt selector during other times, but spurious interrupt conditions may be detected by the CPU on the interrupts affected by the modified fields. For example, if EXT_INT4 is low, EXT_INT5 is high, and INT9 is remapped from EXT_INT4 to EXT_INT5, the low-to-high transition on INT9 is recognized as an interrupt and sets IF9.

Chapter 14

Power-Down Logic

This chapter describes the power-down modes.

Topic **Page**

14.1 Overview

Most of the operating power of CMOS logic is dissipated during circuit switching from one logic state to another. By preventing some or all of chip’s logic from switching, significant power savings can be realized without losing any data or operational context. PD1, PD2, and PD3 are three power-down modes available to perform this function. Power-down mode PD1 blocks the internal clock inputs at the boundary of the CPU, preventing most of its logic from switching. PD1 effectively shuts down the CPU. Additional power savings are accomplished in power-down mode PD2, where the entire on-chip clock structure (including multiple buffers) is “halted” at the output of the PLL (see Figure 14–1). PD3 is like PD2 but also disconnects the external clock source (CLKIN) from reaching the PLL. Wake-up from PD3 takes longer then wake-up from PD2 because the PLL needs to be re-locked, just as it does following power-up.

On the ’C6201/C6202/C6701, both the PD2 and PD3 signals also assert the PD pin for external recognition of these two power-down modes. Although the ’C6211/C6711 has power-down modes identical to the other devices, there is no PD pin driven externally. In addition to power-down modes described in this chapter, the IDLE instruction provides lower CPU power consumption by executing continuous NOPs. The IDLE instruction terminates only upon servicing an interrupt.

Figure 14–1. Power-Down Mode Logic

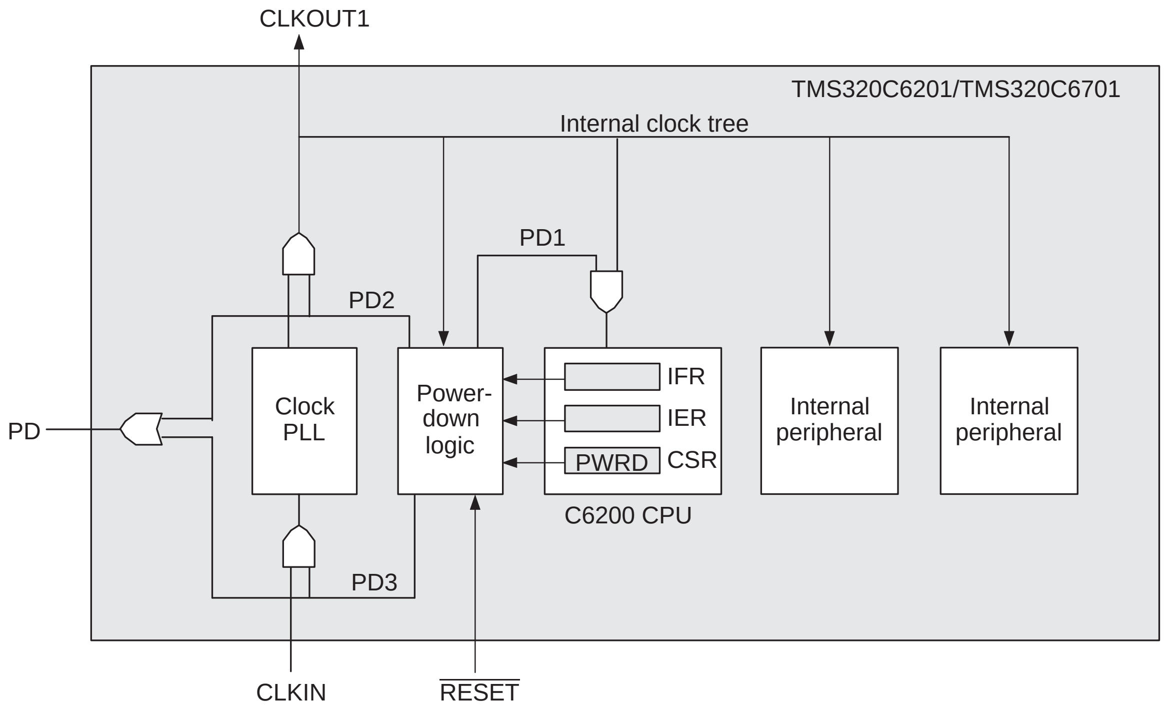


Figure 14–2. PWRD Field of the CSR Register

31 16	15	14	13	12	11	10	9 0
	rsvd	Enabled or non-enabled interrupt wake	Enabled interrupt wake	Pd3	Pd2	Pd1	

Table 14–1. Power-Down Mode and Wake-Up Selection

PRWD	Power-down mode/Wake-up method
000000	no power-down
001001	PD1 / wake by an enabled interrupt
010001	PD1 / wake by an enabled or non-enabled interrupt
011010	PD2
011100	PD3
other	reserved

14.2 Triggering, Wake-Up, and Effects

Power-down mode PD1 takes effect eight to nine clock cycles after the instruction that caused the power down (by setting the idle bits in the CSR). Use the following code segment to enter power down:

```
               B NextInst            ;branch does not effect program flow, but
               NOP                   ;   hides the move to the CSR in the delay
                                     ;   slots
               MVC Breg, CSR         ;power-down mode is set by this instruction
               NOP
               NOP
               NOP
NextInst:      NOP
               NOP5                  ;CPU notifies power-down logic to initiate
                                     ;   power down
               INSTR2                ;normal program exexution resumed here
```

The power-down modes and their wake-up methods are programmed by setting bits 10-15 of the control status register (CSR PWRD field). PD2 and PD3 modes can only be aborted by device reset, while PD1 mode can also be terminated by an enabled interrupt, or any interrupt (enabled or not), as directed by bits 13 and 14 of the CSR. When writing to CSR, all bits of the PWRD field should be set at the same time. Logic 0 should be used when writing to reserved fields (bit 15 of CSR).

The wake-up from PD1 can be triggered by either an enabled interrupt, or any interrupt (enabled or not). The first case is selected by writing a logic 1 to bit 13 of the Control Status Register (PWRD field), and the second case is selected by writing a logic 1 into bit 14 of CSR. If PD1 mode is terminated by a non-enabled interrupt, the program execution returns to the instruction following the NOP 9. Wake-up by an enabled interrupt executes the corresponding interrupt service fetch packet (ISFP) first, prior to returning to the instruction following the NOP 9. CSR register GIE bit and IER register NMIE bit must also be set in order for the ISFP to execute, otherwise execution returns to the previous point, rather than servicing the interrupt.

Table 14–2. Characteristics of the Power-Down Modes

Power-Down Mode	Trigger Action	Wake-up Method	Effect on Chip's Operation
PD1	write logic 001001b or 010001b to bits 15-10 of the CSR	internal interrupt, external interrupt or Reset	CPU halted (except for the interrupt logic)
PD2	write logic 011010b to bits 15-10 of the CSR	Reset only	Output clock from PLL is halted, stopping the internal clock structure from switching and resulting in the entire chip being halted. Signal terminal PD is driven high. All register and internal RAM contents are preserved. All signal terminals behave the same way as during Reset.
PD3	write logic 11100b to bits 15-10 of the CSR	Reset only	Input clock to the PLL stops generating clocks. Signal terminal PD is driven high. All register and internal RAM contents are preserved. All signal terminals behave the same way as during Reset. Following re-set, the PLL needs time to re-lock, just as it does following power-up.

14.3 Additional Power-Saving Modes for the TMS320C6202

In addition to the power down modes common to all of the C6x devices, the 'C6202 has the ability to turn off clocks to individual peripherals on the device. This feature allows the user to selectively turn off peripherals which are not being used for a specific application and not pay the extra price in power consumption for unused peripherals.

This method can have significant savings in power consumption. In a device which is as highly integrated as the C6000 series of DSPs a significant amount of power can be consumed in a reset or no activity state just due to the internal clock distribution. By selectively turning off unused portions of the device, the effects can be minimized.

Table 14–3 shows the peripheral power down register address location, and Figure 14–3 shows the register fields.

Table 14–3. TMS320C6202 Peripheral Power-Down Memory-Mapped Register

Byte Address	Field
019C 0200h	Peripheral Power-Down Control

Figure 14–3. Peripheral Power-Down Control Fields for the TMS320C6202

31 5	4	3	2	1	0
Reserved	PDMCSP2	PDMCSP1	PDMCSP0	PDEMIF	PDDMA
R,+0	RW,+1	RW,+1	RW,+1	RW,+1	RW,+1

Table 14–4 lists and describes the fields in the TMS320C6202 peripheral power-down memory-mapped register.

Table 14–4. Description of TMS320C6202 Power-Down Control Fields

Field	Description	Section
PDDMA	Enable/disable internal DMA clock PDDMA=0: internal DMA clock allowed to clock PDDMA=1: internal DMA clock disabled. DMA is not functional	14.3
PDEMIF	Enable/disable internal EMIF clock PDEMIF=0: internal EMIF clock allowed to clock PDEMIF=1: Internal EMIF clock disabled. EMIF is not functional. The HOLD condition which exists at power down will remain active and external clocks continue to clock.	14.3
PDMCSP0	Enable/disable internal McBSP0 clock PDMCSP0=0: Internal McBSP0 clock allowed to clock. PDMCSP0=1: Internal McBSP0 clock disabled. McBSP0 is not functional.	14.3
PDMCSP1	Enable/disable internal McBSP1 clock PDMCSP1=0: internal McBSP1 clock allowed to clock. PDMCSP1=1: internal McBSP1 clock disabled, McBSP1 is not functional.	14.3
PDMCSP2	Enable/disable internal McBSP2 clock PDMCSP2=0: internal McBSP2 clock allowed to clock PDMCSP2=1: internal McBSP2 clock disabled, McBSP2 is not functional	14.3

You must careful to not disable a portion of the device which is being used, since the peripheral in question will not be operational. A clock-off mode can be entered and exited depending on the needs of the application. For example, if an application does not need the serial ports, the ports can be disabled and then re-enabled when needed.

When re-enabling any of the PD bits, the CPU should wait at least 5 additional clock cycles before attempting to access that peripheral. This delay can be accomplished with a NOP 5 after any write to a peripheral power down register, as shown in Example 14–1.

Example 14–1. Assemble Code for Initializing Peripheral Power-Down Register

```
MVK    0x019C0200, Dest_Ptr_Reg
MVKH   0x019C0200, Dest_Ptr_Reg
STW    SrcReg,   *Dest_Ptr_Reg
NOP    5
```

Chapter 15

Designing for JTAG Emulation

This chapter assists you in meeting the design requirements of the XDS510 emulator with respect to JTAG designs and discusses the XDS510 cable (manufacturing part number 2617698-0001). This cable is identified by a label on the cable pod marked **JTAG 3/5V** and supports both standard 3-volt and 5-volt target system power inputs.

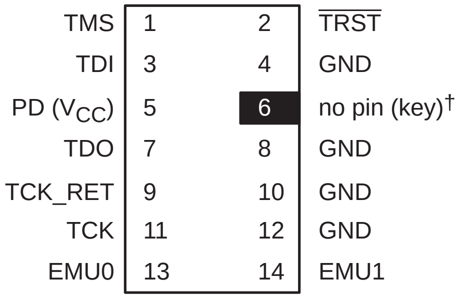
The term *JTAG,* as used in this book, refers to TI scan-based emulation, which is based on the IEEE 1149.1 standard.

Topic **Page**

15.1 Designing Your Target System’s Emulator Connector (14-Pin Header)

JTAG target devices support emulation through a dedicated emulation port. This port is a superset of the IEEE 1149.1 standard and is accessed by the emulator. To communicate with the emulator, **your target system must have a 14-pin header** (two rows of seven pins) with the connections that are shown in Figure 15–1. Table 15–1 describes the emulation signals.

Figure 15–1. 14-Pin Header Signals and Header Dimensions



Header Dimensions:
Pin-to-pin spacing, 0.100 in. (X,Y)
Pin width, 0.025-in. square post
Pin length, 0.235-in. nominal

† While the corresponding female position on the cable connector is plugged to prevent improper connection, the cable lead for pin 6 is present in the cable and is grounded, as shown in the schematics and wiring diagrams in this document.

Table 15–1. 14-Pin Header Signal Descriptions

Signal	Description	Emulator† State	Target† State
TMS	Test mode select	O	I
TDI	Test data input	O	I
TDO	Test data output	I	O
TCK	Test clock. TCK is a 10.368-MHz clock source from the emulation cable pod. This signal can be used to drive the system test clock	O	I
$\overline{\text{TRST}}$‡	Test reset	O	I
EMU0	Emulation pin 0	I	I/O
EMU1	Emulation pin 1	I	I/O
PD(V_{CC})	Presence detect. Indicates that the emulation cable is connected and that the target is powered up. PD should be tied to V_{CC} in the target system.	I	O
TCK_RET	Test clock return. Test clock input to the emulator. May be a buffered or unbuffered version of TCK.	I	O
GND	Ground		

† I = input; O = output

‡ Do not use pullup resistors on $\overline{\text{TRST}}$: it has an internal pulldown device. In a low-noise environment, $\overline{\text{TRST}}$ can be left floating. In a high-noise environment, an additional pulldown resistor may be needed. (The size of this resistor should be based on electrical current considerations.)

Although you can use other headers, recommended parts include:

straight header, unshrouded	DuPont Connector Systems part numbers:	65610–114
		65611–114
		67996–114
		67997–114

15.2 Bus Protocol

The IEEE 1149.1 specification covers the requirements for the test access port (TAP) bus slave devices and provides certain rules, summarized as follows:

❑ The TMS/TDI inputs are sampled on the rising edge of the TCK signal of the device.

❑ The TDO output is clocked from the falling edge of the TCK signal of the device.

When these devices are daisy-chained together, the TDO of one device has approximately a half TCK cycle setup to the next device's TDI signal. This type of timing scheme minimizes race conditions that would occur if both TDO and TDI were timed from the same TCK edge. The penalty for this timing scheme is a reduced TCK frequency.

The IEEE 1149.1 specification does not provide rules for bus master (emulator) devices. Instead, it states that it expects a bus master to provide bus slave compatible timings. The XDS510 provides timings that meet the bus slave rules.

15.3 IEEE 1149.1 Standard

For more information concerning the IEEE 1149.1 standard, contact IEEE Customer Service:

Address: IEEE Customer Service
445 Hoes Lane, PO Box 1331
Piscataway, NJ 08855-1331

Phone: (800) 678–IEEE in the US and Canada
(908) 981–1393 outside the US and Canada

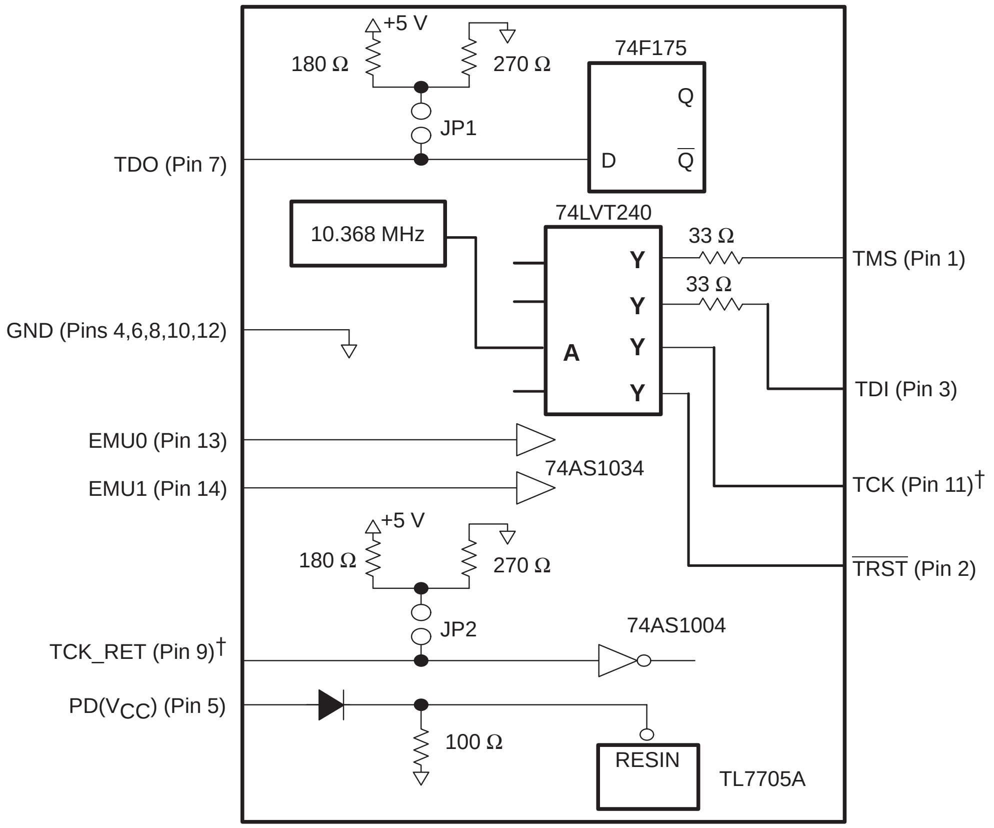
FAX: (908) 981–9667 Telex: 833233

15.4 JTAG Emulator Cable Pod Logic

Figure 15–2 shows a portion of the emulator cable pod. These are the functional features of the pod:

- ❑ Signals TDO and TCK_RET can be parallel-terminated inside the pod if required by the application. By default, these signals are not terminated.
- ❑ Signal TCK is driven with a 74LVT240 device. Because of the high-current drive (32 mA I_{OL}/I_{OH}), this signal can be parallel-terminated. If TCK is tied to TCK_RET, then you can use the parallel terminator in the pod.
- ❑ Signals TMS and TDI can be generated from the falling edge of TCK_RET, according to the IEEE 1149.1 bus slave device timing rules.
- ❑ Signals TMS and TDI are series-terminated to reduce signal reflections.
- ❑ A 10.368-MHz test clock source is provided. You may also provide your own test clock for greater flexibility.

Figure 15–2. JTAG Emulator Cable Pod Interface



† The emulator pod uses TCK_RET as its clock source for internal synchronization. TCK is provided as an optional target system test clock source.

15.5 JTAG Emulator Cable Pod Signal Timing

Figure 15–3 shows the signal timings for the emulator cable pod. Table 15–2 defines the timing parameters. These timing parameters are calculated from values specified in the standard data sheets for the emulator and cable pod and are for reference only. Texas Instruments does not test or guarantee these timings.

The emulator pod uses TCK_RET as its clock source for internal synchronization. TCK is provided as an optional target system test clock source.

Figure 15–3. JTAG Emulator Cable Pod Timings

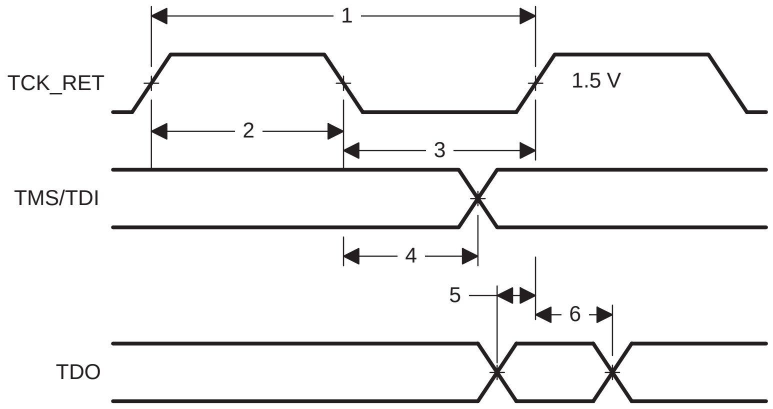


Table 15–2. Emulator Cable Pod Timing Parameters

No.	Reference	Description	Min	Max	Units
1	$t_{c(TCK)}$	TCK_RET period	35	200	ns
2	$t_{w(TCKH)}$	TCK_RET high-pulse duration	15		ns
3	$t_{w(TCKL)}$	TCK_RET low-pulse duration	15		ns
4	$t_{d(TMS)}$	Delay time, TMS/TDI valid from TCK_RET low	6	20	ns
5	$t_{su(TDO)}$	TDO setup time to TCK_RET high	3		ns
6	$t_{h(TDO)}$	TDO hold time from TCK_RET high	12		ns

15.6 Emulation Timing Calculations

The following examples help you calculate emulation timings in your system. For actual target timing parameters, see the appropriate device data sheets.

Assumptions:

$t_{su(TTMS)}$	Target TMS/TDI setup to TCK high	10 ns
$t_{d(TTDO)}$	Target TDO delay from TCK low	15 ns
$t_{d(bufmax)}$	Target buffer delay, maximum	10 ns
$t_{d(bufmin)}$	Target buffer delay, minimum	1 ns
$t_{(bufskew)}$	Target buffer skew between two devices in the same package: $[t_{d(bufmax)} - t_{d(bufmin)}] \times 0.15$	1.35 ns
$t_{(TCKfactor)}$	Assume a 40/60 duty cycle clock	0.4 (40%)

Given in Table 15–2 (on page 15-5):

$t_{d(TMSmax)}$	Emulator TMS/TDI delay from TCK_RET low, maximum	20 ns
$t_{su(TDOmin)}$	TDO setup time to emulator TCK_RET high, minimum	3 ns

There are two key timing paths to consider in the emulation design:

- ❑ The TCK_RET-to-TMS/TDI path, called $t_{pd(TCK\_RET–TMS/TDI)}$, and
- ❑ The TCK_RET-to-TDO path, called $t_{pd(TCK\_RET–TDO)}$.

Of the following two cases, the worst-case path delay is calculated to determine the maximum system test clock frequency.

Case 1: Single processor, direct connection, TMS/TDI timed from TCK_RET low.

$$t_{pd\,(TCK\_RET–TMS/TDI)} = \frac{\left[t_{d\,(TMSmax)} + t_{su\,(TTMS)}\right]}{t_{(TCKfactor)}} = \frac{[20ns + 10ns]}{0.4} = 75ns\ (13.3\ MHz)$$

$$t_{pd\,(TCK\_RET–TDO)} = \frac{\left[t_{d\,(TTDO)} + t_{su\,(TDOmin)}\right]}{t_{(TCKfactor)}} = \frac{[15ns + 3ns]}{0.4} = 45ns\ (22.2\ MHz)$$

In this case, the TCK_RET-to-TMS/TDI path is the limiting factor.

Case 2: Single/multiprocessor, TMS/TDI/TCK buffered input, TDO buffered output, TMS/TDI timed from TCK_RET low.

$$t_{pd\,(TCK\_RET–TMS/TDI)} = \frac{\left[t_{d\,(TMSmax)} + t_{su\,(TTMS)} + t_{(bufskew)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[20ns + 10ns + 1.35ns]}{0.4}$$

$$= 78.4ns\ (12.7\ MHz)$$

$$t_{pd\,(TCK\_RET–TDO)} = \frac{\left[t_{d\,(TTDO)} + t_{su\,(TDOmin)} + t_{d\,(bufmax)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[15ns + 3ns + 10ns]}{0.4}$$

$$= 70ns\ (14.3\ MHz)$$

In this case, the TCK_RET-to-TMS/TDI path is the limiting factor.

In a multiprocessor application, it is necessary to ensure that the EMU0–1 lines can go from a logic low level to a logic high level in less than 10 μs. This can be calculated as follows:

$$\begin{aligned} t_r &= 5(R_{pullup} \times N_{devices} \times C_{load\_per\_device}) \\ &= 5(4.7\ k\Omega \times 16 \times 15\ pF) \\ &= 5.64\ \mu s \end{aligned}$$

15.7 Connections Between the Emulator and the Target System

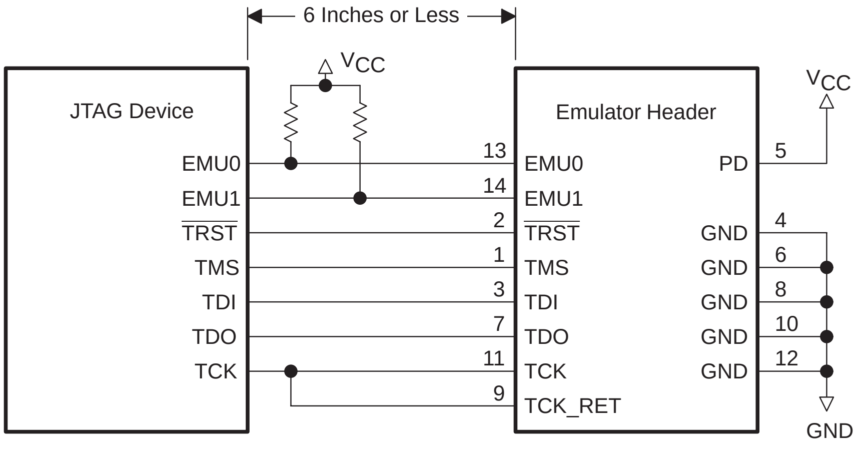
It is extremely important to provide high-quality signals between the emulator and the JTAG target system. Depending upon the situation, you must supply the correct signal buffering, test clock inputs, and multiple processor interconnections to ensure proper emulator and target system operation.

Signals applied to the EMU0 and EMU1 pins on the JTAG target device can be either input or output (I/O). In general, these two pins are used as both input and output in multiprocessor systems to handle global run/stop operations. EMU0 and EMU1 signals are applied only as inputs to the XDS510 emulator header.

15.7.1 Buffering Signals

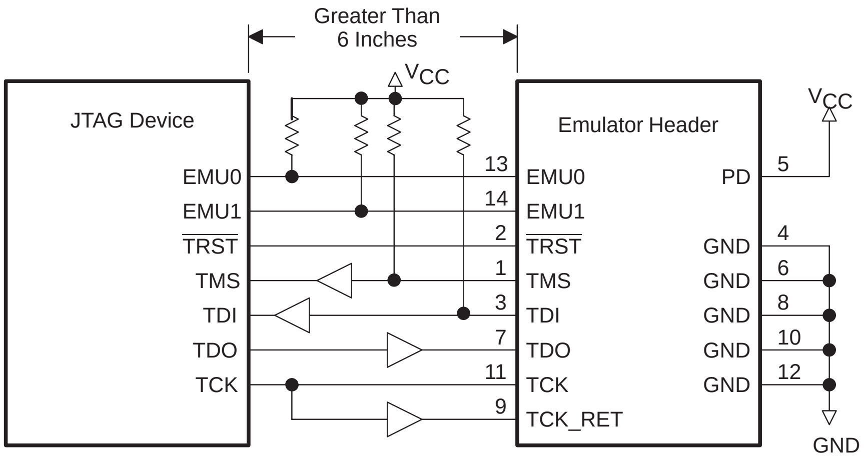
If the distance between the emulation header and the JTAG target device is greater than six inches, the emulation signals must be buffered. If the distance is less than six inches, no buffering is necessary. The following illustrations depict these two situations.

❑ **No signal buffering.** In this situation, the distance between the header and the JTAG target device should be no more than six inches.



The EMU0 and EMU1 signals must have pullup resistors connected to V_{CC} to provide a signal rise time of less than 10 µs. A 4.7-kΩ resistor is suggested for most applications.

- **Buffered transmission signals.** In this situation, the distance between the emulation header and the processor is greater than six inches. Emulation signals TMS, TDI, TDO, and TCK_RET are buffered through the same package.

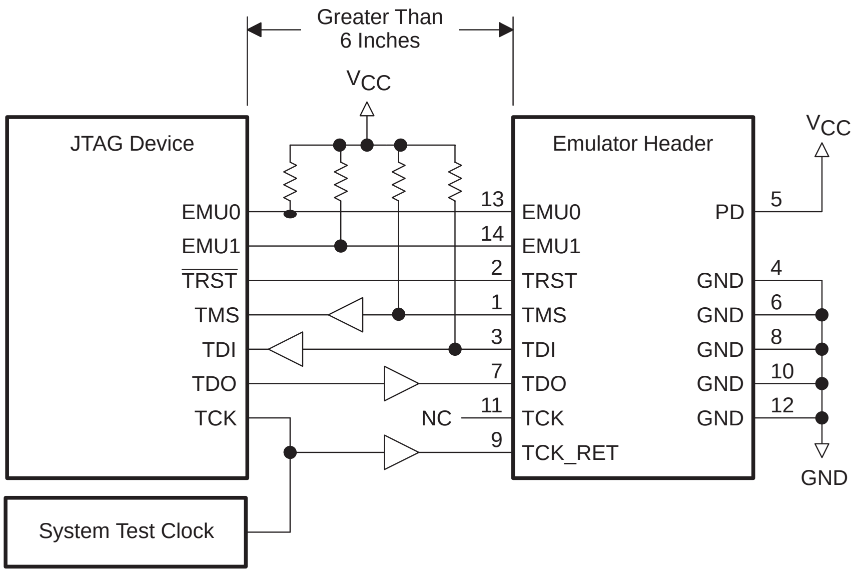


  - The EMU0 and EMU1 signals must have pullup resistors connected to V_{CC} to provide a signal rise time of less than 10 µs. A 4.7-kΩ resistor is suggested for most applications.
  - The input buffers for TMS and TDI should have pullup resistors connected to V_{CC} to hold these signals at a known value when the emulator is not connected. A resistor value of 4.7 kΩ or greater is suggested.
  - To have high-quality signals (especially the processor TCK and the emulator TCK_RET signals), you may have to employ special care when routing the PWB trace. You also may have to use termination resistors to match the trace impedance. The emulator pod provides optional internal parallel terminators on the TCK_RET and TDO. TMS and TDI provide fixed series termination.
  - Since $\overline{\text{TRST}}$ is an asynchronous signal, it should be buffered as needed to insure sufficient current to all target devices.

15.7.2 Using a Target-System Clock

Figure 15–4 shows an application with the system test clock generated in the target system. In this application, the TCK signal is left unconnected.

Figure 15–4. Target-System-Generated Test Clock



Note: When the TMS/TDI lines are buffered, pullup resistors should be used to hold the buffer inputs at a known level when the emulator cable is not connected.

There are two benefits to having the target system generate the test clock:

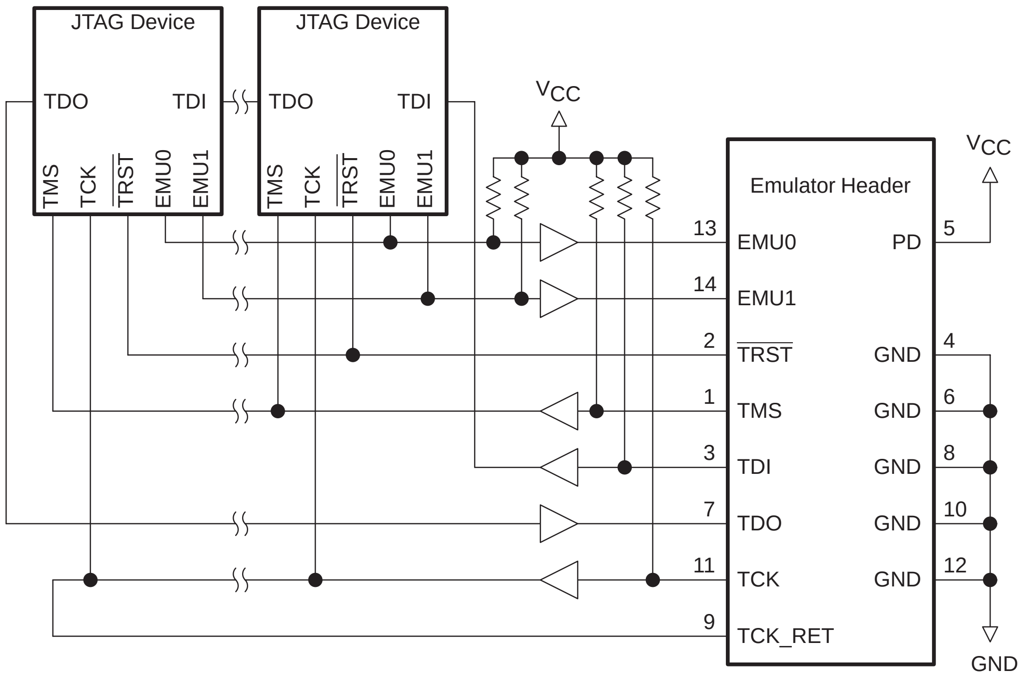
- ❑ The emulator provides only a single 10.368-MHz test clock. If you allow the target system to generate your test clock, you can set the frequency to match your system requirements.
- ❑ In some cases, you may have other devices in your system that require a test clock when the emulator is not connected. The system test clock also serves this purpose.

15.7.3 Configuring Multiple Processors

Figure 15–5 shows a typical daisy-chained multiprocessor configuration, which meets the minimum requirements of the IEEE 1149.1 specification. The emulation signals in this example are buffered to isolate the processors from the emulator and provide adequate signal drive for the target system. One of the benefits of this type of interface is that you can generally slow down the test clock to eliminate timing problems. You should follow these guidelines for multiprocessor support:

- The processor TMS, TDI, TDO, and TCK signals should be buffered through the same physical package for better control of timing skew.
- The input buffers for TMS, TDI, and TCK should have pullup resistors connected to V_{CC} to hold these signals at a known value when the emulator is not connected. A resistor value of 4.7 kΩ or greater is suggested.
- Buffering EMU0 and EMU1 is optional but highly recommended to provide isolation. These are not critical signals and do not have to be buffered through the same physical package as TMS, TCK, TDI, and TDO. Unbuffered and buffered signals are shown in this section (page 15-8 and page 15-9).

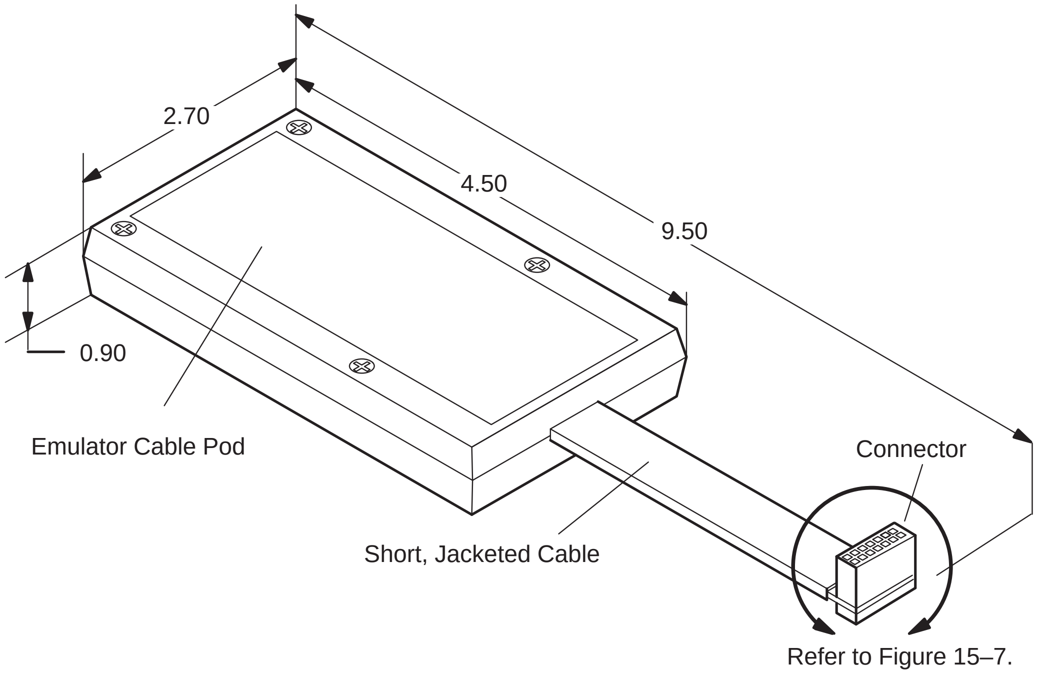
Figure 15–5. Multiprocessor Connections



15.8 Mechanical Dimensions for the 14-Pin Emulator Connector

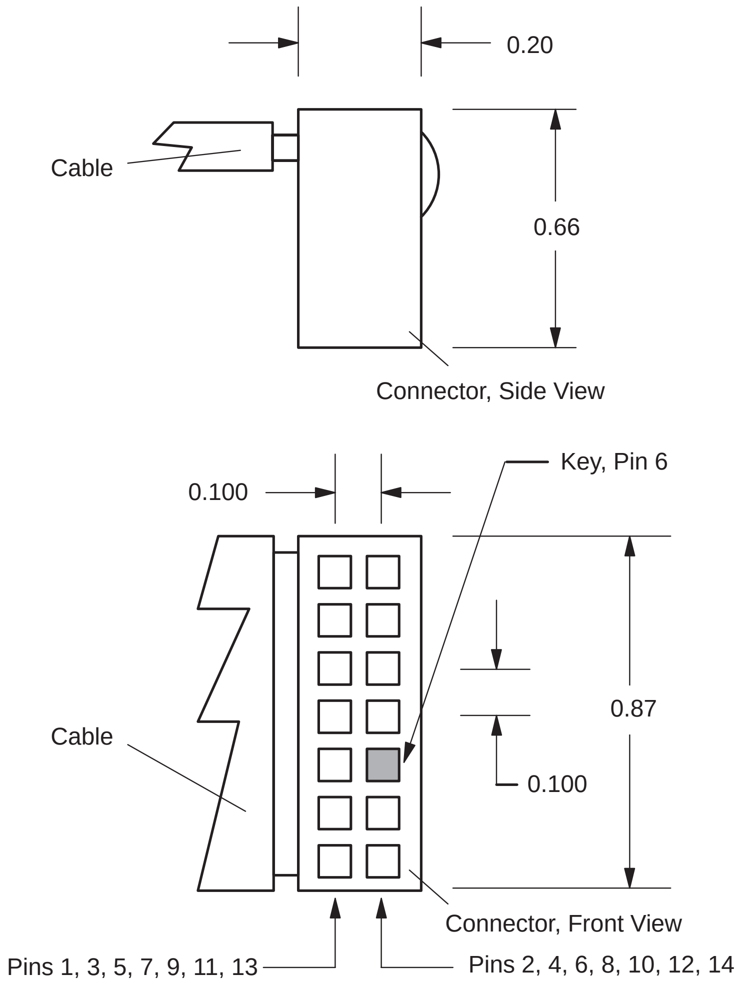
The JTAG emulator target cable consists of a 3-foot section of jacketed cable, an active cable pod, and a short section of jacketed cable that connects to the target system. The overall cable length is approximately 3 feet 10 inches. Figure 15–6 and Figure 15–7 (page 15-13) show the mechanical dimensions for the target cable pod and short cable. Note that the pin-to-pin spacing on the connector is 0.100 inches in both the X and Y planes. The cable pod box is nonconductive plastic with four recessed metal screws.

Figure 15–6. Pod/Connector Dimensions



Note: All dimensions are in inches and are nominal dimensions, unless otherwise specified.

Figure 15–7. 14-Pin Connector Dimensions



Note: All dimensions are in inches and are nominal dimensions, unless otherwise specified.

15.9 Emulation Design Considerations

This section describes the use and application of the scan path linker (SPL), which can simultaneously add all four secondary JTAG scan paths to the main scan path. It also describes the use of the emulation pins and the configuration of multiple processors.

15.9.1 Using Scan Path Linkers

You can use the TI ACT8997 scan path linker (SPL) to divide the JTAG emulation scan path into smaller, logically connected groups of 4 to 16 devices. As described in the *Advanced Logic and Bus Interface Logic Data Book* (literature number SCYD001), the SPL is compatible with the JTAG emulation scanning. The SPL is capable of adding any combination of its four secondary scan paths into the main scan path.

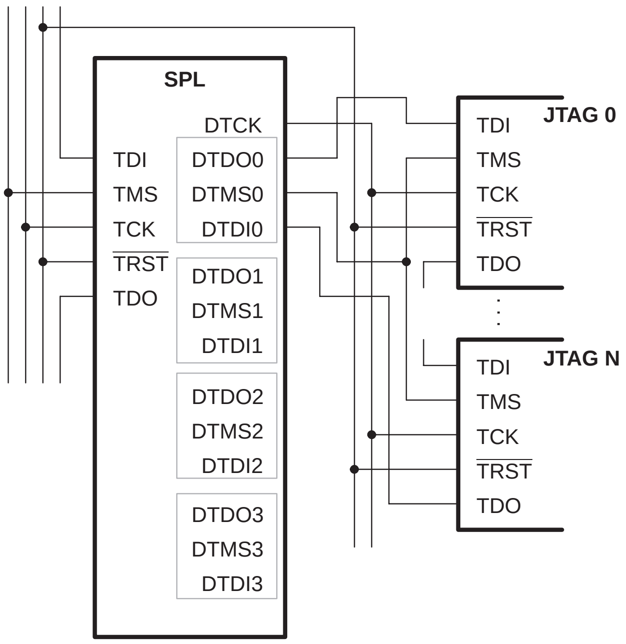
A system of multiple, secondary JTAG scan paths has better fault tolerance and isolation than a single scan path. Since an SPL has the capability of adding all secondary scan paths to the main scan path simultaneously, it can support global emulation operations, such as starting or stopping a selected group of processors.

TI emulators do not support the nesting of SPLs (for example, an SPL connected to the secondary scan path of another SPL). However, you can have multiple SPLs on the main scan path.

Although the ACT8999 scan path selector is similar to the SPL, it can add only one of its secondary scan paths at a time to the main JTAG scan path. Thus, global emulation operations are not assured with the scan path selector. For this reason, scan path selectors are not supported.

You can insert an SPL on a backplane so that you can add up to four device boards to the system without the jumper wiring required with nonbackplane devices. You connect an SPL to the main JTAG scan path in the same way you connect any other device. Figure 15–8 shows you how to connect a secondary scan path to an SPL.

Figure 15–8. Connecting a Secondary JTAG Scan Path to an SPL



The $\overline{\text{TRST}}$ signal from the main scan path drives all devices, even those on the secondary scan paths of the SPL. The TCK signal on each target device on the secondary scan path of an SPL is driven by the SPL's DTCK signal. The TMS signal on each device on the secondary scan path is driven by the respective DTMS signals on the SPL.

DTDO on the SPL is connected to the TDI signal of the first device on the secondary scan path. DTDI on the SPL is connected to the TDO signal of the last device in the secondary scan path. Within each secondary scan path, the TDI signal of a device is connected to the TDO signal of the device before it. If the SPL is on a backplane, its secondary JTAG scan paths are on add-on boards; if signal degradation is a problem, you may need to buffer both the $\overline{\text{TRST}}$ and DTCK signals. Although less likely, you may also need to buffer the DTMS*n* signals for the same reasons.

15.9.2 Emulation Timing Calculations for SPL

The following examples help you to calculate the emulation timings in the SPL secondary scan path of your system. For actual target timing parameters, see the appropriate device data sheets.

Assumptions:

$t_{su(TTMS)}$	Target TMS/TDI setup to TCK high	10 ns
$t_{d(TTDO)}$	Target TDO delay from TCK low	15 ns
$t_{d(bufmax)}$	Target buffer delay, maximum	10 ns
$t_{d(bufmin)}$	Target buffer delay, minimum	1 ns
$t_{(bufskew)}$	Target buffer skew between two devices in the same package: $[t_{d(bufmax)} – t_{d(bufmin)}] \times 0.15$	1.35 ns
$t_{(TCKfactor)}$	Assume a 40/60 duty cycle clock	0.4 (40%)

Given in the SPL data sheet:

$t_{d(DTMSmax)}$	SPL DTMS/DTDO delay from TCK low, maximum	31 ns
$t_{su(DTDLmin)}$	DTDI setup time to SPL TCK high, minimum	7 ns
$t_{d(DTCKHmin)}$	SPL DTCK delay from TCK high, minimum	2 ns
$t_{d(DTCKLmax)}$	SPL DTCK delay from TCK low, maximum	16 ns

There are two key timing paths to consider in the emulation design:

- The TCK-to-DTMS/DTDO path, called $t_{pd(TCK–DTMS)}$, and
- The TCK-to-DTDI path, called $t_{pd(TCK–DTDI)}$.

Of the following two cases, the worst-case path delay is calculated to determine the maximum system test clock frequency.

Case 1: Single processor, direct connection, DTMS/DTDO timed from TCK low.

$$t_{pd\,(TCK–DTMS)} = \frac{\left[t_{d\,(DTMSmax)} + t_{d\,(DTCKHmin)} + t_{su\,(TTMS)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[31ns + 2ns + 10ns]}{0.4}$$

$$= 107.5ns\ (9.3\ MHz)$$

$$t_{pd\,(TCK–DTDI)} = \frac{\left[t_{d\,(TTDO)} + t_{d\,(DTCKLmax)} + t_{su\,(DTDLmin)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[15ns + 16ns + 7ns]}{0.4}$$

$$= 9.5ns\ (10.5\ MHz)$$

In this case, the TCK-to-DTMS/DTDL path is the limiting factor.

Case 2: Single/multiprocessor, DTMS/DTDO/TCK buffered input, DTDI buffered output, DTMS/DTDO timed from TCK low.

$$t_{pd\,(TCK–TDMS)} = \frac{\left[t_{d\,(DTMSmax)} + t_{(DTCKHmin)} + t_{su\,(TTMS)} + t_{(bufskew)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[31ns + 2ns + 10ns + 1.35ns]}{0.4}$$

$$= 110.9ns\ (9.0\ MHz)$$

$$t_{pd\,(TCK–DTDI)} = \frac{\left[t_{d\,(TTDO)} + t_{d\,(DTCKLmax)} + t_{su\,(DTDLmin)} + t_{d\,(bufskew)}\right.}{t_{(TCKfactor)}}$$

$$= \frac{[15ns + 15ns + 7ns + 10ns]}{0.4}$$

$$= 120ns\ (8.3\ MHz)$$

In this case, the TCK-to-DTDI path is the limiting factor.

15.9.3 Using Emulation Pins

The EMU0/1 pins of TI devices are bidirectional, three-state output pins. When in an inactive state, these pins are at high impedance. When the pins are active, they function in one of the two following output modes:

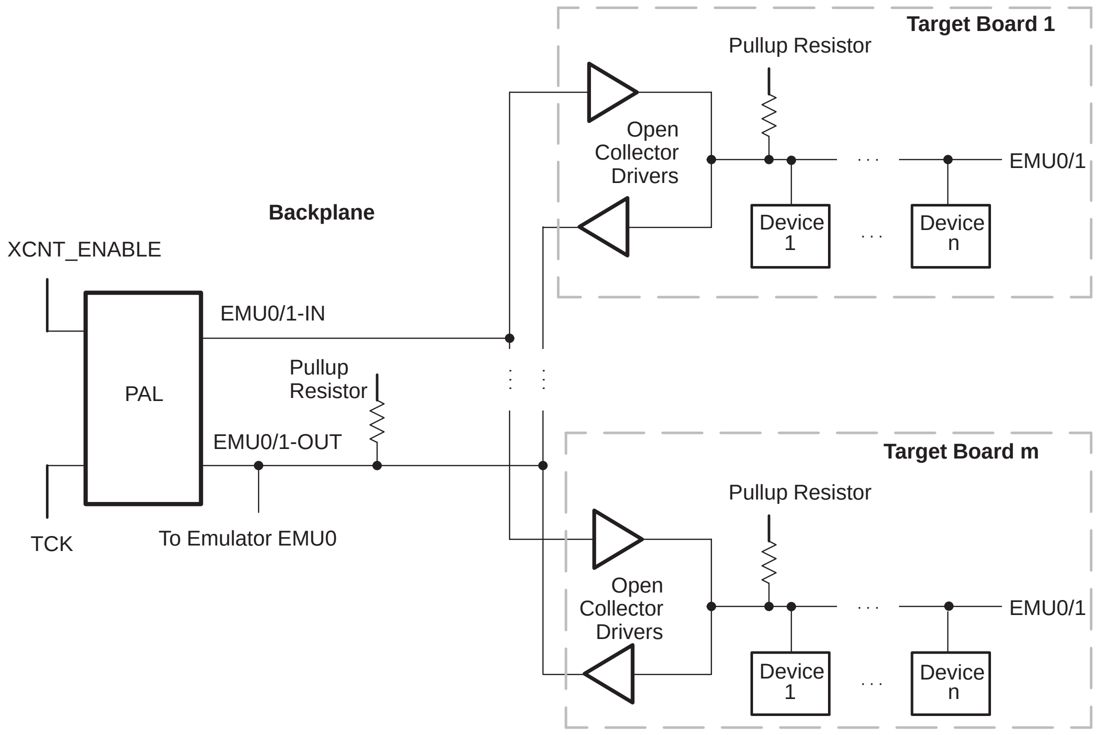
- **Signal Event**
  The EMU0/1 pins can be configured via software to signal internal events. In this mode, driving one of these pins low can cause devices to signal such events. To enable this operation, the EMU0/1 pins function as open-collector sources. External devices such as logic analyzers can also be connected to the EMU0/1 signals in this manner. If such an external source is used, it must also be connected via an open-collector source.

- **External Count**
  The EMU0/1 pins can be configured via software as totem-pole outputs for driving an external counter. If the output of more than one device is configured for totem-pole operation, then these devices can be damaged. The emulation software detects and prevents this condition. However, the emulation software has no control over external sources on the EMU0/1 signal. Therefore, all external sources must be inactive when any device is in the external count mode.

TI devices can be configured by software to halt processing if their EMU0/1 pins are driven low. This feature, in combination with the use of the signal event output mode, allows one TI device to halt all other TI devices on a given event for system-level debugging.

If you route the EMU0/1 signals between boards, they require special handling because these signals are more complex than normal emulation signals. Figure 15–9 shows an example configuration that allows any processor in the system to stop any other processor in the system. Do not tie the EMU0/1 pins of more than 16 processors together in a single group without using buffers. Buffers provide the crisp signals that are required during a RUNB (run benchmark) debugger command or when the external analysis counter feature is used.

Figure 15–9. EMU0/1 Configuration



Notes:
1) The low time on EMUx-IN should be at least one TCK cycle and less than 10 μs. Software will set the EMUx-OUT pin to a high state.
2) To enable the open-collector driver and pullup resistor on EMU1 to provide rising/falling edges of less than 25 ns, the modification shown in this figure is suggested. Rising edges slower than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used.

These seven important points apply to the circuitry shown in Figure 15–9 and Figure 15–10 , and the timing shown in Figure 15–11:

❑ Open-collector drivers isolate each board. The EMU0/1 pins are tied together on each board.

❑ At the board edge, the EMU0/1 signals are split to provide IN/OUT. This is required to prevent the open-collector drivers from acting as a latch that can be set only once.

❑ The EMU0/1 signals are bused down the backplane. Pullup resistors are installed as required.

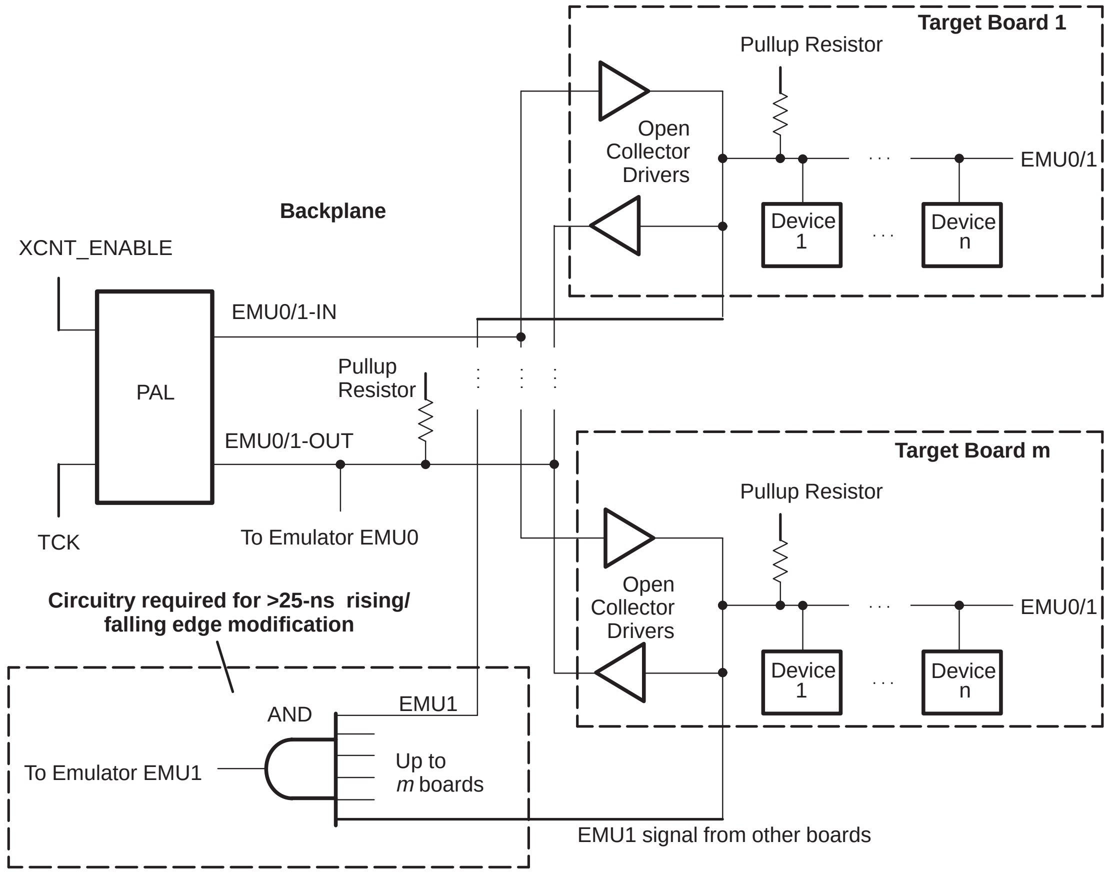
❑ The bused EMU0/1 signals go into a PAL® device, whose function is to generate a low pulse on the EMU0/1-IN signal when a low level is detected on the EMU0/1-OUT signal. This pulse must be longer than one TCK period to affect the devices, but less than 10 µs to avoid possible conflicts or retriggering, once the emulation software clears the device's pins.

- ❑ During a RUNB debugger command or other external analysis count, the EMU0/1 pins on the target device become totem-pole outputs. The EMU1 pin is a ripple carry-out of the internal counter. EMU0 becomes a *processor-halted* signal. During a RUNB or other external analysis count, the EMU0/1-IN signal to all boards must remain in the high (disabled) state. You must provide some type of external input (XCNT_ENABLE) to the PAL to disable the PAL from driving EMU0/1-IN to a low state.

- ❑ If sources other than TI processors (such as logic analyzers) are used to drive EMU0/1, their signal lines must be isolated by open-collector drivers and be inactive during RUNB and other external analysis counts.

- ❑ You must connect the EMU0/1-OUT signals to the emulation header or directly to a test bus controller.

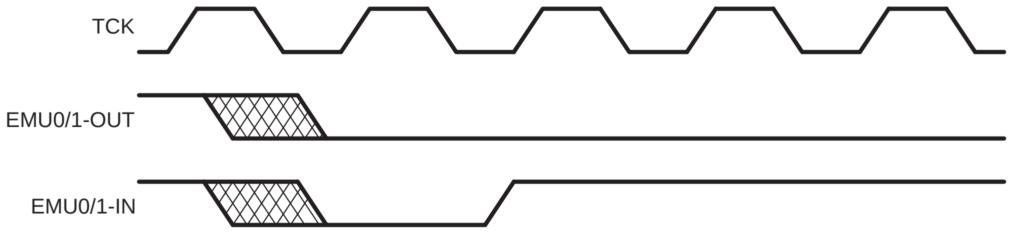
Figure 15–10. EMU0/1 Configuration With Additional AND Gate to Meet Timing Requirements



Notes: 1) The low time on EMUx–IN should be at least one TCK cycle and less than 10 μs. Software will set the EMUx–OUT pin to a high state.

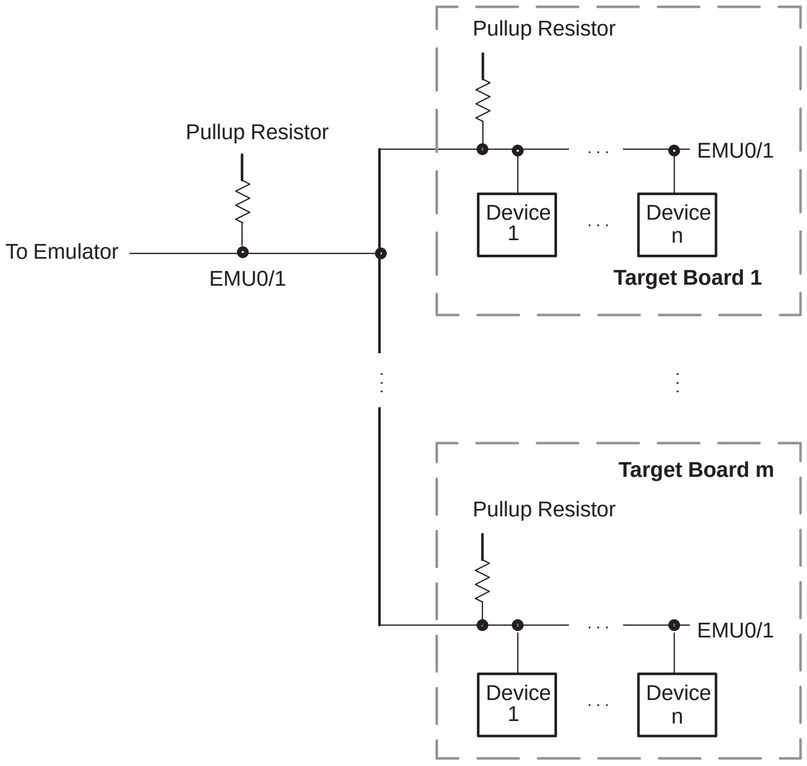
2) To enable the open-collector driver and pullup resistor on EMU1 to provide rising/falling edges of less than 25 ns, the modification shown in this figure is suggested. Rising edges slower than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used.

Figure 15–11. Suggested Timings for the EMU0 and EMU1 Signals



If having devices on one target board stopped by devices on another target board via the EMU0/1 signals is not important, then the circuit in Figure 15–12 can be used. In this configuration, the global-stop capability is lost. It is important not to overload EMU0/1 with more than 16 devices.

Figure 15–12. EMU0/1 Configuration Without Global Stop

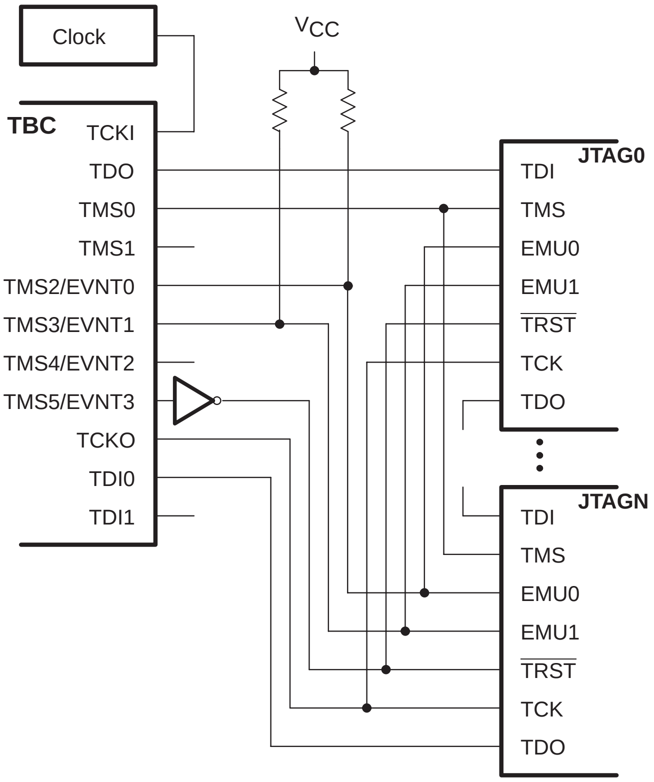


Note: The open-collector driver and pullup resistor on EMU1 must be able to provide rising/falling edges of less than 25 ns. Rising edges slower than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used. If this condition cannot be met, then the EMU0/1 signals from the individual boards should be ANDed together (as shown in Figure 15–10) to produce an EMU0/1 signal for the emulator.

15.9.4 Performing Diagnostic Applications

For systems that require built-in diagnostics, it is possible to connect the emulation scan path directly to a TI ACT8990 test bus controller (TBC) instead of the emulation header. The TBC is described in the Texas Instruments *Advanced Logic and Bus Interface Logic Data Book* (literature number SCYD001). Figure 15–13 shows the scan path connections of *n* devices to the TBC.

Figure 15–13. TBC Emulation Connections for n JTAG Scan Paths



In the system design shown in Figure 1–13, the TBC emulation signals TCKI, TDO, TMS0, TMS2/EVNT0, TMS3/EVNT1, TMS5/EVNT3, TCKO, and TDI0 are used, and TMS1, TMS4/EVNT2, and TDI1 are not connected. The target devices' EMU0 and EMU1 signals are connected to V_{CC} through pullup resistors and tied to the TBC's TMS2/EVNT0 and TMS3/EVNT1 pins, respectively. The TBC's TCKI pin is connected to a clock generator. The TCK signal for the main JTAG scan path is driven by the TBC's TCKO pin.

On the TBC, the TMS0 pin drives the TMS pins on each device on the main JTAG scan path. TDO on the TBC connects to TDI on the first device on the main JTAG scan path. TDI0 on the TBC is connected to the TDO signal of the last device on the main JTAG scan path. Within the main JTAG scan path, the TDI signal of a device is connected to the TDO signal of the device before it. $\overline{\text{TRST}}$ for the devices can be generated either by inverting the TBC's TMS5/EVNT3 signal for software control or by logic on the board itself.

Index